An interior-point approach for primal block-angular problems

Jordi Castro
Dept. of Statistics and Operations Research
Universitat Politècnica de Catalunya
Jordi Girona 1–3, 08034 Barcelona (Catalonia, Spain)
jordi.castro@upc.edu

Research Report DR 2005-20
September 2005

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

An interior-point approach for primal block-angular problems

Jordi Castro*
Dept. of Statistics and Operations Research
Universitat Politècnica de Catalunya
Jordi Girona 1–3, 08034 Barcelona, Catalonia, Spain
jordi.castro@upc.edu
http://www-eio.upc.es/~jcastro

Abstract

Multicommodity flows belong to the class of primal block-angular problems. An efficient interior-point method was developed in the past for this type of network optimization problems. It solved the normal equations, using sparse Cholesky factorizations for the diagonal blocks, and a preconditioned conjugate gradient for the linking constraints. In this work we extend this procedure, showing that the preconditioner initially developed for multicommodity flows applies to any primal block-angular problem, although its efficiency depends of the particular linking constraints structure. We discuss under which conditions the preconditioner is effective. The procedure is implemented in a user-friendly package under the MATLAB environment, reporting computational results for four primal block-angular problems: multicommodity flows, nonoriented multicommodity flows, minimum-distance controlled tabular adjustment for statistical data protection, and the minimum congestion problem. The results show that this procedure is very promising for the efficient solution of large primal-block angular problems.

Key words: interior-point methods, structured problems, normal equations, preconditioned conjugate gradient, large-scale optimization

1 Introduction

Multicommodity flows are challenging linear programming problems and some sets of instances, as the PDS ones, have been widely used in the past for the evaluation of general solvers [5, 6]. Interior-point methods were not considered an efficient choice for this kind of problems until the specialized method of [7]. This

*Work supported by the Spanish MCyT project TIC2003-00997.

approach was recognized as the most efficient interior-point method for general multicommodity problems [4]. Although some of the formerly considered difficult multicommodity models are today solved in seconds with extremely efficient simplex implementations [4, 16], specialized interior-point algorithms still provide the best performance for some recent multicommodity instances [8, 9].

The purpose of this work is the extension of the specialized multicommodity interior-point method of [7] to general primal block-angular problems. As in the multicommodity case, the normal equations will be solved by a scheme that combines Cholesky factorizations for the diagonal blocks, and a preconditioned conjugate gradient (PCG) for the linking constraints. It will be shown that the preconditioner initially developed for multicommodity flows can be applied to any primal block-angular problem. However, unlike multicommodity flows, a diagonal preconditioner can not be guaranteed for general block-angular problems; it depends of the particular linking constraints structure. In general this will not be a significant drawback, since, for most problems, linking constraints are sparse, resulting in sparse and efficient preconditioners.

Specialized interior-point methods have been applied in the past for block-angular and more general block-bordered structures. The most significant attempt is the OOPS system [13, 15]. This approach, tailored for parallel processing, exploits the nested structure of the matrix constraints through a tree representation, and solves the linear systems of equations at nodes of the tree by Cholesky factorizations. It is worth to note that the procedure here described could be used at those nodes associated to primal block-angular structures.

Although the procedure here described makes uses of PCG, it is significantly different from other interior-point algorithms based on iterative solvers. Iterative approaches solve the full set of rows and columns of either the normal equations (see [12, 23] and references therein) or augmented system (see [2, 19] and references therein) through PCG, whereas our method only applies PCG for the rows and columns of normal equations associated to linking constraints. Our approach aims at eliminating the complicating constraints from the normal equations. In this sense, it can viewed as a decomposition approach.

The augmented system offers more freedom for direct and iterative solvers [1], in particular for the solution of general quadratic and nonlinear problems and the design of preconditioners. This was the choice, for instance, in the iterative approaches of [2] and [19]. In particular, the latter states that iterative solvers should be used for the augmented system. However, our method is, first, not merely based on PCG, but also uses Cholesky factorizations; and, second, its purpose is to eliminate the complicating linking constraints, making the problem block separable, not to solve the full system by an iterative solver. For instance, the approach of [19] solved PDS60 in 89000 seconds, whereas ours required 6700 [7] in a machine twice slower (the ratio is thus close to 30). Therefore, although our procedure uses the normal equations, for block-angular problems seems to be a more efficient alternative.

An additional argument for using the normal equations in our approach is that, for linear and separable quadratic problems, which are assumed in this work, they are more efficient than the augmented system formulation, if solved

through direct methods in the absence of dense columns [24, Ch. 11]. As far as we know, normal equations are still used in state-of-the-art commercial codes, as CPLEX. Our method uses Cholesky factorizations for the diagonal blocks; if they don't contain dense columns, normal equations are a sensible choice. If some dense column appears in the linking constraints submatrix, variable splitting techniques can be used to reduce the fill-in of the preconditioner. An example of such procedure will be shown in Subsection 6.3 for the minimum congestion problem.

The structure of the paper is as follows. Section 2 presents the formulation of the primal block-angular problem. Section 3 shows the structure of the normal equations for this problem. Section 4 describes the specialized procedure for the solution of normal equations, combining sparse Cholesky factorizations and PCG. This section also discusses under which conditions the preconditioner will be effective. Section 5 gives details of the implementation developed under the MATLAB environment. Finally, Section 6 compares the specialized approach with the standard one based on Cholesky factorizations of normal equations, using four classes of instances: multicommodity flows, nonoriented multicommodity flows, controlled tabular adjustment in statistical data protection, and the minimum congestion problem.

2 The primal block-angular problem

The primal block-angular formulation considered in this work is

$$\begin{aligned} \min \quad & \sum_{i=1}^{k+1} ({c^i}^T x^i + {x^i}^T Q_i x^i) \\ \text{subject to} \quad & \begin{bmatrix} N_1 & & & & \\ & N_2 & & & \\ & & \ddots & & \\ & & & N_k & \\ L_1 & L_2 & \dots & L_k & I \end{bmatrix} \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^k \\ x^{k+1} \end{bmatrix} = \begin{bmatrix} b^1 \\ b^2 \\ \vdots \\ b^k \\ b^{k+1} \end{bmatrix} \\ & 0 \le x^i \le u^i \quad i = 1, \dots, k+1. \end{aligned} \tag{1}$$

Matrices $N_i \in \mathbb{R}^{m_i \times n_i}$ and $L_i \in \mathbb{R}^{l \times n_i}$, $i = 1, \dots, k$ define respectively the block and linking constraints, k being the number of blocks. Vectors $x^i \in \mathbb{R}^{n_i}, i = 1, \dots, k$, are the variables for each block. $x^{k+1} \in \mathbb{R}^l$ 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 $b^{k+1} \in \mathbb{R}^l$ is for the linking constraints. Upper bounds for each group of variables are defined by $u^i, i = 1, \dots, k+1$. Note that this formulation considers the general form of linking constraints $b^{k+1} - u^{k+1} \le \sum_{i=1}^{k} L_i x^i \le b^{k+1}$. $c^i \in \mathbb{R}^{n_i}$ and $Q_i \in \mathbb{R}^{n_i \times n_i}$, $i = 1, \dots, k$ are the linear and quadratic costs for each group of variables. We also consider linear and quadratic costs $c^{k+1} \in \mathbb{R}^l$ and $Q \in \mathbb{R}^{l \times l}$ for the slacks. Since the

procedure to be developed uses the normal equations, we restrict for efficiency to the separable case where Q_i $i = 1, \ldots, k+1$ are semidefinite positive diagonal matrices. Note that any quadratic problem can be transformed to a separable equivalent one, through the addition of extra variables and constraints. This can significantly reduce the solution time for some instances [22, Ch.23]. (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. We assume that, for some i, m_i is allowed to be 0, i.e., problem (1) includes more general situations where some group of variables only appears in the linking constraints. For instance, if such group of variables corresponds to block k, the matrix structure is

$$\begin{bmatrix} N_1 & & & & & \\ & N_2 & & & & \\ & & \ddots & & & \\ & & & N_{k-1} & & \\ L_1 & L_2 & \ldots & L_{k-1} & L_k & I \end{bmatrix}. \tag{2}$$

This matches the formulation considered, for instance, in [15].

3 Structure of normal equations

Problem (1) can be written in standard form as

$$\begin{array}{ll} \min & c^T x + \frac{1}{2} x^T Q x \\ & Ax = b \\ & x + s = u \\ & x, s \geq 0 \end{array} \tag{3}$$

where $x, s, u \in \mathbb{R}^n$, $Q \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^m$. The dual of (3) is

$$\begin{array}{ll} \max & b^T y - \frac{1}{2} x^T Q x - w^T u \\ & A^T y - Qx + z - w = c \\ & z, w \geq 0 \end{array} \tag{4}$$

where $y \in \mathbb{R}^m$ and $z, w \in \mathbb{R}^n$. For problem (1), vectors c, x, s, u, y, z, w and matrix Q are made of $k+1$ blocks.

Replacing inequalities in (3) by a logarithmic barrier with parameter μ, the first order optimality conditions for the barrier problem, after some manipulations and elimination of constraints $x + s = u$, are

$$\begin{array}{ll} r_{xz} & \equiv \mu e - XZe = 0 \\ r_{sw} & \equiv \mu e - SWe = 0 \\ r_b & \equiv b - Ax = 0 \\ r_c & \equiv c - (A^T y - Qx + z - w) = 0 \\ & (x, s, z, w) \geq 0 \; ; \end{array} \tag{5}$$

$e \in \mathbb{R}^n$ is a vector of 1's, and matrices $X, Z, S, W \in \mathbb{R}^{n \times n}$ are diagonal matrices made from 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 superlinearly converge to those of (3) and (4) The nonlinear system (5) is usually solved by a damped version of Newton's method, reducing the μ parameter at each iteration. This procedure is known as the path-following interior-point algorithm. An excellent discussion about the theoretical properties of this and other interior-point algorithms can be found in [24].

The linearization of (5), implicitly assuming that $s = u - x$, gives rises to the following system of equations

$$\begin{bmatrix} A & & & \\ -Q & A^T & I & -I \\ Z & & X & \\ -W & & & S \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta w \end{bmatrix} = \begin{bmatrix} r_b \\ r_c \\ r_{xz} \\ r_{sw} \end{bmatrix}, \tag{6}$$

right-hand-sides having been defined in (5). After the elimination of Δw and Δz from last two groups of equations of (6), as follows

$$\Delta z = X^{-1} r_{xz} - X^{-1} Z \Delta x \tag{7}$$
$$\Delta w = S^{-1} r_{sw} - S^{-1} W \Delta x, \tag{8}$$

we obtain the augmented system form

$$\begin{bmatrix} A & \\ -\Theta^{-1} & A^T \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_b \\ r \end{bmatrix}, \tag{9}$$

where Θ and r are defined as

$$\Theta = (Q + S^{-1} W + X^{-1} Z)^{-1} \qquad r = r_c + S^{-1} r_{sw} - X^{-1} r_{xz}. \tag{10}$$

Additional elimination of Δx from last group of equations of (9) provides the normal equations form:

$$(A \Theta A^T) \Delta y = r_b + A \Theta r \tag{11}$$
$$\Delta x = \Theta (A^T \Delta y - r). \tag{12}$$

The Newton direction is computed using (7), (8), (11) and (12).

For linear and separable quadratic problems, Θ is a diagonal matrix, thus (12) is easily computed. Exploiting the structure of A and Θ of the primal block-angular problem (1), the matrix of system (11) can be recast as

$$A \Theta A^T = \left[\begin{array}{ccc|c} N_1 \Theta_1 N_1^T & & & N_1 \Theta_1 L_1^T \\ & \ddots & & \vdots \\ & & N_k \Theta_k N_k^T & N_k \Theta_k L_k^T \\ \hline L_1 \Theta_1 N_1^T & \dots & L_k \Theta_k N_k^T & \Theta_{k+1} + \sum_{i=1}^{k} L_i \Theta_i L_i^T \end{array}\right] = \begin{bmatrix} B & C \\ C^T & D \end{bmatrix}, \tag{13}$$

$B \in \mathbb{R}^{\tilde{n}\times\tilde{n}}$ ($\tilde{n} = \sum_{i=1}^{k} n_i$), $C \in \mathbb{R}^{\tilde{n}\times l}$ and $D \in \mathbb{R}^{l\times l}$ being the blocks of $A\Theta A^T$ and Θ_i, $i = 1, \ldots, k+1$, the submatrices of Θ associated to the $k+1$ groups of variables in (1), i.e., $\Theta_i = (Q_i + S_i^{-1}W_i + X_i^{-1}Z_i)^{-1}$. Denoting by g the right-hand-side of (11), and appropriately partitioning g and Δy, the normal equations can be written as

$$\begin{bmatrix} B & C \\ C^T & D \end{bmatrix} \begin{bmatrix} \Delta y_1 \\ \Delta y_2 \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix}. \tag{14}$$

4 Solving the normal equations

Eliminating Δy_1 from the first group of equations of (14) we get

$$\begin{aligned} (D - C^T B^{-1} C)\Delta y_2 &= (g_2 - C^T B^{-1} g_1) && (15) \\ B\Delta y_1 &= (g_1 - C\Delta y_2). && (16) \end{aligned}$$

System (16) is solved by performing one Cholesky factorization for each diagonal block $N_i\Theta_i N_i^T, i = 1 \ldots k$, of B. System with matrix $D - C^T B^{-1} C$, the Schur complement of (14), is solved by a PCG. The dimension of this system is l, the number of linking constraints. The preconditioner obtained in [7] for multicommodity flows, a particular class of primal block-angular problems, can be applied to any primal block-angular problems, as shown by next result. Its proof can be obtained from Theorem 1 and Propositions 3 and 4 of [7].

Proposition 1 *If D is positive semidefinite, and $D + C^T B^{-1} C$ is positive semidefinite then the inverse of $(D - C^T B^{-1} C)$ can be computed as*

$$(D - C^T B^{-1} C)^{-1} = \left(\sum_{i=0}^{\infty} (D^{-1}(C^T B^{-1} C))^i \right) D^{-1}. \tag{17}$$

The hypotheses of Proposition 1 are satisfied by any primal block-angular problem.

The preconditioner M^{-1}, an approximation of $(D - C^T B^{-1} C)^{-1}$, is thus obtained by truncating the infinite power series (17) at some term h. Since the preconditioner is used at each iteration of PCG for the solution of system $Mz = r$ (for some vectors z and r), increasing h by one means solving an additional system with matrix B at each PCG iteration. Therefore, as a rule of thumb, although it is problem dependent, $h = 0$ or $h = 1$ are reasonable choices in practice, obtaining

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

$h = 0$ has been used for all the computational results of this paper.

Up to now the above preconditioner has only been (successfully) applied to linear and quadratic multicommodity problems. Although the expected performance for a general primal block-angular matrix is problem dependent, the effectiveness of the preconditioner is governed by next criteria:

- **The spectral radius of** $D^{-1}(C^T B^{-1} C))$, which is always in $[0, 1)$ (Theorem 1 of [7]). The farthest from 1, the closer is M^{-1} to $(D - C^T B^{-1} C)^{-1}$. Although the particular behaviour of the spectral radius value is problem dependent, in general it becomes closer to 1 as we approach the optimal solution, because of the ill-conditioning of the Θ matrix.

- **The structure of matrix** D. At each PCG iteration, $h + 1$ systems with matrix D must be solved. For multicommodity flows this is an inexpensive step, since $L_i = I$, $i = 1, \ldots, k$, and then D is diagonal. For a general problem, D depends of the structure of side constraints, and we are forced to use a sparse Cholesky factorization (including row and column permutation, and symbolic factorization stages). If the fill-in of D is large the preconditioner can be computationally expensive. Procedures devised to avoid fill-in for $A\Theta A^T$ [1] can also be applied for D to improve the efficiency of the preconditioner. An example of linking constraints with a dense column is presented in Subsection 6.3 for the minimum congestion problem; in that case a dense D matrix is avoided by variable splitting.

- **The structure of matrices** $N_i \Theta_i N_i^T$**,** $i = 1, \ldots, k$ **of** B. Denoting by t the number of PCG iterations, the solution of (15) and (16) requires $2 + t(1 + h)$ systems with matrix B. Although the numerical factorization is performed only once, the large number of backsolve steps can be very expensive if the fill-in is significant. Again, this is problem dependent, and general procedures devised for $A\Theta A^T$ can be applied to $N_i \Theta_i N_i^T$.

- **Products** Cv **and** $C^T v$**, for some vector** v. Denoting by t the number of PCG iterations, the number of such products is $2 + t(1 + h)$. From (13), these operations involve matrix-vector products with N_i, N_i^T, L_i and L_i^T, $i = 1, \ldots, k$. In general, they can be highly tuned for each particular problem, exploiting the matrix structure. This is done, for instance, for multicommodity flows, where N_i are node-arc incidence matrices, and $L_i = I$. The use of generic programming and virtual functions, common tools in object-oriented programming languages as C++, can be very effective for the implementation of these operations. Therefore, the execution time spent in these computations should not be significant, compared to the time needed for systems with B and D. However, this rule does not hold in our MATLAB implementation, where the Cholesky factorizations for B and D are performed through precompiled routines, and operations Cv and $C^T v$ are done within the MATLAB interpreted language. As discussed below, this is the reason we only use the time spent in precompiled Cholesky routines for the computational results.

5 Implementation details

The specialized interior-point algorithm described in previous sections has been implemented under the MATLAB environment. The code has been designed as

a generic solver, named PRBLOCK_IP, which can be hooked to a front-end for each particular primal block-angular problem to be solved. The generic solver receives from the front-end:

- $c \in \mathbb{R}^n$: the linear costs vector.
- $Q \in \mathbb{R}^n$: the quadratic costs vector.
- $u \in \mathbb{R}^n$: the upper bounds vector.
- $b \in \mathbb{R}^m$: the right-hand-side vector.
- N: using the overloading capabilites of MATLAB, N can be a list of matrices $N_i \in \mathbb{R}^{m_i \times n_i}$ $i = 1, \ldots, k$, or a single matrix; in the latter case $N_i = N$ for all i, and a single row ordering and symbolic factorization is required.
- L: as before, L is an overloaded parameter, which can be a list of matrices $L_i \in \mathbb{R}^{l \times n_i}$ $i = 1, \ldots, k$, or a single matrix (and thus $L_i = L$ for all i).

PRBLOCK_IP implements a standard path-following algorithm, which solves the normal equations either through a Cholesky factorization, or through the specialized procedure. For efficiency reasons, Cholesky factorizations are performed through external precompiled routines. In particular, the code uses the Ng-Peyton sparse Cholesky package [18], hooked to MATLAB for the LIPSOL package [25]. Ng-Peyton Cholesky package implements the minimum degree ordering heuristic for the row and columns permutations of normal equations, and exploits the memory hierarchy through the use of supernodes. PRBLOCK_IP is about 1800 lines, aside from the precompiled Ng-Peyton Cholesky package and front-ends for each particular problem. It can be obtained for research purposes from `http://www-eio.upc.es/~jcastro/prblock_ip.html`. The distribution includes also a SCILAB version, a free MATLAB-like environment.

Although a MATLAB implementation is far less efficient that an equivalent C/C++ one, it provides a user-friendly environment for testing the suitability of the specialized algorithm for any primal block-angular problem. Front-ends are easily developed within MATLAB. Looking at the CPU time of precompiled Cholesky routines, which is automatically provided by PRBLOCK_IP, we can forecast the performance of the specialized block-angular interior-point algorithm compared to a standard one (see Section 6 for details). If the results are satisfactory, it is worth to spend time on an ad-hoc implementation for this kind of problems. Up to now, such an ad-hoc code is only available for multicommodity flows [7]. The development of a generic C++ class, with virtual functions that exploit the structure of any block-angular problem is out of the scope of this work and among the further tasks to be done.

Some additional features of the package are:

- The path-following algorithm implemented does not compute Mehrotra's predictor-corrector or higher-order directions. As observed in [7], for the particular case of multicommodity flows, the reduction in number of iterations is not worthwhile, due to the increase of execution time per iteration

for applying twice the PCG. However, in the computational results we also compare the specialized procedure with the native MATLAB interior-point solver, which is based on LIPSOL and makes use of Mehrotra's direction. As it will be shown in Section 6, the specialized procedure is competitive with Cholesky based procedures, whether they rely on Newton or Mehrotra directions.

- The angle criteria of [20] is used as stopping rule for the PCG. At iteration i of the interior-point method, we consider that the jth PCG iterate Δy_2^j solves (15) if the angle of $(D - C^T B^{-1} C)\Delta y_2^j$ and $g_2 - C^T B^{-1} g_1$ is close to 1. This rule is implemented as:

$$1 - \cos\Big((D - C^T B^{-1} C)\Delta y_2^j, g_2 - C^T B^{-1} g_1\Big) < \epsilon_i, \tag{18}$$

  ϵ_i being the PCG tolerance parameter. This tolerance is dynamically updated as

$$\epsilon_i = 0.95\epsilon_{i-1}, \tag{19}$$

  which guarantees better Δy_2 directions as we get closer to the solution. By default PRBLOCK_IP uses an initial tolerance of $\epsilon_0 = 10^{-2}$ and $\epsilon_0 = 10^{-3}$ for respectively linear and quadratic problems.

- The initial dimension of system (15) is l, the number of linking constraints. Early detection of inactive linking constraints, initially suggested in [14], may significantly reduce the number of PCG iterations, and was very effective in practice for multicommodity flows [7]. The inactivation procedure of PRBLOCK_IP is only performed when we are close to the optimal solution, i.e., when $|p-d|/(1+|p|) < 1$, p and d being respectively the primal and dual objective functions. The linking constraint i is considered inactive if (i) its slack x_i^{k+1} is far from bounds; (ii) its slack x_i^{k+1} does not intervene in the objective function; (iii) the upper bound of the slack u_i^{k+1} is far enough from 0 (to avoid the removal of active constraints); (iv) and its Lagrange multiplier y_i^{k+1} is close to 0. This is implemented as:

$$\begin{array}{ll} \text{(i)} & 9/10u_i^{k+1} > x_i^{k+1} > 1/10u_i^{k+1} \\ \text{(ii)} & (c_i^{k+1} = 0) \wedge (Q_{k+1_{ii}} = 0) \\ \text{(iii)} & u_i^{k+1} > 1/10 \\ \text{(iv)} & |y_i^{k+1}| < 1/100. \end{array}$$

  Note that, unlike general Cholesky solvers, the removal of mutual capacity constraints does not imply any additional symbolic refactorization with PCG.

- As we approach an optimal point, system (15) becomes more ill-conditioned, and PCG may provide inaccurate solutions. When this happens, PRBLOCK_IP switches to the solution of normal equations (14) by a Cholesky factorization. This is done when we are close enough to the optimal point, and the duality gap increases from one iteration to the next. If we define the gap at iteration i as $gap^i = |p^i - d^i|/(1+|p^i|)$, p^i and d^i being respectively the

Table 1: Percentage of overall execution time spent in computation of Δy (including Cholesky and PCG), and in Cholesky-related procedures

Instance	k	m_i	n_i	% Δy	% Cholesky
PDS10	11	1398	4792	84.3	74.3
PDS20	11	2856	10858	89.4	79.6
Mnetgen 64-64	64	63	511	65.3	41.3
Mnetgen 128-64	64	127	1171	83.2	59.6

primal and dual objective functions, the code switches to the Cholesky factorization of normal equations if

$$(gap^i < 1/2) \text{ and } (gap^i > 1.05 gap^{i-1}). \tag{20}$$

6 Computational evaluation

We tested the specialized algorithm with four classes of primal block-angular problems: multicommodity problems, nonoriented multicommodity problems, minimum-distance controlled tabular adjustment, and the minimum congestion problem. It is worth to note that for some of the above problems there are more efficient algorithms than our specialized approach. Our purpose is to compare the specialized algorithm with a general interior-point one. Minimum-distance controlled tabular adjustment instances are quadratic problems; the remaining instances are linear ones. They are presented in the following subsections. For each instance we provide results with PRBLOCK_IP solving the normal equations (14) with both the specialized procedure and through a Cholesky factorization. For the linear instances we also provide results with LINPROG, the linear optimization solver of MATLAB, which is based on LIPSOL [25] and computes Mehrotra's predictor-corrector directions through Cholesky factorizations. All runs were carried out on a HP-LC2000 server with Intel Xeon 933 MHz processors and 2 Gb of main memory.

Being an interpreted language, the overall execution time required by MATLAB is meaningless. We only consider the execution time spent in the external precompiled Ng-Peyton Cholesky routines (including minimum degree ordering, symbolic factorization, numerical factorization, and numerical solution). For the runs with LINPROG we provide the overall execution time, since it uses internal Cholesky MATLAB routines. We observed that, when solving the normal equations by a Cholesky factorization, the relative difference between the time spent in Cholesky factorizations and the overall execution time is less than a 1% for large instances. However, when solving (14) by the specialized procedure, the execution time was many times (e.g., 60 times for large instances) the time spent in all Cholesky factorizations (including those required for the PCG). This behaviour is due to the interpreted nature of the MATLAB language. In a C/C++ implementation, the time spent in the solution of (15) by PCG (excluding Cholesky factorizations) is a small fraction of the overall time, and much less than the time required by Cholesky factorizations. For instance,

Table 2: Dimensions of multicommodity instances

				Oriented		Nonoriented	
Instance	k	m'	n'	m	n	m	n
PDS1	11	125	372	1450	4167	1462	8271
PDS5	11	685	2325	7938	25978	8088	51703
PDS10	11	1398	4792	16192	53526	16547	106593
PDS15	11	2124	7756	24634	86586	25176	172444
M32-32	32	31	486	1353	15913	1353	31465
M64-64	64	63	511	4403	33075	4419	65795
M128-64	64	127	1171	8988	75804	9024	150784
M128-128	128	127	1204	17188	155044	17215	309183

Table 1 reports results obtained with the C implementation of the specialized procedure for multicommodity flows of [7]. For four of the instances of next subsections, the table provides the dimensions (number of blocks k, and number of rows m_i and columns n_i of all blocks), the percentage of overall execution time spent in the computation of Δy (which includes both PCG and Cholesky), and the percentage of overall execution time spent only in Cholesky-related procedures. These two figures would get closer for larger instances. They would also approach 100%. Therefore, the time spent in Cholesky procedures can be used to forecast the overall execution time of the specialized algorithm. From Table 1 we see that, even for the smallest instances, the overall execution time will be, as much, about twice the Cholesky time.

6.1 Oriented and nonoriented multicommodity problems

Oriented multicommodity flow problems match the general primal block-angular formulation (1) with $N_i = N$ and $L_i = I$ for all $i = 1, \ldots, k$. $N \in \mathbb{R}^{m' \times n'}$ is a node-arc incidence matrix, of $m' + 1$ nodes (one node is removed to guarantee full row-rank) and n' arcs. Blocks are denoted as commodities in this problem. Vectors b^i, $i = 1, \ldots, k$ are the supply-demands at the nodes for each commodity, and b^{k+1} is the mutual capacity of arcs. l, the number of linking constraints, is n'. The code of [7] is a particular and efficient C implementation of PRBLOCK_IP for multicommodity flows.

We considered a subset of the PDS [6] and Mnetgen [17] instances. These are standard multicommodity problems, widely used in the literature. They can be retrieved from `http://www.di.unipi.it/di/groups/optimize/Data/MMCF.html`. Table 2 shows the dimensions of these instances: number of blocks (k), constraints and variables for each block (m' and n'), and overall number of constraints and variables of the linear problem (m and n, oriented columns). A preprocessing removed inactive linking constraints; therefore, $l < n'$, and $m < km' + n'$ and $n < (k+1)n'$ in Table 2.

Nonoriented multicommodity problems allow flow in both directions on each arc. They are also primal block-angular problems with $N_i = [N \quad -N]$ and

Table 3: Results for oriented multicommodity instances

	PRBLOCK_IP				LINPROG	
	PCG+Chol		Chol			
Instance	Iter	CPU	Iter	CPU	Iter	CPU
PDS1	34	2.5	34	0.6	23	4.4
PDS5	51	22	48	64	40	264
PDS10	68	75	62	790	46	2021
PDS15	76	188	81	4624	54	6365
M32-32	44	7.6	29	44	21	184
M64-64	54	22	39	1744	27	2762
M128-64	58	55	47	31397	31	38869
M128-128	74	111	60	247044	36	188640

Table 4: Results for nonoriented multicommodity instances

	PRBLOCK_IP				LINPROG	
	PCG+Chol		Chol			
Instance	Iter	CPU	Iter	CPU	Iter	CPU
PDS1	31	1.2	26	0.5	24	9
PDS5	47	17	39	66	43	309
PDS10	56	92	47	651	60	2852
PDS15	83	1034	53	2813	61	6994
M32-32	64	39	31	49	23	189
M64-64	62	33	40	1914	27	1643
M128-64	71	105	49	31294	32	35704
M128-128	74	130	61	227612	39	168633

$L_i = [I \quad -I]$ for all $i = 1, \ldots, k$. As before, $N \in \mathbb{R}^{m' \times n'}$ is a node-arc incidence matrix, of $m'+1$ nodes and n' arcs. A description of other specialized algorithms for this problem can be found in [11]. Since, as far as we know, there is no standard set of nonoriented multicommodity problems, we generated them from the previous multicommodity instances. The overall number of constraints and variables is reported in last two columns of Table 2.

Tables 3 and 4 show the results obtained with PRBLOCK_IP and LINPROG (a.k.a. LIPSOL) for respectively oriented and nonoriented multicommodity problems. The columns show the number of interior-point iterations ("Iter") and CPU time ("CPU") for PRBLOCK_IP with the specialized procedure ("PCG+Chol") and with the standard one based on the Cholesky factorization of normal equations ("Chol"), and for LINPROG. For LINPROG we provide the overall execution time; for options "PCG+Chol" and "Chol" we provide the CPU time spent in Cholesky routines. Although in a efficient C/C++ implementation the overall execution time of option "PCG+Chol" might be

about twice the CPU time reported in these tables (mainly for the Mnetgen instances, as discussed above), the specialized procedure is much more efficient than the standard one. Note that either for oriented or nonoriented multicommodity problems, the structure of matrix D is diagonal, and thus computations with the preconditioner are inexpensive.

6.2 Minimum-distance controlled tabular adjustment problems

Minimum-distance controlled tabular adjustment (CTA for short) is a recent technique for the protection of statistical tabular data [9, 10]. This is a main concern of National Statistical Institutes, which must guarantee that individual information can not be disclosed from released data. Tabular data is obtained by crossing two or more variables of a file of microdata, e.g., city, age, and profession. The Cartesian product of values for these variables provide a set of cells. For each cell the table reports the number of individuals (frequency tables), or information about another variable, e.g., average salary (magnitude tables).

Cell values $a = (a_i), i = 1, \ldots, n$, n being the number of cells, must verify some linear relations $Aa = b$. For instance, for a three-dimensional table of $r + 1$, $c + 1$ and $k + 1$ categories for, respectively, the first, second and third variable (last category corresponds to marginal values), the linear relations are

$$\sum_{i_1=1}^{r} a_{i_1 i_2 i_3} = a_{(r+1) i_2 i_3} \qquad i_2 = 1 \ldots c, \quad i_3 = 1 \ldots k \tag{21}$$

$$\sum_{i_2=1}^{c} a_{i_1 i_2 i_3} = a_{i_1 (c+1) i_3} \qquad i_1 = 1 \ldots r, \quad i_3 = 1 \ldots k \tag{22}$$

$$\sum_{i_3=1}^{k} a_{i_1 i_2 i_3} = a_{i_1 i_2 (k+1)} \qquad i_1 = 1 \ldots r, \quad i_2 = 1 \ldots c. \tag{23}$$

Given a subset of cells $\mathcal{P} \subseteq \{1, \ldots, n\}$ to be protected, and lower and upper protection levels lpl_i and upl_i for $i \in \mathcal{P}$, the purpose of CTA is to find the closest safe values $x = (x_i), i = 1, \ldots, n$, according to some distance L, that makes the released table safe. This involves the solution of the following optimization problem

$$\begin{array}{rl} \min\limits_{x} & ||x - a||_L \\ \text{subject to} & Ax = b \\ & \underline{a}_i \leq x_i \leq \overline{a}_i \quad i = 1, \ldots, n \\ & x_i \leq a_i - lpl_i \text{ or } x_i \geq a_i + upl_i \quad i \in \mathcal{P}, \end{array} \tag{24}$$

$\underline{a}_i$ and $\overline{a}_i$ being lower and upper bounds for each cell $i = 1, \ldots, n$, which are considered known by any data-attacker. In practice L_1 or L_2 are used, respectively obtaining either a linear or a quadratic optimization problem. The results reported in this subsection correspond to L_2.

Exploiting the structure of matrix A, (24) can be formulated as a primal block-angular problem (1). The simplest case correspond to the linear relations (21–23) of a three-dimensional table. Appropriately reordering (21–22)

Table 5: Dimensions of minimum-distance tabular adjustment instances

Instance	k	m'	n'	m	n
CTA-15-15-10	10	29	225	515	2475
CTA-15-15-25	25	29	225	950	5850
CTA-25-25-25	25	49	625	1850	16250
CTA-50-25-25	25	74	1250	3100	32500
CTA-50-50-50	50	99	2500	7450	127500

Table 6: Results for minimum-distance tabular adjustment instances

	PRBLOCK_IP			
	PCG+Chol		Chol	
Instance	Iter	CPU	Iter	CPU
CTA-15-15-10	7	0.2	7	0.2
CTA-15-15-25	7	0.3	7	4
CTA-25-25-25	8	0.5	8	42
CTA-50-25-25	7	0.8	8	77
CTA-50-50-50	7	2.7	7	2595

we obtain k blocks of $m' = c + r$ constraints (indeed one is redundant and can be removed) and $n' = cr$ variables. All blocks have the same structure; thus $N_i = N \in \mathbb{R}^{m' \times n'}$, $i = 1, \ldots, k$, in (1). Block i is related to the cells of category i of third variable. Equations (23) are the $l = cr$ linking constraints, and are defined with $L_i = I,\ i = 1, \ldots, k$, in (1). More details can be found in [9].

We generated five three-dimensional CTA instances with a random generator of synthetic tables. It can be retrieved from `http://www-eio.upc.es/~jcastro/CTA_3Dtables.html`. Table 5 reports the dimensions of each instance, which are denoted as CTA-c-r-k. The meaning of the columns is the same that in Table 2. Table 6 shows the results obtained only with PRBLOCK_IP, since LINPROG can not deal with quadratic problems. The meaning of the columns is the same that in Tables 3 and 4. Clearly, for CTA problems, exploiting the problem structure through our procedure is extremely more efficient than solving the normal equations by Cholesky factorizations. It is worth to note that general interior-point solvers have shown to be more efficient than simplex implementations for both linear and quadratic variants of CTA [10].

6.3 Minimum congestion problems

The minimum congestion problem is equivalent to the maximum concurrent flow one. In the literature both problems are usually seen as one, and denoted as the maximum concurrent flow problem [21, 3]. These problems arise in practical applications of telecommunications networks. The maximum concurrent flow problem is defined on, usually, infeasible nonoriented multicommodity networks of k commodities, $m' + 1$ nodes, and n' arcs, i.e., total flow to be sent from sources to destinations exceeds the arc capacities. The purpose of the problem

is to determine the maximum concurrent flow (or throughput) that can be transported. On the other hand the minimum congestion problem finds the minimum increment in arc capacities that makes the problem feasible, i.e., all multicommodity flows can be sent from origins to destinations. The minimum congestion problem can be formulated as

$$\begin{array}{rll} \min & \max\{y_1,\ldots,y_{n'}\} & \\ \text{subject to} & Nx^{i^+} - Nx^{i^-} = b^i & i = 1,\ldots,k \\ & \sum_{i=1}^{k}(x_j^{i^+} + x_j^{i^-}) \le y_j u_j & j = 1,\ldots,n' \\ & x^{i^+}, x^{i^-} \ge 0 & i = 1,\ldots,k \\ & y_j \ge 0 & j = 1,\ldots,n', \end{array} \tag{25}$$

$N \in \mathbb{R}^{m' \times n'}$ being the network matrix, $x^{i^+} \in \mathbb{R}^{n'}$ and $x^{i^-} \in \mathbb{R}^{n'}$ the nonoriented flows of commodity i for respectively the forward and backward directions, $u \in \mathbb{R}^{n'}$ the arc capacities (no individual commodity capacities are considered), $b^i \in \mathbb{R}^{m'}$ the supply-demands for commodity i, and y_j the fraction of the capacity of arc j that has to be increased.

Considering a new variable $z \in \mathbb{R}$, (25) can be written as the following linear problem

$$\begin{array}{rll} \min & z & \\ \text{subject to} & Nx^{i^+} - Nx^{i^-} = b^i & i = 1,\ldots,k \\ & \sum_{i=1}^{k}(x_j^{i^+} + x_j^{i^-}) - y_j u_j \le 0 & j = 1,\ldots,n' \\ & y_j - z \le 0 & j = 1,\ldots,n' \\ & x^{i^+}, x^{i^-} \ge 0 & i = 1,\ldots,k \\ & y_j \ge 0 & j = 1,\ldots,n'. \end{array} \tag{26}$$

(25) is a primal block-angular problem. Its constraints matrix structure, which matches (29), is

$$\begin{array}{c} \begin{array}{cccccccccccc} x^{1^+} & x^{1^-} & x^{2^+} & x^{2^-} & \ldots & x^{k^+} & x^{k^-} & z & y & & & \end{array} \\ \left[\begin{array}{cccccccccccc} N & -N & & & & & & & & & & \\ & & N & -N & & & & & & & & \\ & & & & \ddots & & & & & & & \\ & & & & & N & -N & & & & & \\ I & I & I & I & \ldots & I & I & & -U & I & & \\ & & & & & & & -e & I & & I & \end{array}\right] \end{array}, \tag{27}$$

$e \in \mathbb{R}^{n'}$ being a vector of 1's. Formulation (26) has a dense column in the second group of linking constraints, due to the e vector in (29). Therefore, matrix D in (14) will show a dense submatrix of $n' \times n'$ nonzero elements. Even for $h = 0$, the preconditioner can be expected to be computationally expensive.

A second more efficient formulation is obtained by considering $z_j, j = 1,\ldots,n'$

Table 7: Dimensions of minimum congestion instances, formulation (26)

Instance	k	$\sum_{i=1}^{k} m_i$	$\sum_{i=1}^{k} n_i$	m	n
M32-32	34	992	31591	1964	32563
M64-64	66	4032	65920	5054	66942
M128-64	66	8128	151060	10470	153402
M128-128	130	16256	309429	18664	311837

Table 8: Dimensions of minimum congestion instances, formulation (28)

Instance	k	$\sum_{i=1}^{k} m_i$	$\sum_{i=1}^{k} n_i$	m	n
M32-32	34	992	32076	2449	33533
M64-64	66	4032	66430	5564	67962
M128-64	66	8128	152230	11640	155742
M128-128	130	16256	310632	19867	314243

for each arc, and imposing $n' - 1$ constraints $z_j = z_{j+1}$. This second model is

$$\begin{array}{rll} \min & z_1 & \\ \text{subject to} & Nx^{i^+} - Nx^{i^-} = b^i & i = 1, \ldots, k \\ & \sum_{i=1}^{k} (x_j^{i^+} + x_j^{i^-}) - y_j u_j \leq 0 & j = 1, \ldots, n' \\ & y_j - z_j \leq 0 & j = 1, \ldots, n' \\ & z_j - z_{j+1} = 0 & j = 1, \ldots, n' - 1 \\ & x^{i^+}, x^{i^-} \geq 0 & i = 1, \ldots, k \\ & y_j \geq 0 & j = 1, \ldots, n'. \end{array} \tag{28}$$

The constraints matrix structure of (28) is

$$\begin{array}{c} \begin{array}{cccccccccccc} x^{1^+} & x^{1^-} & x^{2^+} & x^{2^-} & \ldots & x^{k^+} & x^{k^-} & z & y & & & \end{array} \\ \left[\begin{array}{cccccccccccc} N & -N & & & & & & & & & & \\ & & N & -N & & & & & & & & \\ & & & & \ddots & & & & & & & \\ & & & & & N & -N & & & & & \\ I & I & I & I & \ldots & I & I & & -U & I & & \\ & & & & & & & -I & I & & I & \\ & & & & & & & T & & & & I \end{array}\right] \end{array}, \tag{29}$$

$T \in \mathbb{R}^{(n'-1) \times n'}$ being a banded matrix, with a main diagonal of 1's, and a diagonal above the main diagonal of -1's. Although the dimension of linking constraints increases by $n' - 1$ compared to previous formulation, the fill-in of D is significantly reduced, improving the performance of the overall procedure.

Table 9: Results for minimum congestion instances, formulation (26)

	PRBLOCK_IP				LINPROG	
	PCG+Chol		Chol			
Instance	Iter	CPU	Iter	CPU	Iter	CPU
M32-32	24 (2)	29	22	194	45	279
M64-64	25 (0)	25	21	1682	16	1250
M128-64	27 (8)	8549	23	27847	20	19779
M128-128	31 (11)	48429	24	116115	(1)	—

(1) Not enough memory

Table 10: Results for minimum congestion instances, formulation (28)

	PRBLOCK_IP				LINPROG	
	PCG+Chol		Chol			
Instance	Iter	CPU	Iter	CPU	Iter	CPU
M32-32	27 (2)	7	24	57	32	302
M64-64	28 (5)	67	24	358	23	703
M128-64	30 (7)	3481	25	14043	39	16016
M128-128	34 (12)	10541	26	24804	(1)	—

(1) Problem did not converge

We generated minimum congestion instances from the Mnetgen ones of Subsection 6.1, increasing the supplies and demands by a factor of two. Tables 7 and 8 show the dimensions of these instances, for each formulation. Columns $\sum_{i=1}^{k} m_i$ and $\sum_{i=1}^{k} n_i$ show the number of constraints and variables of the diagonal blocks part. The remaining columns have the same meaning as in previous tables. Tables 9 and 10 provide the computational results. The meaning of the columns is the same as in previous tables. Numbers in brackets for the first column "Iter" mean the number of interior-point iterations with normal equations solved by Cholesky factorizations (i.e., the rule (20) was satisfied). It is clear that the second formulation is much more efficient than the first one. We also see that the CPU time of the specialized algorithm seems to be much higher than for previous instances. However, this is due to the significant number of iterations performed with Cholesky factorizations of normal equations. From columns "Iter" and "CPU" of option "Chol", we can estimate the time per interior-point iteration using a direct solution of normal equations. Using this value and the numbers in brackets, we conclude that the specialized procedure was very efficient for the first interior-point iterations.

7 Conclusions

From the computational results of this work it can be concluded that the specialized interior-point algorithm initially developed for multicommodity flows is a very efficient tool for the solution of general primal block-angular problems.

Although the behaviour of the preconditioner is problem dependent, the specialized algorithm was more efficient than the Cholesky factorization of normal equations for the four classes of instances tested. Several tasks are still to be done. The most significant are: the development of an efficient C/C++ replacement to the current MATLAB implementation; improving the efficiency of PCG by adaptive selection of h, the number of terms in the preconditioner; adaptive selection of either Newton or higher-order directions, according to the quality of the preconditioner at each interior-point iteration; specialization of the procedure for other classes of primal block-angular problems; and the inclusion of the procedure in a more general framework for structured problems through interior-point solvers.

References

[1] E. D. Andersen, J. Gondzio, C. Mészáros and X. Xu, Implementation of interior point methods for large scale linear programming, in T. Terlaky (ed.), *Interior Point Methods in Mathematical Programming*, Kluwer: Dordrecht, pp. 189–252, 1996.

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

[3] 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.

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

[5] R.E. Bixby, M. Fenelon, Z. Gu, E. Rothberg and R. Wunderling, MIP: Theory and practice—Closing the gap, in M.J.D. Powell and S. Scholtes (eds.), *System Modelling and Optimization. Methods, Theory and Applications*, Kluwer: Boston, pp. 19–49, 2000.

[6] W.J. Carolan, J.E. Hill, J.L. Kennington, S. Niemi, and S.J. Wichmann, An empirical evaluation of the KORBX algorithms for military airlift applications, *Operations Research*, 38, pp. 240–248, 1990.

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

[8] J. Castro, Solving difficult multicommodity problems through a specialized interior-point algorithm, *Annals of Operations Research*, 124, pp. 35–48, 2003.

[9] J. Castro, Quadratic interior-point methods in statistical disclosure control, *Computational Management Science*, 2, pp. 107–121, 2005.

[10] J. Castro, Minimum-distance controlled perturbation methods for large-scale tabular data protection, *European Journal of Operational Research*, in press, 2006.

[11] P. Chardaire and A. Lisser, Simplex and interior point specialized algorithms for solving nonoriented multicommodity flow problems, *Operations Research*, 50, pp. 260–276, 2002.

[12] P. Chin, Iterative algorithm for solving linear programming from engineering applications, PhD Thesis, University of Waterloo, 1995.

[13] J. Gondzio and A. Grothey, Exploiting structure in parallel implementation of interior point methods for optimization, School of Mathematics, The University of Edinburgh, Technical Report MS-04-004, 2005.

[14] J. Gondzio and M. Makowski, Solving a class of LP problems with a primal-dual logarithmic barrier method, *European Journal of Operational Research*, 80, pp. 184–192, 1995.

[15] J. Gondzio and R. Sarkissian, Parallel Interior Point Solver for Structured Linear Programs, *Mathematical Programming*, 96, pp. 561–584, 2003.

[16] J. Mamer and R. McBride, A decomposition bases pricing procedure for large scale linear programs: an application to the linear multicommodity flow problem, *Management Science* 46, pp. 693–709, 2000.

[17] A. Ali and J.L. Kennington, Mnetgen program documentation, Technical Report 77003, Department of Industrial Engineering and Operations Research, Southern Methodist University, Dallas, TX, 1977.

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

[19] 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, pp. 1–24, 2005.

[20] 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, pp. 516–537, 1993.

[21] F. Shahrokhi and D.W. Matula, The maximum concurrent flow problem, *Journal of the ACM*, 37, pp. 318–334, 1990.

[22] R.J. Vanderbei, *Linear Programming: Foundations and Extensions*, Kluwer: Boston, 1996.

[23] W. Wang and D.P. O'Leary, Adaptive use of iterative methods in predictor-corrector interior point methods for linear programming, *Numerical Algorithms*, 25, pp. 387–406, 2000.

[24] S.J. Wright, *Primal-Dual Interior-Point Methods*, SIAM: Philadelphia, 1996.

[25] Y. Zhang, Solving large-scale linear programs by interior-point methods under the MATLAB environment, *Optimization Methods and Software*, 10, pp. 1–31, 1998.