Interior-point solver for convex separable block-angular problems

Jordi Castro
Dept. of Statistics and Operations Research
Univ. Politècnica de Catalunya
jordi.castro@upc.edu

Research Report UPC-DEIO DR 2014-03
October, 2014; updated January 2015, May 2015

Report available from http://www-eio.upc.es/~jcastro

Interior-point solver for convex separable block-angular problems

Jordi Castro*

Dept. of Statistics and Operations Research, Universitat Politècnica de Catalunya, Jordi Girona 1–3, 08034 Barcelona

Constraints matrices with block-angular structures are pervasive in Optimization. Interior-point methods have shown to be competitive for these structured problems by exploiting the linear algebra. One of these approaches solves the normal equations using sparse Cholesky factorizations for the block constraints, and a preconditioned conjugate gradient (PCG) for the linking constraints. The preconditioner is based on a power series expansion which approximates the inverse of the matrix of the linking constraints system. In this work we present an efficient solver based on this algorithm. Some of its features are: it solves linearly constrained convex separable problems (linear, quadratic or nonlinear); both Newton and second-order predictor-corrector directions can be used, either with the Cholesky+PCG scheme or with a Cholesky factorization of normal equations; the preconditioner may include any number of terms of the power series; for any number of these terms, it estimates the spectral radius of the matrix in the power series (which is instrumental for the quality of the preconditioner). The solver has been hooked to SML, a structure-conveying modelling language based on the popular AMPL modeling language. Computational results are reported for some large and/or difficult instances in the literature: (1) multicommodity flow problems; (2) minimum congestion problems; (3) statistical data protection problems using ℓ_1 and ℓ_2 distances (which are linear and quadratic problems, respectively), and the pseudo-Huber function, a nonlinear approximation to ℓ_1 which improves the preconditioner. In the largest instances, of up to 25 millions of variables and 300000 constraints, this approach is from two to three orders of magnitude faster than state-of-the-art linear and quadratic optimization solvers.

Keywords: interior-point methods; structured problems; normal equations; preconditioned conjugate gradient; large-scale optimization; optimization software

AMS Subject Classification: 90C51, 90C25, 90C06, 90C05, 90C20

1. Introduction

Block-angular structures are used as a modelling tool in many situations, such as multiperiod or multicommodity problems, two-stage stochastic problems, and, in general, models involving linking variables or linking constraints. These structured problems have applications, for instance, in the energy, logistics, telecommunications and big-data fields. The resulting optimization problems have in common a huge number of variables, and a large number of linear constraints. Interior-point methods (IPMs) are in general competitive against other techniques in those cases.

This work presents an efficient interior-point solver for block-angular convex optimization problems, which relies on a combination of Cholesky factorizations and preconditioned conjugate gradient (PCG) for the solution of the linear systems at each interior-point iterations. This Cholesky+PCG combination was initially suggested for the case of multicommodity flows in [10]; it was later extended to general problems in [12], including a Matlab prototype. The solver introduced in this work, named `BlockIP` is not merely a "C++ translation" of this Matlab prototype. It includes many other features,

*Email: jordi.castro@upc.edu

being the three most relevant: (i) it may solve convex nonlinear separable optimization problems; (ii) the addition of proximal point and quadratic regularization, which may significantly improve the quality of the preconditioner as shown in [14]; (ii) an estimation of the spectral radius of a certain matrix which intervenes in the preconditioner, following [9], which can be used as a measure of the quality of preconditioner. This estimation has been extended in this work to more complex preconditioners, i.e., involving more terms of a certain power series; Proposition 3.5 provides this result. In addition, `BlockIP` resulted to be much more efficient than the Matlab prototype, allowing us to solve problems of up to 25 millions of variables and 300000 constraints (see Section 5), which were out of the scope of [12].

`BlockIP` solves the normal equations, unlike other PCG-based IPMs that focus on the augmented system, such as [5, 23, 32]. Normal equations were solved by some of the earlier preconditioners for network flows problems [19, 34], but also by some recent approaches [3]. Although PCG usually provides an approximate solution of the linear system of equations, it has recently been shown that those inexact solutions also guarantee the convergence of the IPM [21].

The structure of the paper is as follows. The formulation of the block-angular problem considered is given in Section 2. Section 3 outlines the PCG-based IPM, including three subsections summarizing some of the main features implemented in the solver, namely, the power series preconditioner; improving the preconditioner by reducing the spectral radius of a certain matrix through quadratic or nonlinear terms in the objective; and the estimation of this spectral radius by Ritz values. Though the material of these three subsections comes, mainly, from [12], [14] and [9], it is included here for completeness. Section 4 describes the specialized solver. Finally, Section 5 provides computational results in the solution of three types of problems: multicommodity flows, minimum congestion, and a disclosure control problem in statistical tabular data.

2. Convex block-angular problems

The standard form of the linearly constrained convex block-angular problems considered in this work is

$$\begin{aligned} \min \ & \sum_{i=0}^{k} f_i(x^i) \\ \text{s. to } & \begin{bmatrix} A_1 & & \\ & \ddots & \\ & & A_k \\ L_1 & \dots & L_k \ I \end{bmatrix} \begin{bmatrix} x^1 \\ \vdots \\ x^k \\ x^0 \end{bmatrix} = \begin{bmatrix} b^1 \\ \vdots \\ b^k \\ b^0 \end{bmatrix} \\ & 0 \le x^i \le u^i \quad i = 0, \dots, k. \end{aligned} \tag{1}$$

Matrices $A_i \in \mathbb{R}^{m_i \times n_i}$ and $L_i \in \mathbb{R}^{l \times n_i}$, $i = 1, \dots, k$ define the block and linking constraints, respectively, k being the number of blocks. Vectors $x^i \in \mathbb{R}^{n_i}, i = 1, \dots, k$, are the variables for each block. $x^0 \in \mathbb{R}^l$ are the slacks of the linking constraints. $b^i \in \mathbb{R}^{m_i}, i = 1, \dots, k$ is the right-hand-side vector for each block of constraints, whereas $b^0 \in \mathbb{R}^l$ is for the linking constraints. The upper bounds for each group of variables are defined by $u^i, i = 0, \dots, k$. Note that with this standard formulation linking constraints are of the form $b^0 - u^0 \le \sum_{i=1}^{k} L_i x^i \le b^0$. As it will be shown later, slacks of linking constraints play a significant role in the quality of the preconditioner, and they should not

be removed. Equality linking constraints can be formulated by setting $u^0 \approx 0$. Functions $f_i : \mathbb{R}^{n_i} \to \mathbb{R}$, $i = 0, \ldots, k$, are assumed to be convex. Although the specialized IPM to be described is valid for any f_i, for the sake of efficiency we will restrict to separable functions, i.e., $\nabla^2 f_i(x^i)$ are (positive semidefinite) diagonal matrices. Note that any convex quadratic problem can be transformed into a separable equivalent one by the addition of extra variables and constraints [36, Ch.23], though this may degrade the sparsity of the problem. It is also worth to remark that slacks may also intervene in the objective function; this may be helpful to preserve the separability of the problem, avoiding terms of the form $f_0(b^0 - \sum_{i=1}^k L_i x^i)$. The particular case of linear and convex quadratic separable problems is obtained with $f_i(x^i) = {c^i}^\top x^i$ and $f_i(x^i) = {c^i}^\top x^i + \frac{1}{2}{x^i}^\top Q_i x^i$, Q_i positive semidefinite and diagonal, for $i = 1 \ldots, k$. (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.

The structure of the constraints matrix of (1) is usually named *primal block-angular*. More general structures (i.e., such as those of [22]) are obtained if, for some i, m_i is 0, i.e., a group of variables only appears in the linking constraints. For instance, if $m_k = 0$ we have

$$\begin{bmatrix} A_1 & & & \\ & \ddots & & \\ & & A_{k-1} & \\ L_1 & \ldots & L_{k-1} & L_k \; I \end{bmatrix}. \tag{2}$$

Dual block-angular structures, as those appearing in two-stage stochastic optimization problems with recourse, can be formulated as (1) by considering the dual problem. Even problems with both linking variables and linking constraints may be transformed to (1) by replicating columns and adding extra linking constraints.

On the other hand, when $k = 1$, the specialized method can be applied to any linearly constrained convex problem by considering a partition of the linear constraints, as follows

$$\begin{array}{l} \min f(x) + f_0(x^0) \\ \text{s. to } Ax = b \\ \qquad Lx + x^0 = b^0 \\ \qquad 0 \le x \le u \quad , \quad 0 \le x^0 \le u^0. \end{array} \tag{3}$$

If $k = 1$ and $l = 0$, (1) reduces to the standard linearly constrained convex problem $\min f(x)$ s. to $Ax = b,\ 0 \le x \le u$.

3. Overview of the interior-point method for block-angular problems

Problem (1) can be written as the general convex optimization problem

$$\begin{array}{l} \min \; f(x) \\ \text{s. to } Ax = b \\ \qquad 0 \le x \le u, \end{array} \tag{4}$$

where $x, u \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $f : \mathbb{R}^n \to \mathbb{R}$. Vectors x, u and b are made up of the $k+1$ vectors x^i, u^i and b^i, $i = 0 \ldots, k$.

By convexity, the first order KKT conditions of (4) are necessary and sufficient. Perturbing its complementarity equations with a parameter $\mu \in \mathbb{R}^+$, the KKT-μ conditions

can be written as

$$\begin{array}{rll} r_c & \equiv \nabla f(x) - (A^\top \lambda + z - w) = 0, \\ r_b & \equiv b - Ax = 0, \\ r_{xz} & \equiv \mu e - XZe = 0, \\ r_{sw} & \equiv \mu e - SWe = 0, \\ & (x, s, z, w) \geq 0, \end{array} \tag{5}$$

where $e \in \mathbb{R}^n$ is a vector of 1's; $\lambda \in \mathbb{R}^m$, $z \in \mathbb{R}^n$ and $w \in \mathbb{R}^n$ are, respectively, the vectors of Lagrange multipliers of the equality constraints, lower and upper bounds; $s = u - x$; and matrices $X, Z, S, W \in \mathbb{R}^{n \times n}$ are diagonal matrices made up of vectors x, z, s, w. The set of unique solutions of (5) for each μ value is known as the central path, and when $\mu \to 0$ these solutions converge to those of (4). The nonlinear system (5) is usually solved by a sequence of damped Newton's directions (i.e., with step length reduction to preserve the nonnegativity of variables), reducing the μ parameter at each iteration. This procedure is known as the primal-dual path-following interior-point algorithm. An excellent discussion about the theoretical properties of this and other interior-point algorithms can be found in [30, 35, 37].

The linearization of (5) provides a linear system of variables Δx, $\Delta\lambda$, Δz and Δw. After eliminating Δw and Δz, as follows:

$$\Delta z = X^{-1} r_{xz} - X^{-1} Z \Delta x \tag{6}$$

$$\Delta w = S^{-1} r_{sw} + S^{-1} W \Delta x, \tag{7}$$

we obtain the augmented system form

$$\begin{bmatrix} -\Theta^{-1} & A^\top \\ A & \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} r \\ r_b \end{bmatrix}, \tag{8}$$

where Θ and r are defined as

$$\Theta = (ZX^{-1} + WS^{-1} + \nabla^2 f(x))^{-1} \qquad r = r_c + S^{-1} r_{sw} - X^{-1} r_{xz}. \tag{9}$$

Note that, if the objective function is separable, Θ is an easily computable diagonal matrix. Additionally, eliminating Δx from the first group of equations of (8), the normal equations are obtained:

$$(A\Theta A^\top)\Delta\lambda = r_b + A\Theta r \tag{10}$$

$$\Delta x = \Theta(A^\top \Delta\lambda - r). \tag{11}$$

The method of this paper solves the normal equations, and thus the Newton direction is computed by (10), (11), (6) and (7).

Exploiting the structure of A, and appropriately partitioning Θ, as follows

$$A = \begin{bmatrix} A_1 & & \\ & \ddots & \\ & & A_k \\ L_1 & \dots & L_k \; I \end{bmatrix} \qquad \Theta = \begin{bmatrix} \Theta_1 & & & \\ & \ddots & & \\ & & \Theta_k & \\ & & & \Theta_0 \end{bmatrix},$$

the matrix of system (10) can be recast as

$$A\Theta A^\top = \left[\begin{array}{ccc|c} A_1\Theta_1 A_1^\top & & & A_1\Theta_1 L_1^\top \\ & \ddots & & \vdots \\ & & A_k\Theta_k A_k^\top & A_k\Theta_k L_k^\top \\ \hline L_1\Theta_1 A_1^\top & \dots & L_k\Theta_k A_k^\top & \Theta_0 + \sum_{i=1}^k L_i\Theta_i L_i^\top \end{array}\right] = \begin{bmatrix} B & C \\ C^\top & D \end{bmatrix}, \tag{12}$$

$B \in \mathbb{R}^{\tilde{n}\times\tilde{n}}$ ($\tilde{n} = \sum_{i=1}^k n_i$), $C \in \mathbb{R}^{\tilde{n}\times l}$ and $D \in \mathbb{R}^{l\times l}$ 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 (1), i.e., $\Theta_i = (Z_i X_i^{-1} + W_i S_i^{-1} + \nabla^2 f_i(x^i))^{-1}$. Denoting by g the right-hand-side of (10), and appropriately partitioning g and $\Delta\lambda$, the normal equations can be written as

$$\begin{bmatrix} B & C \\ C^T & D \end{bmatrix} \begin{bmatrix} \Delta\lambda_1 \\ \Delta\lambda_2 \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix}. \tag{13}$$

3.1 *Solving the normal equations by PCG*

Eliminating $\Delta\lambda_1$ from the first group of equations of (13), we obtain

$$(D - C^\top B^{-1} C)\Delta\lambda_2 = (g_2 - C^\top B^{-1} g_1) \tag{14}$$

$$B\Delta\lambda_1 = (g_1 - C\Delta y_2). \tag{15}$$

System (15) is solved by performing k Cholesky factorizations, one for each diagonal block $A_i\Theta_i A_i^\top, i = 1 \dots k$, of B. System (14) with the Schur complement

$$S = D - C^\top B^{-1} C, \tag{16}$$

of dimension l—the number of linking constraints—, may exhibit a large fill-in, and it is prohibitive if computed by Cholesky factorization. It will be solved by a PCG, which is outlined in Figure 1. A good preconditioner is instrumental. The solver in this work considers the preconditioner derived in [10] for multicommodity flows, and extended to general problems in [12]. The preconditioner relies on the fact that (16) is a *P-regular splitting*, i.e., S is symmetric and positive definite, D is nonsingular and $D + C^\top B^{-1} C$ is positive definite. Therefore the *P-regular splitting theorem* [4] [33, pp. 254–255] guarantees that

$$\rho(D^{-1}(C^\top B^{-1} C)) < 1, \tag{17}$$

where $\rho(\cdot)$ denotes the spectral radius of a matrix (i.e., the maximum absolute eigenvalue). (To simplify the notation, $\rho(D^{-1}(C^\top B^{-1} C)$ will be referred as to simply ρ.) This allows us to compute the inverse of S, as shown by the next result (see [10, Prop. 4] for a proof).

PROPOSITION 3.1 *The inverse of the Schur complement $D - C^\top B^{-1} C$ can be computed as*

$$(D - C^\top B^{-1} C)^{-1} = \left(\sum_{i=0}^{\infty} (D^{-1}(C^\top B^{-1} C))^i\right) D^{-1}. \tag{18}$$

1. **Algorithm** *PCG*$(S, M, \bar{g}, \Delta\lambda_{2_0}, \epsilon, i_{\max})$
2. // Solve $S\Delta\lambda_2 = \bar{g}$ by PCG with preconditioner M
3. **Initializations:** $i := 0$; $r_0 := \bar{g} - S\Delta\lambda_{2_0}$;
4. Solve $Mz_0 = r_0$; $p_0 := z_0$;
5. **while** $||r_k|| > \epsilon$ and $i < i_{\max}$ **do**
6. $q_i := Sp_i$;
7. $\alpha_i := (z_i^\top r_i)/(p_i^\top q_i)$;
8. $\Delta\lambda_{2_{i+1}} := \Delta\lambda_{2_i} + \alpha_i p_i$;
9. $r_{i+1} := r_i - \alpha_i q_i$;
10. Solve $Mz_{i+1} = r_{i+1}$;
11. $\beta_i := (z_{i+1}^\top r_{i+1})/(z_i^\top r_i)$;
12. $p_{i+1} := z_{i+1} + \beta_i p_i$;
13. $i := i + 1$;
14. **end while**
15. Return $\Delta\lambda_2 := \Delta\lambda_{2_i}$;
16. **End_algorithm**

Figure 1. The PCG algorithm for the solution of $S\Delta\lambda_2 = \bar{g} \equiv g_2 - C^\top B^{-1} g_1$ with preconditioner M

1. **Algorithm** $Mz = r(D, C, B, r, h)$
2. $v := D^{-1}r$;
3. $z_0 := v$;
4. **for** $j := 1$ to h **do**
5. $z_j := D^{-1}(C^\top(B^{-1}(Cz_{j-1}))) + v$;
6. **end for**
7. Return $z := z_h$

Figure 2. Algorithm for computing $z = M^{-1}r$

The preconditioner M^{-1} is an approximation of S^{-1} obtained by truncating the infinite power series (18) at some term h. For instance, for $h = 0$ and $h = 1$ we have

$$\begin{aligned} M^{-1} &= D^{-1} && \text{if } h = 0, \\ M^{-1} &= (I + D^{-1}(C^\top B^{-1} C))D^{-1} && \text{if } h = 1. \end{aligned}$$

The larger h, the better the approximation of the inverse. On the other hand, systems $Mz = r$ (for some vectors z and r) have to be solved at each PCG iteration (step 10 of PCG algorithm of Figure 1), and any extra term in the series means an additional linear system solution with matrix B. This is clearly seen in the algorithm of Figure 2, which shows how $Mz = r$ is iteratively computed in the specialized interior-point solver. Note that matrix $C^\top B^{-1} C$ does not need to be built, and aside from the solution of systems with B and D, only matrix-vector products with C and $C^\top$ (i.e., with L_i, $L_i^\top$, A_i and $A_i^\top$, $i = 1, \ldots, k$) are required. This also applies to the PCG algorithm of Figure 1. It is thus possible to partially apply the matrix-free paradigm [20]. Efficient implementations of this matrix-vector products for particular A_i and L_i, $i = 1, \ldots, k$, matrices can significantly speed the computational efficiency. Aside from general matrices (either stored rowwise or columnwise), the solver of this paper includes matrix-vector routines for some particular classes of matrices, such as node-arc incidence matrices (both for oriented and nonoriented flows), diagonal and identity matrices, and diagonal-diagonal (i.e., $[D_1\ D_2]$, D_1, D_2 being diagonal) and identity-identity matrices (i.e., $[I\ I]$, I being the identity matrix). Other matrix types can be easily added.

The value of h that optimizes the tradeoff between a good quality and an efficient preconditioner is problem dependent, and finding a dynamic updating procedure for each interior-point iteration is still work in progress. It is worth noting that, although Proposition 3.1 is valid for any primal block-angular problem, the preconditioner is only useful in practice for separable problems; otherwise, nondiagonal Θ_i matrices make systems with B prohibitive.

3.2 *Improving the spectral radius*

The quality of the preconditioner depends on ρ, which is always in $[0, 1)$: the farther from 1, the closer M^{-1} is to S^{-1}. In practice it has been observed that ρ comes closer to 1 as we approach the optimal solution for most instances. However, as shown in [14], non-zero Hessians reduce ρ, and this opens the possibility for improving the preconditioner by the addition of a quadratic regularization term. This assertion is supported by the following theorem and proposition (see [14] for a proof):

THEOREM 3.2 *The spectral radius ρ of $D^{-1}(C^\top B^{-1}C)$ is bounded by*

$$0 \leq \rho \leq \max_{j\in\{1,\ldots,l\}} \frac{\gamma_j}{\left(\frac{u_j}{v_j}\right)^2 \Theta_{0j} + \gamma_j} < 1, \tag{19}$$

where u is the eigenvector (or one of the eigenvectors) of $D^{-1}(C^\top B^{-1}C)$ for ρ; γ_j, $j = 1, \ldots, l$, and $V = [V_1 \ldots V_l]$, are respectively the eigenvalues and matrix of columnwise eigenvectors of $\sum_{i=1}^{k} L_i \Theta_i {L_i}^\top$; $v = V^\top u$; and, abusing of notation, we assume that for $v_j = 0$, $(u_j/v_j)^2 = +\infty$.

PROPOSITION 3.3 *Consider a linear problem and a non-linear one obtained by adding (likely small) Hessian terms $\nabla^2 f_i(x^i) \succ 0, i = 1, \ldots, k$. Assume $\hat{u}_j/\hat{v}_j \leq u_j/v_j$, $j = 1, \ldots, l$, where "hatted" and "non-hatted" terms refer, respectively, to the linear and non-linear problems, and u and v are defined as in Theorem 3.2. Then bound (19) is smaller for the non-linear than for the linear problem.*

The fulfillment of the technical condition $\hat{u}_j/\hat{v}_j \leq u_j/v_j$, $j = 1, \ldots, l$ in Proposition 3.3 is problem dependent, and it may not be easy to check. However for some important classes of problems, such as those where L_i, $i = 1, \ldots, k$, are diagonal matrices, this assumption may be seen to hold [14]. It is also worth to note that the presence of slacks in the linking constraints is instrumental, otherwise $\Theta_0 = 0$ and the bound (19) would be equal to 1.

Theorem 3.2 and Proposition 3.3 state that the bound on the spectral radius (and thus eventually the spectral radius when it is close to 1) is reduced if we consider a Hessian term to Θ, defined in (9). In the limit, as shown by next proposition (see [14] for a proof), the spectral radius goes to 0 for very large Hessians, and therefore PCG will become extremely efficient:

PROPOSITION 3.4

$$\lim_{\substack{\nabla^2 f_i(x^i)\to+\infty \\ i=1,\ldots,k}} \rho = 0. \tag{20}$$

In practice, if the problem is linear or the Hessian is close to 0, a non-zero Hessian can be added by a quadratic regularization. The interior-point solver of this work implements two types of regularization: a proximal point and a quadratic regularization. They are based on the addition of a quadratic term to the standard logarithmic barrier function $B(x, \mu)$ of (4)

$$B(x, \mu) \triangleq f(x) + \mu \left(-\sum_{i=1}^{n} \ln x_i - \sum_{i=1}^{n} \ln(u_i - x_i) \right), \tag{21}$$

$\mu \in \mathbb{R}^+$ being the barrier parameter. In the proximal point regularization, $B(x,\mu)$ is replaced by

$$B_P(x,\mu) \triangleq f(x) + \frac{1}{2}(x-\bar{x})^\top Q_P(x-\bar{x}) + \mu\left(-\sum_{i=1}^{n}\ln x_i - \sum_{i=1}^{n}\ln(u_i - x_i)\right), \qquad (22)$$

Q_P being a diagonal positive definite matrix (which can be dynamically updated at each interior-point iteration, as done in [1]), and $\bar{x}$ the current point obtained by the interior-point algorithm. The alternative quadratic regularization introduced in [14] considers

$$B_Q(x,\mu) \triangleq f(x) + \mu\left(\frac{1}{2}x^\top Q_R x - \sum_{i=1}^{n}\ln x_i - \sum_{i=1}^{n}\ln(u_i - x_i)\right), \qquad (23)$$

Q_R being a diagonal positive semidefinite matrix. Unlike B_P, B_Q does not depend on the current point, and its reduction to 0 is controlled by μ, the standard barrier parameter.

Using either B, B_P, or B_Q only changes the dual feasibility of KKT conditions and matrix Θ, defined in (5) and (9), respectively. Dual feasibility becomes

$$A^\top\lambda + z - w = \nabla f(x) \qquad \text{for } B, \qquad (24)$$

$$A^\top\lambda + z - w = \nabla f(x) + Q_P(x-\bar{x}) \qquad \text{for } B_P, \text{ and} \qquad (25)$$

$$A^\top\lambda + z - w = \nabla f(x) + \mu Q_R x \qquad \text{for } B_Q. \qquad (26)$$

(26) is equivalent to (24) when μ tends to zero (i.e., when we approach the optimal solution), whereas this only happens for (25) when evaluated at current point ($x = \bar{x}$). The Θ matrices are

$$\Theta = (ZX^{-1} + WS^{-1} + \nabla^2 f(x))^{-1} \qquad \text{for } B, \qquad (27)$$

$$\Theta = (Q_P + ZX^{-1} + WS^{-1} + \nabla^2 f(x))^{-1} \qquad \text{for } B_P, \text{ and} \qquad (28)$$

$$\Theta = (\mu Q_R + ZX^{-1} + WS^{-1} + \nabla^2 f(x))^{-1} \qquad \text{for } B_Q. \qquad (29)$$

The main difference between (28) and (29) is that μQ_R tends to zero with μ and therefore (29) approximates (27) better than (28). Therefore, in general, B_Q should be preferred to B_P. The computational results of this paper have been obtained with B_Q.

3.3 *Estimating the spectral radius*

Although knowing the spectral radius ρ of $D^{-1}(C^\top B^{-1}C)$ would be instrumental to forecast the efficiency of the preconditioner, its computation is impractical. Even computing its upper bound (19) may be prohibitive, except for some particular classes of problems.

However, a procedure to estimate ρ was recently introduced in [9] for $h = 0$, i.e., when the preconditioner M^{-1} only includes one term of the power series, thus being equal to D^{-1}. In this case, the preconditioned system $(D - C^\top B^{-1}C)\Delta\lambda_2 = \bar{g}$ solved by PCG in asymmetric form is

$$(I - D^{-1}(C^\top B^{-1}C))\Delta\lambda_2 = D^{-1}\bar{g}.$$

Clearly, by linear algebra, if $\sigma_{\min}$ is the minimum eigenvalue of $I - D^{-1}(C^\top B^{-1}C)$ then $1 - \sigma_{\min}$ is the spectral radius of $D^{-1}(C^\top B^{-1}C)$. The minimum eigenvalue $\sigma_{\min}$ of $I -$

$D^{-1}(C^\top B^{-1}C)$ can be estimated from the solution of (10) by PCG, using the relation between PCG and Lanczos method [26]. From this relation (see [24, Chapter 9],[28] for details) it is known that the eigenvalues of the tridiagonal matrices

$$T_k = \begin{bmatrix} \gamma_1 & \eta_2 & & & \\ \eta_2 & \gamma_2 & \eta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \eta_{k-1} & \gamma_{k-1} & \eta_k \\ & & & \eta_k & \gamma_k \end{bmatrix}, \tag{30}$$

$k = 1, \ldots, l$, converge to the eigenvalues of the preconditioned matrix of the system solved by PCG as k (the number of PCG iterations) approaches l, where the coefficients γ_i and η_j can be computed from the PCG algorithm of Figure 1 as follows:

$$\gamma_k = \frac{1}{\alpha_{k-1}} + \frac{\beta_{k-1}}{\alpha_{k-2}}, \qquad \beta_0 = 0, \qquad \alpha_{-1} = 0, \qquad \eta_{k+1} = -\frac{\sqrt{\beta_k}}{\alpha_{k-1}}.$$

Eigenvalues of T_k are known as *Ritz values*. In general, the extreme eigenvalues of the preconditioned matrix (the ones we are interested in, for the estimation of the spectral radius) are well approximated already during early PCG iterations [28]. In [9] it was observed that very tight estimations require PCG solutions of high precision.

The previous estimations have been extended in this work to consider any number $h \geq 0$ of terms in the power series preconditioner:

PROPOSITION 3.5 *Let* $M^{-1} = \left(\sum_{i=0}^{h} (D^{-1}(C^\top B^{-1}C))^i \right) D^{-1}$ *be the preconditioner with* h *terms of the power series (18). And let* $\sigma_{\min}$ *be the smallest eigenvalue of the preconditioned matrix* $M^{-1}S$. *Then the spectral radius of* $(D^{-1}(C^\top B^{-1}C))$ *is*

$$\rho = \sqrt[h+1]{1 - \sigma_{\min}}. \tag{31}$$

Proof. Denoting $D^{-1}(C^\top B^{-1}C)$ as Q, we have

$$M^{-1}S = (I + Q + \cdots + Q^h)D^{-1}(D - Q) = I - Q^{h+1}.$$

By linear algebra, if σ is an eigenvalue of $I - Q^{h+1}$ with eigenvector v, then $\sqrt[h+1]{1-\sigma}$ is eigenvalue of Q with the same eigenvector. The spectral radius of Q is thus provided by the smallest eigenvalue $\sigma_{\min}$. ■

By the previous proposition, ρ can be easily estimated as $\sqrt[h+1]{1 - \tilde{\sigma}_{\min}}$, where $\tilde{\sigma}_{\min}$ is the smallest Ritz value.

4. Solver implementation details

The specialized block-angular IPM described in the previous section has been efficiently implemented in a software package named `BlockIP`. `BlockIP` is written in C++, using the object oriented paradigm. It is roughly about 14000 lines of source code, aside from the external package for Cholesky factorization. Actually, `BlockIP` is only linked to the Ng-Peyton block sparse Cholesky package [31], which only implements an approximate

minimum degree ordering; other more recent Cholesky packages can be added in the future, which may likely improve the performance of the solver. The package can be obtained for research purposes from `http://www-eio.upc.edu/~jcastro/BlockIP.html`. The distribution contains a reference manual and an example illustrating the use of the package.

Since `BlockIP` includes the features of the early prototype described in [12], these will be omitted here. Some additional new features of `BlockIP` are:

(1) `BlockIP` may handle linear, quadratic and convex linearly constrained block-angular optimization problems. It may deal with either problems in the standard form (4) and in the more general form

$$\begin{array}{ll} \min & f(x) \\ \text{s. to} & b_l \leq Ax \leq b_r \\ & l \leq x \leq u, \end{array} \tag{32}$$

which are internally transformed to the standard form. Since gradients and Hessians are computed in the original space of variables by user functions, the standard form is preferred (particularly for nonlinear problems) to avoid the extra overhead of transforming the current point to the original space of variables for function evaluations.

(2) `BlockIP` stops at the feasible point of iteration j when the relative gap between the primal and dual objectives, denoted by p^j and d^j, is below some optimality gap. The relative gap is computed as $(p^j - d^j)/(1 + p^j)$. For linear and quadratic problems p^j and d^j are

$$p^j = c^\top x^j + \frac{1}{2} x^{j\top} Q x^j \qquad d^j = b^\top \lambda^j - u^\top w^j - \frac{1}{2} x^{j\top} Q x^j$$

while for nonlinear problems they are computed as

$$p^j = f(x^j) \qquad d^j = \mathcal{L}(x^j, \lambda^j, z^j, w^j) = f(x^j) - \lambda^{j\top}(Ax^j - b) - z^{j\top} x^j - w^{j\top} s^j,$$

$\mathcal{L}$ denoting the Lagrangian function.

(3) `BlockIP` implements two types of directions: the standard Newton direction which was described in Section 3; and the second order heuristic direction of [27], which requires the solution of two systems (10) with different right-hand-sides. Both directions can be computed by solving the normal equations by either a Cholesky factorization or by the PCG-based approach of Subsection 3.1. In general, the reduction of iterations caused by the second order direction is not worthwhile when using PCG, since, as far as we know, the solution of the first system can not be efficiently used as a warm-start for the second one. Indeed, from the results of [10] this strategy was not successful for multicommodity flows problems.

(4) The solver implements the two types of regularizations presented in Subsection 3.2: the proximal point and the quadratic regularization. Although both are similar in practice, the quadratic regularization is preferred by the discussion in Subsection 3.2. Following [14], the quadratic regularization matrix Q_R of (23) is heuristically updated at each interior-point iteration i as

$$Q_R := \delta \cdot i \cdot \mu_i / \mu_0 I,$$

where μ_0 and μ_i are the barrier parameter at the starting and current point, and δ is a (usually small) initial regularization value. The product by i, the iteration counter,

```
...
// declare N (block constraints matrix) as a Matrix for BlockIP
MatrixBlockIP N;
// declare arc source and destination vectors
int *src, *dst;
// N is created as network matrix
// numArcs and numNodes previously assigned; true means oriented
N.create_network_matrix(numArcs, numNodes, src, dst, true);
// fill src and dst; src and dst allocated by create_network_matrix()
...
// declare L (linking constraints matrix) as a Matrix for BlockIP
MatrixBlockIP L;
// L is created as an identity  matrix
L.create_identity_matrix(numArcs);

BlockIP bip; // declare BlockIP problem

double *cost, *qcost, *ub, *rhs;
// creation of BlockIP problem
// numBlocks previosuly assigned; true means same N and L for all blocks
bip.create_problem(BlockIP::QUADRATIC, cost, qcost, NULL, NULL, ub, rhs,
                   numBlocks, true, &N, true, &L);
// fill cost, qcost, ub, rhs.
...
bip.minimize();
```

Figure 3. Piece of code illustrating the usage of the `BlockIP` callable library for the solution of a oriented quadratic multicommodity flow problem

```
#typeobj 0=linear 1=quadratic 2=nonlinear
1
#number of blocks
2
#sameN 1=yes 0=no
1
#Matrix: first line m,n,nnz; next nnz lines i,j,a
3 5 7
1 1 1
1 2 1
1 3 1
2 1 -1
2 4 1
3 2 -1
3 5 1
...
```

Figure 4. Example using the particular `BlockIP` file format

is an attempt to compensate for the quick reduction of μ_i when the optimal solution is approached.

(5) `BlockIP` may compute the Ritz values, and thus the spectral radius ρ, for any number h of terms in the preconditioner, as described in Subsection 3.3. The value h can also be selected by the user. As in [9], Ritz values are efficiently computed by using the SSTEQR LAPACK routine [2]; the extra CPU time needed by this computation is negligible.

(6) Problems can be provided in four different formats.

(a) The most efficient way is using the `BlockIP` callable library, which provides routines to create problems from matrices and vectors. The code of Figure 3 illustrates how the callable library could be used to formulate, in this example, a oriented quadratic multicommodity flow problem. Particular matrix formats (network—oriented and nonoriented—, general rowwise or columnwise, diagonal,

```
ROWS
E Block1:Cons1
...
E LinkCons1
...
COLUMNS
Block1:Var1 obj 1 Block1:Cons1 1
...
Slack1 LinkCons1 1
...
```

Figure 5. Syntax of row and column entries of structured MPS extension for block-angular problems

```
set ORIG; # origins
set DEST; # destinations
set PROD; # products
param supply {PROD,ORIG} >= 0; # amounts available at origins
param demand {PROD,DEST} >= 0; # amounts required at destinations
param limit {ORIG,DEST} >= 0;
param cost {PROD,ORIG,DEST} >= 0; # shipment costs per unit

block Prod{p in PROD}:
  var Trans {ORIG, DEST} >= 0; # units to be shipped
  minimize total_cost: sum {i in ORIG,j in DEST} cost[p,i,j]*Trans[i,j];
  subject to Supply {i in ORIG}: sum {j in DEST} Trans[i,j] = supply[p,i];
  subject to Demand {j in DEST}: sum {i in ORIG} Trans[i,j] = demand[p,j];
end block;

subject to Multi {i in ORIG, j in DEST}:
  sum {p in PROD} Prod[p].Trans[i,j] <= limit[i,j];
```

Figure 6. Example of SML–AMPL code for a multicommodity transportation problem

identity, etc.) can be exploited through the callable library. Nonlinear objective functions—including gradient and Hessian evaluations—have to be provided as C++ routines.

(b) Problems can also be efficiently provided by an input file using a specific format for `BlockIP`. This format consists on a set of scalars, vectors and sparse matrices defining the problem parameters. Sparse matrices are provided in coordinate scheme, i.e, a set of triples (i, j, a_{ij}) where (i, j) denote the row and column location of the nonzero element a_{ij}. An example is shown in Figure 4.

(c) Input files can also be in *structured MPS format*, an extension of the well-known MPS format created for `BlockIP`. Figure 5 illustrates the syntax of this format. All variables and constraints preceded by the same prefix (ended with ":") are associated to the same block. Constraints and variables without a prefix correspond, respectively, to linking constraints and their slacks. Standard packages can read structured MPS files without modification.

(d) The last format is based on SML [15], a structure-conveying modelling language based on the popular AMPL [18] modeling language. In addition to hooking `BlockIP` to it, SML was extended to deal with nonlinear separable problems (see [25] for details). Briefly, SML extends AMPL with the `block` and `end block` keywords. Variables and constraints defined within these keywords are associated to the same block, whereas those outside correspond to linking constraints and their slacks. Figure 6 illustrates how a simple multicommodity transportation problem from [18] could be formulated with SML.

5. Computational results with some applications

Three applications have been considered for testing the performance of `BlockIP`: the multicommodity flow problem (linear optimization model), the minimum congestion problem (linear optimization model), and a statistical tabular data confidentiality problem (linear, quadratic or nonlinear optimization model). `BlockIP` has been compared with the state-of-the-art CPLEX 12.5 package for the linear and quadratic instances. Both the simplex and (interior-point) barrier CPLEX algorithms have been tried for all the instances, and the tables of next subsections report the results with the most efficient option for each instance. Actually, the most efficient CPLEX option was always the barrier algorithm with the nested dissection ordering, except for instance "gridgen1" of Table 1 where it was outperformed by the barrier with the approximate minimum degree ordering. It is worth to note that the Cholesky solver of `BlockIP` does not implement the nested dissection ordering, so there is room for significant improvement. For the nonlinear problems we only report results with `BlockIP`, since other nonlinear interior-point packages could not solve them. A value $h = 0$ was considered for the number of terms of the power series preconditioner. The quadratic regularization (23) has been used for `BlockIP`. Default options have been used for the other CPLEX and `BlockIP` parameters, unless explicitly stated. The CPU time provided for CPLEX is only for the barrier iterations, without crossover. All runs were carried out on a Fujitsu Primergy RX300 server with 3.33 GHz Intel Xeon X5680 CPUs with 144 gigabytes of memory, under a GNU/Linux operating system (OpenSuse 11.4), without exploitation of parallelism capabilities.

The optimality gap was set to 10^{-5} for both `BlockIP` and CPLEX, unless otherwise stated, since tighter tolerances may be problematic for the power series preconditioner. Indeed, as discussed in previous sections, since the spectral radius approaches 1 near the optimal solution, the performance of the preconditioner may degrade close to optimality. It is worth to note that with other preconditioners it is possible to achieve optimality tolerances of 10^{-8}, which is the standard with direct methods. This is the case, for instance, for the splitting preconditioner of [32], though it is more time consuming than the power series one, and it is devised for general linear optimization problems, thus, it does not exploit the problem structure. A hybrid approach combining both preconditioners was successfully tested in [9]; the inclusion of the splitting preconditioner within `BlockIP` for the last interior-point iterations is part of the further tasks to be done. We remark that, according to Propositions 3.3 and 3.4, for quadratic optimization problems the power series preconditioner may be very efficient and tighter optimality tolerances can be achieved.

5.1 *Multicommodity problems*

The purpose of the multicommodity problem is to route a set of items (the commodities) at a minimum cost over a capacitated network. This problem is formulated as

$$\begin{aligned} \min & \sum_{i=1}^{k} {c^i}^\top x^i \\ \text{s. to } & Nx^i = b^i \quad i = 1, \ldots, k \\ & \sum_{i=1}^{k} x^i + s \leq u \\ & 0 \leq x^i \leq u^i \quad i = 1, \ldots, k, \qquad 0 \leq s \leq u, \end{aligned} \tag{33}$$

Table 1. Dimensions and results for multicommodity instances

				BlockIP			CPLEX 12.5		BlockIP switch		
Instance	k	m	n	Iter	CPU	PCG	Iter	CPU	Iter	CPU	PCG
tripart1	16	3294	33774	52	0.8	1260	19	**0.3**	47	0.9	497
tripart2	16	13301	135941	69	10	4034	17	**4**	67	10	756
tripart3	20	25541	329161	79	20	3363	19	**13**	71	38	1717
tripart4	35	38004	869814	132	268	20791	24	**34**	121	223	4688
gridgen1	320	329831	985191	196	**216**	3938	20	545	163	110[†]	1299

[†] Stopped after switching: Cholesky factor is almost dense (fill-in is 99.6%)

where N is a node-arc incidence matrix of $m'+1$ nodes (one node is removed to guarantee full row-rank) and n' arcs, b^i are the vectors of supply-demand at the nodes for each commodity, s are the slacks of the linking constraints, u^i are the vectors of arc capacities for each commodity, and u is the vector of arc total capacities for all the commodities. The number of linking constraints is $l = n'$. Multicommodity problems match the general block-angular formulation (1) with $A_i = N$, $L_i = I$, and $f_i(x^i) = {c^i}^\top x^i$, for $i = 1, \ldots, k$. Note that D, defined in (12), is diagonal, since L_i are identities, and thus step 10 of the PCG algorithm of Figure 1 only involves a diagonal matrix when $h = 0$.

Several standard classes of multicommodity instances are available from the literature. Many of them are already satisfactorily solved by generic solvers [8]. Therefore, we selected a small set of—small/medium—five instances which are considered difficult in the literature [6, 11]. Those instances are available from `http://www-eio.upc.edu/~jcastro/mmcnf_data.html`. Table 1 reports, for each instance, the number of blocks (k), constraints (m) and variables (n); the number of IPM iterations ("Iter") and CPU time ("CPU") required by `BlockIP` and CPLEX; and the overall number of PCG iterations ("PCG") performed by `BlockIP`. The CPU of the faster run is marked in boldface. Though not very large, those five instances are difficult for `BlockIP` since the spectral radius quickly goes to 1, thus requiring a large number of PCG iterations in the last IPM iterations. This mainly happened for the last "gridgen1" instance, where the optimality gap was reduced to 10^{-3} (for both `BlockIP` and CPLEX).

To avoid this large number or PCG iterations in `BlockIP` when we are close to the solution, we considered the switching to a Cholesky factorization of the normal equations when the optimality gap is below some threshold value (in particular we used 10^{-2}). Results are reported in the last three columns of Table 1, showing that this strategy saved most of PCG iterations in the two largest instances. However, for "gridgen1" the code was stopped after the switching since the Cholesky factorization contained about 491 million nonzero coefficients, thus being very time consuming. This number for the Cholesky factorization of CPLEX, also using a minimum degree ordering, was of 98 million. The efficiency of `BlockIP` could thus be significantly improved by replacing the Ng-Peyton Cholesky package [31] with some more recent alternative.

To observe the effect of h—the number of terms of the power series (18) included in the preconditioner—we ran again the instances for $h \in \{1, 2, 3, 4\}$ (the default value $h = 0$ was used in the executions of Table 1). Results are shown in Figure 7. The left plot reports the ratio "PCG iterations"/"IP iterations" for each instance and value of h. Clearly, the ratio decreases with h, that is, the more terms we consider, the better is the preconditioner, and therefore less PCG iterations are required in average to get the desired accuracy. On the other hand, the computational cost of the system with the preconditioner increases with h. The right plot of Figure 7 shows the overall CPU time for each instance and value of h. In general, $h = 0$ is the most efficient option, and the CPU time usually increases with h. We remark that when $h = 0$ the preconditioner is equal to D, which is diagonal in this problem. However, for the most difficult instance "gridgen1" the best results (though by a small margin) were obtained with $h = 3$ and $h = 4$. This can be explained from the

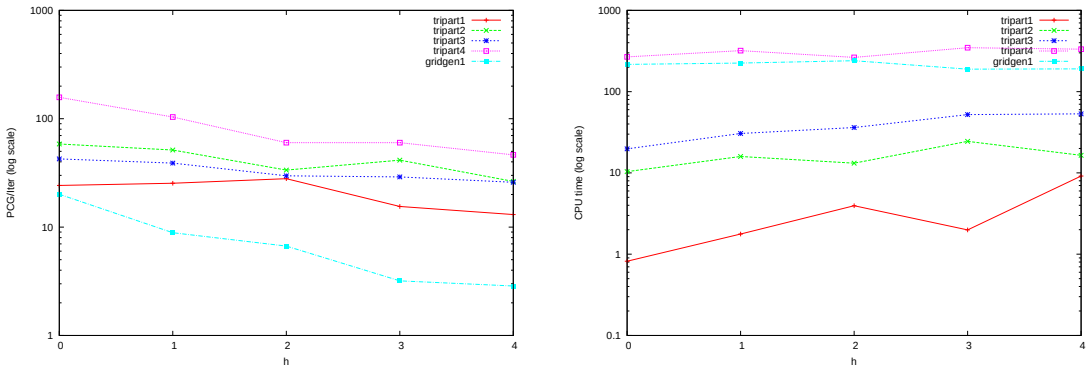

Figure 7. PCG iterations per IP iteration (left) and CPU time (right) time for different values of h

Table 2. Results for quadratic multicommodity instances

	BlockIP			CPLEX 12.5	
Instance	Iter	CPU	PCG	Iter	CPU
tripart1	33	**0.2**	130	18	0.4
tripart2	46	**1.5**	216	24	4.7
tripart3	65	**5**	279	31	17
tripart4	81	**18**	374	29	36
gridgen1	196	**226**	4223	48	1252

left plot, by the significant reduction in the number of PCG iterations per IP iteration in this instance.

As seen in Subsection 3.2, quadratic terms in the objective reduce the spectral radius, thus increasing the quality of the preconditioner. To support this claim, we generated five quadratic multicommodity cases by adding the quadratic term $\frac{1}{2}x^\top Qx$, with $Q = \alpha I$, to the above linear instances. The results obtained with $\alpha = 0.1$ and an optimality gap of 10^{-4} for `BlockIP` and CPLEX are reported in Table 2, clearly showing the good performance of `BlockIP` as a quadratic multicommodity flow solver.

5.2 *Minimum congestion problems*

The minimum congestion problem (also known as the maximum concurrent flow [7]) is defined on an infeasible nonoriented multicommodity flow problems. Its purpose is to minimize $\|y\|_\infty$, where y is the vector of relative increments in arc capacities needed to make the multicommodity flow problem feasible. This problem, with practical applications in telecommunications, has proved to be difficult for simplex algorithms [6].

Denoting by x^{i^+} and x^{i^-} the forward and backward flows for each commodity, and considering the extra variable $t \in \mathbb{R}$, it can be formulated as:

$$\begin{aligned} &\min\ t \\ &\text{s. to } Nx^{i^+} - Nx^{i^-} = b^i && i = 1,\dots,k \\ &\quad \sum_{i=1}^{k}(x_j^{i^+} + x_j^{i^-}) - y_j u_j \le 0 && j = 1,\dots,n' \\ &\quad y_j - t \le 0 && j = 1,\dots,n' \\ &\quad x^{i^+}, x^{i^-} \ge 0 && i = 1,\dots,k \\ &\quad y_j \ge 0 && j = 1,\dots,n' \end{aligned} \tag{34}$$

This formulation contains a dense column because of t, and thus the matrix D (needed for the preconditioner) becomes very dense. A more appropriate formulation is obtained

Table 3. Dimensions and results for minimum congestion instances

				BlockIP			CPLEX 12.5	
Instance	k	m	n	Iter	CPU	PCG	Iter	CPU
M32-32	34	2449	33533	94	**0.9**	289	17	1.3
M64-64	66	5564	67962	95	**2**	183	17	4
M128-64	66	11640	155742	98	**7**	234	19	22
M128-128	130	19867	314243	98	**15**	213	20	52
M256-256	258	71891	1139467	111	**161**	891	22	627
M512-64	66	470075	634143	132	**95**	1223	21	1071
M512-128	130	79765	1249145	132	**244**	2090	25	2520

by considering $t_i, i = 1, \ldots, n'$ for each arc, with the extra constraints $t_i = t_{i+1}$. The resulting model is:

$$\begin{array}{ll} \min\ t_1 & \\ \text{s. to } Nx^{i^+} - Nx^{i^-} = b^i & i = 1, \ldots, k \\ \sum_{i=1}^{k} (x_j^{i^+} + x_j^{i^-}) - y_j u_j \leq 0 & j = 1, \ldots, n' \\ y_j - t_j \leq 0 & j = 1, \ldots, n' \\ t_j - t_{j+1} = 0 & j = 1, \ldots, n' - 1 \\ x^{i^+}, x^{i^-} \geq 0 & i = 1, \ldots, k \\ y_j \geq 0 & j = 1, \ldots, n' \end{array} \tag{35}$$

Matrix D of the preconditioner for (35) is of larger dimension but sparser. Unlike in Subsection 5.1, the linking constraints, and thus D, are not diagonal matrices.

Table 3 reports the dimensions and results for some instances. They were obtained by making infeasible some multicommodity instances (produced with the standard Mnetgen generator) increasing the demands and supplies. The meaning of the columns is the same as in Table 1. The two numbers in the instance name denote the number of arcs and nodes of the network. `BlockIP` was competitive even for the smaller instances, being up to ten times faster for the larger ones.

As in Subsection 5.1, we also ran the instances for different values of h. The results are shown in Figure 8. The meaning of the plots is the same than for Figure 7. For the four smaller instances, there is a clear reduction in the average number of PCG iterations per IP iteration as h increases. The CPU time was very similar for all h, the best results being obtained for $h = 1$. This can be explained by D not being a diagonal matrix, so $h = 0$ is not as efficient as in Subsection 5.1. However this behaviour is not observed for the three larger instances, where $h = 0$ outperformed all the other versions, and the average number of PCG iterations did not monotonically reduce with h. For cases "M512-64" and "M512-128" and some $h > 0$ values, PCG did not converge and the executions were prematurely stopped; those cases correspond to the empty segments in the plots.

5.3 *Statistical tabular data confidentiality problems*

National statistical agencies have to guarantee that confidential information can not be be disclosed from tabular data released. One of the data protection approaches consists, broadly speaking, in computing the closest safe table to the original one, in an attempt to minimize the information loss (see, for instance, [13] for a recent survey on these topics). Representing a table of n' cells as a vector $a \in \mathbb{R}^{n'}$ that satisfies the constraints $Aa = b, l_a \leq a \leq u_a$, the goal is to find cell perturbations $x \in \mathbb{R}^{n'}$ such that:

- minimize $||x||_\ell$ for some distance ℓ;

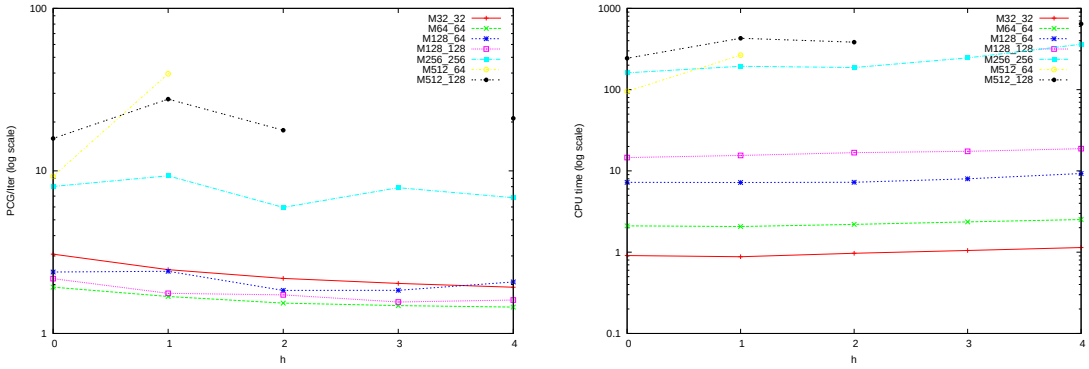

Figure 8. PCG iterations per IP iteration (left) and CPU time (right) time for different values of h

- $x+a$ satisfy the table constraints, that is, $A(x+a) = b,\ l_a \leq x+a \leq u_a$, or equivalently $Ax = 0,\ l \leq x \leq u$, where $l = l_a - a$ and $u = u_a - a$;
- a subset of *sensitive cells* $\mathcal{S} \subseteq \{1, \ldots, n'\}$ is safely shifted from their original values, that is, $\alpha_i \leq x_i \leq \beta_i \quad i \in \mathcal{S}$, where α_i, β_i are some given parameters such that $0 \notin [\alpha_i, \beta_i]$.

This results in the following optimization problem:

$$\begin{array}{ll} \min\limits_{x} & ||x||_\ell \\ \text{s. to} & Ax = 0 \\ & l \leq x \leq u \\ & \alpha_i \leq x_i \leq \beta_i \ i \in \mathcal{S}, \end{array} \tag{36}$$

which exhibits a block-angular structure for some classes of tables, such as three-dimensional ones, i.e., boxes of data obtained by crossing three categorical variables (see [13] for details). Since all the linking constraints of these instances are equalities, the upper bounds u^0 of the slacks were successfully set to a small value ($2.22 \cdot 10^{-16}$, the machine epsilon).

Using ℓ_1, and considering the splitting $x = x^+ - x^-, x^+ \geq 0, x^- \geq 0$, the objective function of (36) becomes

$$\|x\|_{\ell_1} = \sum_{i=1}^{n} |x_i| = \sum_{i=1}^{n} (x_i^+ + x_i^-), \tag{37}$$

obtaining a linear optimization problem.PCG-based IPMs have been used in other ℓ_1 distance applications, such as, for instance, [17] for the solution of compressed sensing problems. Like `BlockIP`, the IPM of [17] also solved the normal equations and its preconditioner resulted to be very efficient . However, both approaches significantly differ in the type of problems they can deal with: the IPM of [17] considers constraints matrices A without any particular structure, but being close to orthonormality (which is a strong requirement, though it is satisfied by compressed sensing instances), while the power series preconditioner of `BlockIP` is, in principle, only appropriate for matrices A with block-angular structure.

For the Euclidean distance ℓ_2, no splitting of variables is necessary, resulting in a quadratic optimization problem of objective

$$\|x\|_{\ell_2}^2 = \sum_{i=1}^{n} x_i^2. \tag{38}$$

Table 4. Dimensions and results for ℓ_1

Instance	k	m	n	BlockIP Iter	BlockIP CPU	BlockIP PCG	CPLEX 12.5 Iter	CPLEX 12.5 CPU
25-25-25	25	1850	31875	168	4	16475	13	**1**
25-25-50	50	3075	63125	172	12	22430	14	**2**
25-50-25	25	3100	63750	194	19	34863	13	**2**
25-50-50	50	4950	126250	200	61	57641	15	**10**
50-25-25	25	3100	63750	200	28	53667	14	**1**
50-25-50	50	4950	126250	62	**1**	526	15	7
50-50-25	25	4975	127500	187	33	28669	15	**9**
50-50-50	50	7450	252500	133	**16**	5523	16	41
100-100-100	100	29900	2010000	23	**8**	25	8	986
100-100-200	200	49800	4010000	32	**25**	35	9	2262
200-100-200	200	79800	8020000	32	**49**	42	10	8789
200-200-200	200	119800	16040000	35	**144**	49	8	64521
500-500-50	50	299950	25250000	40	**424**	57	9	19595
500-50-500	500	299500	25025000	54	**227**	76	9	17415

By Subsection 3.2, the PCG is expected to be more efficient with the quadratic than the linear objective because of the nonzero Hessian. According to that, we also considered a strictly convex nonlinear approximation of ℓ_1 given by the pseudo-Huber function

$$\varphi_\delta(x_i) = \sqrt{\delta^2 + x_i^2} - \delta, \tag{39}$$

δ being a small positive value. The objective function of the resulting convex optimization problem is

$$f(x) = \sum_{i=1}^{n} \varphi_\delta(x_i) \approx \|x\|_{\ell_1}. \tag{40}$$

Properties of the pseudo-Huber function and its application to compressed sensing problems are described in [16]. That work also introduced an efficient primal-dual PCG-based Newton method for ℓ_1-regularization optimization problems, using the pseudo-Huber function as a replacement for the non-differentiable regularization term. However both approaches and problems are significantly different. Our problem (36) is linearly constrained, with lower and upper bounds in the variables, while that of [16] is unconstrained, but involving a second nonlinear term in the objective function. As for the solution method, that of [16] is highly tuned for this kind of ℓ_1-regularization problems—in particular, it considers an ad-hoc reformulation of the KKT conditions. On the other hand, `BlockIP` implements a generic IPM, which is valid for a wider class of problems, but not expected to be as efficient as the approach of [16] for unconstrained ℓ_1-regularization problems.

Tables 4, 5 and 6 report the results with, respectively, (37), (38) and (40). We generated two types of instances, small and large, clearly separated in the tables. The name of the instances denote the number of categories of the three categorical variables. The instance generator of three-dimensional tables can be retrieved from `http://www-eio.upc.edu/~jcastro/CTA_3Dtables.html`. Large instances could not be solved with the pseudo-Huber function. Other state-of-the-art nonlinear IPM packages could not even solve the small instances with the pseudo-Huber function. The meaning of the columns is the same as in tables of previous subsections. For ℓ_1 we see that `BlockIP` is far more efficient as the size of the optimization problem becomes really large (millions of variables), being one to two orders of magnitude faster than CPLEX. For ℓ_2 the results are even better, being two to three orders of magnitude faster in the largest instances. The benefit of a nonzero Hessian in the quality of the preconditioner is clearly seen from the number of PCG iterations: it is smaller for ℓ_2 than for ℓ_1. Table 6 shows the results

Table 5. Dimensions and results for ℓ_2

Instance	k	m	n	BlockIP Iter	BlockIP CPU	BlockIP PCG	CPLEX 12.5 Iter	CPLEX 12.5 CPU
25-25-25	25	1850	16250	11	**0.0**	22	9	0.8
25-25-50	50	3075	31875	10	**0.1**	13	9	1.4
25-50-25	25	3100	32500	10	**0.1**	14	8	1.2
25-50-50	50	4950	63750	10	**0.1**	12	7	5.8
50-25-25	25	3100	32500	10	**0.1**	15	9	1.2
50-25-50	50	4950	63750	10	**0.1**	13	8	4.2
50-50-25	25	4975	65000	10	**0.1**	12	8	5.1
50-50-50	50	7450	127500	10	**0.2**	12	7	19
100-100-100	100	29900	1010000	10	**3**	10	7	874
100-100-200	200	49800	2010000	10	**6**	10	7	1802
200-100-200	200	79800	4020000	10	**11**	10	8	7319
200-200-200	200	119800	8040000	9	**29**	9	8	65467
500-500-50	50	299950	12750000	10	**91**	10	7	15437
500-50-500	500	299500	12525000	10	**28**	10	8	14784

Table 6. Dimensions and results for pseudo-Huber function and ℓ_1 in small instances

Instance	k	m	n	φ_δ Iter	φ_δ CPU	φ_δ PCG	ℓ_1 CPU	ℓ_1 PCG
25-25-25	25	1850	16250	156	**1**	3285	4	16475
25-25-50	50	3075	31875	152	**2**	2940	12	22430
25-50-25	25	3100	32500	146	**2**	2525	19	34863
25-50-50	50	4950	63750	159	**5**	4658	61	57641
50-25-25	25	3100	32500	150	**2**	2404	28	53667
50-25-50	50	4950	63750	143	4	4392	**1**	526
50-50-25	25	4975	65000	163	**4**	3298	33	28669
50-50-50	50	7450	127500	152	**6**	1831	16	5523

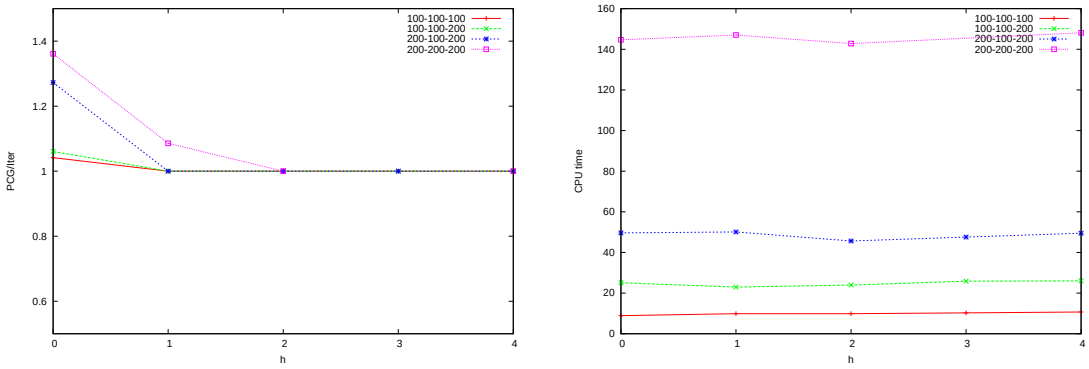

Figure 9. PCG iterations per IP iteration (left) and CPU time (right) time for different values of h

with the pseudo-Huber function. We see again the advantage of a nonzero Hessian: the nonlinear problem is solved more efficiently than the linear one for all the instances (but one), requiring less PCG iterations.

Finally, as in previous subsections, we solved some of the large ℓ_1 instances for different values of h. From left plot of Figure 9 we see that for $h \geq 1$ ($h \geq 2$ for instance "200-200-200") only 1 PCG iteration was needed at each IP iteration with the improved preconditioner. The CPU time was very similar for all h, though the best execution time was always for $h > 0$ except for the smallest instance. Although in those instances D was diagonal for $h = 0$ and already very efficient, it was worth adding an extra term to the preconditioner. This behaviour differs from that observed in Subsection 5.1 with another diagonal preconditioner. Therefore, although as a matter of fact a small h is recommended, the best value is likely to be problem dependent.

6. Conclusions

The `BlockIP` implementation of the PCG-based IPM using the power series preconditioner has shown to be a very efficient tool for the solution of some classes of large convex separable block-angular problems. We do not claim such good efficiencies can be observed in all block-angular problems; this depends, among other factors, on the value of the spectral radius of $D^{-1}(C^\top B^{-1}C)$ which is problem dependent. Therefore, the regularization strategies included in `BlockIP` (either the proximal point or the quadratic terms) may be instrumental for some problems. The estimation of the spectral radius through Ritz values can also be used as a guidance for the suitability of this approach to some particular problem.

Among the further tasks to be done we find: improving the efficiency of the PCG by adaptive selection of h, the number of terms in the preconditioner, using the estimation of ρ by the Ritz values; adaptive selection of either Newton or second-order directions, according to the quality of the preconditioner at each interior-point iteration; testing other (linear and nonlinear) classes of block-angular problems (e.g., routing problems in telecommunication networks, formulated as nonlinear multicommodity flows); using this approach within a more general framework for the solution of large mixed integer problems with block-angular structure (e.g., [29]); and implementing within `BlockIP` other type of preconditioners, such as, e.g., the hybrid approach described in [9]. Some of these tasks are already under development.

Acknowledgments

The interface between `BlockIP` and SML–AMPL and the extension of SML–AMPL to nonlinear separable problems was implemented by Xavi Jiménez. This work has been supported by grant MTM2012-31440 of the Spanish Ministry of Economy and Competitiveness.

References

[1] A. Altman and J. Gondzio, *Regularized symmetric indefinite systems in interior point methods for linear and quadratic optimization*, Optimization Methods & Software 11 (1999), pp. 275–302.

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney and D. Sorensen, *LAPACK Users' Guide, Third edition*, SIAM, Philadelphia, PA, 1999.

[3] S. Bellavia, J. Gondzio and B. Morini, *A matrix-free preconditioner for sparse symmetric positive definite systems and least-squares problems*, SIAM Journal on Scientific Computing 35 (2013), pp. A192-A211.

[4] M. Benzi, *Splittings of symmetric matrices and a question of Ortega*, Linear Algebra and its Applications 429 (2008), pp. 2340-2343.

[5] L. Bergamaschi, J. Gondzio and G. Zilli, *Preconditioning indefinite systems in interior point methods for optimization*, Computational Optimization and Applications 28 (2004), pp. 149–171.

[6] D. Bienstock, *Potential Function Methods for Approximately Solving Linear Programming Problems. Theory and Practice*, Kluwer: Boston, 2002.

[7] D. Bienstock and O. Raskina, Asymptotic analysis of the flow deviation method for the maximum concurrent flow problem, *Mathematical Programming*, 91, pp. 479–492, 2002.

[8] R. E. Bixby, *Solving real-world linear programs: a decade and more of progress*, Operations Research, 50 (2002), pp. 3–15.

[9] 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 (2013), pp. 263–273.

[10] J. Castro, *A specialized interior-point algorithm for multicommodity network flows*, SIAM Journal on Optimization 10 (2000), pp. 852–877.
[11] J. Castro, *Solving difficult multicommodity problems through a specialized interior-point algorithm*, Annals of Operations Research, 124 (2003), pp. 35–48.
[12] J. Castro, *An interior-point approach for primal block-angular problems*, Computational Optimization and Applications 36 (2007), pp. 195–219.
[13] J. Castro, *Recent advances in optimization techniques for statistical tabular data protection*, European Journal of Operational Research 216 (2012), pp. 257-269.
[14] J. Castro and J. Cuesta, *Quadratic regularizations in an interior-point method for primal block-angular problems*, Mathematical Programming 130 (2011), pp. 415–445.
[15] M. Colombo, A. Grothey, J. Hogg, K. Woodsend and J. Gondzio, *A structure-conveying modelling language for mathematical and stochastic programming*, Mathematical Programming Computation, 1 (2009), pp. 223–247.
[16] K. Fountoulakis and J. Gondzio, A second-order method for strongly convex l1-regularization problems, Technical Report ERGO-14-005, School of Mathematics, The University of Edinburgh, 2014.
[17] K. Fountoulakis, J. Gondzio and P. Zhlobich, *Matrix-free interior point method for compressed sensing problems*, Mathematical Programming Computation 6 (2014), 1–31.
[18] R. Fourer, D.M. Gay and D.W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming, Second edition*, Thomson Brooks/Cole, Toronto, Canada, 2003.
[19] A. Frangioni and C. Gentile, *New preconditioners for KKT systems of network flow problems*, SIAM Journal on Optimization 14 (2004), 894–913.
[20] J. Gondzio, *Matrix-free interior point method*, Computational Optimization and Applications 51 (2012), pp. 457–480.
[21] J. Gondzio, *Convergence analysis of an inexact feasible interior point method for convex quadratic programming*, SIAM Journal on Optimization 23 (2013), pp. 1510–1527.
[22] J. Gondzio and R. Sarkissian, *Parallel Interior Point Solver for Structured Linear Programs*, Mathematical Programming 96 (2003) pp. 561–584.
[23] C. Keller, N.I.M. Gould and A.J. Wathen, *Constraint preconditioning for indefinite linear systems*, SIAM Journal on Matrix Analysis and Applications 21 (2000), pp. 1300–1317.
[24] G.H. Golub and C.F. Van Loan, *Matrix Computations, Third edition*, The Johns Hopkins University Press, Baltimore, MA, 1996.
[25] X. Jiménez, *A modelling and optimization environment for large-scale block-angular problems* (in Catalan), MSc thesis, Barcelona School of Informatics, Universitat Politècnica de Catalunya, 2012.
[26] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, Journal of Research of the National Bureau of Standards 45 (1950), pp.225–280.
[27] S. Mehrotra, *On the implementation of a primal–dual interior point method*, SIAM Journal on Optimization 2 (1992), pp. 575–601.
[28] G. Meurant, *The Lanczos and Conjugate Gradient Algorithms: from Theory to Finite Precision Computations*, SIAM, Philadelphia, PA, 1006.
[29] P. Munari and J. Gondzio, *Using the primal-dual interior point algorithm within the branch-price-and-cut method*, Computers & Operations Research 40 (2013), pp. 2026–2036.
[30] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, Kluwer, Boston, MA, 2004.
[31] E. Ng and B.W. Peyton, *Block sparse Cholesky algorithms on advanced uniprocessor computers*, SIAM Journal on Scientific Computing 14 (1993), pp. 1034–1056.
[32] A.R.L. Oliveira and D.C. Sorensen, *A new class of preconditioners for large-scale linear systems from interior point methods for linear programming*, Linear Algebra and its Applications 394 (2005), pp. 1–24.
[33] J.M. Ortega, *Introduction to Parallel and Vector Solutions of Linear Systems*, Plenum Press, New York, NY, 1988.
[34] M.G.C. 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 (1993), pp. 516–537.
[35] C. Roos, T. Terláky and J.-P. Vial, *Interior Point methods for linear optimization, 2nd Ed.*, Springer, Boston, MA, 2006.
[36] R.J. Vanderbei, *Linear Programming: Foundations and Extensions*, Kluwer, Boston, MA, 1996.
[37] S.J. Wright, *Primal-Dual Interior-Point Methods*, SIAM, Philadelphia, PA, 1996.