Improving an interior-point algorithm for multicommodity flows by quadratic regularizations

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

Jordi Cuesta
Dept. of Chemical Engineering
Universitat Rovira i Virgili
jordi.cuesta@urv.cat

Research Report UPC-DEIO DR 2009-09
July 2009

Improving an interior-point algorithm for multicommodity flows by quadratic regularizations [*]

Jordi Castro[†]
Dept. of Statistics and Operations Research
Universitat Politècnica de Catalunya
c. Jordi Girona 1–3, 08034 Barcelona
`jordi.castro@upc.edu`

Jordi Cuesta
Unit of Statistics and Operations Research
Dept. of Chemical Engineering
Universitat Rovira i Virgili
`jordi.cuesta@urv.cat`

Abstract

One of the best approaches for some classes of multicommodity flow problems is a specialized interior-point method that solves the normal equations by a combination of Cholesky factorizations and preconditioned conjugate gradient. Its efficiency depends on the spectral radius—in [0,1)—of a certain matrix in the definition of the preconditioner. In a recent work the authors improved this algorithm (i.e., reduced the spectral radius) for general block-angular problems by adding a quadratic regularization to the logarithmic barrier. This barrier was shown to be self-concordant, which guarantees the convergence and polynomial complexity of the algorithm. In this work we focus on linear multicommodity problems, a particular case of primal block-angular ones. General results are tailored for multicommodity flows, allowing a local sensitivity analysis on the effect of the regularization. Extensive computational results on some standard and some difficult instances, testing several regularization strategies, are also provided. These results show that the regularized interior-point algorithm is more efficient than the nonregularized one. From this work it can be concluded that, if interior-point methods based on conjugate gradients are used, linear multicommodity flow problems are most efficiently solved as a sequence of quadratic ones.

Key words: interior-point methods, multicommodity network flows, preconditioned conjugate gradient, regularizations, large-scale computational optimization

[*]Work supported by Spanish MEC project MTM2006-05550, and by Catalan Research Council grant 2009-SGR-1122.

[†]Corresponding author

1 Introduction

Multicommodity flows are widely used as a modeling tool in many fields as, e.g., in telecommunications and transportation problems. This kind of models are usually very large linear programming problems, and some difficult instances have shown to be challenging for state-of-the-art solvers [8]. For these difficult instances, the specialized interior-point algorithm of [7] was a very efficient choice. In this work that approach is improved by adding a quadratic regularization. In particular, as it will be shown, the quality of the preconditioner of the PCG solver used by the algorithm is improved by the regularization. The resulting multicommodity flow code is more efficient than the original nonregularized one of [7]. The new multicommodity algorithm relies on theoretical results developed in [11] for a more general class of problems.

In the last two decades there has been a significant amount of research in the field of multicommodity flows, mainly for linear problems. Some of the solution strategies can be broadly classified into four main categories: simplex-based methods [12, 18], decomposition methods [4, 14, 15], approximation methods [5], and interior-point methods [4, 7, 15]. Some of the approaches for linear multicommodity flows were compared in [13]. Significant advances have also been made for nonlinear multicommodity flows. Among them we find active set methods [12], ACCPM approaches [3, 15], interior-point methods for quadratic problems [9], proximal point algorithms [21], and bundle-type decomposition [17]. A description and empirical evaluation of additional nonlinear multicommodity algorithms can be found in the survey [22].

The specialized interior-point algorithm for multicommodity flows extended in this work was first suggested in [7]. Given a directed network of n' arcs and $m'+1$ nodes, the algorithm considers this general formulation for multicommodity flows:

$$\min \quad \sum_{i=0}^{k} ({c^i}^T x^i + {x^i}^T Q_i x^i) \tag{1a}$$

$$\text{subject to} \quad \begin{bmatrix} N & & & & 0 \\ & N & & & 0 \\ & & \ddots & & \vdots \\ & & & N & 0 \\ I & I & \dots & I & I \end{bmatrix} \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^k \\ x^0 \end{bmatrix} = \begin{bmatrix} b^1 \\ b^2 \\ \vdots \\ b^k \\ u \end{bmatrix} \tag{1b}$$

$$0 \le x^i \le u^i \qquad i = 0, \dots, k. \tag{1c}$$

$x^i \in \mathbb{R}^{n'}, i = 1, \dots, k$, are the flows per commodity i, while $x^0 \in \mathbb{R}^{n'}$ are the slacks of capacity constraints. $N \in \mathbb{R}^{m' \times n'}$ is the node-arc incidence matrix of the directed graph. Note we assume N has full row-rank, which can always be achieved by removing one of the redundant constraints associated to some node. I is the $n' \times n'$ identity matrix, used in the formulation of linking constraints. $u \in \mathbb{R}^{n'}$ is the vector of arc capacities for all the commodities, while $u^i \in \mathbb{R}^{n'}, i = 1, \dots, n'$, are the individual capacities per commodity; $u^0 \in \mathbb{R}^{n'}$ are the upper bounds of slacks x^0, and in general we have $u^0 = u$. Vectors $b^i \in \mathbb{R}^{m'}, i = 1, \dots, k$, are the node supply/demands for each commodity. $c^i \in \mathbb{R}^{n'}, i = 1, \dots, k$, are the arc linear costs per commodity, and the diagonal positive semidefinite matrices $Q_i \in \mathbb{R}^{n' \times n'}, i = 1, \dots, k$, denote the arc quadratic costs. Note that the algorithm can also deal with linear costs $c^0 \in \mathbb{R}^{n'}$ and quadratic costs $Q_0 \in \mathbb{R}^{n' \times n'}$ (Q_0 diagonal and positive semidefinite) for slacks; this can be useful for problems that involve quadratic costs for the total flow on arcs, since $\sum_{i=1}^k x^i = u - x^0$. Clearly, for linear multicommodity problems $Q_i = 0$. However, the regularized algorithm will make use of this quadratic term.

The structure of this paper is as follows. Section 2 outlines the specialized interior-point algorithm for primal block-angular problems, provides the main theoretical results about the improvement due to a quadratic term, and describes the regularized variant of the specialized algorithm and its main properties. Section 3 particularizes general results for primal block-angular problems to multicommodity flows. Using these particular results, Section 4 performs a sensitivity analysis to the addition of a quadratic regularization term. Section 5 evaluates several regularization strategies. Finally, Section 6 provides computational results with an implementation of the regularized algorithm.

2 Outline of the regularized interior-point algorithm for multicommodity flows

The specialized algorithm, initially developed for multicommodity flows [7], was extended for general primal block-angular problems in [10]. The improved regularized version [11] was developed for this more general formulation:

$$\min \quad \sum_{i=0}^{k} ({c^i}^T x^i + {x^i}^T Q_i x^i) \tag{2a}$$

$$\text{subject to} \quad \begin{bmatrix} N_1 & & & & 0 \\ & N_2 & & & 0 \\ & & \ddots & & \vdots \\ & & & N_k & 0 \\ L_1 & L_2 & \dots & L_k & I \end{bmatrix} \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^k \\ x^0 \end{bmatrix} = \begin{bmatrix} b^1 \\ b^2 \\ \vdots \\ b^k \\ b^0 \end{bmatrix} \tag{2b}$$

$$0 \le x^i \le u^i \qquad i = 0, \dots, k. \tag{2c}$$

The main differences between (2) and (1) are: (i) matrices $N_i \in \mathbb{R}^{m_i \times n_i}$ and $L_i \in \mathbb{R}^{l \times n_i}$ may have any structure, and be of different dimensions for each $i = 1, \dots, k$, l being the number of linking constraints; (ii) vectors $x^i \in \mathbb{R}^{n_i}$ and $b^i \in \mathbb{R}^{m_i}$, $i = 1, \dots, k$, are no longer related to flows and flow injections, respectively; and (iii), the right-hand-side of linking constraints is $b^0 \in \mathbb{R}^l$ instead of a mutual capacity (and in general, $b^0 = u^0$). We also restrict our considerations to the separable case where $Q_i \in \mathbb{R}^{n_i \times n_i}$, $i = 0, \dots, k$, are diagonal positive semidefinite matrices.

2.1 The specialized algorithm

Problem (2) can be written as

$$\begin{array}{rl} \min & c^T x + \frac{1}{2} x^T Q x \\ \text{subject to} & Ax = b \\ & 0 \le x \le u \end{array} \tag{3}$$

where $c, x, u \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $Q \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^m$. Note that $n = \tilde{n} + l$ and $m = \tilde{m} + l$, where $\tilde{n} = \sum_{i=1}^k n_i$ and $\tilde{m} = \sum_{i=1}^k m_i$; for the particular case of multicommodity problems (1), $\tilde{n} = kn'$, $\tilde{m} = km'$ and $l = n'$, and thus $n = (k+1)n'$ and $m = km' + n'$. Replacing inequalities in (3) by a logarithmic barrier with parameter $\mu > 0$ we obtain the logarithmic barrier problem

$$\begin{array}{ll} \min & B(x,\mu) \triangleq c^T x + \frac{1}{2} x^T Q x + \mu \left(-\sum_{i=1}^n \ln x_i - \sum_{i=1}^n \ln(u_i - x_i) \right) \\ \text{subject to} & Ax = b. \end{array} \tag{4}$$

The KKT conditions of (4) are [24]:

$$\begin{aligned} Ax &= b, && (5a) \\ A^T y - Qx + z - w &= c, && (5b) \\ XZe &= \mu e, && (5c) \\ (U - X)We &= \mu e, && (5d) \\ (z, w) &> 0 \quad u > x > 0; && (5e) \end{aligned}$$

$e \in \mathbb{R}^n$ is a vector of 1's; $y \in \mathbb{R}^m$, $z, w \in \mathbb{R}^n$ are the Lagrange multipliers (or dual variables) of $Ax = b$, $x \geq 0$ and $x \leq u$, respectively; and matrices $X, Z, U, W \in \mathbb{R}^{n \times n}$ are diagonal matrices made up of vectors x, z, u, w. Equations (5a)–(5b) impose, respectively, primal and dual feasibility; (5c)–(5d) impose complementarity. The normal equations for the Newton direction $(\Delta x, \Delta y, \Delta z)$ of (5) reduce to (see [10] for details)

$$\begin{aligned} (A\Theta A^T)\Delta y &= g && (6) \\ \Theta &= (Q + (U - X)^{-1} W + X^{-1} Z)^{-1}, && (7) \end{aligned}$$

for some right-hand-side g. For linear (i.e., $Q = 0$) or separable quadratic problems Θ is a diagonal positive definite matrix and it can be easily computed. Exploiting the structure of A and Θ in (2), the matrix of (6) can be written as

$$\begin{aligned} A\Theta A^T &= \left[\begin{array}{ccc|c} N_1 \Theta_1 N_1^T & & & N_1 \Theta_1 L_1^T \\ & \ddots & & \vdots \\ & & N_k \Theta_k N_k^T & N_k \Theta_k L_k^T \\ \hline L_1 \Theta_1 N_1^T & \dots & L_k \Theta_k N_k^T & \Theta_0 + \sum_{i=1}^k L_i \Theta_i L_i^T \end{array} \right] \\ &= \begin{bmatrix} B & C \\ C^T & D \end{bmatrix}, \end{aligned} \tag{8}$$

$B \in \mathbb{R}^{\tilde{m} \times \tilde{m}}$, $C \in \mathbb{R}^{\tilde{m} \times l}$ and $D \in \mathbb{R}^{l \times l}$ being the blocks of $A\Theta A^T$, and Θ_i, $i = 0, \dots, k$, the submatrices of Θ associated with the $k+1$ groups of variables in (2), i.e., $\Theta_i = (Q_i + (U_i - X_i)^{-1} W_i + X_i^{-1} Z_i)^{-1}$. Appropriately partitioning g and Δy in (6), the normal equations can be written as

$$\begin{bmatrix} B & C \\ C^T & D \end{bmatrix} \begin{bmatrix} \Delta y_1 \\ \Delta y_2 \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix}. \tag{9}$$

By eliminating Δy_1 from the first group of equations of (9), we obtain

$$\begin{aligned}(D - C^T B^{-1} C)\Delta y_2 &= (g_2 - C^T B^{-1} g_1) \qquad (10a)\\ B\Delta y_1 &= (g_1 - C\Delta y_2). \qquad (10b)\end{aligned}$$

System (10b) is solved by 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 (9), is solved by a precondintioned conjugate gradient (PCG). The dimension of this system is l, which is the number of linking constraints. In [7] it was proved that the inverse of $(D - C^T B^{-1} C)$ can be computed as

$$(D - C^T B^{-1} C)^{-1} = \left(\sum_{i=0}^{\infty} (D^{-1}(C^T B^{-1} C))^i \right) D^{-1}. \qquad (11)$$

The preconditioner M^{-1}, an approximation of $(D - C^T B^{-1} C)^{-1}$, is thus obtained by truncating the infinite power series (11) at some term h. In practice, $h = 0$ or $h = 1$ provide the best computational results.

2.2 Effect of the quadratic term

The effectiveness of the preconditioner depends on the spectral radius of matrix $D^{-1}(C^T B^{-1} C)$, which is always in $[0, 1)$ [7, Theorem 1]. The farther away from 1 is the spectral radius of $D^{-1}(C^T B^{-1} C)$, the better is the quality of the approximation of (11) obtained by truncation with $h = 0$ or $h = 1$. The next theorem and proposition from [11] show that the quadratic term in the objective function effectively reduces this spectral radius.

Theorem 1 *Let A be the constraint matrix of problem (2), with full row rank matrices $N_i \in \mathbb{R}^{m_i \times n_i}$ $i = 1, \dots, k$, and at least one full row rank matrix $L_i \in \mathbb{R}^{l \times n_i}$, $i = 1, \dots, k$. Let Θ be the diagonal positive definite matrix defined in (7), and $B \in \mathbb{R}^{\tilde{m} \times \tilde{m}}$, $C \in \mathbb{R}^{\tilde{m} \times l}$ and $D \in \mathbb{R}^{l \times l}$ the submatrices of $A\Theta A^T$ defined in (8). Then, the spectral radius ρ of $D^{-1}(C^T B^{-1} C)$ is bounded by*

$$0 \le \rho \le \max_{j \in \{1,\dots,l\}} \frac{\gamma_j}{\left(\frac{u_j}{v_j}\right)^2 \Theta_{0j} + \gamma_j} < 1, \qquad (12)$$

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

Proposition 1 *Let assume the hypotheses of Theorem 1, and consider a linear problem and a quadratic one obtained by adding (likely small) quadratic costs $Q_i \succ 0$, $i = 1, \dots, k$. Assume $\hat{u}_j/\hat{v}_j \le u_j/v_j$, $j = 1, \dots, l$, where "hatted" and "non-hatted" terms refer, respectively, to the linear and quadratic problems, and u and v are defined as in Theorem 1. Then bound (12) is smaller for the quadratic than for the linear problem.*

Preliminary computational results showed that the quadratic term in practice reduces the spectral radius, as predicted by the theory, and the overall number of PCG iterations and CPU time is significantly reduced. This explained the empirical results of previous works [9], where the specialized algorithm was more efficient for quadratic than for linear instances, where the quadratic instances were obtained from the linear ones by adding a separable quadratic convex cost.

2.3 The regularized algorithm

To reproduce the good behaviour of quadratic problems in linear ones a quadratic regularization term is added to the linear formulation (i.e., with $Q = 0$) of (3). Previously used regularized variants replaced $B(x, \mu)$ in (4) by a proximal point regularization

$$B_P(x,\mu) \triangleq c^T x + \frac{1}{2}(x-\bar{x})^T Q_P (x-\bar{x}) + \mu\left(-\sum_{i=1}^n \ln x_i - \sum_{i=1}^n \ln(u_i - x_i)\right), \quad (13)$$

Q_P being a positive definite matrix and $\bar{x}$ the current point obtained by the interior-point algorithm. For instance, Q_P was the identity matrix in [23]; and Q_P was a diagonal matrix with small entries—dynamically updated at each interior-point iteration—in [2]. Unfortunately, these proximal point regularizations depend on the current point $\bar{x}$, and then they do not fit the general theory of structural optimization for interior-point methods [19]. In [11] the authors suggested the alternative regularized barrier problem

$$B_Q(x,\mu) \triangleq c^T x + \mu\left(\frac{1}{2}x^T Q x - \sum_{i=1}^n \ln x_i - \sum_{i=1}^n \ln(u_i - x_i)\right), \quad (14)$$

Q being a diagonal positive semidefinite matrix. In this variant the regularization affects to the variables x (flows and slacks of (1)) instead to the directions as in (13). The regularized barrier function (14) was shown to be a self-concordant barrier [11] for upper-bounded problems and thus it fits the general interior-point theory of [19]. Since Q is diagonal, the self-concordant barrier

$$F_Q(x) = \frac{1}{2}x^T Q x - \sum_{i=1}^n \ln x_i - \sum_{i=1}^n \ln(u_i - x_i)$$

of (14) can be written as a sum of self-concordant barriers for each component:

$$F_Q(x) = \sum_{i=1}^n F_{q_i}(x_i) = \sum_{i=1}^n \left(\frac{1}{2}q_i x_i^2 - \ln x_i - \ln(u_i - x_i)\right), \quad (15)$$

q_i being the diagonal entry of Q. The complexity of the interior-point algorithm in number of iterations is $O(\sqrt{\nu}\ln 1/\epsilon)$, where ϵ is the accuracy of the solution, and ν is the *parameter* of the self-concordant barrier of (14), which can be computed as $\nu = \sum_{i=1}^n \nu_i$, where ν_i is the parameter of the barrier $F_{q_i}(x_i)$ of (15) for component i (see [19] for details). In [11] the following result about ν_i was proved:

Proposition 2 *The parameter of the self-concordant barrier $F_{q_i}(x_i)$ of (15) in its domain $\{x_i : 0 < x_i < u_i\}$ is*

$$\begin{array}{lll} \nu_i = 1 & \text{if} & 0 \le q_i \le 1/u_i^2, \\ \nu_i = q_i u_i^2 & \text{if} & q_i \ge 1/u_i^2. \end{array} \quad (16)$$

The value $\nu_i = 1$ is the lowest possible one for any self-concordant barrier [19, Lemma 4.3.1], and therefore for small regularizations the regularized algorithm is in theory as efficient as the standard interior-point one. When $q_i \geq 1/u_i^2$ the complexity increases, but, as will be shown in next subsections, there is a wide range of values for which the number of interior-point iterations do not increase with the regularization term. Since the regularization term means less PCG iterations, the overall CPU time is reduced, making the regularized algorithm an effective approach for primal block-angular problems. It is also worth noting that the only (minor) change in the interior-point algorithm due to the regularized barrier problem (14) is that the dual feasibility condition (5b) is replaced by

$$A^T y - \mu Q x + z - w = c. \tag{17}$$

For linear problems, (17) and (5b) are equivalent when μ tends to zero. If the proximal point regularization (13) is used, the dual feasibility becomes

$$A^T y - Q_P(x - \bar{x}) + z - w = c. \tag{18}$$

Although, for linear problems, (18) is equal to (5b) when $x = \bar{x}$, the expression of Θ associated to (18) is

$$\Theta = (Q_P + (U - X)^{-1} W + X^{-1} Z)^{-1}, \tag{19}$$

while for (17) is

$$\Theta = (\mu Q + (U - X)^{-1} W + X^{-1} Z)^{-1}. \tag{20}$$

Note that when μ tends to zero (20) is a better approximation than (19) of the linear version of (7) (i.e., when $Q = 0$ in (7)).

3 The case of multicommodity flow problems

The bound provided by Theorem 1 is difficult to compute for general primal block-angular problems. However, for the particular case of multicommodity flow problems it reduces to a simple and computable form. Indeed, since $l = n'$, $N_i = N$ and $L_i = I$ for $i = 1, \ldots, k$ we have that: (i) N_i and L_i have full row-rank; (ii) $\sum_{i=1}^k L_i \Theta_i {L_i}^T = \sum_{i=1}^k \Theta_i$ is a diagonal matrix, and its eigenvalues are $\gamma_j = \sum_{i=1}^k \Theta_{ij}$ with eigenvectors $V_j = e_j$ (e_j being the jth column of I), $j = 1, \ldots, n'$; (iii) $V = [V_1 \ldots V_{n'}] = I$, and then $v = V^T u = u$, i.e., $u_j/v_j = 1$ for $j = 1, \ldots, n'$. Therefore, for multicommodity problems bound (12) can be written as

$$\rho \leq \max_{j \in \{1,\ldots,n'\}} \frac{\displaystyle\sum_{i=1}^k \Theta_{ij}}{\Theta_{0j} + \displaystyle\sum_{i=1}^k \Theta_{ij}} < 1, \tag{21}$$

where Θ was defined in (7).

In addition, for multicommodity flows the strong assumption $\hat{u}_j/\hat{v}_j \leq u_j/v_j$ of Proposition 1 is satisfied, since $u_j/v_j = \hat{u}_j/\hat{v}_j = 1$. Therefore bound (21) is effectively reduced by adding even a small quadratic term $Q_i \succ 0$, $i = 1, \ldots, k$, to a linear multicommodity problem. The quadratic term of the regularized algorithm thus guarantees such a reduction. Although a reduction in the bound does not mean a reduction in the spectral radius (which is the instrumental factor), we note that in the last interior-point iterations, because of the ill-conditioning of Θ, the spectral radius tends to one [7], and then a reduction in the bound will also mean a reduction in the spectral radius. It is also worth noting that if N, the node-arc incidence matrix, is a square matrix then

$$\begin{aligned}(D^{-1}(C^TB^{-1}C)) &= \left(\Theta_0 + \textstyle\sum_{i=1}^k \Theta_i\right)^{-1} \left(\textstyle\sum_{i=1}^k \Theta_i N_i^T \left(N_i\Theta_i N_i^T\right)^{-1} N_i\Theta_i\right) \\ &= \left(\Theta_0 + \textstyle\sum_{i=1}^k \Theta_i\right)^{-1} \left(\textstyle\sum_{i=1}^k \Theta_i\right),\end{aligned}$$

which is equal to a diagonal matrix whose jth component is $\left(\Theta_{0j} + \sum_{i=1}^k \Theta_{ij}\right)^{-1} \left(\sum_{i=1}^k \Theta_{ij}\right)$. In this case, (21) does not actually provide a bound, but the true spectral radius. Although problems with N square are not of practical interest (they have at most one feasible solution), it shows how tight the bound is in a limit situation. Another interesting observation from this result is that slacks are instrumental: otherwise Θ_0 would be 0, and the bound would be one, independently of the regularization performed. This suggests that, even for primal block-angular (or multicommodity problems) with equality linking constraints (i.e. saturated arcs) it is worth to consider slacks with negligible upper bounds. This justifies what was empirically observed in [10], where a quadratic multicommodity flow problem—from the statistical disclosure control field— with equality capacity constraints (all arcs were saturated) was solved very efficiently with this algorithm considering slacks with very small upper bounds. Additional arguments over the benefits of the regularization term for decreasing the spectral radius are provided by the computational results of next sections.

4 Local sensitivity analysis

The simple expression of bound (21) allows us to perform a local sensitivity analysis on small regularizations. Let us consider that $j \in \{1, \dots, n'\}$ is the index providing the maximum in (21). By (7), the elements Θ_{ij}, $i = 0, \dots, k$, are

$$\Theta_{ij} = \frac{1}{Q_{ij} + (U_{ij} - X_{ij})^{-1} W_{ij} + X_{ij}^{-1} Z_{ij}}, \tag{22}$$

where Q_{ij}, U_{ij}, W_{ij}, X_{ij} and Z_{ij} are scalars. For linear problems $Q_{ij} = 0$. Adding a small quadratic regularization $Q_{ij} = \delta_i$, $i = 0, \dots, k$, both the spectral radius of matrix $D^{-1}(C^TB^{-1}C)$ and the bound (21) will change. Performing an accurate sensitivity analysis on the spectral radius is not possible [16, Section 8.1.2], but it can be done for the bound. Defining $t_i = (U_{ij} - X_{ij})^{-1} W_{ij} + X_{ij}^{-1} Z_{ij} > 0$, $i = 0, \dots, k$, the bound (21) can be written as a continuous function of $\vec{\delta} = (\delta_0, \dots, \delta_k)$:

$$f(\vec{\delta}) = \frac{\displaystyle\sum_{i=1}^k \frac{1}{\delta_i + t_i}}{\displaystyle\frac{1}{\delta_0 + t_0} + \sum_{i=1}^k \frac{1}{\delta_i + t_i}}. \tag{23}$$

We next show the effect of regularizing either the flows, slacks, or both of them.

In the first case we consider a regularization on flows only, thus $\delta_0 = 0$, and to simplify the notation we assume that $\delta_i = \delta$, $i = 1, \ldots, k$. In this case (23) and its derivative are

$$f(\delta) = \frac{\displaystyle\sum_{i=1}^{k} \frac{1}{\delta + t_i}}{\displaystyle\frac{1}{t_0} + \sum_{i=1}^{k} \frac{1}{\delta + t_i}}, \qquad f'(\delta) = \frac{\displaystyle\frac{-1}{t_0} \sum_{i=1}^{k} \frac{1}{(\delta + t_i)^2}}{\left(\displaystyle\frac{1}{t_0} + \sum_{i=1}^{k} \frac{1}{\delta + t_i}\right)^2}. \tag{24}$$

Since $t_0 > 0$, $f'(\delta) < 0$ and $f(\delta)$ is a monotonically decreasing function. This holds not only for the jth component associated to the maximum in (21), but for all the components. Therefore the bound is always reduced if flows only are regularized. This is consistent with Proposition 1, which only considered the addition of quadratic costs $Q_i \succ 0$, $i = 1, \ldots, k$, with $Q_0 = 0$.

In the second case, if we only regularize the slacks, i.e., $\delta_0 = \delta$ and $\delta_i = 0, i = 1, \ldots, k$, (23) and its derivative become

$$f(\delta) = \frac{\displaystyle\sum_{i=1}^{k} \frac{1}{t_i}}{\displaystyle\frac{1}{\delta + t_0} + \sum_{i=1}^{k} \frac{1}{t_i}}, \qquad f'(\delta) = \frac{\displaystyle\sum_{i=1}^{k} \frac{1}{t_i}}{\left(1 + (\delta + t_0) \displaystyle\sum_{i=1}^{k} \frac{1}{t_i}\right)^2}. \tag{25}$$

Since $t_i > 0$, $i = 1, \ldots, k$, $f'(\delta) > 0$, and thus $f(\delta)$ is monotonically increasing, which means that locally the bound on the spectral radius will get worse.

Finally, the more general case considers a regularization on both the flows and slacks. To simplify the notation we assume that $\delta_i = \delta$, $i = 0, \ldots, k$. In this case (23) and its derivative are

$$f(\delta) = \frac{\displaystyle\sum_{i=1}^{k} \frac{1}{\delta + t_i}}{\displaystyle\frac{1}{\delta + t_0} + \sum_{i=1}^{k} \frac{1}{\delta + t_i}}, \qquad f'(\delta) = \frac{\displaystyle\frac{1}{(\delta + t_0)^2} \sum_{i=1}^{k} \frac{(t_i - t_0)}{(\delta + t_i)^2}}{\left(\displaystyle\frac{1}{\delta + t_0} + \sum_{i=1}^{k} \frac{1}{\delta + t_i}\right)^2}. \tag{26}$$

In this case $f'(\delta)$ can be either positive or negative depending on $\sum_{i=1}^{k} \frac{(t_i - t_0)}{(\delta + t_i)^2}$.

According to the previous results, the safest option for multicommodity flow problems is to perform a regularization on the flows only. Regularizing both flows and slacks can be even more effective, but it will depend on the particular values of Θ (and it may be computationally expensive to perform a check). Regularizing only the slacks is in general not a good choice. Indeed, in the implementation developed, and tested in next Subsection, the default option is to regularize the flows only. It is worth noting, however, that, first, this sensitivity analysis is local; and second, it is only valid for the bound on the spectral radius, not the spectral radius. Therefore, it might happen that for some instances the regularization of slacks even provided good results.

5 Evaluating the regularized algorithm

The original code IPM [7] implementing the specialized interior-point algorithm for multicommodity flows has been extended with the regularized barrier (14). The new code will be denoted as RIPM. RIPM is mainly written in C, with only the sparse Cholesky factorization routines coded in Fortran [20]. RIPM is available from the authors on request. The three main parameters to be adjusted in the algorithm are h, the number of terms (minus one) of the power series (11) considered in the preconditioner; ϵ_0, the initial PCG tolerance requested, which is updated at each interior-point iteration; and Q, the diagonal positive semidefinite regularization matrix of (14). As for IPM, the default values for h and ϵ_0 in RIPM are 0 and 10^{-2} respectively. They have been used in all the computational results of Section 6, excluding some few that are clearly marked. For the third and new parameter, an empirical study—based on the results of Subsection 2.3—has been performed for an appropriate choice of Q; this is shown in below Subsection 5.3.

The termination criteria for RIPM are the same than for IPM: the code stops when the current primal and dual feasible point (i.e., it solves (5a), (5b) and (5e)) has a relative optimality gap

$$\frac{\left|c^T x - \left(b^T y - u^T w\right)\right|}{1 + |(c^T x)|} \tag{27}$$

below some optimality tolerance (by default 10^{-6}). We note that in theory the code should stop when $\mu = 0$ in (5c) and (5d), such that (5) corresponds to the KKT conditions of (3). However, such a strong condition $\mu = 0$ can not be imposed, specially using a PCG for the solution of normal equations. Therefore, in practice the regularization term $\mu x^T Q x$ could take a non-negligible value in the optimal point, perturbing the optimizer of the original problem. For this reason RIPM adds an additional control at the optimal solution: it checks that

$$\frac{x^T Q x}{|c^T x|} \tag{28}$$

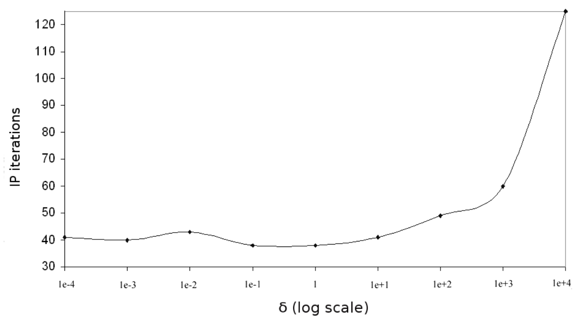
is below some tolerance (by default 10^{-6}). If this check fails, the problem should be solved by reducing or removing the regularization term, or increasing the code tolerances (optimality tolerance and ϵ_0).

5.1 Problem instances

We considered three kind of problems. They are used both in this Section 5 and next Section 6.

The first type corresponds to the well-known PDS problems [6]. Problems obtained with this generator are denoted as PDSt, where t is associated to the planning horizon in days of a military logistic problem. The PDS instances can be retrieved from `http://www.di.unipi.it/di/groups/optimize/Data/MMCF.html`. The second kind was obtained with the implementation of [14] of the Mnetgen generator [1]. It can be retrieved from the above URL. These instances will be denoted as m'-k-d, where m' is the number of nodes, k the number of commodities, and d is related to the density of the network; the larger d the denser is the network. The last set of instances was obtained with

Figure 1: Interior-point iterations for instance PDS1 using $Q = \delta I$



the Tripartite generator and with a variation for multicommodity flows of the Gridgen generator. They are known to be difficult linear programming instances, and interior-point algorithms outperformed simplex variants on them [5, 8]. Five such test examples are available. They can be obtained from `http://www-eio.upc.es/~jcastro/mmcnf_data.html`.

5.2 Effect of Q on the number of iterations

According to Proposition 1 and Section 4, the bound on the spectral radius is reduced if a regularization is considered, and we can expect a reduction in the number of PCG iterations needed. On the other hand, by Proposition 2, the diagonal elements Q_{ij}, $i = 1, \ldots, k$, $j = 1, \ldots, n'$, of the regularization matrix should be less or equal than $1/u_{ij}^2$ to have the same complexity result in number of iterations than the nonregularized interior-point algorithm, u_{ij} being the capacity of arc j for commodity i. The complexity increases for larger Q_{ij} values. In many instances the term $1/u_{ij}^2$ would be very small—almost negligible—, causing no reduction in the spectral radius.

Fortunately, in practice it has been observed that there is wide range of regularization values (much larger than $1/u_{ij}^2$) that maintain the number of interior-point iterations; this number of iterations only increases when large regularizations are used. For instance, let us consider problem PDS1, one of the smallest instances considered, and the simple regularization matrix $Q = \delta I$ for some $\delta \in \mathbb{R}$, $\delta \geq 0$. Figure 1 shows the number of iterations for several δ. Note that for many arcs of PDS1, the value $1/u_{ij}^2$ was about 10^{-7}, which is much smaller than the values used in Figure 1.

It is also worth noting how Q affects to the number of PCG iterations. Figure 2 shows the average number of PCG iterations needed per interior-point iteration, again for instance PDS1 and the simple regularization matrix $Q = \delta I$ for several δ. It is shown that for small regularizations this average ratio is kept constant, it decreases for δ between 10^{-1} and 10^2, and it significantly increases when $\delta \geq 5 \cdot 10^2$. This does not contradict that the regularization term decreases the number of PCG iterations; indeed, the significant increment of PCG iterations happened in the last interior-point iterations, when the regularization term is very small. This is observed in Figure 3 which shows the evolution of the

Figure 2: Ratio between PCG and interior-point iterations for instance PDS1 using $Q = \delta I$

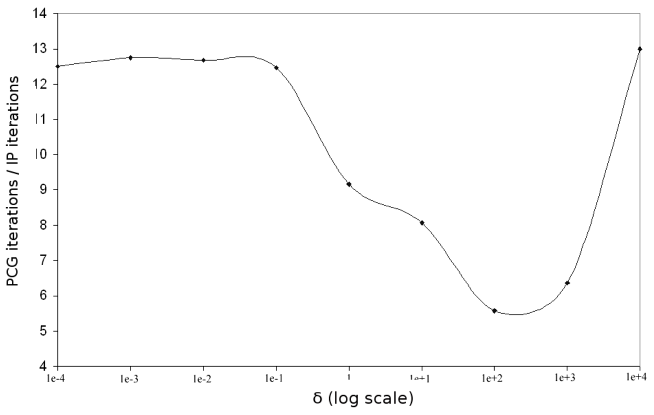
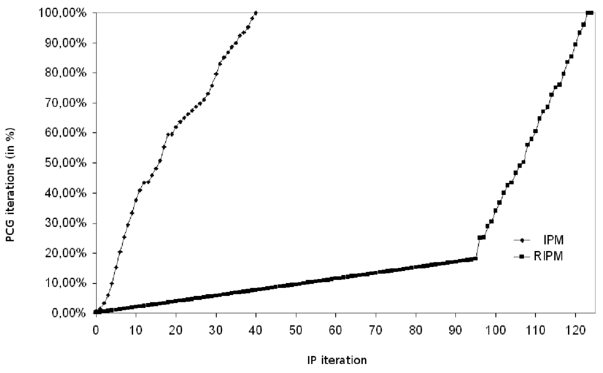


Figure 3: Evolution of percentage of PCG iterations for instance PDS1



number of PCG iterations (in percentage) for instance PDS1 using both RIPM (regularized algorithm with rule $Q = \delta I$ and $\delta = 10^4$) and IPM (nonregularized algorithm). As stated before, the number of PCG iterations for RIPM only increased significantly (i.e., with the same slope that for IPM) in last iterations; the overall number of interior-point iterations of RIPM was also much larger due to the too large regularization considered. Indeed, this example suggests that such large initial regularizations should be avoided in practice.

5.3 Selection of Q

For the selection of a good rule for matrix Q in RIPM, four alternatives were evaluated. The first, the simplest one, is

$$Q = \delta/\mu_0 I, \tag{29}$$

where $\delta \in \mathbb{R}$ is a positive value, and μ_0 is the value of the centrality parameter at the first iterate: since Q is multiplied by μ, this term guarantees that at the first iteration $\mu Q = \delta I$.

The second variant computed the regularization matrix as

$$Q = \delta/\mu_0 X^{(0)}(Z^{(0)})^{-1}, \tag{30}$$

where the diagonals of $X^{(0)}$ and $Z^{(0)}$ are the starting values of x and z. This choice satisfies that, excluding the upper bounds term of (7), $\Theta = (Q+X^{-1}Z)^{-1}$ will be initially well conditioned (since Q is large when $(X^0)^{-1}Z^0$ is small, and vice-versa).

The third and fourth variants are obtained from (29) and (30) by multiplying them by the iteration counter, i.e., they are, respectively,

$$Q^{(t)} = t\delta/\mu_0 I, \tag{31}$$

and

$$Q^{(t)} = t\delta/\mu_0 X^{(0)}(Z^{(0)})^{-1}, \tag{32}$$

t being the number of interior-point iteration. Note that the definition of Q changes with t. These two variants are justified because it was observed that the effect of the regularization term (which is multiplied by μ) could disappear too early when the solution is being reached (i.e., μ approaches zero). For instance, Figure 4 shows for instance PDS1 and $Q = 1/\mu_0 I$ the evolution of μ_t/μ_0, $t\mu_t/\mu_0$ and $t^2\mu_t/\mu_0$, using a log scale for the vertical axis. Compared to μ_t/μ_0, $t\mu_t/\mu_0$ provides a smoother decrement at last iterations, and it does not result in a very large regularization term, mainly at first iterations, unlike $t^2\mu_t/\mu_0$.

Other variants were tried, but are not reported here since they did not improve the nonregularized algorithm in IPM. One of them, based on Proposition 2, consisted on $Q = U^{-2}$, such that the parameter of the regularized barrier would be 1, the best possible one. In practice, it provided poor results, since in many instances this resulted in a negligible regularization.

The four regularization variants (29)–(30) were implemented and applied to a subset of 16 instances of the PDS, Mnetgen and Tripartite suite. The dimensions of these 16 instances are provided in Table 1: columns k, m' and n' provide the number of commodities, nodes, and arcs, respectively; columns

Figure 4: Evolution of μ_t/μ_0, $t\mu_t/\mu_0$ and $t^2\mu_t/\mu_0$ for instance PDS1 and $Q = 1/\mu_0 I$

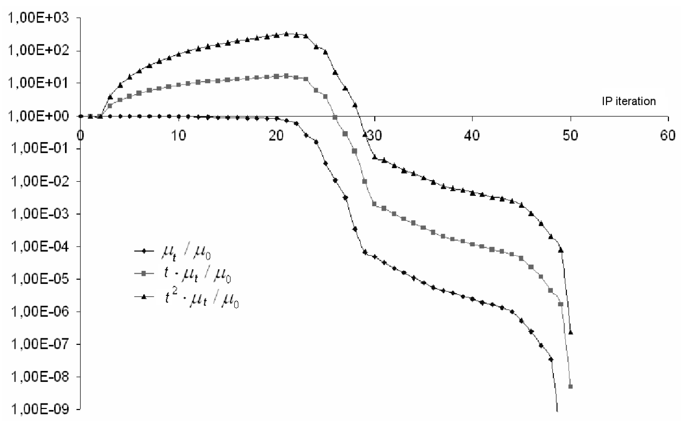


Table 1: Dimensions of the subset of 16 instances

Instance	k	m'	n'	n	m
PDS1	11	126	372	4464	1758
PDS5	11	686	2325	27900	9871
PDS10	11	1399	4792	57504	20181
PDS15	11	2125	7756	93072	31131
PDS20	11	2857	10858	130296	42285
PDS25	11	3554	13580	162960	52674
PDS30	11	4223	16148	193776	62601
32-32-12	32	32	486	16038	1510
64-64-12	64	64	511	33215	4607
128-64-12	64	128	1171	76115	9363
256-64-12	256	64	2320	150190	18030
256-256-12	256	256	2204	566428	67740
tripart1	16	192	2096	35632	5168
tripart2	16	768	8432	143344	20720
tripart3	20	1200	16380	343980	40380
tripart4	35	1050	24815	893340	61565

Table 2: Best results for regularization (29): $Q = \delta/\mu_0 I$

	RIPM					PIPM				
instance	δ	ϵ_0	it.	PCG	CPU	δ	ϵ_0	it.	PCG	CPU
PDS1	10^{-1}	10^{-2}	38	463	0.08	10^{-2}	10^{-2}	39	489	0.08
PDS5	10^{-1}	10^{-2}	59	965	1.54	10^{-1}	10^{-2}	56	863	1.42
PDS10	10^{-2}	10^{-2}	76	1601	7.13	10^{-3}	10^{-2}	77	1536	6.84
PDS15	10^{-2}	10^{-2}	86	2304	19.0	10^{-1}	10^{-2}	89	2481	19.4
PDS20	10^{-1}	10^{-2}	105	4333	49.7	10^{-3}	10^{-2}	107	3877	47.2
PDS25	1	10^{-2}	105	2374	55.6	1	10^{-2}	108	2882	67.7
PDS30	10^{-1}	10^{-2}	113	3050	92.0	10^{-2}	10^{-2}	111	2808	89.6
32-32-12	10^{-1}	10^{-2}	48	2309	0.66	1	10^{-2}	37	1213	0.42
64-64-12	10^{-1}	10^{-2}	63	1445	1.87	10^{-2}	10^{-2}	48	700	1.14
128-64-12	10^{-1}	10^{-2}	70	3793	12.3	10^{-1}	10^{-2}	66	2632	9.19
256-64-12	10^{-2}	10^{-3}	62	3762	34.7	10^{-2}	10^{-3}	62	3964	35.9
256-256-12	10^{-1}	10^{-2}	115	3820	165	10^{-1}	10^{-2}	113	3905	165
tripart1	10^{-3}	10^{-2}	74	3711	2.78	10^{-1}	10^{-2}	63	2682	2.07
tripart2	10^{-1}	10^{-3}	67	2894	11.8	10^{-1}	10^{-2}	72	2368	10.9
tripart3	10^{-1}	10^{-2}	97	5233	47.2	10^{-2}	10^{-2}	83	5490	48.0
tripart4	1	10^{-2}	123	4381	113	1	10^{-2}	127	8313	178

n and m show the overall number of variables and constraints of the resulting linear problem. The four regularizations were tested for 10 different values of $\delta \in \{10^{-8}, 10^{-7}, \ldots, 1, 10^{1}\}$, and two values for the PCG tolerance $\epsilon_0 \in \{10^{-2}, 10^{-3}\}$. Each resulting combination was also solved with the proximal point regularization barrier problem (13), defining $Q_P = \mu Q$, and computing Q using the four previous regularization variants; that implementation will be denoted by PIPM. This way, RIPM and PIPM are compared under the same conditions, being the only differences the dual feasibility conditions (17) and (18), and the definitions of Θ (19) and (20)—theoretically, there is an important difference: the barrier in RIPM is self-concordant, unlike that of PIPM. This amounts to 2560 executions. Tables 2–5 show respectively for each regularization variant, the best results obtained (i.e., best combination of δ and ϵ_0) for each instance, and for both RIPM and PIPM. Columns "it." and "PCG" report the number of interior-point and overall number of PCG iterations. Columns "CPU" provide the CPU time; all the executions were carried on a Linux SUN Fire V20Z server, credited of 367 Mflops, with two AMD Opteron 2.46GHz processors and 8 GB of RAM (multiprocessor capabilites were not exploited in these runs).

Looking at Tables 2–5 there is not a definitive best approach: neither RIPM nor PIPM always outperformed the other approach; and any of the four regularizations was the most efficient choice for some instance. To have a clearer picture, the results of Tables 2–5 are summarized in Tables 6 and 7. Table 6 shows the CPU time of the best variant for both RIPM and PIPM, and the CPU of the nonregularized algorithm IPM obtained by setting $Q = 0$. The fastest execution is marked in boldface. Last row reports the total time for all the instances. It is clearly shown that the regularization that provided more "fastest executions" is (32); it is also the variant with the minimum total CPU time. The nonregularized variant never provided the best run. Table 7 shows the information of Table 6 in relative performance with respect to the nonregularized code IPM, i.e., CPU time of the regularization variant divided by the

Table 3: Best results for regularization (30): $Q = \delta/\mu_0 X^{(0)}(Z^{(0)})^{-1}$

	RIPM					PIPM				
instance	δ	ϵ_0	it.	PCG	CPU	δ	ϵ_0	it.	PCG	CPU
PDS1	1	10^{-2}	43	506	0.09	1	10^{-2}	41	479	0.09
PDS5	10^{1}	10^{-2}	57	726	1.33	1	10^{-2}	61	962	1.61
PDS10	10^{-2}	10^{-2}	74	1294	6.18	10^{-2}	10^{-2}	73	1481	6.93
PDS15	10^{-1}	10^{-2}	77	1610	14.3	10^{-1}	10^{-2}	89	2326	18.7
PDS20	10^{-2}	10^{-2}	96	2947	38.0	1	10^{-2}	106	4899	56.0
PDS25	10^{1}	10^{-2}	98	1741	44.3	1	10^{-2}	112	3808	73.2
PDS30	10^{-3}	10^{-2}	118	3568	102	1	10^{-2}	118	3852	109
32-32-12	10^{-3}	10^{-2}	39	1179	0.43	1	10^{-2}	40	1523	0.48
64-64-12	10^{-3}	10^{-2}	53	947	1.41	1	10^{-2}	81	5649	4.80
128-64-12	10^{-2}	10^{-2}	60	2304	8.42	1	10^{-2}	68	2968	10.4
256-64-12	1	10^{-2}	86	5149	48.3	1	10^{-2}	96	8886	74.9
256-256-12	1	10^{-2}	113	3774	164	1	10^{-2}	115	3855	167
tripart1	10^{-4}	10^{-2}	53	1646	1.46	1	10^{-2}	71	2440	2.14
tripart2	10^{-2}	10^{-2}	79	3368	13.5	1	10^{-2}	119	11162	36.6
tripart3	10^{-2}	10^{-2}	80	4401	39.6	1	10^{-2}	94	5055	46.5
tripart4	10^{-1}	10^{-2}	131	5835	138	1	10^{-2}	124	7107	153

Table 4: Best results for regularization (31): $Q^{(t)} = t\delta/\mu_0 I$

	RIPM					PIPM				
instance	δ	ϵ_0	it.	PCG	CPU	δ	ϵ_0	it.	PCG	CPU
PDS1	10^{-2}	10^{-2}	43	579	0.11	10^{-2}	10^{-2}	47	563	0.09
PDS5	10^{-2}	10^{-2}	57	911	1.46	10^{-2}	10^{-2}	63	1262	1.82
PDS10	10^{-3}	10^{-2}	78	1784	7.51	10^{-2}	10^{-2}	73	1481	6.58
PDS15	10^{-3}	10^{-2}	84	2494	19.4	10^{-2}	10^{-2}	85	2036	17.0
PDS20	10^{-2}	10^{-2}	106	4260	49.1	10^{-1}	10^{-2}	103	4408	51.8
PDS25	10^{-4}	10^{-2}	114	3929	73.3	10^{-2}	10^{-2}	113	3625	72.2
PDS30	10^{-2}	10^{-2}	120	4275	116	10^{-3}	10^{-2}	115	3235	96.5
32-32-12	10^{-3}	10^{-2}	38	1260	0.42	1	10^{-2}	40	924	0.36
64-64-12	10^{-4}	10^{-2}	54	1173	1.52	10^{-1}	10^{-2}	54	1203	2.11
128-64-12	10^{-2}	10^{-2}	65	2301	9.65	10^{-1}	10^{-2}	64	1736	7.00
256-64-12	1	10^{-2}	85	3108	36.0	10^{-5}	10^{-2}	82	3849	38.5
256-256-12	$5^{(*)}$	10^{-2}	113	2965	140	10	10^{-2}	114	2764	135
tripart1	10^{-1}	10^{-2}	86	1305	1.87	1	10^{-2}	88	1662	2.01
tripart2	10^{-4}	10^{-2}	79	3517	13.9	10^{-2}	10^{-2}	75	3036	12.5
tripart3	10^{-2}	10^{-2}	95	2935	32.6	10^{-3}	10^{-2}	80	4154	38.2
tripart4	10^{-2}	10^{-2}	131	5065	126	10^{-1}	10^{-2}	124	6438	148

$^{(*)}$using $\delta = 10$ check (28) failed, i.e., $x^T Q x / |c^T x| > 10^{-6}$

Table 5: Best results for regularization (32): $Q^{(t)} = t\delta/\mu_0 X^{(0)}(Z^{(0)})^{-1}$

	RIPM					PIPM				
instance	δ	ϵ_0	it.	PCG	CPU	δ	ϵ_0	it.	PCG	CPU
PDS1	10^{-3}	10^{-2}	38	472	0.08	1	10^{-2}	42	413	0.08
PDS5	10^{-4}	10^{-2}	56	927	1.47	1	10^{-2}	54	755	1.31
PDS10	10^{-1}	10^{-2}	77	1777	7.44	10^{-2}	10^{-2}	73	1481	6.58
PDS15	1	10^{-2}	81	1607	14.8	1	10^{-2}	84	1961	16.5
PDS20	1	10^{-2}	96	2364	33.9	1	10^{-2}	97	3036	38.6
PDS25	1	10^{-2}	94	1633	42.4	1	10^{-2}	100	2116	49.0
PDS30	1	10^{-2}	99	1667	64.5	1	10^{-2}	121	4388	116
32-32-12	1	10^{-3}	35	1165	0.38	10^{-2}	10^{-2}	41	1137	0.40
64-64-12	10	10^{-3}	45	1235	1.45	10^{-1}	10^{-2}	54	1203	1.54
128-64-12	10	10^{-3}	51	1833	6.79	1	10^{-2}	65	2557	9.06
256-64-12	10	10^{-3}	59	2112	22.8	1	10^{-2}	86	4071	39.8
256-256-12	10	10^{-3}	98	3772	154	1	10^{-2}	110	3354	148
tripart1	1	10^{-3}	142	3844	3.7	1	10^{-2}	86	1907	2.01
tripart2	1	10^{-2}	80	2121	10.5	1	10^{-2}	125	5371	21.3
tripart3	1	10^{-2}	114	1755	28.0	1	10^{-2}	115	7857	66.0
tripart4	0.01	10^{-2}	127	6222	146	1	10^{-2}	149	8191	176

Table 6: Comparison of regularizations: CPU time (seconds)

	Reg. (29)		Reg. (30)		Reg. (31)		Reg. (32)		No Reg.
instance	RIPM	PIPM	RIPM	PIPM	RIPM	PIPM	RIPM	PIPM	IPM
PDS1	**0.08**	**0.08**	0.09	0.09	0.11	0.09	**0.08**	**0.08**	0.09
PDS5	1.54	1.42	1.33	1.61	1.46	1.82	1.47	**1.31**	1.66
PDS10	7.13	6.84	**6.18**	7.92	7.51	6.58	7.44	6.58	7.25
PDS15	19	19.4	**14.3**	19.9	19.4	17	14.8	16.5	21.9
PDS20	49.7	47.2	38	56	49.1	51.8	**33.9**	38.6	56.5
PDS25	55.6	67.7	44.3	73.2	73.3	72.2	**42.4**	49	74.6
PDS30	92	89.6	102	109	116	96.5	**64.5**	116	111
32-32-12	0.66	0.42	0.43	0.48	0.42	**0.36**	0.38	0.4	0.44
64-64-12	1.87	**1.14**	1.41	4.8	1.52	2.11	1.45	1.54	1.49
128-64-12	12.3	9.19	8.42	10.4	9.65	7	**6.79**	9.06	13.2
256-64-12	59.6	47.1	35.9	74.9	36	38.5	**22.8**	39.8	62.4
256-256-12	165	165	164	167	158	**135**	154	148	203
tripart1	2.78	2.07	**1.46**	2.14	1.87	1.01	3.7	2.01	1.7
tripart2	15.5	10.9	13.5	36.6	13.9	12.5	**10.5**	21.3	17.3
tripart3	47.2	48	39.6	46.5	32.6	38.2	**28**	66	62.4
tripart4	**113**	178	138	153	126	148	146	176	265
Sum	643	694	609	764	647	629	**538**	692	900

Reg. (29): $Q = \delta/\mu_0 I$
Reg. (30): $Q = \delta/\mu_0 X^{(0)}(Z^{(0)})^{-1}$
Reg. (31): $Q^{(t)} = t\delta/\mu_0 I$
Reg. (32): $Q^{(t)} = t\delta/\mu_0 X^{(0)}(Z^{(0)})^{-1}$
No Reg.: $Q = 0$

Table 7: Comparison of regularizations: relative performance with respect to IPM

	Reg. (29)		Reg. (30)		Reg. (31)		Reg. (32)	
instance	RIPM	PIPM	RIPM	PIPM	RIPM	PIPM	RIPM	PIPM
PDS1	**0.89**	**0.89**	1	1	1.22	1	**0.89**	**0.89**
PDS5	0.93	0.86	0.8	0.97	0.88	1.1	0.89	**0.79**
PDS10	0.98	0.94	**0.85**	1.09	1.04	0.91	1.03	0.91
PDS15	0.87	0.89	**0.65**	0.91	0.89	0.78	0.68	0.75
PDS20	0.88	0.84	0.67	0.99	0.87	0.92	**0.60**	0.68
PDS25	0.75	0.91	0.59	0.98	0.98	0.97	**0.57**	0.66
PDS30	0.83	0.81	0.92	0.98	1.05	0.87	**0.58**	1.05
32-32-12	1.5	0.95	0.98	1.09	0.95	**0.82**	0.86	0.91
64-64-12	1.26	**0.77**	0.95	3.22	1.02	1.42	0.97	1.03
128-64-12	0.93	0.7	0.64	0.79	0.73	0.53	**0.51**	0.69
256-64-12	0.96	0.75	0.58	1.2	0.58	0.62	**0.37**	0.64
256-256-12	0.81	0.81	0.81	0.82	0.78	**0.67**	0.76	0.73
tripart1	1.64	1.22	**0.86**	1.26	1.1	0.59	2.18	1.18
tripart2	0.9	0.63	0.78	2.12	0.8	0.72	**0.61**	1.23
tripart3	0.76	0.77	0.63	0.75	0.52	0.61	**0.45**	1.06
tripart4	**0.43**	0.67	0.52	0.58	0.48	0.56	0.55	0.66
average	0.96	0.84	**0.76**	1.17	0.87	0.82	0.78	0.87

Reg. (29): $Q = \delta/\mu_0 I$
Reg. (30): $Q = \delta/\mu_0 X^{(0)}(Z^{(0)})^{-1}$
Reg. (31): $Q^{(t)} = t\delta/\mu_0 I$
Reg. (32): $Q^{(t)} = t\delta/\mu_0 X^{(0)}(Z^{(0)})^{-1}$

CPU time of IPM (last column of Table 6 would result in a column of 1's and it is removed from Table 7). The best runs are also marked in boldface, and the average relative performance is provided in last row. Unlike for Table 6, the variant that provides the best results is (30), but closely followed by regularization (32). Since (32) was the best regularization in Table 6 and was very close to the best one in Table 7, it was chosen as the default option in RIPM for the computational results of Section 6.

This section is concluded by noting that, from previous tables, and for these multicommodity instances, the quadratic self-concordant regularization in RIPM was a bit more efficient than the proximal point one in PIPM; and both of them outperformed the nonregularized algorithm. This is shown in Figure 5, which plots the CPU time of IPM, RIPM and PIPM (for the above selected variant, i.e., it plots the last three columns of Table 6) by the number of variables of the instances, dividing them into two groups: "smaller" (left plot) and bigger instances (right plot). Note that RIPM is more efficient for larger problems. In general, however, there is not a significant difference between the two regularized approaches, and the results may be different by tuning some of the parameters. For instance, looking at the number of PCG iterations required for RIPM and PIPM for the particular instance PDS5, for different δ values ranging from 0 to 5000, the results of Figure 6 are obtained. The (unexpected) oscillatory behaviour of RIPM and PIPM shows there is not a definitive better approach, though it is worth noting that the minimum and maximum number of PCG iterations are achieved by RIPM and PIPM, respectively.

Figure 5: CPU time for RIPM, PIPM and IPM. Left plot: smaller instances. Right plot: bigger instances

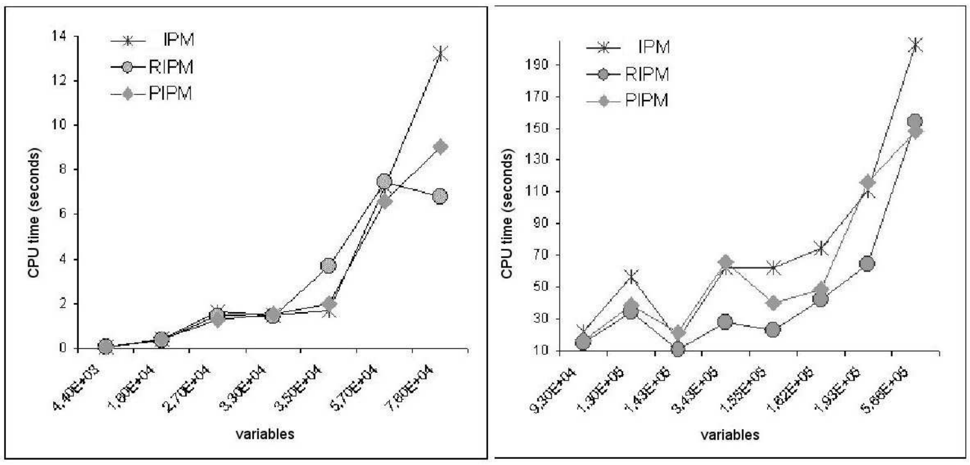


Figure 6: PCG iterations of RIPM and PIPM for different δ, in problem PDS5

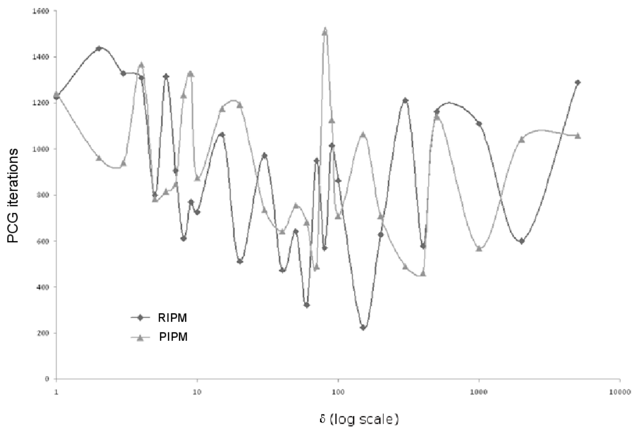


Table 8: Results for PDS instances with IPM and RIPM

Problem dimensions						IPM			RIPM		
Instance	k	m'	n'	n	m	it.	PCG	CPU	it.	PCG	CPU
PDS10	11	4792	1399	53526	16192	80	3830	12.2	86	2964	**10.9**
PDS20	11	10858	2857	121137	33115	106	11011	100	100	3667	**44.9**
PDS30	11	16148	4223	180027	48841	126	9717	204	114	4287	**114**
PDS40	11	22059	5652	245848	65360	135	12700	469	135	9261	**367**
PDS50	11	27668	7031	308281	81263	135	13603	718	138	8707	**518**
PDS60	11	33388	8423	371945	97319	135	19264	1350	147	15577	**1170**
PDS70	11	38369	9750	427663	12546	150	18526	1830	158$^{(*)}$	9998	**1230**
PDS80	11	42472	10989	472863	126539	152	19701	2340	162	10360	**1570**
PDS90	11	46161	12186	513635	139899	149	18644	2560	171	13207	**2180**

$^{(*)}$ check (28) failed, i.e., $x^TQx/|c^Tx| >= 1.5071 > 10^{-6}$

6 Computational results

From the empirical analysis of Section 5, for the computational results we have considered RIPM with the regularized version (32), and $\epsilon_0 = 10^{-2}$ as default options. The parameter δ was set to 1 for the Mnetgen and Tripartite/Gridgen instances, while it was 0.1 for the PDS ones. These default settings have been used for all the runs of this Section, unless otherwise stated for some few executions (due to numerical issues associated to PCG). Note that tuning those parameters it is possible to obtain better results (as in Section 5). However, the purpose of this Section is to show that RIPM with default values may be an efficient interior-point approach for some multicommodity flows, much more than the nonregularized algorithm. As in Section 5, executions were performed on a Linux SUN Fire V20Z server with two AMD Opteron 2.46GHz processors and 8 GB of RAM (without exploiting multiprocessor capabilites).

Tables 8, 9 and 10 provide the results obtained for some PDS, Mnetgen and Tripartite/Gridgen instances, described in Subsection 5.1. The meaning of the columns is the same than in previous tables. The fastest execution for each instance is marked in boldface. For the PDS and Tripartite/Gridgen problems RIPM was always more efficient than IPM. The efficiency is more notorious in the Tripartite/Gridgen instances, when for the larger problems RIPM was more than twice faster. For the Mnetgen problems RIPM was more efficient in all the cases but four. In some instances the benefit added by the regularization term to the solution of systems with PCG is instrumental: in the largest Mnetgen instance 512-512-12, IPM required an average number of 67 PCG iterations per interior point iteration, while RIPM only needed 29.

It is known than interior-point methods are not the best approach for PDS and Mnetgen instances. For this reason Tables 11 and 12 show the results obtained with CPLEX-11 for, respectively, the PDS and Mnetgen instances. Results are provided for all the CPLEX-11 options: "primal" simplex, "dual" simplex, "hybrid" (network simplex followed by dual), and "barrier" (interior-point). The fastest execution is marked in boldface. For the PDS instances not only the "dual", the fastest CPLEX-11 option, outperformed RIPM, but also the generic "barrier" did. This can be explained by the highly efficient ordering and factorization routines in CPLEX-11. For the Mnetgen instances, the "dual"

Table 9: Results for Mnetgen instances with IPM and RIPM

Problem dimensions						IPM			RIPM		
Instance	k	m'	n'	n	m	it.	PCG	CPU	it.	PCG	CPU
128-64-10	64	128	1182	76566	9046	70	4953	14.6	69	4441	**13.4**
128-64-11	64	128	1201	77786	9050	76	5481	16.5	64	3020	**10.3**
128-128-12	128	128	1204	155044	17188	97	3839	26.8	90	2779	**21.2**
256-64-10	64	256	2336	151293	18109	90	5770	54.3	85$^{(*)}$	5204	**50.4**
256-64-11	64	256	2334	151154	18098	83	4748	**46.3**	73	6467	59.9
256-64-12	64	256	2320	150190	18030	132	87698	620	61	3018	**29.5**
512-128-12	128	512	4786	616189	68989	117	6164	396	113	4105	**344**
512-256-12	256	512	4810	1234949	134405	139	6945	883	139	6433	**829**
512-512-12	512	512	4786	2454022	265222	179	12074	2760	163$^{(*)}$	4782	**1560**

$^{(*)}$ $\delta = 2$

Table 10: Results for Tripartite/Gridgen instances with IPM and RIPM

Problem dimensions						IPM			RIPM		
Instance	k	m'	n'	n	m	it.	PCG	CPU	it.	PCG	CPU
tripart1	16	192	2096	35632	5168	58	1976	1.7	74	587	**1.26**
tripart2	16	768	8432	143344	20720	87	4092	17.3	142	2299	**14.7**
tripart3	20	1200	16380	343980	40380	90	6978	62.4	146	3236	**42.3**
tripart4	35	1050	24815	893340	61565	133	14660	265	151	2405	**96.8**
gridgen1	340	1025	3072	986112	331072	242	96877	7400	241	31280	**2420**

simplex is also the fastest CPLEX-11 option. However, in those problems the "barrier" solver is significantly slower than RIPM (an academic code with standard factorization routines), specially for the larger instances, which exceeded a time limit of 3000 seconds. On the other hand, the Tripartite/Gridgen instances are known to be difficult instances for simplex-like methods, and interior-point algorithms outperform them. This is clearly seen in Table 13 which shows the results with CPLEX-11 on these instances. The results for the simplex are those of the fastest variant (either "primal" or "dual"), which is reported in column "solver". It can be seen that the "barrier" solver was by far the most efficient CPLEX-11 approach. However, RIPM (and also IPM) was significantly faster than CPLEX-11 "barrier". As far as we know, up to now IPM was the most efficient algorithm for these difficult instances [8]; this no longer holds, since RIPM is a more efficient approach.

As stated above, the results of Tables 8, 9 and 10 were obtained with default options, since this was the purpose of this Section. However, we note they are not the best results that can be obtained with RIPM; specially for the larger instances, there is room for improvement by tuning the parameters. For instance, problem PDS60 could be solved in 147 iterations, 5954 PCG iterations and 610 seconds (instead of the 1170 seconds of Table 8); problem PDS90 could be solved in 140 iterations, 6018 PCG iterations and 1280 seconds (instead of the 2180 seconds of Table 8); and problem "gridgen1" was solved in 219 iterations, 5703 PCG iterations and 618 seconds (significantly reducing the 2520 seconds of Table 10, and making it even more competitive against general solvers like

Table 11: Results for PDS instances with CPLEX-11

	primal		hybrid		dual		barrier	
Instance	it.	CPU	it.	CPU	it.	CPU	it.	CPU
PDS10	8409	0.94	9176	1.69	5116	**0.88**	33	7.07
PDS20	78508	15.77	29798	14.30	17830	**7.37**	39	30.76
PDS30	298932	105.38	49264	34.09	29336	**17.27**	38	123.76
PDS40	451783	331.52	78014	80.95	49160	**46.25**	38	191.87
PDS50	218455	339.26	95327	130.32	62983	**62.34**	38	297.66
PDS60	700718	1138.4	122125	172.54	73610	**76.50**	40	485.10
PDS70	764918	957.4	154221	251.69	97271	**99.46**	38	645.24
PDS80	828482	1144.8	197605	281.48	117281	**138.36**	40	719.63
PDS90	864758	1378.3	197754	336.34	118797	**139.14**	40	814.56

Table 12: Results for Mnetgen instances with CPLEX-11

	primal		hybrid		dual		barrier	
Instance	it.	CPU	it.	CPU	it.	CPU	it.	CPU
128-64-10	242539	421.2	33383	148.96	17686	**12.98**	18	55.92
128-64-11	272249	534.0	36518	95.03	19295	**11.72**	18	57.97
128-128-12	275263	519.5	73941	308.4	39987	**33.83**	21	145.6
256-64-10	534984	2823.6	90611	1220.14	36462	**40.05**	18	360.34
256-64-11	589623	2821.9	78142	535.97	35246	**31.27**	17	362.11
256-64-12	—	>3000	68890	208.64	36193	**33.96**	11	302.49
512-128-12	—	>3000	—	>3000	98856	**88.27**	—	>3000
512-256-12	—	>3000	—	>3000	181380	**157.3**	—	>3000
512-512-12	—	>3000	—	>3000	342622	**399.0**	—	>3000

Table 13: Results for Tripartite/Gridgen instances with CPLEX-11

	barrier		simplex		
instance	it.	CPU	it.	CPU	solver
tripart1	21	3.99	4197	**1.12**	dual
tripart2	25	**36.01**	58316	106.97	dual
tripart3	28	**138.8**	96592	382.47	dual
tripart4	29	**1323.2**	165668	1638.12	dual
gridgen1	64	**12288**	—	>15000	any

CPLEX-11).

7 Conclusions

From the results of this work, it is clear that the new regularized version outperforms the specialized interior-point method for multicommodity flows implemented in IPM. This means that linear multicommodity flow problems are more efficiently solved by specialized interior-point methods based on PCG, if they are dealt with as a sequence of quadratic multicommodity flow problems. However, for some standard classes of multicommodity flow problems, as the PDS and Mnetgen ones, dual simplex algorithms are still more efficient than the new regularized approach. However, for some classes of difficult multicommodity problems, interior-point methods outperform simplex variants; for those instances, the regularized specialized interior-point algorithm could be considered one of the most efficient available approaches. The automatic tuning of parameters of the algorithm for particular instances, and the application of the regularized algorithm to nonlinear convex separable multicommodity flow problems is part of the future research to be done.

References

[1] A. Ali and J.L. Kennington, Mnetgen Program Documentation. Technical Report 77003, Department of Industrial Engineering and Operations Research, Southern Methodist University, Dallas, 1977.

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

[3] F. Babonneau and J.-P. Vial, ACCPM with a nonlinear constraint and an active set strategy to solve nonlinear multicommodity flow problems, *Math. Prog.* 120 (2009), 179-210.

[4] F. Babonneau, O. du Merle and J.-P. Vial, Solving large-scale linear multicommodity flow problems with an active set strategy and proximal-ACCPM, *Oper. Res.* 54 (2006), 184–197.

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

[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, *Oper. Res.* 38 (1990), 240–248.

[7] J. Castro, A specialized interior-point algorithm for multicommodity network flows, *SIAM J. Optim.* 10 (2000), 852–877.

[8] J. Castro, Solving difficult multicommodity problems through a specialized interior-point algorithm, *Ann. Oper. Res.* 124 (2003), 35–48.

[9] J. Castro, “Solving quadratic multicommodity problems through an interior-point algorithm”, System Modelling and Optimization XX, E.W. Sachs and R. Tichatschke (Editors), Kluwer, Boston, 2003, pp. 199–212.

[10] J. Castro, An interior-point approach for primal block-angular problems, *Comput. Optim. Appl.* 36 (2007), 195–219.

[11] J. Castro and J. Cuesta, Quadratic regularizations in an interior-point method for primal block-angular problems, *Math. Prog.* (2009), conditionally accepted. Also available as Research Report DR2008-07, Dept. of Statistics and Operations Research, Universitat Politècnica de Catalunya, 2008.

[12] J. Castro and N. Nabona, An implementation of linear and nonlinear multicommodity network flows, *Eur. J. Oper. Res.* 92 (1996), 37–53.

[13] P. Chardaire and A. Lisser, Simplex and interior point specialized algorithms for solving nonoriented multicommodity flow problems, *Oper. Res.* 50 (2002), 260–276.

[14] A. Frangioni and G. Gallo, A bundle type dual-ascent approach to linear multicommodity min cost flow problems, *INFORMS J. Comp.* 11 (1999), 370–393.

[15] J.-L. Goffin, J. Gondzio, R. Sarkissian and J.-P. Vial, Solving nonlinear multicommodity flow problems by the analytic center cutting plane method, *Math. Prog.* 76 (1996), 131–154.

[16] G.H. Golub and C.F. Van Loan, Matrix Computations, Third Ed., Johns Hopkins Univ. Press, Baltimore, 1996.

[17] C. Lemaréchal, A. Ouorou and G. Petrou, A bundle-type algorithm for routing in telecommunication data networks, *Comput. Optim. Appl.* in press. DOI 10.1007/s10589-007-9160-7.

[18] R.D. McBride, Progress made in solving the multicommodity flow problem, *SIAM J. Optim.* 8 (1998), 947–955.

[19] Y. Nesterov, Introductory Lectures on Convex Optimization: A Basic Course, Kluwer, Boston, 2004.

[20] E. Ng and B.W. Peyton, Block sparse Cholesky algorithms on advanced uniprocessor computers, *SIAM J. Sci. Comput.* 14 (1993), 1034–1056.

[21] A. Ouorou, Implementing a proximal point algorithm to some nonlinear multicommodity flow problems, *Networks* 18 (2007), 18–27.

[22] A. Ouorou, P. Mahey and J.-P. Vial, A survey of algorithms for convex multicommodity flow problems, *Manag. Sci.*, 46 (2000), 126–147.

[23] R. Setiono, Interior proximal point algorithm for linear programs, *J. Optim. Theory Appl.* 74 (1992), 425–444.

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