A new interior-point approach for large separable convex quadratic two-stage stochastic problems

Jordi Castro Paula de la Lama
Dept. of Statistics and Operations Research
Universitat Politècnica de Catalunya
Barcelona, Catalonia
`jordi.castro@upc.edu` `paula.de.la.lama@upc.edu`

Research Report UPC-DEIO-JC DR 2020-01
April 2020; updated September 2020

A new interior-point approach for large separable convex quadratic two-stage stochastic problems

Jordi Castro* Paula de la Lama-Zubirán
Dept. of Statistics and Operations Research
Universitat Politècnica de Catalunya
Jordi Girona 1–3, 08034 Barcelona, Catalonia
jordi.castro@upc.edu paula.de.la.lama@upc.edu

Two-stage stochastic models give rise to very large optimization problems. Several approaches have been devised for efficiently solving them, including interior-point methods (IPMs). However, using IPMs, the linking columns associated to first-stage decisions cause excessive fill-in for the solution of the normal equations. This downside is usually alleviated if variable splitting is applied to first-stage variables. This work presents a specialized IPM that applies variable splitting and exploits the structure of the deterministic equivalent of the stochastic problem. The specialized IPM combines Cholesky factorizations and preconditioned conjugate gradients for solving the normal equations. This specialized IPM outperforms other approaches when the number of first-stage variables is large enough. This paper provides computational results for two stochastic problems: (1) a supply chain system and (2) capacity expansion in an electric system. Both linear and convex quadratic formulations were used, obtaining instances of up to 38 million variables and six million constraints. The computational results show that our procedure is more efficient than alternative state-of-the-art IPM implementations (e.g., CPLEX) and other specialized solvers for stochastic optimization.

Keywords: interior-point methods; stochastic optimization; structured problems; large-scale optimization

AMS Subject Classification: 90C51, 90C15, 90C06

1. Introduction

Stochastic programming is a framework for modelling optimization problems with uncertain data, and it has a wide range of applications in real-life situations. The variability in these models may appear in the objective or the constraints of the decision-making process, and it is due to their relationships to random elements [4]. Such problems can be solved for realistic instances—which are both large and complex—because advances in computing power and algorithm development allow finding solutions in a reasonable amount of time. However, larger instances increasingly demand more efficient solution techniques [1].

A two-stage stochastic model includes two types of variables. The *first-stage variables* occur at the beginning of the period and are the decisions to be made before the outcome of random events; and the *second-stage variables* are decided after the outcomes of uncertain events are known. In the extensive form of the model, all scenarios (i.e., possible realization of the random elements) are incorporated explicitly. This entails a special structure of the constraints matrix with separable blocks.

*Corresponding author

Two of the most common structures in large-scale optimization correspond to problems with *linking variables* between groups of constraints blocks (also named *dual block-angular* structure), and *linking constraints* between blocks (or *primal block-angular* structure) [6]. Two-stage stochastic problems belong to the class of problems with linking variables. The two classical methods of resolution for each type of structure are Benders decomposition for dual block-angular [3], and Dantzig-Wolfe for primal block-angular [13]. Interior-point methods (IPMs) [31] exploiting problem structure is a third technique that has also proven to be useful for these kinds of problems [8, 17–19] Hence, large-scale refers not only to size but also to problem structure.

Linking variables of two-stage stochastic problems are usually dense columns, thus making them difficult to be solved by standard IPMs. The reformulation technique used in [24], named *splitting formulation*, removes this issue by considering copies of the first-stage variables, equating them through simple and sparse linking constraints. The specialized IPM implemented in the BlockIP package is tailored for problems with primal block-angular structure and linking constraints. It solves the normal equations by combining Cholesky factorizations for the diagonal blocks and a preconditioned conjugate gradient (PCG) for the linking constraints [8]. The preconditioner considered has proven to perform efficiently when the linking constraints are, precisely, simple and sparse [10, 11].

Therefore, the approach we propose in this works aims to efficiently solve two-stage stochastic problems by using the algorithm of the BlockIP solver for a *splitting formulation* of two-stage stochastic problems. Although this approach is not expected to be the most efficient algorithm for any stochastic problem, we will show that, in problems with a large enough number of first-stages variables, it outperforms the alternative state-of-the-art methods. Two standard problems from the literature were used to test this approach: a supply chain model, and a capacity expansion in an electric system. Both linear and quadratic formulations were used to obtain instances of up to 38 million variables and six million constraints.

The remainder of this document is organized as follows. Section 2 reviews the two-stage stochastic problem and its splitting formulation. Section 3 outlines the specialized interior-point method in BlockIP. Section 4 provides preliminary computational results comparing our approach with other specialized methods for two-stage stochastic problems in the solution of standard instances from the literature. Next, Section 5 presents the two stochastic problems and the computational results obtained, for both linear and quadratic instances. Finally, Section 6 summarizes the main outcomes of this paper and suggests directions for further research.

2. Two-stage stochastic optimization problems

The general idea behind a two-stage stochastic formulation is to compute some initial (first-stage) decisions that must be made prior to the occurrence of uncertainty, taking into account the expected value of future recourse (second-stage) actions. This work focuses on two-stage stochastic problems with either linear or convex quadratic objective functions. The *first-stage* problem manages the so-called "here-and-now" decisions. The *second-stage problem* refers to "wait-and-see" decisions, which represent the recourse actions. It deals with uncertainty by analyzing possible outcomes [4].

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

$$\begin{aligned} \min_x \ & c^\top x + \tfrac{1}{2} x^\top D_x x + \mathcal{Q}(x) \\ \text{s. to } & Mx = b \\ & u_x \geq x \geq 0, \end{aligned} \tag{1}$$

where

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

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 $c \in \mathbb{R}^{n_x}$, $D_x \in \mathbb{R}^{n_x \times n_x}$, $b \in \mathbb{R}^{m_x}$, $M \in \mathbb{R}^{m_x \times n_x}$, where D_x 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, $q_\xi \in \mathbb{R}^{n_y}$, $D_\xi \in \mathbb{R}^{n_y \times n_y}$, $W \in \mathbb{R}^{m_y \times n_y}$, $T_\xi \in \mathbb{R}^{m_y \times n_x}$, and $h_\xi \in \mathbb{R}^{m_y}$; D_ξ is symmetric and positive definite. The stochastic random vector is comprised of q_ξ, D_ξ, h_ξ and T_ξ.

For some particular problems (i.e., simple problems usually involving only a few variables), a closed form solution can be obtained for $Q(x,\xi) = q_\xi^\top y^* + \frac{1}{2} {y^*}^\top D_\xi y^*$, where y^* is the optimum of the second-stage future decisions. In these cases, if the vector ξ contains only a few random variables, it might be possible to compute $\mathcal{Q}(x) = E_\xi[Q(x,\xi)]$ by solving the low-dimensional integral associated to the expectation operator, thus obtaining an analytical expression for $\mathcal{Q}(x)$. This would allow the solution of (1) only in terms of the first-stage decisions. For example, the stochastic optimization formulation of the *newsvendor* model can be solved analytically using the above procedure (see [4][Chapter 1]). In general, however, no closed form exists for $Q(x,\xi)$, or, even if it exists, the computation of the expectation cannot be done analytically, 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 $\xi_i, i = 1, \ldots, k$ is usually known as a scenario. Next, second-stage variables and constraints are replicated for each scenario (i.e., $y_i, i = 1, \ldots, k$ for variables), combining problems (1) and (2) into a single one, as follows:

$$\begin{array}{ll} \min\limits_{x,y_i} & c^\top x + \dfrac{1}{2} x^\top D_x x + \displaystyle\sum_{i=1}^{k} p_i \left(q_{\xi_i}^\top y_i + \dfrac{1}{2} y_i^\top D_{\xi_i} y_i \right) \\ \text{s. to} & Mx = b \\ & u_x \geq x \geq 0 \\ & \left. \begin{array}{l} T_{\xi_i} x + W y_i = h_{\xi_i} \\ u_y \geq y_i \geq 0 \end{array} \right\} i = 1, \ldots, k. \end{array} \tag{3}$$

For real problems, (3) can be very large and needs to be solved by specialized procedures that exploit the particular problem structure [4, Ch. 5–8], [22, 27].

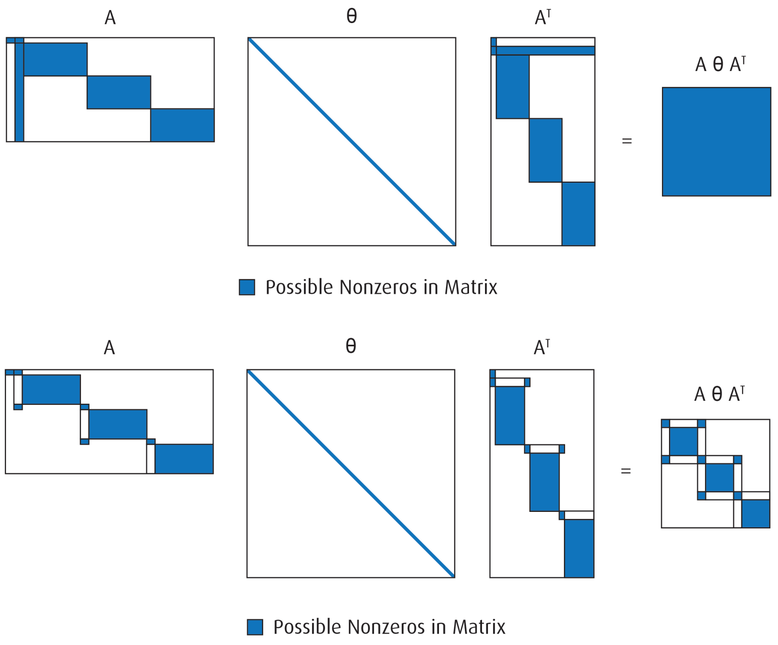


Figure 1.: Multiplication of $A\Theta T^T$ with and without splitting [24]

2.1. *The constraints structure of the two-stage problem*

The constraints matrix in (3) shows the following dual block-angular structure

$$A = \begin{array}{c} \begin{array}{ccccc} x & y_1 & y_2 & \cdots & y_k \end{array} \\ \begin{bmatrix} M & & & & \\ T_1 & W & & & \\ T_2 & & W & & \\ \vdots & & & \ddots & \\ T_k & & & & W \end{bmatrix} \end{array}, \tag{4}$$

where T_{ξ_i} in (3) has been renamed as T_i to simplify the notation. As shown in the top picture of Figure 1, the linking columns associated with the first-stage variables give 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. This is true even for good permutation matrices. We note that D_x and D_ξ must be diagonal matrices if Θ is assumed to be diagonal; indeed, D_x and D_ξ are considered diagonal in the specialized IPM of Section 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 $AA^\top$ [24].

2.1.1. *Splitting Formulation*

The linking columns ensure that the first-stage decision variables x are feasible under each future scenario. Since the same first-stage decision variables x are used throughout, those necessarily satisfy the nonanticipativity requirement. Second-stage decisions $y_i, i = 1, \ldots, k$, depend on both the first-stage variables and the realization of stochastic events.

In [24], variable splitting was suggested for reducing fill-in. Two variants were attempted: full and partial splitting. The full version splits all first-stage variables, while in partial splitting only the dense first-stage variables are replicated. We will make use of full splitting, which is more appropriate than partial splitting for the specialized interior-point algorithm in this work, because it 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 [24]. The bottom picture of Figure 1 shows the effect of splitting on the normal equations matrix.

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

$$\begin{array}{ll}\min\limits_{x_i,y_i} & c^\top x_1 + \frac{1}{2} x_1^\top D_x x_1 + \sum_{i=1}^{k} p_i \left(q_{\xi_i}^\top y_i + \frac{1}{2} y_i^\top D_{\xi_i} y_i \right) \\ \text{s. to} & M x_1 = b \\ & u_x \geq x_1 \geq 0 \\ & \left.\begin{array}{l} T_{\xi_i} x_i + W y_i = h_{\xi_i} \\ u_y \geq y_i \geq 0 \end{array}\right\} i = 1, \dots, k \\ & \left.\begin{array}{l} x_1 - x_i = 0 \\ u_x \geq x_i \geq 0 \end{array}\right\} i = 2, \dots, k. \end{array} \tag{5}$$

Reordering columns by the number of scenarios, the constraint matrix in (5) can be written as:

$$A_{(5)} = \begin{array}{c} \begin{array}{cccccccc} x_1 & y_1 & x_2 & y_2 & \cdots & x_k & y_k \end{array} \\ \left[\begin{array}{ccccccc} M & & & & & & \\ T_1 & W & & & & & \\ & & T_2 & W & & & \\ & & & & \ddots & & \\ & & & & & T_k & W \\ I & & -I & & & & \\ \vdots & & & & \ddots & & \\ I & & & & & -I & \end{array}\right] \end{array}. \tag{6}$$

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 (5) with $x_i = x_{i+1}, i = 1, \dots, k-1$, thus obtaining

$$\begin{array}{ll}\min\limits_{x_i,y_i} & c^\top x_1 + \frac{1}{2} x_1^\top D_x x_1 + \sum_{i=1}^{k} p_i \left(q_{\xi_i}^\top y_i + \frac{1}{2} y_i^\top D_{\xi_i} y_i \right) \\ \text{s. to} & M x_1 = b \\ & u_x \geq x_1 \geq 0 \\ & \left.\begin{array}{l} T_{\xi_i} x_i + W y_i = h_{\xi_i} \\ u_y \geq y_i \geq 0 \end{array}\right\} i = 1, \dots, k \\ & \left.\begin{array}{l} x_i - x_{i+1} = 0 \\ u_x \geq x_i \geq 0 \end{array}\right\} i = 1, \dots, k-1. \end{array} \tag{7}$$

The constraints matrix of formulation (7) is

$$A_{(7)} = \begin{array}{c} \begin{array}{cccccccccc} x_1 & y_1 & x_2 & y_2 & x_3 & y_3 & \cdots & x_{k-1} & y_{k-1} & x_k \; y_k \end{array} \\ \left[\begin{array}{ccccccccccc} M & & & & & & & & & & \\ T_1 & W & & & & & & & & & \\ & & T_2 & W & & & & & & & \\ & & & & T_3 & W & & & & & \\ & & & & & & \ddots & & & & \\ & & & & & & & T_{k-1} & W & & \\ & & & & & & & & & T_k & W \\ I & & -I & & & & & & & & \\ & & I & & -I & & & & & & \\ & & & & & & \ddots & & & & \\ & & & & & & & I & & -I & \end{array}\right] \end{array} . \tag{8}$$

Although (5) and (7) are equivalent, the latter is computationally more efficient for the specialized IPM that will be used, as will be shown in Subsection 3.2.

Both constraint matrices (6) and (8) have a primal block-angular structure with $(k-1)n_x$ very sparse linking constraints. The linear programming problems (5) and (7) match the following general block-angular formulation:

$$\begin{array}{rl} \min\limits_{x^1,\ldots,x^k,x^0} & \sum\limits_{i=1}^{k} \left({c^i}^\top x^i + \frac{1}{2} {x^i}^\top Q^i x^i \right) \\ \text{s. to} & \begin{bmatrix} N_1 & & & & \\ & N_2 & & & \\ & & \ddots & & \\ & & & N_k & \\ R_1 & R_2 & \ldots & R_k & I \end{bmatrix} \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^k \\ x^0 \end{bmatrix} = \begin{bmatrix} b^1 \\ b^2 \\ \vdots \\ b^k \\ b^0 \end{bmatrix} \\ & 0 \le x^i \le u^i \quad i = 1,\ldots,k \\ & 0 \le x^0 \le b^0. \end{array} \tag{9}$$

For both formulations (5) and (7) $x^i, u^i \in \mathbb{R}^{n_x+n_y}$ are, respectively, $(x_i^\top \;\; y_i^\top)^\top$ and $(u_x^\top \;\; u_y^\top)^\top$ $i = 1,\ldots,k$; 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 $i = 2,\ldots,k$. The quadratic cost matrices $Q^i \in \mathbb{R}^{(n_x+n_y)\times(n_x+n_y)}$ are $Q^1 = \begin{bmatrix} D_x & \\ & D_{\xi_1} \end{bmatrix}$, and $Q^i = \begin{bmatrix} \mathbf{0} & \\ & D_{\xi_i} \end{bmatrix}$ for $i = 2,\ldots,k$; we assume Q^i are diagonal matrices. The right-hand-side terms are $\mathbb{R}^{m_x+m_y} \ni b^1 = (b^\top \;\; h_{\xi_1}^\top)^\top$, while $\mathbb{R}^{m_y} \ni b^i = h_{\xi_i}$ $i = 2,\ldots,k$, and $\mathbb{R}^{(k-1)n_x} \ni b^0 = 0$. 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 $[T_i \;\; W]$, $i = 2,\ldots,k$.

Linking constraints for (5) are defined by

$$R_1 = \begin{bmatrix} I_1 & 0^y_1 \\ \vdots & \vdots \\ I_{k-1} & 0^y_{k-1} \end{bmatrix} \quad R_i = \begin{bmatrix} 0^x_1 & 0^y_1 \\ \vdots & \vdots \\ 0^x_{i-2} & 0^y_{i-2} \\ -I_{i-1} & 0^y_{i-1} \\ 0^x_i & 0^y_i \\ \vdots & \vdots \\ 0^x_{k-1} & 0^y_{k-1} \end{bmatrix} \; i = 2, \dots, k, \tag{10}$$

where $0^x_i, I_i \in \mathbb{R}^{n_x \times n_x}$, $0^y_i \in \mathbb{R}^{n_x \times n_y}$, and the subindex denotes the block row position. For (7) the linking constraints are defined by

$$R_1 = \begin{bmatrix} I_1 & 0^y_1 \\ 0^x_2 & 0^y_2 \\ \vdots & \vdots \\ 0^x_{k-1} & 0^y_{k-1} \end{bmatrix} \quad R_i = \begin{bmatrix} 0^x_1 & 0^y_1 \\ \vdots & \vdots \\ 0^x_{i-2} & 0^y_{i-2} \\ -I_{i-1} & 0^y_{i-1} \\ I_i & 0^y_i \\ 0^x_{i+1} & 0^y_{i+1} \\ \vdots & \vdots \\ 0^x_{k-1} & 0^y_{k-1} \end{bmatrix} \quad R_k = \begin{bmatrix} 0^x_1 & 0^y_1 \\ \vdots & \vdots \\ 0^x_{k-2} & 0^y_{k-2} \\ -I_{k-1} & 0^y_{k-1} \end{bmatrix}, \tag{11}$$

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

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 (9).

3. The specialized interior-point approach

The specialized IPM is based on a primal-dual path-following algorithm [30], that solves the normal equations by means of a scheme that sensibly combines direct sparse Cholesky factorizations [26] and iterative preconditioned conjugate gradient (PCG for short) solvers. This procedure was initially suggested for multicommodity flow problems [11] and later extended to primal block-angular problems [10]. Thus, it can be applied to problem (9). This specialized IPM was recently implemented in the BlockIP solver [8].

For completeness, we provide an outline of the primal-dual path-following algorithm. Problem (9) can be written in a compact form as

$$\begin{array}{ll} \min\limits_{x} & c^\top x + \frac{1}{2} x^\top Q x \\ \text{s. to} & Ax = b \\ & 0 \le x \le u, \end{array} \tag{12}$$

where $c, x, u \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $Q \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^m$, m and n being respectively the overall number of constraints and variables of the problem. The dual of (12) is

$$\begin{array}{l} \max b^\top \lambda - \frac{1}{2} x^\top Q x - w^\top u \\ \text{s. to } A^\top \lambda - Qx + z - w = c \\ \qquad z, w \geq 0, \end{array} \tag{13}$$

where $\lambda \in \mathbb{R}^m$ and $z, w \in \mathbb{R}^n$.

Replacing inequalities in (12) by a logarithmic barrier with parameter $\mu > 0$, the KKT conditions for the barrier problem become

$$\begin{array}{ll} r_c & \equiv c - (A^\top \lambda - Qx + z - w) = 0, \\ r_b & \equiv b - Ax = 0, \\ r_{xz} & \equiv \mu e - XZe = 0, \\ r_{sw} & \equiv \mu e - SWe = 0, \\ & (x, s, z, w) \geq 0 \ ; \end{array} \tag{14}$$

$e \in \mathbb{R}^n$ is a vector of 1's, matrices $X, Z, U, W \in \mathbb{R}^{n \times n}$ are diagonal matrices made up of vectors x, z, u, w, and S is defined as $S = U - X$. The first two sets of equations of (14) impose, respectively, dual and primal feasibility; the last two impose complementarity. The set of unique solutions of (14) for each μ value is known as the central path, and when $\mu \to 0$ these solutions converge to those of (12) and (13). The primal-dual path-following algorithm solves the nonlinear system (14) by a sequence of damped Newton's directions (that is, with step length reduction to preserve the nonnegativity of variables), reducing the μ parameter at each iteration, and staying close to the central path. An excellent discussion about the theoretical properties of path-following algorithms can be found in [30].

Computing the Newton's directions $(\Delta x, \Delta\lambda, \Delta z, \Delta w)$ for (14) requires the solution of linear systems of this form:

$$\begin{bmatrix} -Q & A^\top & I & \\ A & & & \\ Z & & X & \\ -W & & & S \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta z \\ \Delta w \end{bmatrix} = \begin{bmatrix} r_c \\ r_b \\ r_{xz} \\ r_{sw} \end{bmatrix}. \tag{15}$$

After eliminating Δw and Δz, as follows,

$$\Delta z = X^{-1} r_{xz} - X^{-1} Z \Delta x \tag{16}$$

$$\Delta w = S^{-1} r_{sw} + S^{-1} W \Delta x, \tag{17}$$

we obtain a symmetric indefinite system known as the *augmented system*:

$$\begin{bmatrix} -\Theta^{-1} & A^\top \\ A & \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} r \\ r_b \end{bmatrix}, \tag{18}$$

where Θ and r are defined as

$$\Theta = (Q + S^{-1} W + X^{-1} Z)^{-1} \qquad r = r_c + S^{-1} r_{sw} - X^{-1} r_{xz}. \tag{19}$$

If, in addition, we eliminate Δx from the last group of equations of (18), the *normal* *equations* form is obtained:

$$(A\Theta A^\top)\Delta\lambda = g \quad \text{where } g = r_b + A\Theta r \tag{20}$$

$$\Delta x = \Theta(A^\top \Delta\lambda - r). \tag{21}$$

The Newton's direction is computed using (16), (17), (20) and (21). For linear (i.e., $Q = 0$) or separable quadratic problems Θ is a positive diagonal matrix and can be easily computed and inverted.

Computationally, the most time consuming step of the algorithm is the solution at each interior-point iteration of the normal equations system (20). Exploiting the structure of A in (9) and appropriately partitioning Θ, the matrix of the normal equations can be recast as

$$A\Theta A^\top = \left[\begin{array}{ccc|c} N_1\Theta_1 N_1^\top & & & N_1\Theta_1 R_1^\top \\ & \ddots & & \vdots \\ & & N_k\Theta_k N_k^\top & N_k\Theta_k R_k^\top \\ \hline R_1\Theta_1 N_1^\top & \dots & R_k\Theta_k N_k^\top & \Theta_0 + \sum_{i=1}^k R_i\Theta_i R_i^\top \end{array}\right] = \begin{bmatrix} B & C \\ C^\top & E \end{bmatrix}, \tag{22}$$

$B \in \mathbb{R}^{\tilde m\times\tilde m}$ ($\tilde m = \sum_{i=1}^k m_i$), $C \in \mathbb{R}^{\tilde m\times l}$ and $E \in \mathbb{R}^{l\times l}$ 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 (9). Appropriately partitioning g and $\Delta\lambda$, the normal equations can be written as

$$\begin{bmatrix} B & C \\ C^T & E \end{bmatrix}\begin{bmatrix} \Delta\lambda_1 \\ \Delta\lambda_2 \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix}. \tag{23}$$

Eliminating $\Delta\lambda_1$ from the first group of equations of (23), we get

$$(E - C^\top B^{-1}C)\Delta\lambda_2 = (g_2 - C^\top B^{-1} g_1) \tag{24}$$

$$B\Delta\lambda_1 = (g_1 - C\Delta\lambda_2). \tag{25}$$

System (25) is solved by performing one Cholesky factorization for each diagonal block $N_i\Theta^i N_i^\top, i = 1,\dots,k$ of B. System (24) is solved by a PCG. The dimension of this system is l, i.e., the number of linking constraints. A good preconditioner is imperative for the efficient solution of (24).

The preconditioner obtained in [11] for multicommodity flows can be applied to any primal block-angular problem [10]. The Neumann series preconditioner is based on the following result (see [11] for a proof):

$$(E - C^\top B^{-1}C)^{-1} = \left(\sum_{i=0}^{\infty}(E^{-1}(C^\top B^{-1}C))^i\right)E^{-1}. \tag{26}$$

The preconditioner, an approximation of $(E - C^\top B^{-1}C)^{-1}$, is thus obtained by truncating the infinite power series (26) at some term ϕ. From (26), the quality of the preconditioner depends on the spectral radius ρ (i.e., the largest eigenvalue) of matrix $(E^{-1}(C^\top B^{-1}C))$, which is known to be in $[0,1)$ [11]. 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. 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. $\phi = 0$ has been used for all the computational results in this paper (unless otherwise stated). 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 next two sections we analyze the structure and efficient factorization of E for both formulations (5) and (7).

One of the important parameters for the efficient solution of (24) 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 Sections 4 and 5.

3.1. Structure of E for formulation (5)

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 (10), the structure of E is given by

$$E = \Theta_0 + \begin{bmatrix} \Theta_1^x + \Theta_2^x & \Theta_1^x & \dots & \Theta_1^x & \Theta_1^x \\ \Theta_1^x & \Theta_1^x + \Theta_3^x & \dots & \Theta_1^x & \Theta_1^x \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \Theta_1^x & \Theta_1^x & \dots & \Theta_1^x + \Theta_{k-1}^x & \Theta_1^x \\ \Theta_1^x & \Theta_1^x & \dots & \Theta_1^x & \Theta_1^x + \Theta_k^x \end{bmatrix}. \quad (27)$$

E in (27) 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., [14, Ch. 4]). Figure 2.a shows the structure of (27) 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 (27); 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 (7), which is a more efficient alternative.

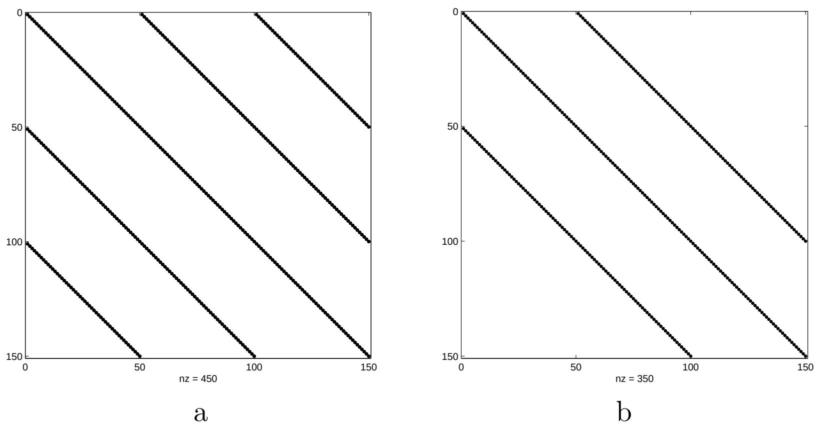


Figure 2.: Problem with $n_x = 50$ first-stage variables and $k = 4$ scenarios: a) Structure of (27) for formulation (5); b) Structure of (28) for formulation (7)

3.2. *Structure of E for formulation (7)*

From the definition of E in (22) and those of R_i in (11), and considering as above that Θ_i is made up of Θ_i^x and Θ_i^y, by block multiplication we get

$$E = \Theta_0 + \begin{bmatrix} \Theta_1^x + \Theta_2^x & -\Theta_2^x & & & \\ -\Theta_2^x & \Theta_2^x + \Theta_3^x & -\Theta_3^x & & \\ & \ddots & \ddots & \ddots & \\ & & -\Theta_{k-2}^x & \Theta_{k-2}^x + \Theta_{k-1}^x & -\Theta_{k-1}^x \\ & & & -\Theta_{k-1}^x & \Theta_{k-1}^x + \Theta_k^x \end{bmatrix}. \tag{28}$$

Figure 2.b shows the structure of (28) for the same problem in Subsection 3.1, reformulated as in (7). The fundamental difference between (27) and (28) 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 (28) 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.

4. Preliminary computational results

The problem formulation (7) was implemented and solved using the BlockIP package [8, 10], which is based on the the specialized IPM of Section 3. The instances used for preliminarily testing the performance of the approach were a subset of the instances proposed in [1] and [15]. They are publicly available in SMPS format [16]. These two-stage stochastic instances have been used to measure performance in several solvers, such as in [17] and [32].

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. Unlike BlockIP, CPLEX computes directions by direct solvers, instead of a

					BlockIP			CPLEX	
Instance	$1st$	k	m	n	Iter	CPU	PCG	Iter	CPU
LandS	4	3	23	62	9	0.00	41	6	0.00
AIRL_first	4	25	152	402	12	0.00	121	7	0.00
pltexpA2	188	6	686	2760	60	0.05	492	10	0.01
4node128	52	128	9486	32008	35	0.97	1167	25	0.30
4node4096	52	4096	303118	1024008	49	76.06	2938	45	17.30
4node16384	52	16384	1212430	4096008	59	522.10	5131	43	57.89
env_15	49	15	768	2046	21	0.03	78	14	0.01
env_1875	49	1875	90048	251286	64	9.09	1096	26	1.17
envDiss8232	49	8232	395184	1103124	135	154.36	4066	62	15.02
phone	8	32768	753665	3309569	21	38.02	255	23	26.00

Table 1.: Results of linear instances with BlockIP and CPLEX barrier

combination of Cholesky and PCG. CPLEX executions were made without crossover for a fair comparison with BlockIP. In this preliminary experiments, for BlockIP the problems were modelled using the splitting formulation (7), whereas for CPLEX the dual of the extensive form (3) was used, in an attempt to avoid dense columns due to the first-stage variables. All the experiments in this work 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 exploitation of multithreading capabilities.

Table 1 shows a sample of the results with the instances referred to above. The first column indicates the name of each instance; the second column (named $1st$) gives the number of variables in the first stage; columns k, m, and n refer to, respectively the number of scenarios, total constraints, and total variables. The given values of m and n correspond to the splitting formulation (7), which is the one used by BlockIP, while CPLEX solved the formulation without splitting—i.e., the dual of the extensive form (3)— which involves fewer variables and constraints. In the case of BlockIP, we show the number of interior-point iterations (column "Iter"), CPU time (column "CPU") and total number of PCG iterations (column "PCG"). For CPLEX we have the number of interior-point iterations and CPU time. 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 were difficult to achieve by BlockIP in those instances due to the approximate solution of the Newton direction by PCG). The results of Table 1 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.

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 2 reports the characteristics of these quadratic instances and the results obtained with BlockIP and CPLEX. 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 [9]. Then, as expected, the performance of BlockIP improved in these quadratic stochastic instances. However, it was still outperformed by CPLEX.

					BlockIP			CPLEX	
Instance	$1st$	k	m	n	Iter	CPU	PCG	Iter	CPU
LandS	4	3	23	62	9	0.00	48	6	0.00
AIRL_first	4	25	152	402	11	0.00	173	10	0.00
pltexpA2	188	6	686	2760	54	0.03	241	10	0.2
4node128	52	128	9486	32008	29	0.61	534	7	0.12
4node4096	52	4096	303118	1024008	31	36.60	984	7	6.81
4node16384	52	16384	1212430	4096008	31	137.87	1219	8	14.29
env_15	49	15	768	2046	14	0.01	46	13	0.02
env_1875	49	1875	90048	251286	15	2.24	261	17	1.11
envDiss8232	49	8232	395184	1103124	17	19.44	393	18	4.60
phone	8	32768	753665	3309569	13	21.95	169	18	12.88

Table 2.: Results of quadratic instances with BlockIP and CPLEX barrier

					BlockIP			DSP	
Instance	$1st$	k	m	n	Iter	CPU	PCG	Algorithm	CPU
LandS	4	3	23	62	9	0.00	41	Benders	0
								Dual	0.3
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
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	—
phone	8	32768	753665	3309569	21	38.02	255	Benders	—
								Dual	—

— Instance exceeded the maximum memory allotted by Neos Server to a job

Table 3.: Results of linear instances with BlockIP and DSP

4.1. 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 [32]; the primal-dual column generation approach of [17]; and the dual decomposition approach (with an interior-point cutting-plane generator) implemented in the DSP (Decomposition for Structured Programming) stochastic solver [21]. Comparing BlockIP with these other approaches is not straightforward since: the solvers of [32] and [17] are not freely available; the DSP solver had to be used remotely from the *Neos Server* [12] (we did not succeed in installing it locally due to its many dependencies with third-party software); and the four hardwares (i.e., those of [32], [17], the Neos Server, and our work) were different, so only an indirect comparison can be performed.

Table 3 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 1. CPU times for DSP were provided by the Neos Server [25], 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 3. From the times reported, DSP seems not to be competitive with our approach, specially for the largest instances. Possible explanations might be that DSP was mainly designed for mixed integer stochastic problems, and that it is a research code.

Concerning the other two specialized methods, and according to the information provided by their authors, the processors used in [17] and [32] were, in single-thread mode, respectively a 27% and a 11% faster (in terms of Mflops, millions of floating point operations per second) than the one used in our work. The approach of [17] outperformed in general all the methods tested in [32], from the results in those papers. Therefore we will focus on the primal-dual column generation method of [17]. This approach required (in the hardware used in [17]) for the 10 instances of Table 1, 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 ran the approach of [17] 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 1 that the performance of CPLEX was very similar in those instances (excluding the last two, where the approach of [17] was significantly faster—specially 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 [32], where the CPLEX barrier ranked among the best three algorithms for stochastic optimization). In addition, CPLEX can solve quadratic instances, while the approaches of [17] and [32] dealt only with linear problems. CPLEX will thus be the solver used to test our approach in Section 5, for the solution of more difficult instances provided by two particular applications.

4.2. 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 (18)—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) [20].

Table 4 provides the number of iterations and CPU time with the package Ipopt (v. 3.9.3) [29], which solves the augmented system using MA57, for the same instances used in previous Table 2. 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. 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 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 5.3.3 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.

5. Computational results for two particular applications

The instances of Tables 1 and 2 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

					Ipopt		CPU/Iter		
Instance	1*st*	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.: Results of linear instances with Ipopt, and CPU time per iteration for BlockIP, CPLEX and Ipopt

order to get more difficult instances, we modified two problems from the literature: (1) a stochastic supply chain problem based on [2], which will be described in Section 5.1; (2) and instance *LandS* of Section 4, which corresponds to a stochastic electricity generation problem and it is is explained below in Section 5.2. 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. The resulting instances had around 600 first-stage variables, which is significantly more than the number of first-stage variables of the cases of Tables 1 and 2. We note that in some applications (e.g., some real-world electricity generation problems [7]) the number of first-stage variables involved in the coupling with the second-stage may be even larger, up to $O(10000)$. Both problems were solved with CPLEX and BlockIP. For the solution with BlockIP they were modelled with the splitting formulation (7), thus obtaining the constraints structure (8). For CPLEX we considered two formulations: the splitting formulation (7) (as for BlockIP); and the dual of the original extensive form (3) (whose constraints matrix is the transpose of (4) 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.

5.1. The stochastic supply chain problem

This problem was created from the models in [2] and [28]. 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 product 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 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. In addition, 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:

$$\begin{array}{lll} \min\limits_{x,u} & \sum\limits_{i\in\mathcal{S}\cup\mathcal{P}} c_i x_i + \sum\limits_{(ij)\in\mathcal{A}} f_{ij}u_{ij} + \mathcal{Q}(x,u) & \\ \text{s. to} & 0 \le x_i \le 1 \quad \forall i \in \mathcal{S}\cup\mathcal{P} & \\ & 0 \le u_{ij} \le M \quad \forall (ij) \in \mathcal{A} & \end{array} \qquad \text{and} \qquad \mathcal{Q}(x,u) = E_\xi[Q(x,u,\xi)], \tag{29}$$

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

$$Q(x,u,\xi) = \min_{y,z} \quad \sum_{(ij)\in\mathcal{A}} q_{ij}y_{ij} + \sum_{j\in\mathcal{C}} h_j z_j \tag{30a}$$

$$\text{s. to} \quad \sum_{i\in\mathcal{S}} y_{ij} - \sum_{l\in\mathcal{C}} y_{jl} = 0 \qquad \forall j \in \mathcal{P} \tag{30b}$$

$$\sum_{i\in\mathcal{P}} y_{ij} + z_j \ge d_j \qquad \forall j \in \mathcal{C} \tag{30c}$$

$$\sum_{j\in\mathcal{P}} y_{ij} \le s_i x_i \qquad \forall i \in \mathcal{S} \tag{30d}$$

$$\sum_{i\in\mathcal{S}} y_{ij} \le m_j x_j \qquad \forall j \in \mathcal{P} \tag{30e}$$

$$0 \le y_{ij} \le u_{ij} \qquad \forall (ij) \in \mathcal{A} \tag{30f}$$

$$z_j \ge 0 \qquad \forall j \in \mathcal{C}, \tag{30g}$$

whose parameters are:

- c_i is the investment cost for supply and processing 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 demand of the product at 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 random vector ξ contains the parameters $(s_i)_{i\in\mathcal{S}}$, $(d_j)_{j\in\mathcal{C}}$, $(h_j)_{j\in\mathcal{C}}$, and $(q_{ij})_{(ij)\in\mathcal{A}}$.

The first-stage problem (29) 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 (30) involves the processing and the transportation of the product, where the objective function (30a) minimizes the transportation and shortage costs. The first set of constraints (30b) enforces the flow conservation across each processing node j. The next group of constraints (30c) requires that the total flow for the customer j plus its shortfall should be at least the demand d_j. Constraints (30d) 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 (30e) establishes that the number of units

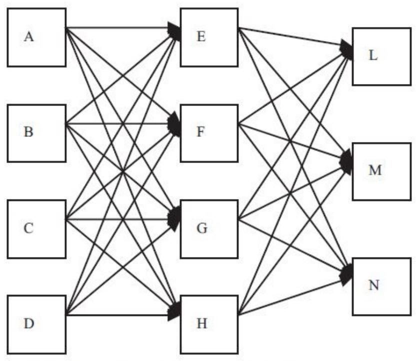


Figure 3.: The network of the wine company used in [2]

to process should not be greater than the capacity m_j of facility j if it is built. If facility $i \in S \cup P$ is not built, the above two groups of constraint will force all flow variables $y_{ij} = 0$ for all $i \in \mathcal{S} \cup \mathcal{P}$. Finally, constraints (30f) and (30g) are the upper and lower bounds.

5.1.1. Test instances

The original model in [2] 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 Fig. 3. The product considered in [2] was uniform-quality wine in bulk (raw material). Four scenarios were considered (boom, good, fair and poor), with different probabilities associated to each one. It should be stressed that in [2] the supplies, transportation costs and shortage costs were considered as deterministic parameters for the extensive form (or deterministic equivalent) formulation.

We extended the original model in [2] in several ways. First, all the parameters of the problem were created by means of 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 for 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 5.3.1 and 5.4.1 are obtained by replicating the number of scenarios.

5.2. The stochastic electricity generation problem

This problem (based on [23]) consists of finding the optimal investment over various types of power plants to satisfy an 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 large number of modes for a good approximation to the reality. The several power plants use one of the available technologies. See [1, 4] 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:

$$\begin{aligned} \min_x \quad & \sum_{i\in\mathcal{T}} c_i x_i + \mathcal{Q}(x) \\ \text{s. to} \quad & \sum_{i\in\mathcal{T}} x_i \geq C \\ & \sum_{i\in\mathcal{T}} c_i x_i \leq B \\ & x_i \geq 0 \quad i \in \mathcal{T} \end{aligned} \qquad \text{and} \qquad \mathcal{Q}(x) = E_\xi[Q(x,\xi)], \tag{31}$$

where

$$Q(x,\xi) = \min_y \quad \sum_{i\in\mathcal{T}} \sum_{j\in\mathcal{M}} q_i t_j y_{ij} \tag{32a}$$

$$\text{s. to} \quad \sum_{j\in\mathcal{M}} y_{ij} \leq x_i \quad i \in \mathcal{T} \tag{32b}$$

$$\sum_{i\in\mathcal{T}} y_{ij} = \xi \quad j \in \{1\} \tag{32c}$$

$$\sum_{i\in\mathcal{T}} y_{ij} = d_j \quad j \in \mathcal{M} \setminus \{1\} \tag{32d}$$

$$y_{ij} \geq 0 \qquad i \in \mathcal{T}, j \in \mathcal{M}, \tag{32e}$$

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

The first-stage problem (31) considers the investment cost for each technology, subject to a minimum total capacity and a maximum budget constraints. The objective function (32a) of the second-stage problem consists of minimizing the operational costs for each technology effectively used in every mode. The constraints (32b) impose that available capacities cannot be exceeded, for each technology. The other two sets of constraints (32c)–(32d) impose demand satisfaction for every mode; the first constraint is for the first mode, whose demand is stochastic.

5.2.1. Test Instances

The instance presented in [1, 4] 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 investment 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

			BlockIP			CPLEX no splitting		CPLEX 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 5.: Results for linear supply chain problem up to 1000 scenarios

modes and scenarios for every instance. The rest of 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. The number of technologies (that is, first-stage decisions), was increased 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.

5.3. Results for linear instances

Next two subsections present results for the linear instances of the supply chain and electricity generation problems. 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.

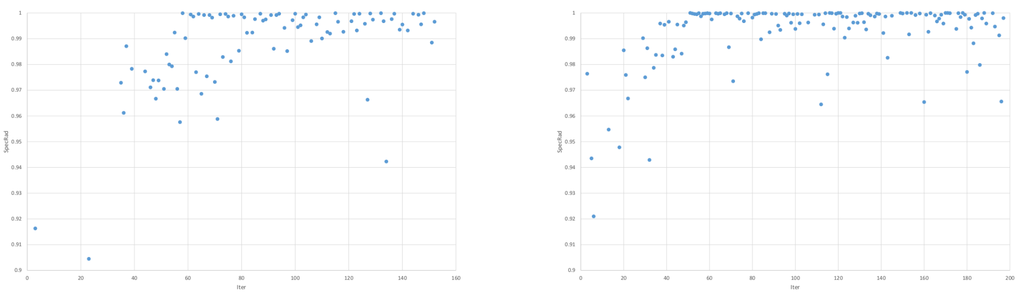
5.3.1. Linear instances of the stochastic supply chain problem

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 5.1.1, we generated a first set of stochastic instances with 200, 400, 600, 800 and 1000 scenarios. Table 5 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 (we recall that, according to 26, the further from one is the spectral radius of matrix $E^{-1}(C^\top B^{-1}C)$, the better is the preconditioner). 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 5 it is clearly observed that BlockIP was more efficient than CPLEX either with or without splitting.

As stated above, the number of PCG iterations was high for the instances of Table 5 because the spectral radii were close to one. To give more insight on this fact, Figure 4 shows the evolution of the spectral radius (vertical axis) along the interior-point iterations (horizontal axis) for the two instances with 600 and 1000 scenarios of Table 5. Indeed, the information provided corresponds to estimations of the spectral radius (the true value would be very expensive to compute numerically for this large instances), using the *Ritz values* obtained during the solution of the PCG (this technique was developed in [5], and full details can be obtained there). It is worth remarking that when the number of PCG

			BlockIP			CPLEX no splitting		CPLEX 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 6.: Results for linear supply chain problem up to 5000 scenarios



(a) 600 scenarios

(b) 1000 scenarios

Figure 4.: Evolution of the spectral radius along interior-point iterations for instances with 600 and 1000 scenarios (plots (a) and (b), respectively) of Table 5

iterations in an interior-point iteration is small (e.g., less than five in our instances), the estimation of the spectral radius can be wrong and it is not reported in Figure 4 (this explains the "holes" for some interior-point iterations). From Figure 4 we observe that the values tend to one as we increase the iteration number, and that for the instance with 1000 scenarios the values are closer to one, which explains why this instance is more difficult than the 600 scenarios one (note that it takes 45 more interior-point iterations, and the overall number of PCG iteration is three times higher than for the 600 scenarios instance).

Table 6 reports results for a second set of larger instances with a number of scenarios between 2000 and 5000; the largest instance has 6.5M constraints and 12.8M variables. The optimality tolerance was increased to 10^{-3} for both BlockIP and CPLEX, 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 6 we see that BlockIP clearly outperformed both CPLEX variants. It is also worth to note that, unlike in Table 5, the CPLEX runs for the model with splitting were much slower than those without splitting. This fact will be even more evident in below Table 11 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 11 for quadratic instances).

As summarized in Figure 5 (which plots the solution time against the number of scenarios for the instances of Tables 5 and 6), BlockIP always outperformed CPLEX, and one could infer that the difference between them will increase if the number of scenarios

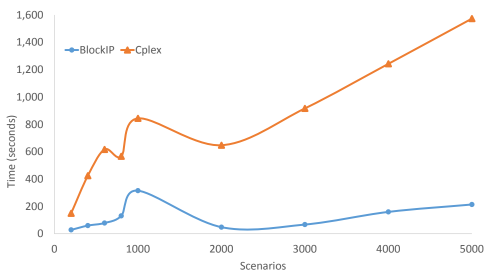


Figure 5.: BlockIP vs CPLEX for linear supply chain problem (CPLEX without splitting)

			BlockIP			CPLEX	
$1st$	m	n	Iter	CPU	PCG	Iter	CPU
400	80602	519602	83	7.14	559	23	87.50
450	90552	584552	85	8.20	533	20	86.42
500	100502	649502	81	9.02	470	23	110.02
550	110452	714452	90	12.45	714	28	139.01
600	120402	779402	85	12.00	491	19	108.76

Table 7.: Results for linear electricity generation problem for different number of first-stage variables

			BlockIP			CPLEX no splitting		CPLEX 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 8.: Results for linear electricity generation problem up to 1000 scenarios

grows higher.

5.3.2. Linear instances of the stochastic electricity generation problem

We generated a preliminary set of instances where the first-stage variables (number of available technologies) were increased up to 600, the number of modes of electricity demand were 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 7.

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 8, which shows that the performance of BlockIP is much better than the one of CPLEX either with or without splitting. It is seen that CPLEX with either model required large executions in those instances. For this reason, the second set

			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 9.: Results for linear electricity generation problem up to 5000 scenarios

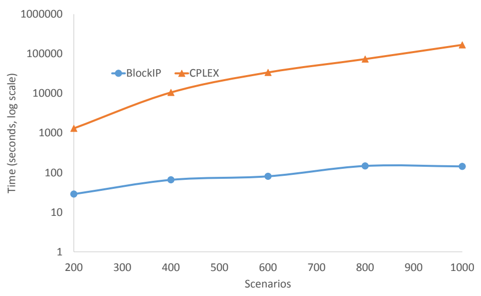


Figure 6.: BlockIP vs CPLEX for linear electricity generation problems (CPLEX with splitting). Vertical axis (solution time) in log scale

of even larger instances obtained by considering a number of scenarios between 2000 and 5000 were only ran with BlockIP. The results are provided in Table 9. 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). The plot in Figure 6 (which reports the solution time (in log scale) against the number of scenarios) helps better appreciate that BlockIP is by far faster than CPLEX in those instances.

5.3.3. Normal equations vs augmented system

As it was advanced in previous Section 4.2, Table 10 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 10 correspond, respectively, to the first two instances of Tables 5 and 8). 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 10 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 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

problem	k	Ipopt no splitting Iter	Ipopt no splitting CPU	Ipopt with splitting Iter	Ipopt with splitting CPU	CPU/Iter BlockIP	CPU/Iter CPLEX no	CPU/Iter CPLEX with	CPU/Iter Ipopt no	CPU/Iter Ipopt with
Supply 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 10.: 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

k	m	n	BlockIP Iter	BlockIP CPU	BlockIP PCG	CPLEX no splitting Iter	CPLEX no splitting CPU	CPLEX with splitting Iter	CPLEX with splitting 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 11.: Results for quadratic supply chain problem up to 1000 scenarios

(that is, combining Cholesky and PCG) is the best option.

5.4. 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 [9] for details). In all the runs, the optimality tolerance for BlockIP and CPLEX was 10^{-6}. For BlockIP the initial PCG tolerance and reduction factor 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 (7), while for CPLEX both the splitting formulation (7) and the (no splitting) dual of the extensive form (3) were considered (note that in the latter case the constraints matrix contains both the transpose of (4) and the terms associated to quadratic costs).

5.4.1. 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 11. Clearly, BlockIP outperformed both CPLEX runs (with and without splitting). It is also observed that CPLEX is much less efficient with splitting in those instances. 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 11, 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

			BlockIP			CPLEX 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 12.: Results for quadratic supply chain problem up to 5000 scenarios

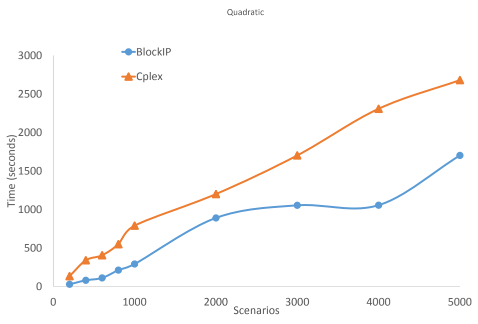


Figure 7.: BlockIP vs CPLEX for quadratic supply chain problems (CPLEX without splitting)

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 (therefore, same order as for the no-splitting case), but the number of nonzeros in lower triangle of $AA^\top$ is 312500000 (14 times higher than for the model without splitting), and the number of nonzeros in the factor (after reordering) is 10855849207 (31 times higher than without splitting). This explains the poor performance of the splitting model with CPLEX in some cases.

Next, we raised the number of scenarios up to 5000. The results are shown in Table 12. 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 7, where BlockIP remains always below the time of CPLEX, even in small instances.

5.4.2. Quadratic instances of the stochastic electricity generation problem

We considered as the base case the instance with 100 first-stage variables, such that the number or 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 13 presents the results for up to 10000 scenarios. In these executions the model with 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 behaviour. The good results obtained in most cases with BlockIP are evident in Figure 8.

An explanation of the bad behaviour of BlockIP for the 1000 and 2000 scenarios instances of Table 13 can be found in the spectral radius ρ. Figure 9 shows the evolution of the (estimation of the) spectral radius (vertical axis) along the interior-point iterations

			BlockIP			CPLEX no splitting		CPLEX 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 13.: Results for quadratic electricity generation problem up to 10000 scenarios

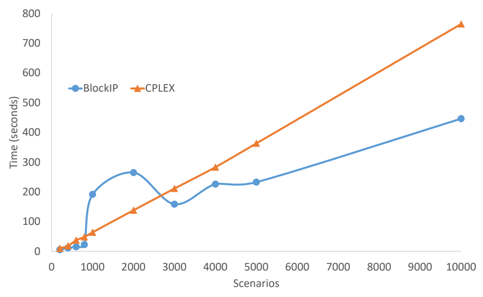
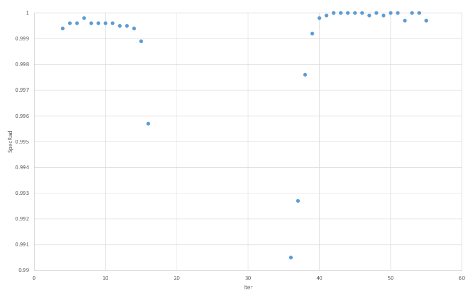


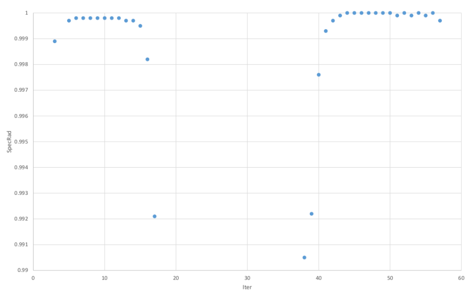
Figure 8.: BlockIP vs CPLEX for quadratic electricity generation problems (CPLEX with splitting)

(horizontal axis). Note that the vertical range is in [0.99,1.0], that is, all values of ρ reported in the plots were very close to 1 (specially in the last iterations). This explains the poor performance of BlockIP in this two particular instances.

As stated in Section 3, all the computational results obtained (so far) in this work with BlockIP used a value of $\phi = 0$ for the preconditioner (that is, the preconditioner was only the first term of the power series (26)). In general, $\phi = 0$ has been empirically observed to be the best choice [8]: the preconditioner is good enough, and the solution of extra systems with matrix B is avoided, which would make the preconditioner more time consuming. However, in some cases a value of $\phi = 1$ (i.e., the preconditioner contains the first two terms of (26)) provides a better trade-off between goodness and efficiency. For this purpose, the instances of Table 13 were also solved with BlockIP using $\phi = 1$. The results obtained are given in Table 14; the results with $\phi = 0$ are those of Table 13. It is observed that (excluding the instances with 400 and, mainly, 800 scenarios) the new term of the preconditioner reduces the total number of PCG iterations, but at the same time, it increases the overall solution times. The only exception is instance $k = 2000$: in this case the PCG iterations are divided by 10, and the CPU time by 4.4. Indeed, BlockIP with $\phi = 1$ becomes more efficient than CPLEX for this instance.



(a) 1000 scenarios



(b) 2000 scenarios

Figure 9.: Evolution of the spectral radius along interior-point iterations for instances with 1000 and 2000 scenarios (plots (a) and (b), respectively) of Table 13

			$\phi = 0$			$\phi = 1$		
k	m	n	Iter	CPU	PCG	Iter	CPU	PCG
200	41902	259902	43	5.45	1760	42	7.97	1695
400	83902	519902	42	11.19	1911	42	18.41	1916
600	125902	779902	43	14.77	1471	41	20.63	1178
800	167902	1039902	46	22.42	1575	53	201.58	9388
1000	209902	1299902	56	191.98	13604	56	338.89	12826
2000	419902	2599902	58	258.42	9037	44	58.40	855
3000	629902	3899902	60	153.12	3205	58	257.60	2948
4000	839902	5199902	59	219.14	3493	58	289.41	2410
5000	1049902	6499902	61	227.20	2738	61	333.89	2177
10000	2099902	12999902	59	457.10	2509	60	627.37	1895

Table 14.: Results for quadratic electricity generation problem with BlockIP and $\phi = 0$ and $\phi = 1$

6. Conclusions

In this work, we explored the performance of a specialized interior-point algorithm for block-angular problems (implemented in the BlockIP package) for two-stage stochastic programming 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 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 [1], BlockIP was outperformed by the other approaches.

This new approach for stochastic optimization has potential in areas where decision-makers need to analyze many first-stage decisions, with a large number of future scenarios, and in a very short time.

This specialized interior-point method allows the factorization and solution of the k Cholesky factorizations in parallel. A parallel version of BlockIP is being developed, and the preliminary results are promising (but it has not yet been tested in two-stage stochastic problems). This is part of the future work to be done.

Future studies should also focus on the applicability of the specialized interior-point algorithm to block-angular reformulations of multistage stochastic problems.

Acknowledgments

This work has been supported by the grants MINECO/FEDER MTM2015-65362-R and MCIU/AEI/FEDER RTI2018-097580-B-I00. The second author was supported by the CONACyT (Consejo Nacional de Ciencia y Tecnologia, México) grant CVU-394291. We also thank the two anonymous reviewers, whose suggestions and comments improved the quality of the paper.

References

[1] 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.

[2] A. Azaron, K.N. Brown, S.A. Tarim and M. Modarres, *A multi-objective stochastic programming approach for supply chain design considering risk*, International Journal of Production Economics 116(1) (2008), pp. 129–138.

[3] J. F. Benders, *Partitioning procedures for solving mixed-variables programming problems*, Numerische Mathematik 4(1) (1962), pp. 238–252.

[4] J.R. Birge and F. Louveaux, *Introduction to Stochastic Programming*, Springer Science & Business Media, (2011).

[5] S. Bocanegra, J. Castro and A.R.L. Oliveira, *Improving an interior-point approach for large block-angular problems by hybrid preconditioners*, European Journal of Operational Research 231.2 (2013), pp 263–273.

[6] S. Bradley, A. Hax and T. Magnanti, *Applied mathematical programming*, Addison Wesley, (1977).

[7] Y. Cao, C.D. Laird, V.M. Zavala, *Clustering-based preconditioning for stochastic programs*, Computational Optimization and Applications 64 (2016), pp. 379–406.

[8] J. Castro, *Interior-point solver for convex separable block-angular problems*, Optimization Methods and Software 31(1) (2016), pp. 88–109.

[9] 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.

[10] J. Castro, *An interior-point approach for primal block-angular problems*, Computational Optimization and Applications 36(2) (2007), pp. 195–219.

[11] J. Castro, *A specialized interior-point algorithm for multicommodity network flows*, SIAM Journal on Optimization 10(3) (2000), pp. 852–877.

[12] J. Czyzyk, M.P. Mesnier, J.J. Moré, *The NEOS Server*, IEEE Journal on Computational Science and Engineering 5(3) (1998), pp. 68–75.

[13] G.B. Dantzig and P. Wolfe, *The decomposition algorithm for linear programs*, Econometrica 29(4) (1961), pp. 767–778.

[14] G.H. Golub and C.F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, (2012).

[15] D. Holmes, *A Portable Stochastic programming Test Set*, available in: `http://users.iems.northwestern.edu/~jrbirge/html/dholmes/post.html`, (1995) [Accessed on 2018-11-15].

[16] H.I. Gassmann and B. Kristjánsson, *The SMPS format explained*, IMA Journal of Management Mathematics 19(4) (2007), pp. 1–31.

[17] 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.

[18] J. Gondzio, *Interior point methods 25 years later*, European Journal of Operational Research 218(3) (2012), pp. 587–601

[19] 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.

[20] HSL. A collection of Fortran codes for large scale scientific computation. `http://www.hsl.rl.ac.uk/`.

[21] 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.

[22] 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.

[23] F.V. Louveau and Y. Smeers, *Optimal investments for electricity generation: A stochastic model and a test-problem*, in *Numerical Techniques for Stochastic Optimization*, R. J-B. Wets and Y. Ermoliev, eds., Springer, 1988, pp. 445–453.

[24] I.J. Lustig, J.M. Mulvey and T.J. Carpenter, *Formulating stochastic programs for interior point methods*, Operations Research 39 (1991), pp. 757–770.

[25] 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].

[26] E. Ng and B.W. Peyton, *Block sparse Cholesky algorithms on advanced uniprocessor computers*, SIAM Journal on Scientific Computing 14(5) (1993), pp. 1034–1056.

[27] A. Ruszczyński, *Interior point methods in stochastic programming*, International Institute for Applied Systems analysis, Working paper 93-8, (1993).

[28] T. Santoso, S. Ahmed, M Goetschalckx and A. Shapiro, *A stochastic programming approach for supply chain network design under uncertainty*, European Journal of Operational Research 167(1) (2005), pp. 96–115.

[29] A. Wächter and L.T. Biegler, *On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming*, Mathematical Programming, 106(1) (2006), pp.25–57.

[30] S.J. Wright, *Primal-Dual Interior-Point Methods*, SIAM, (1997).

[31] F. A. Potra and S. J. Wright, *Interior-point methods*, Journal of Computational and Applied Mathematics 124 (2000), pp. 281–302.

[32] V. Zverovich, C. I Fábián, E. F Ellison and G. Mitra, *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.