Comput Optim Appl (2007) 36: 195–219
DOI 10.1007/s10589-006-9000-1

An interior-point approach for primal block-angular problems

Jordi Castro

Published online: 21 February 2007

Abstract Multicommodity flows belong to the class of primal block-angular problems. An efficient interior-point method has already been developed for linear and quadratic network optimization problems. It solved normal equations, using sparse Cholesky factorizations for diagonal blocks, and a preconditioned conjugate gradient for 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 on each particular linking constraints structure. We discuss the conditions under which the preconditioner is effective. The procedure is implemented in a user-friendly package in the MATLAB environment. Computational results are reported 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 holds great potential for solving large primal-block angular problems efficiently.

Keywords 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, such as PDS, have been widely used in the past for the evaluation of

Work supported by the Spanish MCyT Project TIC2003-00997.

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

general solvers [7, 8]. Interior-point methods were not considered an efficient choice for this kind of problem until the specialized method of [9]. This approach was recognized as being the most efficient interior-point method for general multicommodity problems [6]. Although some of the multicommodity models that were formerly considered to be difficult are today solved in seconds with extremely efficient simplex implementations [6, 19], specialized interior-point algorithms still perform better for some recent multicommodity instances [10, 12].

The purpose of this work is to expand on the specialized multicommodity interior-point method of [9] for general primal block-angular problems. As in the case of multicommodity flows, normal equations will be solved using a method 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, the existence of a good diagonal preconditioner cannot be guaranteed for general block-angular problems: it depends on the particular structure of linking constraints. In general this will not be a significant drawback, because for most problems linking constraints are sparse, which results in sparse, efficient preconditioners.

Specialized interior-point methods have been applied in the past to block-angular and more general block-bordered structures. The most significant attempt is the OOPS system [16, 18]. 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. The procedure described here could be used at the nodes associated with primal block-angular structures.

Although the procedure applied in this work makes use of the 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 both the normal equations (see [15, 25] and references therein) and the augmented system (see [3, 15, 21] and references therein) through the PCG, whereas our method only applies the PCG for the rows and columns in normal equations associated with linking constraints. It aims to eliminate the complicating constraints from normal equations. It can thus be viewed as a decomposition approach.

The augmented system offers more freedom for direct and iterative solvers [2], both solving general quadratic and nonlinear problems and for designing preconditioners. This was the choice made, for instance, in the iterative approaches of [3] and [21]. Specifically, the latter states that iterative solvers should be used for the augmented system. However, our method is, firstly, not merely based on the PCG, but also uses Cholesky factorizations; and, secondly, its purpose is to eliminate the complicating linking constraints, making the problem block separable rather than solving the full system using an iterative solver. For instance, an indirect comparison between the approaches of [21] and [9] for some PDS problems is presented in Table 1. The results of [9] are divided into two since they were obtained from a machine twice as slow. Clearly, however, although our procedure uses normal equations, it seems to be a more efficient alternative for solving block-angular problems.

An additional argument for using normal equations in our approach is that for linear and separable quadratic problems, which are dealt with in this work, they are more

Table 1 Comparison of the approaches of [21] and [9] for some PDS problems

Instance	CPU of [21]	CPU of [9]
PDS10	1316	44
PDS15	4285	118
PDS20	8371	193
PDS50	60215	2083
PDS60	89651	3380

efficient than the augmented system formulation, if solved through direct methods in the absence of dense columns [26, Chap. 11]. As far as we know, normal equations are still used in state-of-the-art commercial codes such as CPLEX. Our method uses Cholesky factorizations for the diagonal blocks; if they do not contain dense columns, normal equations are a sensible choice. Should a dense column appear 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 normal equations for this problem. Section 4 describes the specialized procedure for the solution of normal equations, combining sparse Cholesky factorizations and the PCG. This section also discusses the conditions under which the preconditioner will be effective. Section 5 gives details of the implementation developed in the MATLAB environment. Finally, Sect. 6 compares the specialized approach with the standard one based on Cholesky factorizations of normal equations and with alternative preconditioners, using four sets 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

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

Matrices $N_i \in \mathbb{R}^{m_i \times n_i}$ and $L_i \in \mathbb{R}^{l \times n_i}$, $i = 1, \dots, k$, respectively define 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. $b^i \in \mathbb{R}^{m_i}, i = 1, \ldots, k$ is the right-hand-side vector for each block of constraints, whereas $b^{k+1} \in \mathbb{R}^l$ is for the linking constraints. The upper bounds for each group of variables are defined by $u^i, i = 1, \ldots, k+1$. Note that this formulation considers the general form of linking constraints $b^{k+1} - u^{k+1} \leq \sum_{i=1}^{k} L_i x^i \leq b^{k+1}$. Equality constraints are not considered to simplify the exposition, although the procedure to be developed is equally valid for problems with both types of linking constraints. Equality constraints can be defined by imposing zero (or almost zero) upper bounds on the slacks. $c^i \in \mathbb{R}^{n_i}$ and $Q_i \in \mathbb{R}^{n_i \times n_i}, i = 1, \ldots, k$, 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 $Q_{k+1} \in \mathbb{R}^{l \times l}$ for the slacks. Since the procedure to be developed uses normal equations, for the sake of efficiency it has been restricted to the separable case where $Q_i, i = 1, \ldots, k+1$, are positive semidefinite diagonal matrices, although the underlying method is valid for any positive semidefinite matrix. Note that any quadratic problem can be transformed into a separable equivalent one, through the addition of extra variables and constraints. This can significantly reduce the solution time in some instances [24, Chap. 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 a 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 primal block-angular formulation of, for instance, [18].

Note that more general structures, such as block-bordered structures, can also be attempted with the procedure described in this work if they are previously transformed to a primal block-angular form. This can be done, for instance, by replicating the linking columns for each block and imposing extra linking constraints that impose the same value on the replicated variables. A similar strategy is used in stochastic optimization with first-stage variables and nonanticipativity constraints.

3 Structure of normal equations

Problem (1) can be written in standard form as

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

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

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

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 up 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 manipulating and eliminating the constraints $x + s = u$, are

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

$e \in \mathbb{R}^n$ is a vector of 1's, and matrices $X, Z, S, W \in \mathbb{R}^{n \times n}$ are diagonal matrices made 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 superlinearly to those of (3) and (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 path-following interior-point algorithm. An excellent discussion about the theoretical properties of this and other interior-point algorithms can be found in [26].

The linearization of (5), based on the implicit assumption that $s = u - x$, results in a linear system of variables Δx, Δy, 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

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

where Θ and r are defined as

$$\Theta = (Q + S^{-1} W + X^{-1} Z)^{-1}, \qquad r = r_c + S^{-1} r_{sw} - X^{-1} r_{xz}. \tag{9}$$

If, in addition, we eliminate Δx from the last group of equations of (8), the normal equations form is obtained:

$$(A \Theta A^T) \Delta y = r_b + A\Theta, r, \tag{10}$$

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

The Newton direction is computed using (6), (7), (10) and (11).

For linear and separable quadratic problems, Θ is a diagonal matrix, thus (11) is easily computed. Exploiting the structure of A and Θ of the primal block-angular problem (1), the matrix of system (10) can be recast as

$$A\Theta A^T = \left[\begin{array}{ccc|c} N_1\Theta_1 N_1^T & & & N_1\Theta_1 L_1^T \\ & \ddots & & \vdots \\ & & N_k\Theta_k N_k^T & N_k\Theta_k L_k^T \\ \hline L_1\Theta_1 N_1^T & \dots & L_k\Theta_k N_k^T & \Theta_{k+1} + \sum_{i=1}^{k} L_i\Theta_i L_i^T \end{array}\right] = \left[\begin{array}{cc} B & C \\ C^T & D \end{array}\right], \tag{12}$$

$B \in \mathbb{R}^{\tilde{n}\times\tilde{n}}$ ($\tilde{n} = \sum_{i=1}^{k} n_i$), $C \in \mathbb{R}^{\tilde{n}\times l}$ and $D \in \mathbb{R}^{l\times l}$ being the blocks of $A\Theta A^T$, and Θ_i, $i = 1, \dots, k+1$, the submatrices of Θ associated with 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 (10), and appropriately partitioning g and Δy, the normal equations can be written as

$$\left[\begin{array}{cc} B & C \\ C^T & D \end{array}\right]\left[\begin{array}{c} \Delta y_1 \\ \Delta y_2 \end{array}\right] = \left[\begin{array}{c} g_1 \\ g_2 \end{array}\right]. \tag{13}$$

4 Solving the normal equations

By eliminating Δy_1 from the first group of equations of (13), we obtain

$$(D - C^T B^{-1} C)\Delta y_2 = (g_2 - C^T B^{-1} g_1) \tag{14}$$

$$B\Delta y_1 = (g_1 - C\Delta y_2). \tag{15}$$

System (15) is solved by performing a Cholesky factorization for each diagonal block $N_i\Theta_i N_i^T$, $i = 1, \dots, k$, of B. The system with matrix $D - C^T B^{-1} C$, the Schur complement of (13), is solved by a PCG. The dimension of this system is l, which is the number of linking constraints. The preconditioner obtained in [9] for multicommodity flows, a special class of primal block-angular problems, can be applied to any primal block-angular problem, as shown by following result. Proof of this is upheld by Theorem 1, and Propositions 3 and 4 of [9].

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

$$(D - C^T B^{-1} C)^{-1} = \left(\sum_{i=0}^{\infty} (D^{-1}(C^T B^{-1} C))^i\right) D^{-1}. \tag{16}$$

The hypotheses of Proposition 1 are satisfied by any primal block-angular problem. Note that the above preconditioner is also valid if $Q_i, i = 1, \ldots, k$ are not diagonal, although in practice it would be prohibitive because Θ_i is no longer diagonal in systems with matrix B.

The preconditioner M^{-1}, an approximation of $(D - C^T B^{-1} C)^{-1}$, is thus obtained by truncating the infinite power series (16) at some term h. Since the preconditioner is used at each iteration of the 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. The more the terms included, the better the preconditioner will be, at the expense of increasing the execution time for $Mz = r$. The value of h that optimizes the tradeoff between a good and an efficient preconditioner is problem dependent. However, in general, few terms from the power series should be considered. $h = 0$ or $h = 1$ are reasonable choices, which in practice yield

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

$h = 0$ has been used for all the computational results in this paper. As will be shown in Sects. 6.1, 6.2 and 6.3, which report the overall number of PCG iterations for each instance, the PCG converges in few iterations for $h = 0$. However, the number of PCG iterations is not evenly distributed for all the interior-point iterations: very few PCG iterations are required for the first interior-point iterations, and the number usually increases only when the relative duality gap (defined as the relative difference between the primal and dual objective functions) is less than 10^{-1}.

Up to now, the above preconditioner has only been (successfully) applied to linear and quadratic multicommodity problems [9–11]. Although the expected performance for a general primal block-angular matrix is problem dependent, the effectiveness of the preconditioner is governed by the following criteria:

- **The spectral radius of** $D^{-1}(C^T B^{-1} C))$, which is always in $[0, 1)$ (Theorem 1 of [9]). The farther away from 1, the closer M^{-1} is to $(D - C^T B^{-1} C)^{-1}$. Although the particular behaviour of the spectral radius value is problem dependent, in general, it comes 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, therefore, D is diagonal. For general problems, D depends on the structure of side constraints, and we have to use a sparse Cholesky factorization (including row and column permutation, and symbolic factorization stages). If the fill-in in the factors of D is large, the preconditioner can be computationally expensive. Procedures devised to avoid fill-in for $A\Theta A^T$ [2] 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 Sect. 6.3 for the minimum congestion problem; in this 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. By using t to denote the number of PCG iterations, the solution for (14) and (15) requires $2 + t(1 + h)$ back-solves with matrix B. Although the numerical factorization is performed only once, the large number of back-solve 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_1 and C^Tv_2, **for some vectors** v_1 **and** v_2. If we denote the number of PCG iterations with t, the number of such products is $2+t(1+h)$. From (12), 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, in which N_i are node–arc incidence matrices, and $L_i = I$. The use of generic programming and virtual functions, which are common tools in object-oriented programming languages such as C++, can be highly effective for implementing 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, in which the Cholesky factorizations for B and D are performed through precompiled routines, and operations Cv_1 and C^Tv_2 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 the previous sections has been implemented in 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 the following from the front-end:

- $c \in \mathbb{R}^n$: the linear cost vector.
- $Q \in \mathbb{R}^n$: the quadratic cost matrix.
- $u \in \mathbb{R}^n$: the upper bounds vector.
- $b \in \mathbb{R}^m$: the right-hand-side vector.
- N: using the overloading capabilities 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 in the above case, 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 infeasible path-following algorithm, which solves normal equations either through a Cholesky factorization, or through the specialized procedure. For reasons of efficiency, Cholesky factorizations are performed through external precompiled routines. Specifically, the code uses the Ng–Peyton sparse Cholesky package [20], hooked to MATLAB for the LIPSOL package [27]. 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 2300 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 also includes a SCILAB version, a free MATLAB-like environment.

Although a MATLAB implementation is far less efficient than 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 Sect. 6 for details). If the results are satisfactory, it is worth spending time on an ad-hoc implementation for these kinds of problems. Up to now, such an ad-hoc code is only available for multicommodity flows [9]. The development of a generic C++ class, with virtual functions that exploit the structure of any block-angular problem does not fall within the scope of this paper, and remains among other pending tasks to be undertaken.

Some additional features of the package are:

1. The path-following algorithm implemented does not compute Mehrotra's predictor-corrector or higher-order directions. As observed in [9], in the particular case of multicommodity flows, the reduction in the number of iterations is not worthwhile, due to the increase in execution time per iteration as the PCG has to be applied twice. However, in the computational results, we also compare the specialized procedure with the native MATLAB interior-point solver, which is based on LIPSOL, and with CPLEX 9.1; both codes make use of higher-order directions. As will be shown in Sect. 6, the specialized procedure is competitive with Cholesky based procedures, whether they rely on Newton or higher-order directions.
2. As we approach an optimal point, system (14) becomes more ill-conditioned, and the PCG may give inaccurate solutions. When this happens, PRBLOCK_IP switches to the solution of normal equations (13) using a Cholesky factorization. This is done when we are close enough to the optimal point, and $\text{gap}^i = |p^i - d^i|/(1 + |p^i|)$ increases from one iteration to the next—p^i and d^i being the primal and dual objective functions at iteration i respectively. This rule is implemented as

$$(\text{gap}^i < 1/2) \quad \text{and} \quad (\text{gap}^i > 1.05\,\text{gap}^{i-1}). \tag{17}$$

   The tables of computational results in Sect. 6 provide both the overall number of interior-point iterations and the number of iterations performed before the satisfaction of rule (17). In general, the rule was only applied for the last iterations for some instances in Sect. 6.1; it was never used for those in Sect. 6.2; and it was required for the last iterations of all the instances in Sect. 6.3. It is worth noting that any other effective procedure can be selected for computing the directions, once condition (17) is satisfied. For instance, the solution of the augmented system by the PCG and the preconditioner of [21] would suitably complement our approach, since it proved to be very effective for the last interior-point iterations. Combining the two procedures is one of the further pending tasks.
3. The angle criteria of [22] 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 (14) 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 0. 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

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

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, and a minimum tolerance of $\min_\epsilon = 10^{-8}$. A tighter value is considered for quadratic instances because the preconditioner was shown to be more effective for them [11]. These tolerances are looser and tighter than those of [22] and [3], respectively. In principle, the initial tolerance and the updating factor could be fine-tuned for specific problems. For instance, Table 20 in Sect. 6.3 reports results with $\epsilon_0 = 10^{-4}$ for some instances. These results are significantly better than those obtained with the default tolerance. On the other hand, good results are obtained with default tolerances for the instances in Sects. 6.1 and 6.2. We decided to test the algorithm with standard values for all the problems, in order to demonstrate the robustness of the approach.

4. The initial dimension of system (14) is l, the number of linking constraints. Early detection of inactive linking constraints, initially suggested in [17], may significantly reduce the number of PCG iterations, and was very effective in practice for multicommodity flows [9]. The constraint elimination procedure of PRBLOCK_IP is only performed when $\text{gap}^i < 1$. The linking constraint i is considered inactive if (i) its slack x_i^{k+1} is far enough from the upper bound; (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); and (iv) its Lagrange multiplier y_i^{k+1} is close to 0. This is implemented as:
   (i) $9/10u_i^{k+1} > x_i^{k+1} > 1/10u_i^{k+1}$
   (ii) $(c_i^{k+1} = 0) \wedge (Q_{k+1_{ii}} = 0)$
   (iii) $u_i^{k+1} > 1/10$
   (iv) $|y_i^{k+1}| < 1/100$.
   Note that, unlike in general Cholesky solvers, the removal of linking constraints does not imply any additional symbolic refactorization with the PCG.
5. In addition to the preconditioner discussed in Sect. 4, PRBLOCK_IP includes three others:

   - Incomplete Cholesky factorization of $A\Theta A^T$, for the solution of normal equations by PCG without exploiting its block structure. The angle criteria (18) and the update formula (19) for the PCG were also used. The initial and minimum tolerances were set to $\epsilon_0 = 10^{-3}$ and $\min_\epsilon = 10^{-6}$, respectively.
   - Incomplete Cholesky factorization of matrix D, instead of its Cholesky factorization, which was the choice in Sect. 4. Formulae (18) and (19), and the same tolerances as the Cholesky factorization of D were used.
   - Incomplete Cholesky factorization of matrix $D - C^T \text{diag}(B)^{-1} C$. This preconditioner is similar to that suggested in [3] for the augmented system. Again, the same criteria (18) and (19), and tolerances described for the Cholesky factorization of D were used.

   For the incomplete Cholesky factorization, an initial drop tolerance of 10^{-3} was considered. It was reduced at each interior-point iteration by a factor of 0.8, in

an attempt to obtain more accurate preconditioners and to avoid instabilities as we approached the optimal solution. The built-in MATLAB incomplete Cholesky factorization was used.

6. The code does not consider standard optimality and feasibility tolerances in the range $[10^{-8}, 10^{-6}]$, since they are difficult to achieve using an iterative solver. By default, PRBLOCK_IP stops when $\text{gap}^i < 10^{-4}$. Primal and dual infeasibilities are not checked, although, for most instances tested, they were between 10^{-3} and 10^{-6} when rule (17) was not used. When this rule was applied, i.e., when the last iterations were performed using Cholesky for normal equations, the infeasibilities were reduced to less than 10^{-8} in few iterations. Solutions with smaller infeasibilities and relative duality gaps can be obtained by improving the tolerance of the PCG, at the expense of increasing the number of PCG iterations. A different approach is to use a loose tolerance for the PCG, rapidly obtaining a "quasi-optimal" solution. From this solution, very few interior-point iterations with a direct solver are required to reduce the infeasibilities and the duality gap. Another alternative is to detect the optimal face from the quasi-optimal solution (such a strategy has been used for minimum cost network flow problems [22]). Whatever option is considered, the suggested procedure is efficient for computing, at the very least, a point close to the optimal solution.

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. For some of the above problems there are more efficient algorithms than our specialized approach, but our aim 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 obtained with PRBLOCK_IP for solving normal equations (13) with a Cholesky factorization and with the PCG using an incomplete Cholesky factorization. Results are also given for the specialized procedure using the Cholesky factorization of D (the first term of the power series (16)), the incomplete Cholesky factorization of D, and the incomplete Cholesky factorization of $D - C^T \text{diag}(B)^{-1} C$. For the linear instances we also provide results with LINPROG, the linear optimization solver of MATLAB, which is based on LIPSOL [27] and computes Mehrotra's predictor–corrector directions through Cholesky factorizations. Results with the barrier algorithm of CPLEX 9.1 are also reported for all the instances. (We updated a free for research purposes MATLAB-CPLEX interface, available from `http://www-eio.upc.es/~jcastro/software.html`). It is worth noting that the implementations of PRBLOCK_IP and LINPROG are comparable, whereas CPLEX 9.1 is a state-of-the-art code with highly optimized routines. Therefore, the comparison between the procedures for computing Δy of PRBLOCK_IP and CPLEX 9.1 is biased due to the quality of the implementation. All runs were carried out on an HP-LC2000 server under the Linux operating system, with 2 Intel

Table 2 Percentage of overall execution time spent on the computation of Δy (including Cholesky and the 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

Xeon 933 MHz processors, and 2 GB of main memory. They were performed on a single processor, without exploiting parallelism capabilities.

As it is 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 CPLEX 9.1 and LINPROG we provide the overall execution time. We observed that in solving the normal equations by a Cholesky factorization, the relative difference between the time spent on Cholesky factorizations and the overall execution time is less than a 1% for large instances. However, in solving (13) by the specialized procedure, the execution time was occasionally many times greater (e.g., 60 times for large instances) than the time spent on 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 on the solution of (14) 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, the results obtained with the C implementation of the specialized procedure for multicommodity flows of [9] are presented in Table 2. For four of the instances in the following subsections, the table provides the dimensions (number of blocks k, number of rows m_i and columns n_i in all blocks), the percentage of the overall execution time spent on the computation of Δy (which includes both the PCG and Cholesky), and the percentage of the overall execution time spent on Cholesky-related procedures. These two figures would be closer for larger instances. They would also approach 100%. Therefore, the time spent on Cholesky procedures can be used to forecast the overall execution time of the specialized algorithm. From Table 2 we can see that, even for the smallest instances, the overall execution time would be, at the very most, about twice that needed for the Cholesky procedure.

6.1 Oriented and nonoriented multicommodity problems

Multicommodity flows are a standard source of difficult and large linear programming instances. The purpose of these problems is to route a set of commodities at a minimum cost over a capacitated oriented or nonoriented network. Oriented multicommodity flows 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–demand at the nodes for each commodity, and b^{k+1} is the mutual capacity of the arcs. l, the number of linking constraints, is n'. Note that D, defined in (12), is

diagonal, since L_i are diagonal matrices. The code of [9] is a particularly efficient C implementation of PRBLOCK_IP for linear multicommodity flows.

We considered a subset of the PDS [8] and Mnetgen [1] 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 3 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). Preprocessing removed inactive linking constraints; this explains why $l < n'$, $m < km' + n'$ and $n < (k+1)n'$ in Table 3.

Nonoriented multicommodity problems allow flow in both directions, and need two variables for each arc of the network. They are also primal block-angular problems with $N_i = [N \; -N]$ and $L_i = [I \;\; I]$ 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. Note that, as in the oriented case, D is a diagonal matrix, since $L_i L_i^T = [I \;\; I][I \;\; I]^T = 2I$ is diagonal. A description of other specialized algorithms for this problem can be found in [14]. 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 the last two columns of Table 3.

The results obtained with PRBLOCK_IP, LINPROG (a.k.a. LIPSOL) and CPLEX for oriented and nonoriented multicommodity problems are presented in Tables 4 and 5, respectively. 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"), for LINPROG and for CPLEX. The numbers in brackets in the first column "Iter" show the number of interior-point iterations with normal equations solved by the combined PCG and Cholesky approach, i.e., before the satisfaction of rule (17). Column "PCG" reflects the overall number of PCG iterations for option "PCG + Chol". For LINPROG and CPLEX we provide the overall execution time; for options "PCG + Chol" and "Chol" we provide the CPU time spent on Cholesky routines. Although in an efficient C/C++ implementation the overall execution time

Table 3 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 4 Results for oriented multicommodity instances

	PRBLOCK_IP					LINPROG		CPLEX 9.1	
	PCG + Chol			Chol					
Instance	Iter	CPU[a]	PCG	Iter	CPU[a]	Iter	CPU	Iter	CPU
PDS1	34	2.5	160	34	0.6	23	4.4	14	0.7
PDS5	51	22	337	48	64	40	264	32	26
PDS10	68	75	495	62	790	46	2021	50	115
PDS15	76	188	492	81	4624	54	6365	34	344
M32-32	44	7.6	164	29	44	21	184	12	10
M64-64	54	22	643	39	1744	27	2762	15	30
M128-64	58	55	1584	47	31397	31	38869	17	231
M128-128	74	111	1688	60	247044	36	188640	22	1072

[a]CPU time spent on Cholesky routines

Table 5 Results for nonoriented multicommodity instances

	PRBLOCK_IP					LINPROG		CPLEX 9.1	
	PCG + Chol			Chol					
Instance	Iter	CPU[a]	PCG	Iter	CPU[a]	Iter	CPU	Iter	CPU
PDS1	31 (28)	1.2	101	26	0.5	24	9	19	2.2
PDS5	47 (41)	17	147	39	66	43	309	23	69
PDS10	56 (51)	92	214	47	651	60	2852	23	449
PDS15	83 (66)	1034	538	53	2813	61	6994	26	1262
M32-32	64 (43)	39	729	31	49	23	189	23	225
M64-64	62	33	1210	40	1914	27	1643	17	77
M128-64	71	105	2600	49	31294	32	35704	21	848
M128-128	74	130	1312	61	227612	39	168633	24	3522

[a]CPU time spent on Cholesky routines

of option "PCG + Chol" could take about twice the CPU time shown in these tables (mainly for the Mnetgen instances, as discussed above), the specialized procedure is much more efficient than the standard LINPROG procedure. Compared to CPLEX, option "PCG + Chol" is also remarkably more efficient for the largest instances when rule (17) was not applied. Note that both for oriented and nonoriented multicommodity problems, the structure of matrix D is diagonal, and thus computations with the preconditioner are inexpensive.

The results obtained with two of the three alternative preconditioners discussed in Sect. 5 are presented in Tables 6 and 7: incomplete Cholesky factorizations of normal equations (columns "IC NE") and of $D - C^T \mathrm{diag}(B)^{-1} C$ (columns "IC $D - C^T \mathrm{diag}(B)^{-1} C$"). The third option, an incomplete Cholesky factorization of D, was not tested since D is diagonal. It is observed that option "IC NE" is never competitive. "IC $D - C^T \mathrm{diag}(B)^{-1} C$" is also less effective than "PCG +

Table 6 Results for oriented multicommodity instances and alternative preconditioners

Instance	IC NE			IC $D - C^T \text{diag}(B)^{-1} C$		
	Iter	CPU[a]	PCG	Iter	CPU[a]	PCG
PDS1	56	5	207	32	2.2	472
PDS5	100[b]	509	1236	49 (46)	48	2191
PDS10	100[b]	2844	2333	64	283	6303
PDS15	100[b]	7781	1714	82	1260	15215
M32-32	38	24	142	51	15	1330
M64-64	62	418	260	63	55	2303
M128-64	100[b]	8755	389	69	153	3444
M128-128	100[b]	35396	379	86	392	4217

[a]CPU time spent on Cholesky and incomplete Cholesky routines

[b]Maximum number of iterations reached without a solution

Table 7 Results for nonoriented multicommodity instances and alternative preconditioners

Instance	IC NE			IC $D - C^T \text{diag}(B)^{-1} C$		
	Iter	CPU[a]	PCG	Iter	CPU[a]	PCG
PDS1	43	6	344	31(27)	1	78
PDS5	43	221	33747	48(42)	22	247
PDS10	[c]	—	—	55	58	649
PDS15	53	4327	127912	76(67)	747	1862
M32-32	39	36	54	66(57)	32	2541
M64-64	100[b]	1306	240	98(82)	800	5233
M128-64	100[b]	12729	296	98	568	9720
M128-128	100[b]	54647	358	100[b]	685	5571

[a]CPU time spent on Cholesky and incomplete Cholesky routines

[b]Maximum number of iterations reached without a solution

[c]Execution aborted: the PCG did not converge

Chol", but for the nonoriented PDS10 and PDS15 instances. For these two instances "IC $D - C^T \text{diag}(B)^{-1} C$" reduced the number of iterations performed after the satisfaction of rule (17). This means that the PCG found better solutions with this preconditioner than with the power series preconditioner, i.e., the Cholesky factorization of D.

Finally, to evaluate PRBLOCK_IP in problems with different N_i matrices, we generated variants of the oriented instances in Table 3 with different networks for each commodity (denoted as "vPDS*" and "vM*-*" in Table 8). We added 10% of new links to the network of each commodity, with random low costs, random high capacities for each commodity, and random source and target nodes (loops were not permitted). Mutual capacity constraints of the new links were considered inactive.

Table 8 Results for oriented multicommodity instances with different N_i matrices

	PRBLOCK_IP			LINPROG		CPLEX 9.1	
	PCG + Chol						
Instance	Iter	CPU[a]	PCG	Iter	CPU	Iter	CPU
vPDS1	44	1.2	160	29	7.4	23	2
vPDS5	75	25	337	49	425	35	82
vPDS10	88	62	495	54	3951	33	385
vPDS15	92	151	492	69	17398	35	1107
vM32-32	38	1.7	164	23	133	12	16
vM64-64	57(48)	17	643	29	534	16	47
vM128-64	71	74	1584	35	4525	19	342
vM128-128	86	168	1688	41	14285	25	1419

[a]CPU time spent on Cholesky routines

This generator is included in the distribution of the code discussed in Sect. 5. The results obtained are reported in Table 8. The meaning of the columns is the same as in previous tables. Results for PRBLOCK_IP with the "Chol" option were not computed to avoid excessive execution time. Comparing the results in Table 8 with those in Tables 4 and 5, it can be concluded that the behaviour of the specialized approach is independent, regardless of whether N_i matrices are the same.

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 [12, 13]. This is a major concern for National Statistical Institutes, which must guarantee that individual information cannot be disclosed from released data. Tabular data is obtained by crossing two or more variables in a file of microdata, e.g., city, age, and profession. The Cartesian product of values for these variables provides a set of cells. For each cell, the table reveals 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 several linear relations $Aa = b$. For instance, for a three-dimensional table of $r+1$, $c+1$ and $k+1$ categories for the first, second and third variable respectively (the last category corresponds to marginal values), the linear relations are

$$\sum_{i_1=1}^{r} a_{i_1 i_2 i_3} = a_{(r+1) i_2 i_3}, \quad i_2 = 1, \ldots, c, \ i_3 = 1, \ldots, k, \tag{20}$$

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

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

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

$$\begin{array}{rl} \min_x & \|x - a\|_L \\ \text{subject to} & Ax = b, \\ & \underline{a}_i \le x_i \le \overline{a}_i, \quad i = 1, \ldots, n, \\ & x_i \le a_i - lpl_i \quad \text{or} \quad x_i \ge a_i + upl_i, \quad i \in \mathcal{P}, \end{array} \tag{23}$$

$\underline{a}_i$ and $\overline{a}_i$ being the lower and upper bounds for each cell $i = 1, \ldots, n$, which are considered to be known by any data-attacker. In practice norms $\|.\|_1$ and $\|.\|_2$ are used, obtaining either a linear or a quadratic optimization problem. The results reported in this subsection correspond to $\|.\|_2$. More details about this model can be found in [13].

Exploiting the structure of matrix A, (23) can be formulated as a primal block-angular problem (1). The simplest case corresponds to the linear relations (20–22) of a three-dimensional table, whose primal block-angular structure is obtained as follows. Firstly, variables $x_{i_1 i_2 i_3},\ i_1 = 1, \ldots, r,\ i_2 = 1, \ldots, c,\ i_3 = 1, \ldots, k$ are reordered according to i_3, i.e., $x = (x_{i_1 i_2 1}^T, \ldots, x_{i_1 i_2 k}^T)^T\ i_1 = 1, \ldots, r,\ i_2 = 1, \ldots, c$. Each group for a particular i_3 contains $n' = rc$ variables. Secondly, constraints (20–21) are set first, and ordered according to i_3. Each group for a particular i_3 contains $m' = r + c$ constraints. The remaining rc constraints (22) are moved to end positions. The resulting constraint matrix structure is

$$\begin{array}{c} \begin{array}{cccc} x_{i_1 i_2 1} & x_{i_1 i_2 2} & \ldots & x_{i_1 i_2 k} \end{array} \\ A = \left[\begin{array}{cccc} N & & & \\ & N & & \\ & & \ddots & \\ & & & N \\ I & I & \ldots & I \end{array}\right] \end{array} \quad \begin{array}{l} \text{(20–21) for } i_3 = 1, \\ \text{(20–21) for } i_3 = 2, \\ \vdots \\ \text{(20–21) for } i_3 = k, \\ \text{(22)}. \end{array} \tag{24}$$

$N \in \mathbb{R}^{m' \times n'}$ denotes the structure of constraints (20–21). $I \in \mathbb{R}^{rc \times rc}$ are identity matrices related to constraints (22). (24) matches the constraint matrix structure of (1), for $N_i = N$ and $L_i = I$, $i = 1, \ldots, k$. The number of linking constraints is $l = rc$. Note that, since the L_i matrices are diagonal, D is also diagonal. Additional details can be found in [12].

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 9 reports the dimensions of each instance, which are denoted as CTA-*c*-*r*-*k*. The meaning of the columns is the same as in Table 3. The results obtained with PRBLOCK_IP and CPLEX only are presented

Table 9 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 10 Results for minimum-distance tabular adjustment instances

	PRBLOCK_IP					CPLEX 9.1	
	PCG + Chol			Chol			
Instance	Iter	CPU[a]	PCG	Iter	CPU[a]	Iter	CPU
CTA-15-15-10	7	0.2	14	7	0.2	10	0.9
CTA-15-15-25	7	0.3	14	7	4	10	2.3
CTA-25-25-25	8	0.5	16	8	42	8	17
CTA-50-25-25	7	0.8	12	8	77	9	50
CTA-50-50-50	7	2.7	9	7	2595	7	967

[a]CPU time spent on Cholesky routines

Table 11 Results for minimum-distance tabular adjustment instances and alternative preconditioners

	IC NE			IC $D - C^T \text{diag}(B)^{-1} C$		
Instance	Iter	CPU[a]	PCG	Iter	CPU[a]	PCG
CTA-15-15-10	7	0.5	23	7	0.8	90
CTA-15-15-25	7	1.7	25	7	1.1	87
CTA-25-25-25	8	8.7	25	8	24	195
CTA-50-25-25	7	32	19	7	135	115
CTA-50-50-50	7	312	16	7	1010	271

[a]CPU time spent on Cholesky and incomplete Cholesky routines

in Table 10, since LINPROG cannot deal with quadratic problems. The meaning of the columns is the same as in Tables 4 and 5. The results with alternative preconditioners are presented in Table 11. The meaning of the columns is the same as in Tables 6 and 7. The incomplete factorization of D was not tested since D is diagonal.

Clearly, for CTA problems, exploiting the problem structure through our procedure is significantly more efficient than solving the normal equations by Cholesky factorizations. Moreover, the specialized algorithm is far more efficient than a general state-of-the-art solver such as CPLEX. This is mainly due to the very small number of PCG iterations required. It is also concluded from Tables 10 and 11 that options "IC NE" and "IC $D - C^T \text{diag}(B)^{-1} C$" are less effective than the power series pre-

conditioner, i.e., Cholesky factorization of D. It is worth noting that general interior-point solvers have proved to be more efficient than simplex implementations for both linear and quadratic variants of CTA [13]. Therefore, the specialized procedure is one of the most efficient algorithms for this kind of problem.

6.3 Minimum congestion problems

The minimum congestion problem is equivalent to the maximum concurrent flow problem. In the literature, both problems are usually seen as one, and denoted as the maximum concurrent flow problem [5, 23]. These problems arise in practical applications on telecommunications networks. They have proved to be difficult for simplex algorithms [4]. The maximum concurrent flow problem is usually defined on infeasible nonoriented multicommodity networks of k commodities, $m'+1$ nodes, and n' arcs, i.e., the 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. Furthermore, the minimum congestion problem finds the minimum of the maximum relative increments in arc capacities, for each arc of the network, that makes the problem feasible, i.e., all multicommodity flows can be sent from sources to destinations. This min–max problem can be formulated as the following linear program

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

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

(25) is a primal block-angular problem. Its constraint matrix structure, which matches (2), is

$$\begin{array}{c} \begin{array}{cccccccccc} x^{1^+} & x^{1^-} & x^{2^+} & x^{2^-} & \ldots & x^{k^+} & x^{k^-} & z & y & \end{array} \\ \left(\begin{array}{cccccccccc} N & -N & & & & & & & & \\ & & N & -N & & & & & & \\ & & & & \ddots & & & & & \\ & & & & & N & -N & & & \\ I & I & I & I & \ldots & I & I & & -U & I \\ & & & & & & & -e & I & & I \end{array} \right), \end{array} \tag{26}$$

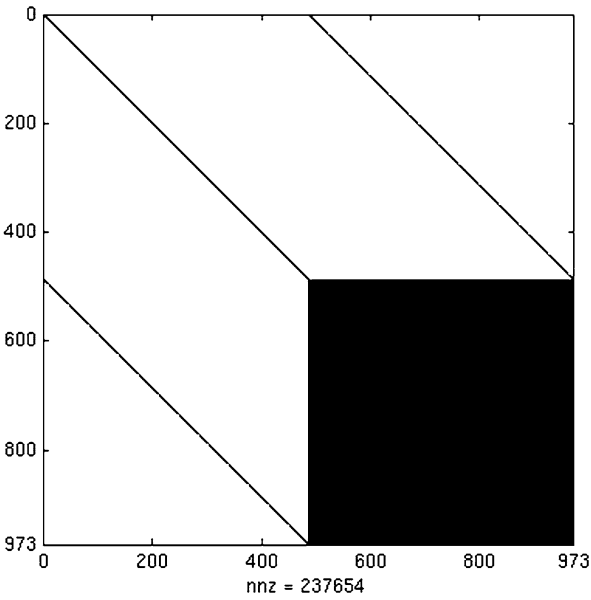

Fig. 1 Sparsity pattern of D for instance M32-32 and formulation (25)

Table 12 Dimensions of minimum congestion instances, formulation (25)

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

$e \in \mathbb{R}^{n'}$ being a vector of 1's. Formulation (25) has a dense column in the second group of linking constraints, due to the e vector in (26). Therefore, matrix D in (12) will show a dense submatrix of $n' \times n'$ nonzero elements. For example, Fig. 1 plots the sparsity pattern of D for instance M32-32 of Table 12. Even for $h = 0$, the preconditioner can be expected to be computationally expensive.

A second more efficient formulation is obtained by considering z_j, $j = 1, \ldots, n'$, for each arc, and imposing $n' - 1$ constraints $z_j = z_{j+1}$. This second model is

$$
\begin{array}{rll}
\min & z_1 & \\
\text{subject to} & Nx^{i^+} - Nx^{i^-} = b^i, & i = 1, \ldots, k, \\
& \sum_{i=1}^{k} (x_j^{i^+} + x_j^{i^-}) - y_j u_j \leq 0, & j = 1, \ldots, n', \\
& y_j - z_j \leq 0, & j = 1, \ldots, n', \\
& z_j - z_{j+1} = 0, & j = 1, \ldots, n' - 1, \\
& x^{i^+}, x^{i^-} \geq 0, & i = 1, \ldots, k, \\
& y_j \geq 0, & j = 1, \ldots, n'.
\end{array}
\tag{27}
$$

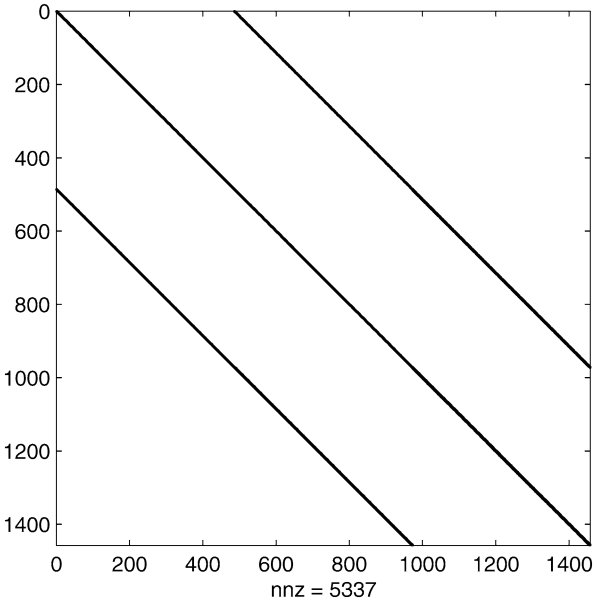

Fig. 2 Sparsity pattern of D for instance M32-32 and formulation (27)

Table 13 Dimensions of minimum congestion instances, formulation (27)

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

The constraint matrix structure of (27) is

$$\begin{array}{c} \begin{array}{cccccccccccc} x^{1+} & x^{1-} & x^{2+} & x^{2-} & \dots & x^{k+} & x^{k-} & z & y & & & \end{array} \\ \left(\begin{array}{cccccccccccc} N & -N & & & & & & & & & & \\ & & N & -N & & & & & & & & \\ & & & & \ddots & & & & & & & \\ & & & & & N & -N & & & & & \\ I & I & I & I & \dots & I & I & & -U & I & & \\ & & & & & & & -I & I & & I & \\ & & & & & & & T & & & & I \end{array} \right) \end{array}, \tag{28}$$

$T \in \mathbb{R}^{(n'-1)\times n'}$ being a band 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. For example, Fig. 2 plots the sparsity pattern of D for instance M32-32 in Table 13. Compared to the matrix in Fig. 1, the number of nonzeros decreased from 237654 to 5337.

We generated minimum congestion instances from the Mnetgen instances described in Sect. 6.1, increasing the supply and demand by a factor of two. The dimensions of these instances for each formulation are presented in Tables 12 and 13. Columns $\sum_{i=1}^{k} m_i$ and $\sum_{i=1}^{k} n_i$ show the number of constraints and variables for the diagonal blocks. The remaining columns have the same meaning as in previous tables. Tables 14, 15, 16 and 17 provide the computational results. The meaning of the columns is the same as in the tables in previous subsections. Columns "IC D" in Tables 16 and 17 show results with the incomplete Cholesky factorization of the preconditioner of D. It is clear that formulation (27) is much more efficient than (25) for all the options, except for CPLEX. This is probably explained by the handling of dense columns by CPLEX. We can also observe that the CPU time of the specialized algorithm is not as competitive as for instances in previous subsections: CPLEX is a more efficient choice, and the alternative preconditioners give a similar performance. Preconditioner "IC D" in particular is as effective as the Cholesky factorization of D (same number of PCG and interior-point iterations); the differences in CPU time can be explained by the implementations (Ng–Peyton Cholesky package against the built-in MATLAB incomplete Cholesky factorization routine). Unlike instances in previous subsections, the preconditioner "IC $D - C^T \mathrm{diag}(B)^{-1} C$" provides better execution times for the largest M128-128 instances. This is due to the lower number

Table 14 Results for minimum congestion instances, formulation (25)

	PRBLOCK_IP					LINPROG		CPLEX 9.1	
	PCG + Chol			Chol					
Instance	Iter	CPU[a]	PCG	Iter	CPU[a]	Iter	CPU	Iter	CPU
M32-32	24 (22)	29	156	22	194	45	279	9	40
M64-64	25	25	103	21	1682	16	1250	12	135
M128-64	27 (19)	8549	37	23	27847	20	19779	18	1326
M128-128	31 (20)	48429	27	24	116115	[b]	—	17	4461

[a]CPU time spent on Cholesky routines

[b]Not enough memory

Table 15 Results for minimum congestion instances, formulation (27)

	PRBLOCK_IP					LINPROG		CPLEX 9.1	
	PCG + Chol			Chol					
Instance	Iter	CPU[a]	PCG	Iter	CPU[a]	Iter	CPU	Iter	CPU
M32-32	27 (25)	7	188	24	57	32	302	9	51
M64-64	28 (23)	67	56	24	358	23	703	12	156
M128-64	30 (23)	3481	83	25	14043	39	16016	18	1566
M128-128	34 (22)	10541	31	26	24804	[b]	—	17	4674

[a]CPU time spent on Cholesky routines

[b]Code did not converge

Table 16 Results for minimum congestion instances, formulation (25), and alternative preconditioners

Instance	IC NE			IC D			IC $D - C^T \text{diag}(B)^{-1} C$		
	Iter	CPU[a]	PCG	Iter	CPU[a]	PCG	Iter	CPU[a]	PCG
M32-32	40	222	1443	24(22)	77	156	20	54	584
M64-64	50	1015	2414	25	89	103	26(20)	526	56
M128-64	[b]	—	—	27(19)	8938	37	29(24)	5237	132
M128-128	[b]	—	—	31(20)	50885	27	34(25)	41668	43

[a]CPU time spent on Cholesky and incomplete Cholesky routines

[b]Code got stuck

Table 17 Results for minimum congestion instances, formulation (27), and alternative preconditioners

Instance	IC NE			IC D			IC $D - C^T \text{diag}(B)^{-1} C$		
	Iter	CPU[a]	PCG	Iter	CPU[a]	PCG	Iter	CPU[a]	PCG
M32-32	54	80	18002	27 (25)	8	188	26	11	415
M64-64	61	775	23386	28 (23)	66	56	32 (23)	125	32
M128-64	85	5104	49934	30 (23)	3446	83	33 (25)	3364	81
M128-128	[b]	—	—	34 (22)	12962	31	29 (27)	1192	96

[a]CPU time spent on Cholesky and incomplete Cholesky routines

[b]Code got stuck

of iterations performed after rule (17) was satisfied. This is an indication that the PCG obtained better solutions with this preconditioner and the default PCG tolerances.

Since the efficiency of the PCG with the various preconditioners is obscured by the effect of rule (17), the CPU time spent on Cholesky factorization routines of interior-point iterations previous to the satisfaction of this rule is presented in Tables 18 and 19. Firstly, these tables, show that the approach is very competitive compared to one solely based on Cholesky factorizations, at least for rapidly computing a point close to the optimal solution. Secondly, they reaffirm that the second formulation is far more efficient than the first one for all the preconditioners. Moreover, the CPU times in Tables 18 and 19 are better than those reported by CPLEX, although the directions obtained were not so accurate. The main reason the preconditioner is not so effective is because D is nondiagonal. To avoid this drawback, we solved formulation (27) by setting an initial PCG tolerance $\epsilon_0 = 10^{-4}$ (instead of the default $\epsilon_0 = 10^{-2}$). Results are shown in Table 20 for the Cholesky factorization of the D preconditioner. With this tighter tolerance, rule (17) was only needed for instance M64-64. The number of PCG iterations clearly increased compared to results with the default PCG tolerances in Table 15 (e.g., for the largest instance M128-128, from 31 PCG iterations in 22 interior-point iterations to 141 in 29). However, since the PCG solutions were more accurate and rule (17) was not satisfied, the overall execution time was significantly reduced, being orders of magnitude less than for LINPROG and CPLEX. These results also show that, for problems with nondiagonal D matrices, a more accurate ϵ_0 tolerance may be advisable.

Table 18 CPU time in Cholesky and incomplete Cholesky routines of interior-point iterations before satisfaction of rule (17), for formulation (25)

Instance	PCG + Chol	IC D	IC $D - C^T \text{diag}(B)^{-1} C$
M32-32	18	70	54
M64-64	24	89	80
M128-64	204	777	1000
M128-128	237	897	1125

Table 19 CPU time in Cholesky and incomplete Cholesky routines of interior-point iterations before satisfaction of rule (17), for formulation (27)

Instance	PCG + Chol	IC D	IC $D - C^T \text{diag}(B)^{-1} C$
M32-32	3	4	11
M64-64	5	6	10
M128-64	11	18	50
M128-128	16	25	73

Table 20 Results for option "PCG+Chol", formulation (27), using initial PCG tolerance $\epsilon_0 = 10^{-4}$

	PCG + Chol		
Instance	Iter	CPU[a]	PCG
M32-32	24	2.9	188
M64-64	26 (21)	66	73
M128-64	27	15	175
M128-128	29	27	141

[a]CPU time spent on Cholesky and incomplete Cholesky routines

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 still remain to be done. The most significant are: the development of an efficient C/C++ replacement for the current MATLAB implementation; improving the efficiency of the PCG by adaptive selection of h and ϵ_0; 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; the inclusion of the procedure in a more general framework for structured problems through interior-point solvers; and combining the suggested preconditioner with alternative PCGs for the last interior-point iterations, in order to yield a reliable and efficient interior-point solver based on iterative methods.

Acknowledgements The author is indebted to the anonymous referees, whose several suggestions greatly improved the presentation of the paper. In particular, the alternative preconditioner $D - C^T \text{diag}(B)^{-1} C$ was suggested by one of the referees.

References

1. Ali, A., Kennington, J.L.: Mnetgen program documentation. Technical Report 77003, Department of Industrial Engineering and Operations Research, Southern Methodist University, Dallas (1977)
2. Andersen, E.D., Gondzio, J., Mészáros, C., Xu, X.: Implementation of interior point methods for large scale linear programming. In: Terlaky, T. (ed.) Interior Point Methods in Mathematical Programming, pp. 189–252. Kluwer, Dordrecht (1996)
3. Bergamaschi, L., Gondzio, J., Zilli, G.: Preconditioning indefinite systems in interior point methods for optimization. Comput. Optim. Appl. **28**, 149–171 (2004)
4. Bienstock, D.: Potential Function Methods for Approximately Solving Linear Programming Problems. Theory and Practice. Kluwer, Boston (2002)
5. Bienstock, D., Raskina, O.: Asymptotic analysis of the flow deviation method for the maximum concurrent flow problem. Math. Program. **91**, 479–492 (2002)
6. Bixby, R.E.: Solving real-world linear programs: a decade and more of progress. Oper. Res. **50**, 3–15 (2002)
7. Bixby, R.E., Fenelon, M., Gu, Z., Rothberg, E., Wunderling, R.: MIP: Theory and practice—Closing the gap. In: Powell, M.J.D., Scholtes, S. (eds.) System Modelling and Optimization. Methods, Theory and Applications, pp. 19–49. Kluwer, Boston (2000)
8. Carolan, W.J., Hill, J.E., Kennington, J.L., Niemi, S., Wichmann, S.J.: An empirical evaluation of the KORBX algorithms for military airlift applications. Oper. Res. **38**, 240–248 (1990)
9. Castro, J.: A specialized interior-point algorithm for multicommodity network flows. SIAM J. Optim. **10**, 852–877 (2000)
10. Castro, J.: Solving difficult multicommodity problems through a specialized interior-point algorithm. Ann. Oper. Res. **124**, 35–48 (2003)
11. Castro, J.: Solving quadratic multicommodity problems through an interior-point algorithm. In: Sachs, E.W., Tichatschke, R. (eds.) System Modelling and Optimization XX, pp. 199–212. Kluwer, Boston (2003)
12. Castro, J.: Quadratic interior-point methods in statistical disclosure control. Comput. Manag. Sci. **2**, 107–121 (2005)
13. Castro, J.: Minimum-distance controlled perturbation methods for large-scale tabular data protection. Eur. J. Oper. Res. **171**, 39–52 (2006)
14. Chardaire, P., Lisser, A.: Simplex and interior point specialized algorithms for solving nonoriented multicommodity flow problems. Oper. Res. **50**, 260–276 (2002)
15. Chin, P.: Iterative algorithm for solving linear programming from engineering applications. Ph.D. thesis, University of Waterloo (1995)
16. Gondzio, J., Grothey, A.: Exploiting structure in parallel implementation of interior point methods for optimization. Technical Report MS-04-004, School of Mathematics, University of Edinburgh (2005)
17. Gondzio, J., Makowski, M.: Solving a class of LP problems with a primal–dual logarithmic barrier method. Eur. J. Oper. Res. **80**, 184–192 (1995)
18. Gondzio, J., Sarkissian, R.: Parallel interior point solver for structured linear programs. Math. Program. **96**, 561–584 (2003)
19. Mamer, J., McBride, R.: A decomposition bases pricing procedure for large scale linear programs: an application to the linear multicommodity flow problem. Manag. Sci. **46**, 693–709 (2000)
20. Ng, E., Peyton, B.W.: Block sparse Cholesky algorithms on advanced uniprocessor computers. SIAM J. Sci. Comput. **14**, 1034–1056 (1993)
21. Oliveira, A.R.L., Sorensen, D.C.: A new class of preconditioners for large-scale linear systems from interior point methods for linear programming. Linear Algebra Appl. **394**, 1–24 (2005)
22. Resende, M.G.C., Veiga, G.: An implementation of the dual affine scaling algorithm for minimum-cost flow on bipartite uncapacitated networks. SIAM J. Optim. **3**, 516–537 (1993)
23. Shahrokhi, F., Matula, D.W.: The maximum concurrent flow problem. J. ACM **37**, 318–334 (1990)
24. Vanderbei, R.J.: Linear Programming: Foundations and Extensions. Kluwer, Boston (1996)
25. Wang, W., O'Leary, D.P.: Adaptive use of iterative methods in predictor–corrector interior point methods for linear programming. Numer. Algorithms **25**, 387–406 (2000)
26. Wright, S.J.: Primal–Dual Interior-Point Methods. SIAM, Philadelphia (1996)
27. Zhang, Y.: Solving large-scale linear programs by interior-point methods under the MATLAB environment. Optim. Methods Software **10**, 1–31 (1998)