SIAM J. OPTIM.
Vol. 10, No. 3, pp. 852–877

A SPECIALIZED INTERIOR-POINT ALGORITHM FOR MULTICOMMODITY NETWORK FLOWS*

JORDI CASTRO†

Abstract. Despite the efficiency shown by interior-point methods in large-scale linear programming, they usually perform poorly when applied to multicommodity flow problems. The new specialized interior-point algorithm presented here overcomes this drawback. This specialization uses both a preconditioned conjugate gradient solver and a sparse Cholesky factorization to solve a linear system of equations at each iteration of the algorithm. The ad hoc preconditioner developed by exploiting the structure of the problem is instrumental in ensuring the efficiency of the method. An implementation of the algorithm is compared to state-of-the-art packages for multicommodity flows. The computational experiments were carried out using an extensive set of test problems, with sizes of up to 700,000 variables and 150,000 constraints. The results show the effectiveness of the algorithm.

Key words. interior-point methods, linear programming, multicommodity flows, network programming

AMS subject classifications. 90C05, 90C06, 90C35

PII. S1052623498341879

1. Introduction.

Multicommodity problems usually have many variables and constraints, which makes it difficult for them to be solved by general procedures. This has led to the formulation of specialized methods. However, some of the largest and most difficult multicommodity problems are still challenging even for these specializations. The algorithm presented in this paper has three main features. First, it has proven to be computationally efficient and robust in the solution of a wide range of problems, not just for some specific kind of multicommodity instances. Second, it is a specialized primal-dual interior-point algorithm, so it globally converges to the optimum in polynomial time, unlike other methods that provide an ϵ-approximate solution (e.g., [17]). And finally, it has been able to efficiently solve large instances of Patient Distribution System (PDS) problems [7]. This class of problems is commonly used as a de facto standard for testing the performance of multicommodity codes. With our algorithm we solved the PDS90 and PDS100 instances in a relatively reasonable amount of time. (In [17] an approximate solution is provided at most for PDS80.)

Most of the specialized methods attempt to exploit in some way the block structure of the multicommodity problem. Among the earlier approaches, we find primal partitioning and the price and resource directive decompositions (see [2, Chap. 17] and [21] for details). Of these three methods, the first two were regarded as the most successful in [3]. Despite this, no implementation of primal partitioning has been able to solve large problems significantly faster than the state-of-the-art simplex codes. For instance, the recent primal partitioning package PPRN [6] was, on average, no more than an order of magnitude faster than the primal simplex code of MINOS 5.3. In some cases, accurate implementations of the dual simplex—preceded by a warm

*Received by the editors July 13, 1998; accepted for publication (in revised form) October 6, 1999; published electronically May 2, 2000. This work was partially supported by Iberdrola S.A. grant 95-005 and by CICYT project TAP96-1044-J02-93.

http://www.siam.org/journals/siopt/10-3/34187.html

†Statistics and Operations Research Department, Universitat Politècnica de Catalunya, Campus Sud, Pau Gargallo 5, 08028 Barcelona, Spain (jcastro@eio.upc.es).

start based on solving minimum-cost network problems for each commodity—can even outperform primal partitioning multicommodity specializations (see [13] for a comparison of PPRN and the network+dual solver of CPLEX 3.0). In this paper, we show that our algorithm, in general, outperforms both PPRN and CPLEX 4.0 and, in some cases, by more than an order of magnitude.

The other method regarded as successful in [3], the price directive or Dantzig–Wolfe decomposition, belongs to the class of cost decomposition approaches for multicommodity flows (see [13], [15], and [33] for recent variants based on bundle methods, analytic centers, and smooth penalty functions, respectively). A recent computational study [13] showed that these are promising approaches for solving a wide variety of problems. However, for some classes of instances—typically difficult problems with large networks and not many commodities such as the PDS ones—our interior-point approach seems to give considerably better performance. Furthermore, Frangioni [12] noted that the particular cost decomposition method of [13] might sometimes require the algorithmic parameters to be tuned if performances are to be the best possible, whereas our algorithm works well with default values.

Interior-point methods have also been applied in the past. For the single-commodity case, efficient specializations were developed by Resende and others [26, 28, 29, 30]. These specializations relied on the use of preconditioned conjugate gradient (PCG) solvers. The preconditioners developed, though efficient, were appropriate only for single-commodity problems. The first reported attempt at solving multicommodity problems by an interior-point method was probably that described in [1]. However, the general implementation of Karmarkar's projective algorithm used there was outperformed by a simplex specialized algorithm in the solution of small-size multicommodity instances. Alternative and more efficient approaches were developed in the following years. In fact, the best complexity bound known for multicommodity problems is provided by the two interior-point algorithms described in [19] and [20], though none of these papers provided computational results. In [18], Kamath et al. applied a variant of Karmarkar's projective algorithm using a PCG solver. However, their preconditioner did not take advantage of the multicommodity structure. An attempt to exploit this structure was made in [9] by Choi and Goldfarb. Though the decomposition scheme they presented is similar to the one in this paper, the solution procedure differs substantially. Choi and Goldfarb suggest solving a fairly dense matrix positive definite linear system that appears during the decomposition stage by means of parallel and vector processing, whereas we apply a PCG method, which enables large problems to be solved efficiently using a midsize workstation. A different interior-point approach was developed in [31], using a barrier function to decompose the problem. This strategy provided approximate solutions for some of the large PDS problems (up to PDS70). However, as will be shown, our method gives more accurate solutions. Finally, Portugal et al. introduced in [27] a specialized interior-point algorithm based solely on a PCG, unlike our method, which combines PCG with direct factorizations. The proposed preconditioner was an extension of that developed in [26] by the same authors for single-commodity flows. No computational results were reported in [27] for the solution of multicommodity problems using this preconditioner.

This paper is organized as follows. Section 2 presents the formulation of the problem to be solved. Section 3 outlines the primal-dual algorithm, and in section 4 we develop the specialization for multicommodity problems. Section 5 describes some implementation details of this specialization. Finally, section 6 gives computational results that show the efficiency of the algorithm.

2. Problem formulation. Let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ be a directed graph, where $\mathcal{N}$ is a set of $m+1$ nodes and $\mathcal{A}$ is a set of n arcs, and let $\mathcal{K}$ be a set of k commodities to be routed through the network represented by $\mathcal{G}$. We shall also consider that the arcs of the network have a capacity for all the commodities, which will be known as the mutual capacity. So, the multicommodity network flow (MCNF) problem can be formulated as follows:

$$\min_{x^{(1)},\ldots,x^{(k)}} \quad \sum_{i=1}^{k} {c^{(i)}}^T x^{(i)} \tag{1}$$

$$\text{subject to } A_N x^{(i)} = b^{(i)}, \qquad i = 1, \ldots, k, \tag{2}$$

$$\sum_{i=1}^{k} x^{(i)} \leq b_{mc}, \tag{3}$$

$$0 \leq x^{(i)} \leq \overline{x}^{(i)}, \qquad i = 1, \ldots, k. \tag{4}$$

Vectors $x^{(i)} \in \mathbb{R}^n$ and $c^{(i)} \in \mathbb{R}^n$ are the flow and cost arrays for each commodity i, $i = 1, \ldots, k$. $A_N \in \mathbb{R}^{m \times n}$ is the node-arc incidence matrix, where each column is related to an arc $a \in \mathcal{A}$, and has only nonzero coefficients in those rows associated with the origin and destination nodes of a (with coefficients 1 and -1, respectively). We shall assume that A_N is a full row-rank matrix. This can always be guaranteed by removing any of the (redundant) node balance constraints. $b^{(i)} \in \mathbb{R}^m$ is the vector of supplies and demands for commodity i at the nodes of the network. Equation (3) represents the mutual capacity constraints, where $b_{mc} \in \mathbb{R}^n$. Constraints (4) are simple bounds on the flows, $\overline{x}^{(i)} \in \mathbb{R}^n$, $i = 1, \ldots, k$, being the upper bounds. These upper bounds represent individual capacities of the arcs for each commodity.

Introducing the slacks s_{mc} for the mutual capacity constraints, (3) can be rewritten as

$$\sum_{i=1}^{k} x^{(i)} + s_{mc} = b_{mc}. \tag{5}$$

We can consider that the slacks s_{mc} are upper bounded by b_{mc}, since all the vectors $x^{(i)}$ in (5) have nonzero components. This gives

$$0 \leq s_{mc} \leq b_{mc}. \tag{6}$$

The MCNF problem can then be recast as

$$\text{min (1) subject to (2), (4), (5), and (6).} \tag{7}$$

3. Outline of the primal-dual interior-point algorithm. Let us consider the linear programming problem

$$\begin{aligned} \min \quad & c^T x \\ \text{subject to} \quad & Ax = b, \\ & \overline{x} \geq x \geq 0, \end{aligned} \tag{8}$$

where $x \in \mathbb{R}^{\tilde{n}}$, $\overline{x} \in \mathbb{R}^{\tilde{n}}$ are the upper bounds, $c \in \mathbb{R}^{\tilde{n}}$, $b \in \mathbb{R}^{\tilde{m}}$, and $A \in \mathbb{R}^{\tilde{m} \times \tilde{n}}$ is a full row-rank matrix. The dual of (8) is

$$\begin{aligned} \max \quad & b^T y - \overline{x}^T w \\ \text{subject to} \quad & A^T y + z - w = c, \\ & z \geq 0, \quad w \geq 0, \end{aligned} \tag{9}$$

where $y \in \mathbb{R}^{\tilde{m}}$ are the dual variables and $z \in \mathbb{R}^{\tilde{n}}$ and $w \in \mathbb{R}^{\tilde{n}}$ are the dual slacks. Note that the MCNF problem, as defined in (7), fits the formulation of (8), where $\tilde{n} = (k+1)n$ and $\tilde{m} = km + n$.

Replacing the inequalities in (8) by a logarithmic barrier in the objective function, with parameter μ, and considering the slacks $f = \overline{x} - x$, it can be seen that the KKT first order optimality conditions of (8) and (9) are equivalent to the following system of nonlinear equations (see [32] for a comprehensive description):

$$\begin{aligned} b_{xz} &\equiv \mu e_{\tilde{n}} - XZe_{\tilde{n}} = 0, \\ b_{fw} &\equiv \mu e_{\tilde{n}} - FWe_{\tilde{n}} = 0, \\ b_b &\equiv b - Ax = 0, \\ b_c &\equiv c - (A^T y + z - w) = 0, \\ (x, z, w) &\geq 0, \quad \overline{x} \geq x, \end{aligned} \tag{10}$$

where $e_{\tilde{n}}$ is the $\tilde{n}$-dimensional vector of 1's; X, Z, F, and W are diagonal matrices defined as $M \in \mathbb{R}^{\tilde{n}\times\tilde{n}} = \text{diag}(m_1, \ldots, m_{\tilde{n}})$; and the vectors b_* define the left-hand-side terms of (10). Note that we did not include the slacks equation $x + f = \overline{x}$ in (10). Instead we replaced the slacks f by $\overline{x} - x$, reducing by $\tilde{n}$ the number of equations and variables. This forces the primal variables x of the iterates obtained during the solution of (10) to always be interior in relation to their upper bounds.

The solutions of system (10)—considering inequalities as strict inequalities—for different μ values gives rise to an arc of strictly feasible points known as the central path. As μ tends to 0, the solutions of (10) converge to those of the original primal and dual problems. A path-following algorithm attempts to follow the central path, computing (10)—in long-step methods—through a damped Newton's method together with the reduction of the barrier parameter μ at each iteration of the algorithm. The path-following algorithm considered for the specialization uses the reduction formula $\mu = 0.1(x^T z + f^T w)/2\tilde{n}$. It can be seen [32] that obtaining Newton's direction amounts to finding dy and then computing dx, dw, dz, in

$$\begin{aligned} (A\Theta A^T)dy &= b_b + A\Theta r, \\ dx &= \Theta(A^T dy - r), \\ dw &= F^{-1}(b_{fw} + W dx), \\ dz &= b_c + dw - A^T dy, \end{aligned} \tag{11}$$

where

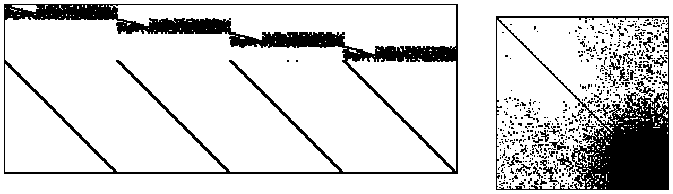
$$r = F^{-1}b_{fw} + b_c - X^{-1}b_{xz}, \quad r \in \mathbb{R}^{\tilde{n}}, \tag{12}$$

$$\Theta = FX(ZF + XW)^{-1}, \quad \Theta \in \mathbb{R}^{\tilde{n}\times\tilde{n}}. \tag{13}$$

Note that Θ is a positive definite diagonal matrix, since it is nothing but a product of positive definite diagonal matrices. Since A is a full row-rank matrix, $A\Theta A^T$ is also positive definite. It is quite clear that the main computational burden of the algorithm is the repeated solution of the linear system

$$(A\Theta A^T)dy = \bar{b}, \tag{14}$$

where $\bar{b}$ denotes $b_b + A\Theta r$ in (11). The performance of any primal-dual multicommodity specialization relies on the efficient solution of (14).



(a) (b)

FIG. 1. (a) *Sparsity pattern of a multicommodity constraint matrix A.* (b) *Sparsity pattern of the factorization of* $PA\Theta A^T P^T$.

4. Primal-dual specialization for the MCNF problem.

4.1. Motivation. General interior-point codes for linear programming attempt to solve (14) through sparse Cholesky factorizations. To reduce the fill-in, they factorize $PA\Theta A^T P^T$, instead of $A\Theta A^T$, where P is a permutation matrix obtained by some heuristic. However, when applied to multicommodity problems, even the best P matrices, such as those provided by good heuristics like the minimum-local-fill-in or minimum-degree orderings, cannot prevent a fairly large dense submatrix from appearing in $LL^T = PA\Theta A^T P^T$. For instance, Figure 1(a) shows the sparsity pattern of the constraint matrix A for a multicommodity problem with 4 commodities, 64 nodes, and 524 arcs (it corresponds to problem M_1 in Table 3 of section 6). Using the state-of-the-art interior-point code BPMPD [23], the sparsity pattern obtained for $L + L^T$ is depicted in Figure 1(b). The dense submatrix created makes the factorization of $PA\Theta A^T P^T$ computationally expensive and, for large problems, its storage in the memory completely prohibitive. Then it is clear that, to be competitive, interior-point methods must exploit the structure of the multicommodity problem to efficiently solve (14).

4.2. Exploiting the multicommodity structure. The constraint matrix A of the MCNF problem defined in (7) has the following structure:

$$A = \begin{array}{|c|c|c|c|c|} \hline A_N & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & A_N & \dots & \mathbf{0} & \mathbf{0} \\ \hline \vdots & \vdots & \ddots & \vdots & \vdots \\ \hline \mathbf{0} & \mathbf{0} & \dots & A_N & \mathbf{0} \\ \hline \mathbb{1}_n & \mathbb{1}_n & \dots & \mathbb{1}_n & \mathbb{1}_n \\ \hline \end{array}, \tag{15}$$

where $\mathbb{1}_n$ denotes the $n \times n$ identity matrix and $\mathbf{0}$ is the zero matrix. Moreover, matrix Θ, as defined in (13), can be partitioned as

$$\Theta = \begin{array}{|c|} \hline \Theta^{(1)} \\ \hline \end{array} \quad \ddots \quad \begin{array}{|c|} \hline \Theta^{(k)} \\ \hline \end{array} \quad \begin{array}{|c|} \hline \Theta_{mc} \\ \hline \end{array}, \tag{16}$$

where $\Theta^{(i)} \in \mathbb{R}^{n\times n}$, $i = 1, \ldots, k$, and $\Theta_{mc} \in \mathbb{R}^{n\times n}$ are associated with the flows $x^{(i)}$ of commodity i and the mutual capacity slacks s_{mc}, respectively. Using (15) and (16), it is straightforward to see that $A\Theta A^T$ in (14) has the following structure:

$$A\Theta A^T = \begin{array}{|c|c|c|c|} \hline A_N\Theta^{(1)}A_N^T & \ldots & \mathbf{0} & A_N\Theta^{(1)} \\ \hline \vdots & \ddots & \vdots & \vdots \\ \hline \mathbf{0} & \ldots & A_N\Theta^{(k)}A_N^T & A_N\Theta^{(k)} \\ \hline \Theta^{(1)}A_N^T & \ldots & \Theta^{(k)}A_N^T & \Theta_{mc} + \sum_{i=1}^k \Theta^{(i)} \\ \hline \end{array} = \begin{array}{|c|c|} \hline B & C \\ \hline C^T & D \\ \hline \end{array}, \tag{17}$$

where $B \in \mathbb{R}^{km\times km}$ is the block diagonal matrix

$$B = \operatorname{diag}(A_N\Theta^{(i)}A_N^T, \quad i = 1, \ldots, k), \tag{18}$$

each block being a square matrix of dimension m, $C \in \mathbb{R}^{km\times n}$ is defined as

$$C = \begin{bmatrix} \Theta^{(1)}A_N^T & \ldots & \Theta^{(k)}A_N^T \end{bmatrix}^T, \tag{19}$$

and $D \in \mathbb{R}^{n\times n}$ corresponds to the lower diagonal submatrix of $A\Theta A^T$:

$$D = \Theta_{mc} + \sum_{i=1}^{k} \Theta^{(i)}. \tag{20}$$

Since Θ is diagonal and positive definite, it holds that D is a positive definite diagonal matrix as well.

The above decomposition of $A\Theta A^T$ can be applied to the solution of (14), partitioning appropriately the dual variables direction dy and the right-hand-side vector $\bar{b}$:

$$\begin{array}{|c|c|} \hline B & C \\ \hline C^T & D \\ \hline \end{array} \quad \begin{array}{|c|} \hline dy_1 \\ \hline dy_2 \\ \hline \end{array} = \begin{array}{|c|} \hline \bar{b}_1 \\ \hline \bar{b}_2 \\ \hline \end{array}, \tag{21}$$

where dy_1, $\bar{b}_1 \in \mathbb{R}^{km}$ and dy_2, $\bar{b}_2 \in \mathbb{R}^n$. The solution of (21) can be directly obtained by block multiplication, yielding

$$(D - C^TB^{-1}C)dy_2 = (\bar{b}_2 - C^TB^{-1}\bar{b}_1), \tag{22}$$

$$Bdy_1 = (\bar{b}_1 - Cdy_2). \tag{23}$$

Matrix $D - C^TB^{-1}C$ is known as the Schur complement, and it will be denoted by S:

$$S = D - C^TB^{-1}C. \tag{24}$$

To efficiently solve (22) and (23)—and obtain the solution to (14)—we only need to deal with systems involving matrices B and S. Systems with matrix B can be considered not too difficult. In fact, exploiting the block structure of B shown in (18), these systems can be decomposed into k smaller ones of dimension m with matrices $A_N\Theta^{(i)}A_N^T$, $i = 1, \ldots, k$. Each of these matrices can be easily obtained. If we denote by $\mathcal{I}_v$ the set of arcs incident to node v and by $a \equiv (v, w)$ the arc of $\mathcal{A}$ that has v and w as origin and destination nodes and consider the structure of the

node-arc incidence matrix A_N, it is straightforward to see that $A_N\Theta^{(i)}A_N^T$ can be easily computed as follows:

$$(25)\ (A_N\Theta^{(i)}A_N^T)_{\substack{vw\\ v=1,\dots,m\\ w=1,\dots,m}} = \begin{cases} \displaystyle\sum_{\forall a} -\Theta_a^{(i)} & \text{if } a \equiv (v,w) \in \mathcal{A}, (w,v) \notin \mathcal{A}, \\ \displaystyle\sum_{\forall a,b} \left(-\Theta_a^{(i)} - \Theta_b^{(i)}\right) & \text{if } a \equiv (v,w) \in \mathcal{A}, b \equiv (w,v) \in \mathcal{A}, \\ \displaystyle\sum_{\forall a \in I_v} \Theta_a^{(i)} & \text{if } (v = w), \\ 0 & \text{otherwise,} \end{cases}$$

where $\Theta_a^{(i)}$ is the diagonal term of $\Theta^{(i)}$ associated to arc a. Moreover, since $\Theta^{(i)}$ is symmetric and positive definite and A_N is a full row-rank network matrix, we have that matrices $A_N\Theta^{(i)}A_N^T$ are symmetric and positive definite as well. Therefore, their Cholesky factorizations exist. In practice, to reduce the fill-in, instead of $A_N\Theta^{(i)}A_N^T$, we shall factorize $P_NA_N\Theta^{(i)}A_N^TP_N^T$, where P_N is a permutation matrix of the nodes of the network. Note that P_N will have to be computed only once, since the nonzero pattern of $A_N\Theta^{(i)}A_N^T$ is the same for all the commodities. In general, due to the high sparsity of the network matrix A_N, we can expect that these k Cholesky factorizations—and, hence, the factorization of B—will not be too computationally expensive. Additionally, in a parallel computing environment, these k factorizations—and their respective backward and forward substitutions—can be carried out independently for each commodity.

System (22) still remains to be solved. We could consider computing and factorizing S. However, this would mean solving n systems of equations with matrix B, n being the number of arcs of the network. In addition, S could become fairly dense. In fact, as the proposition below shows, if we perform symbolic computations, matrix S turns out to be completely dense, increasing the solution cost of (22) with a direct method. With no loss of generality and to simplify the notation we will consider a problem with only one commodity and where $P_N = \mathbb{1}$ (no node permutation is required to reduce the fill-in for $A_N\Theta^{(i)}A_N^T$).

PROPOSITION 1. *Let $L^{(1)}L^{(1)^T} = A_N\Theta^{(1)}A_N^T$ be the Cholesky factorization of $B = A_N\Theta^{(1)}A_N^T$. If we apply this factorization to remove the subdiagonal elements of B and submatrix C in* (17) *by symbolic Gaussian elimination*

$$\begin{array}{|c|c|} \hline L^{(1)^{-1}} & \mathbf{0} \\ \hline -C^TB^{-1} & \mathbb{1} \\ \hline \end{array} \quad \begin{array}{|c|c|} \hline B & C \\ \hline C^T & D \\ \hline \end{array} = \begin{array}{|c|c|} \hline L^{(1)^T} & L^{(1)^{-1}}C \\ \hline \mathbf{0} & D - C^TB^{-1}C \\ \hline \end{array},$$

submatrix $D - C^TB^{-1}C$—the Schur complement—becomes completely dense.

Proof. Let $\mathcal{N}_v$ be the set of nodes adjacent to node $v \in \mathcal{N}$, i.e.,

$$\mathcal{N}_v = \{w \in \mathcal{N} \text{ such that } (v,w) \in \mathcal{A} \text{ or } (w,v) \in \mathcal{A}\};$$

this set will be associated to matrix $B = A_N\Theta^{(1)}A_N^T$. Let $\mathcal{I}_v$ be the set of arcs incident to node v, i.e.,

$$\mathcal{I}_v = \{a \in \mathcal{A} \text{ such that } a \equiv (w,v) \in \mathcal{A} \text{ or } a \equiv (v,w) \in \mathcal{A}\};$$

this set will be associated to matrix $C = A_N\Theta^{(1)}$. And let $\mathcal{C}_a$ be the set of nodes connected to arc $a \in \mathcal{A}$ (initially $\mathcal{C}_a = \{v, w\}$, where $a \equiv (v, w)$); this set will be associated to matrix $C^T = \Theta^{(1)}A_N^T$. Moreover we will denote as $M^{j)}/\mathcal{M}^{j)}$ the matrix/set $M/\mathcal{M}$ after j elimination stages—the original sets and matrices correspond to $M^{0)}/\mathcal{M}^{0)}$—and by v_i and a_j the node and arc associated to row i and column j of A_N, respectively.

Let us assume we are starting the Gaussian elimination and we have to remove the subdiagonal terms of the first column of (17). This will be done through the first row of $B^{0)}$ (which corresponds to the first node v_1 of A_N). From (25) it can be seen that we shall have to remove the elements of the rows of $B^{0)}$ related to the nodes in $\mathcal{N}_{v_1}^{0)}$. Two new nonzero elements will then appear, one in the upper and the other in the lower diagonal parts of $B^{1)}$, for each pair of nodes (v_i, v_j) in $\mathcal{N}_{v_1}^{0)}$ not yet connected by any arc. The adjacent node sets are suitably updated as

$$\text{for all } v_i \in \mathcal{N}_{v_1}^{0)} \quad \mathcal{N}_{v_i}^{1)} = \mathcal{N}_{v_i}^{0)} \bigcup \mathcal{N}_{v_1}^{0)} - \{v_1\}.$$

(A comprehensive explanation of this result can be found in [14, Chap. 5].) New nonzero elements will appear in matrix $C^{1)}$ as well. Initially, the only nonzero elements in row i of $C^{0)}$ are found in the columns of the arcs $\mathcal{I}_{v_i}^{0)}$. After the first elimination stage we find that

$$\text{for all } v_i \in \mathcal{N}_{v_1}^{0)} \quad \mathcal{I}_{v_i}^{1)} = \mathcal{I}_{v_i}^{0)} \bigcup \mathcal{I}_{v_1}^{0)}.$$

Similarly, when eliminating the first column of $C^{T0)}$, new nonzero elements will appear in $C^{T1)}$. Unlike $C^{1)}$, these new entries are related to arcs, thus having

$$\text{for all } a_j \in \mathcal{I}_{v_1}^{0)} \quad \mathcal{C}_{a_j}^{1)} = \mathcal{C}_{a_j}^{0)} \bigcup \mathcal{N}_{v_1}^{0)}.$$

Repeating the above procedure, it is not difficult to see that, after $m - 1$ elimination stages, all the nodes collapsed into the last one v_m (see [14, Chap. 5] again for a detailed description), yielding

$$\mathcal{N}_{v_m}^{m-1)} = \bigcup_{i=1}^{m} \mathcal{N}_{v_i} = \mathcal{V}.$$

It also holds, for the last row in matrix $C^{m-1)}$, that

$$\mathcal{I}_{v_m}^{m-1)} = \bigcup_{i=1}^{m} \mathcal{I}_{v_i} = \mathcal{A}$$

and, analogously for the last column in matrix $C^{Tm-1)}$, that

$$\text{for all } a_j \in \mathcal{A} \quad v_m \in \mathcal{C}_{a_j}^{m-1)}.$$

Therefore, the last row in $C^{m-1)}$ and the last column in $C^{Tm-1)}$ become dense. It is now clear that, if we attempt to eliminate the last column of $C^{Tm-1)}$ from the last row in the $C^{m-1)}$ matrix, $D - C^T B^{-1} C$ becomes dense. □

In practice, however, if we perform numerical instead of symbolic computations, S will not be completely dense due to cancellations. As shown by the next proposition,

the numerical sparsity pattern of S depends on the structure of the network and, in the simplest case, can even be diagonal.

PROPOSITION 2. *If the network is a spanning tree (thus, it is connected and $m = n + 1$), the Schur complement is diagonal.*

Proof. In this case, matrix A_N is square and nonsingular. Using (18), (19), (20), and the nonsingularity of A_N, the Schur complement can be written as

$$S = D - C^T B^{-1} = \Theta_{mc} + \sum_{i=1}^{k} \Theta^{(i)} - \sum_{i=1}^{k} \Theta^{(i)} A_N^T (A_N \Theta^{(i)} A_N^T)^{-1} A_N \Theta^{(i)}$$
$$= \Theta_{mc} + \sum_{i=1}^{k} \Theta^{(i)} - \sum_{i=1}^{k} \Theta^{(i)} = \Theta_{mc}. \qquad \square$$

The density of S increases with the complexity of the network. If each pair of nodes of the network is connected by at least one arc, S can be shown to be numerically completely dense for most Θ matrices. Leaving aside these extreme cases, for general networks the Schur complement will be numerically fairly dense. This fact, together with the cost associated with building matrix S, makes the solution of (22) with a direct method prohibitive. A similar system had to be solved in the approach suggested in [9]. However, no procedure was given there to circumvent this difficulty, and the solution of (22) was addressed through parallel and vector processing. Rather than use a direct method, the specialization we propose attempts to solve (22) through a PCG.

4.3. Solution via a preconditioned conjugate gradient method. Before applying a PCG method to (22) we must guarantee that S is symmetric and positive definite at each iteration of the algorithm.

LEMMA 1. *Let $T \in \mathbb{R}^{t \times t}$ be a square matrix partitioned as follows:*

$$T = \begin{array}{|c|c|} \hline B & C \\ \hline C^T & D \\ \hline \end{array}.$$

Then, if T is symmetric and positive definite and B is positive definite, it holds that the Schur complement $S = D - C^T B^{-1} C$ is symmetric and positive definite.

PROPOSITION 3. *The Schur complement matrix $S = D - C^T B^{-1} C$ defined in (22) is symmetric and positive definite at each iteration of the primal-dual algorithm.*

Proof. The primal and dual variables—x, and z and w—are interior at each iteration of the primal-dual algorithm. So we have that Θ, as defined in (13), is a positive definite diagonal matrix. Moreover, since the network matrix A_N was assumed to be a full row-rank matrix, the constraint matrix of the MCNF problem defined in (15) is a full row-rank matrix as well. Therefore, matrices $A^T \Theta A$ and B defined in (17) and (18) are both symmetric and positive definite. Applying Lemma 1, with $T = A^T \Theta A$, we get that S is symmetric and positive definite. $\square$

The preconditioner that we propose in this paper, denoted by M, consists of using an approximation of the inverse of S. The development of this preconditioner relies on the following theorem.

THEOREM 1 (P-regular splitting theorem). *If R is symmetric positive definite and $R = P - Q$ is a P-regular splitting—i.e., P is nonsingular and $P + Q$ is positive definite—then $\rho(P^{-1}Q) < 1$ (where $\rho(T)$ denotes the spectral radius of T).*

Proof. See [25, pp. 254–255]. □

PROPOSITION 4. *The inverse of* $S = D - C^T B^{-1} C$ *can be computed as*

$$S^{-1} = \left(\sum_{i=0}^{\infty} (P^{-1}Q)^i \right) P^{-1}, \tag{26}$$

where

$$P = D, \qquad Q = C^T B^{-1} C. \tag{27}$$

Proof. Premultiplying S by S^{-1} as defined in (26) we get

$$\begin{aligned} S^{-1}S &= \left(\left(\sum_{i=0}^{\infty} (P^{-1}Q)^i \right) P^{-1} \right) (P - Q) \\ &= \sum_{i=0}^{\infty} (P^{-1}Q)^i - \sum_{i=1}^{\infty} (P^{-1}Q)^i. \end{aligned} \tag{28}$$

Since $P = D$ is a diagonal positive definite matrix, it is nonsingular. $P + Q = D + C^T B^{-1} C$ is positive definite as well because both D and B are positive definite. Thus, $P - Q$ is a regular splitting of S. Moreover, S is symmetric and positive definite, as stated by Proposition 3. By Theorem 1, we have that $\rho(P^{-1}Q) < 1$, and then the geometric power series of (28) converge, obtaining the desired result:

$$S^{-1}S = (P^{-1}Q)^0 + \sum_{i=1}^{\infty} (P^{-1}Q)^i - \sum_{i=1}^{\infty} (P^{-1}Q)^i = \mathbb{1}. \qquad \square$$

The preconditioner is then obtained by truncating the infinite geometric power series (26) at some term $\phi \geq 0$, which will be referred to as the order of the preconditioner:

$$M^{-1} = (\mathbb{1} + (P^{-1}Q) + (P^{-1}Q)^2 + \cdots + (P^{-1}Q)^{\phi}) P^{-1}, \tag{29}$$

where P and Q are defined in (27). Note that M is an adequate preconditioner for the PCG, since it is symmetric and positive definite. (This can be easily proved by showing that, from the symmetry and positive definiteness of both P and Q, M^{-1} is symmetric and positive definite as well.) The main drawback of the preconditioner is that matrices P and Q both become ill-conditioned as the iterates approach a solution, which can lead to values $\rho(P^{-1}Q)$ very close to 1; consequently, (29) would be a poor approximation of S^{-1}. Despite this, the preconditioner has shown to be an efficient solution strategy, being able to significantly reduce the number of iterations required by nonpreconditioned conjugate gradient (CG) methods. For instance, Figure 2 shows the evolution of $\rho(P^{-1}Q)$ for M_1 and PDS1, the smallest problems in Tables 3 and 4 in section 6. Both problems required 30 interior-point iterations. It can be seen that, though it tends to decrease for the central iterations, $\rho(P^{-1}Q)$ is close to 1 throughout the execution of the algorithm (especially for problem PDS1). As shown below (next two paragraphs, and Figures 4 and 5), even in these situations the goodness of the preconditioner increases with ϕ, and we obtain a better performance than with a nonpreconditioned CG method.

Clearly, the higher ϕ is, the better the preconditioning and the fewer iterations of the PCG will be required. However, at each iteration of the PCG, we have to solve

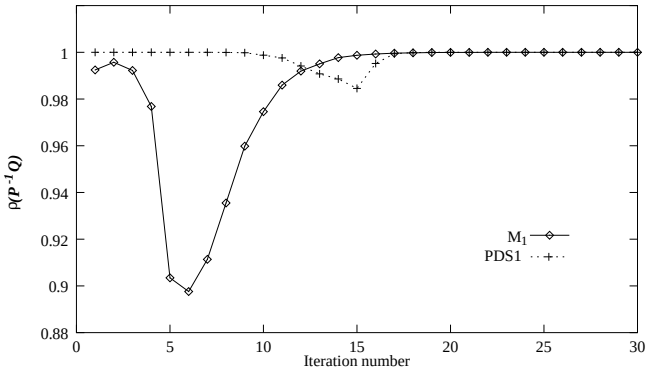


FIG. 2. *Evolution of $\rho(P^{-1}Q)$ for the M_1 and PDS1 problems.*

Procedure MZ=r (P, Q, r, z, ϕ)
$v := P^{-1}r$
$z_0 := v$
for j=1 to ϕ do
    $z_j := P^{-1}Qz_{j-1} + v$
end_do
$z := z_\phi$

FIG. 3. *Procedure for computing $z = M^{-1}r$.*

the system $Mz = r$, with r being any vector. This system can be easily computed through the procedure presented in Figure 3, which involves solving ϕ systems with matrix B, and thus a total of $k\phi$ systems with matrices $A_N^T\Theta^{(i)}A_N$, $i = 1, \ldots, k$. Then ϕ must be chosen to balance two objectives: to reduce both the number of PCG iterations and the number of systems to be solved. In practice, performances are best for $\phi = 0$ and, in some cases, for $\phi = 1$. For instance, Figure 4 shows the evolution of the CPU time and overall number of PCG iterations required to solve problem M_1 in Table 3 of section 6 for different ϕ values. Clearly, there are fewer PCG iterations when ϕ increases, but the performance tends to be poorer. This is the usual behavior observed in most problems tested. The algorithm uses $\phi = 0$ as the default value, though this parameter can be modified by the user. All the numerical results in section 6 were obtained with this default value. Note that when $\phi = 0$ the preconditioner is nothing but $M = P = D$, the diagonal matrix defined in (20). In this case the computation of $Mz = r$ is reduced to n products.

Despite its simplicity, the diagonal preconditioner obtained for $\phi = 0$ has proven to be very efficient compared to a nonpreconditioned CG method. For instance, Figure 5 shows the number of overall CG iterations required to solve the first 10 PDS problems in Table 4 in section 6, by both the PCG with $\phi = 0$ and a standard CG method. The number of interior-point iterations was almost the same in both types of executions. However, it is clear from the figure than the CG required many more CG iterations to achieve the same accuracy in the solution of (22). For the 10 problems, the code with the PCG was, on average, 3.7 times faster than that with the CG and performed 7.5 times fewer CG iterations.

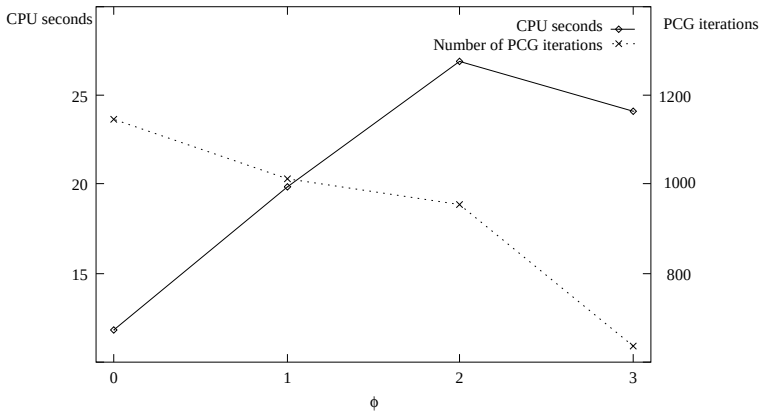


FIG. 4. *CPU time and overall number of PCG iterations for different ϕ values.*

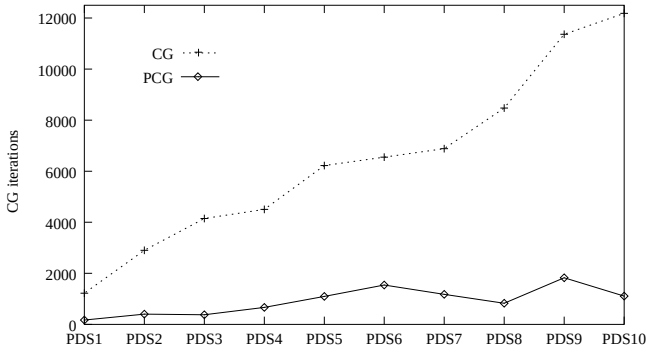


FIG. 5. *Overall number of CG iterations for the PCG with $\phi = 0$ and a CG method.*

Three remarks must be made about this solution strategy.

(i) Though designed for multicommodity instances, it can be applied to other block-structured problems where a similar decomposition to that of (21) is possible. We mention three of them. First, a direct extension of the MCNF problem consisting of replacing the mutual capacity constraints by the more general ones

$$\sum_{i=1}^{k} W^{(i)} x^{(i)} \leq b_{mc},$$

where $W^{(i)}$ is a diagonal matrix of positive weights. Second, the nonoriented multicommodity problem—arcs have no orientation—which commonly appears, for instance, in telecommunication networks [8]. And finally, multicommodity problems with convex separable quadratic objective functions (e.g., $\sum_{i=1}^{k} \sum_{a \in \mathcal{A}} c_a^{(i)} (x_a^{(i)})^2$, $c_a^{(i)} \geq 0$), which only imply a slight modification of the Θ diagonal matrix. Note that simplex-based specializations for multicommodity flows cannot deal with this last class of problems.

(ii) At each iteration of the PCG, $\phi + 1$ systems of equations with matrix B must be solved for computing $Mz = r$ and $q = Sp(z)$, $p(z)$ being a vector that depends on z. Since the k blocks of B have already been factorized, only the forward and backward substitutions must be performed. The solutions to these k systems, however, can

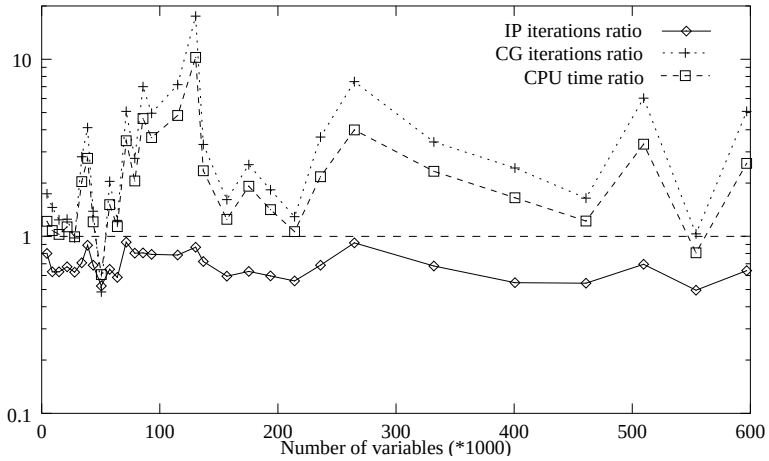


FIG. 6. *Predictor-corrector vs. pure path-following, for the PDS problems.*

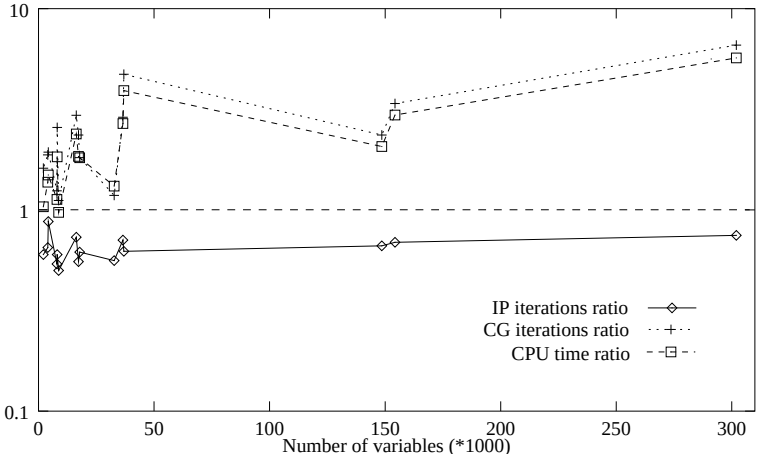


FIG. 7. *Predictor-corrector vs. pure path-following, for the Mnetgen problems.*

be efficiently parallelized since each of them requires the same computational effort (load balancing). We can expect that a coarse-grain parallel implementation of the algorithm would significantly reduce the solution time (at most by a factor of k).

(iii) The solution to (22) using a PCG algorithm forces us to use a pure primal-dual path-following algorithm instead of other more successful approaches, such as Mehrotra's predictor-corrector method. The predictor-corrector method requires two solutions to (21) with different right-hand sides. In our specialization, the benefit obtained by computing this better direction is not worthwhile, since it means applying the PCG method twice. Figures 6 and 7 compare a version of the algorithm using Mehrotra's predictor-corrector with a pure path-following algorithm for the PDS and Mnetgen problems in Tables 3 and 4 in section 6. The predictor-corrector strategy was implemented as described in [22] and [32]. The direction computed by the predictor step was used as the starting point for the PCG of the corrector step in an attempt to reduce the number of CG iterations. Figures 6 and 7 show the ratio of the number of interior-point iterations, CG iterations, and execution time between the predictor-

corrector and pure path-following algorithms. The dashed horizontal line separates the executions according to whether the ratio was favorable to the predictor-corrector (region below the line) or the pure path-following algorithm (region above the line). Clearly, the predictor-corrector heuristic reduced the number of iterations (on average, it performed 1.45 and 1.55 times fewer iterations for the PDS and Mnetgen problems, respectively). However, the overall number of CG iterations significantly increased (with average ratios of 3.5 and 2.6 for each class of problems). The execution time ratio, highly correlated with the CG iteration ratio, was favorable to the pure path-following algorithm for most of the problems. Of the larger instances executed, the predictor-corrector strategy was only better for PDS90. The larger Mnetgen instances were not executed with the predictor-corrector method due to their large execution times (e.g., problem M_{18} was stopped after 21 hours of execution, whereas the pure path-following algorithm required only 2.55 hours). On average, the pure path-following algorithm was 2.3 times faster than the predictor-corrector strategy for the PDS problems and 2.2 times faster for the Mnetgen ones.

5. Implementation details. We have developed an implementation of the algorithm presented in section 4 that will be referred to as IPM. This code is mainly written in C, with only the Cholesky factorization routines coded in Fortran. It can be freely obtained for academic purposes from http://www-eio.upc.es/~jcastro, at software entry. Below, we discuss some of the implementation aspects of IPM that have proved to be instrumental in the performance of the algorithm.

5.1. Cholesky factorizations. The factorizations of matrices $A_N^T \Theta^{(i)} A_N$, $i = 1, \ldots, k$, and, mainly, their backward and forward substitutions at each iteration of the PCG solver are the most computationally expensive steps of the algorithm. Note that the symbolic factorization must be performed only once, since matrices $A_N^T \Theta^{(i)} A_N$ have the same nonzero pattern for all the commodities. IPM uses the sparse Cholesky package by E. Ng and B. Peyton [24]. For large networks these routines provided significantly better solution times than alternative ones like the Sparspak package [14]—although both share the same minimum-degree-ordering heuristic for computing the permutation of the nodes of the network. Note that, unlike general interior-point methods, the main computational burden is not the Cholesky factorizations but the repeated forward and backward substitutions. Indeed, in practice, and due to the (large) number of PCG iterations, these substitutions represent about 60% of the execution time, whereas the factorizations amount to no more than 20%. (For instance, in problems M_5, M_{11}, PDS10, and PDS30 of Tables 3 and 4 of section 6, these figures were 45.3/0.4, 47.4/4.5, 48.8/2.8, and 65.7/18.7, respectively.) Since the Ng–Peyton package concentrates its effort on the factorization stage, it may be possible to improve the performance of the algorithm by either using or developing a Cholesky solver focused on the solution phase.

5.2. Accuracy of the PCG method. The tolerance of the stopping criteria of the PCG is the most influential parameter in the overall performance of the algorithm. It determines the accuracy required to solve system (22) and, hence, the number of PCG iterations performed. We followed a similar approach to that used by Resende and Veiga in [29] for single-commodity network problems. At iteration i of the interior-point method, we consider that the jth PCG iterate dy_2^j solves (22) if

$$1 - \cos(S dy_2^j, \bar{b}_2 - C^T B^{-1} \bar{b}_1) < \epsilon_i, \tag{30}$$

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

$$\epsilon_i = 0.95 \epsilon_{i-1}, \tag{31}$$

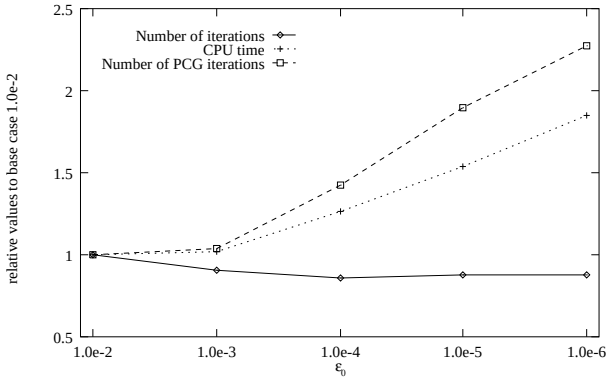


FIG. 8. *CPU time, PCG iterations, and primal-dual iterations for different* ϵ_0 *(problem* M_4*).*

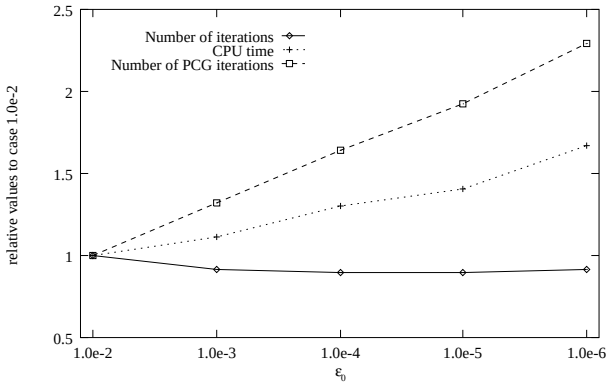


FIG. 9. *CPU time, PCG iterations, and primal-dual iterations for different* ϵ_0 *(problem PDS*5*).*

which guarantees better dy_2 directions as we get closer to the solution. By default IPM uses an initial tolerance of $\epsilon_0 = 10^{-2}$. Smaller ϵ_0 values provide better movement directions, which reduce the sequence of primal-dual points but considerably increase the number of PCG iterations. On the other hand, if large ϵ_0 are used, the primal-dual algorithm can fail to converge. The default value of 10^{-2} was good enough to solve most of the problems tested (only a few required $\epsilon_0 = 10^{-3}$) and provided the best execution times. For instance, Figures 8 and 9 show the CPU time and the number of PCG and primal-dual iterations required to solve problems M_4 and PDS5 of Tables 3 and 4 of section 6, respectively, for different ϵ_0 values (all data are relative to the base case $\epsilon_0 = 10^{-2}$). Though both problems are very different (M_4 has a much smaller network but three times more commodities), the behavior of IPM was almost the same: for small ϵ_0 values the CPU time and PCG iterations increased significantly whereas the primal-dual iterations hardly decreased.

In our computational experience, we have seen that the ϵ_0 value slightly affects the precision of the optimizer provided. In general, IPM stops with a point where the dual infeasibilities, computed as

$$\frac{\|A^T y + z - w - c\|_2}{1 + \|c\|_2},$$

are about 10^{-6}, regardless of whether ϵ_0 was chosen. In most tests performed, the primal infeasibilities

$$\frac{\|Ax-b\|_2}{1+\|b\|_2}$$

were about 10^{-5} for $\epsilon_0 = 10^{-2}$ and 10^{-6} for $\epsilon_0 = 10^{-6}$. The gain of this digit in the primal accuracy was at the expense of approximately doubling the solution time. IPM stops when the relative duality gap of the current iterate

$$\frac{|c^T x - (b^T y - \overline{x}^T w)|}{1+|c^T x|} \tag{32}$$

is less than an optimality tolerance, by default set to 10^{-6}. Unlike general interior-point solvers, and due to the use of a PCG, it is difficult to obtain more accurate solutions. One possible way of overcoming this drawback would be to develop a procedure for detecting the optimal face once we are close to the optimizer. Such a strategy already exists for network linear programs [30], but, to the best of our knowledge, there is not an equivalent result for multicommodity flows. In connection to this, the inclusion of a crossover procedure is also part of further work to be done on IPM.

5.3. Removing inactive mutual capacity constraints. The dimension of the Schur complement S is the number of mutual capacity constraints n. Should we have a procedure for detecting the inactive constraints, these could be removed, thus reducing the computational effort required by (22). IPM implements two kinds of strategy for the detection of inactive constraints. The first one is applied at the beginning, as a preprocessing stage. The second follows the suggestions in [16] and consists of detecting the inactive mutual capacity constraints during the execution of the algorithm, using the complementarity condition $y_j s_{mc_j} = 0$, where y_j and s_{mc_j} denote the dual variable and primal slack of the jth mutual capacity constraint. At iteration i of the primal-dual algorithm, we will remove the mutual capacity constraint of arc a_j if

$$y_j^i \approx 0 \quad \text{and} \quad s_{mc_j}^i \gg 0.$$

IPM implements these conditions as $|y_j^i| < 0.01$ and $s_{mc_j}^i > 0.1 b_{mc_j}$. This removal is only active when the relative duality gap (32) is less than 1.0 (primal and dual functions agree in one figure), in an attempt to guarantee that the current iterate is sufficiently close to the optimizer. Note that, unlike general interior-point solvers, removing mutual capacity constraints does not imply any additional symbolic refactorization.

5.4. Starting point. Following the suggestions in [5], an initial estimate of the primal variables is computed by solving

$$\begin{aligned} \min \quad & c^T x + \frac{\rho}{2}\,(x^T x + (\overline{x}-x)^T(\overline{x}-x)) \\ \text{subject to} \quad & Ax = b, \end{aligned}$$

with $|\rho| = 100$, yielding

$$\lambda = (AA^T)^{-1}\left(\frac{b}{2\rho} + A\left(c - \frac{\overline{x}}{\rho}\right)\right),$$

$$x = \frac{1}{2}\left(\overline{x} + \frac{A^T\lambda - c}{\rho}\right).$$

Thereafter, the components x_i out of bounds are replaced by $\min\{\overline{x}_i/2, 100\}$.

Dual estimators are obtained from the dual feasibility and complementarity slackness conditions

$$\begin{aligned}(A^T y)_i + z_i - w_i &= c_i,\\ x_i z_i &= \mu_0,\\ (\overline{x}_i - x_i)w_i &= \mu_0,\end{aligned} \tag{33}$$

where μ_0 is set to a large value (e.g., 100). Dual variables are initialized as $y = 0$. Dual slacks are computed from (33), yielding

$$z_i = \frac{\mu_0}{\overline{x}_i} + \frac{c_i}{2} + \sqrt{\frac{\mu_0^2}{\overline{x}_i^2} + \frac{c_i^2}{4}},$$

$$w_i = \frac{\mu_0 z_i}{\overline{x}_i z_i - \mu_0}.$$

Note that for $\mu_0 > 0$ the above equations provide strictly positive values for w and z.

6. Computational results. To test the performance of the algorithm, IPM has been compared with the NetOpt routine of CPLEX 4.0 [10] (which uses the solution to k minimum-cost network problems, one for each commodity, as a warm start of a dual simplex solver), and with PPRN [6], a primal partitioning code for linear and nonlinear multicommodity flows. For all the three codes, we used the default tolerances. All runs were carried out on a Sun/Ultra2 2200 workstation with 200 MHz clock, 256 Mbytes of main memory, ≈68 Mflops Linpack, 14.7 Specfp95, and 7.8 Specint95.

For the comparison, we considered three kinds of problem. The first one was obtained from the meta-generator Dimacs2pprn (see [6]). This meta-generator requires a previous minimum-cost network flow problem that is converted to a multicommodity one. It can be obtained from ftp://ftp-eio.upc.es/pub/onl/codes/pprn/tests (an enhanced version is described in [13]). We used four minimum-cost network generators from the DIMACS suite [11]: Rmfgen (D. Goldfarb and M. Grigoriadis), Grid-on-Torus (A. V. Goldberg), Gridgraph (M. G. C. Resende), and Gridgen (Y. Lee and J. Orlin). They are freely distributed and can be obtained via anonymous ftp from dimacs.rutgers.edu at directory /pub/netflow. We generated two kinds of problem for each generator: with few commodities (small problems) and with many commodities (large problems). The small problems are represented by S_k^i, where $i = 1, \ldots, 4$ denotes the DIMACS generator used (1 = Rmfgen, 2 = Grid-on-Torus, 3 = Gridgraph, 4 = Gridgen) and $k \in \{1, 4, 8, 16, 50, 100, 150, 200\}$ is the number of commodities considered. The large problems are called L_k^i, where i and k have the same meaning as

TABLE 1
Dimensions and results obtained for the small Dimacs2pprn problems.

Pr.	m	n	k	$\tilde{n}$	$\tilde{m}$	CPU time (seconds) IPM	CPLEX	PPRN	f^*	$\frac{f^*-f^*_{\rm IPM}}{1+f^*}$
S^1_1	2472	9048	1	9048	2472	37.	1.	1.	375675.2	−4.5e−6
S^1_4	2472	9048	4	45240	18936	273.	8.	9.	2027285.0	−5.2e−6
S^1_8	2472	9048	8	81432	28824	1058.	28.	149.	4506263.3	−5.4e−6
S^1_{16}	2472	9048	16	153816	48600	1377.	73.	1166.	9870432.8	−7.8e−6
S^1_{50}	128	496	50	25296	6896	17.	21.	39.	11839382.2	3.9e−5
S^1_{100}	128	496	100	50096	13296	76.	258.	465.	27150952.6	2.9e−5
S^1_{150}	128	496	150	74896	19696	137.	681.	1245.	39835825.1	8.3e−6
S^1_{200}	128	496	200	99696	26096	174.	1204.	2368.	54343948.3	−5.6e−6
S^2_1	1500	9000	1	9000	1500	28.	1.	1.	36896.8	–1.5e–6
S^2_4	1500	9000	4	45000	15000	623.	22.	85.	187962.0	–5.2e–6
S^2_8	1500	9000	8	81000	21000	2499.	647.	814.	1197048.8	4.9e–6
s^2_{16}	1500	9000	16	153000	33000	7550.	12872.	6721.	5876840.3	1.8e–5
S^2_{50}	100	600	50	30600	5600	28.	14.	21.	5207622.7	2.2e–6
S^2_{100}	100	600	100	60600	10600	78.	110.	222.	12922703.9	1.4e–5
S^2_{150}	100	600	150	90600	15600	263.	652.	1137.	22663204.t	–5.9e–6
S^2_{200}	100	600	200	120600	20600	565.	2393.	3495.	36829147.5	1.3e–5
S^3_1	2502	5000	1	5000	2502	2.	1.	1.	94212753.2	–2.5e–6
S^3_4	2502	5000	4	25000	15008	184.	55.	118.	355884986.5	–3.8e–6
S^3_8	2502	5000	8	45000	25016	247.	85.	215.	128743093.7	9.2e–5
S^3_{16}	2502	5000	16	8500	45032	956.	1171.	2666.	253615755.9	8.3e–5
S^3_{50}	227	450	50	22950	11800	60.	21.	56.	27853327.9	6.0e–5
S^3_{100}	227	450	100	45450	23150	173.	290.	670.	65144564.1	6.8e–5
S^3_{150}	227	450	150	67950	34500	104.	144.	745.	27066715.3	4.5e–6
S^3_{200}	227	450	200	90450	45850	247.	550.	1922.	37964963.8	1.6e–5
S^4_1	976	7808	1	7808	976	25.	1.	1.	5541980.3	–5.5e–7
S^4_4	976	7808	4	39040	11712	747.	(a)	41.	23223474.9	–2.6e–6
S^4_8	976	7808	8	70272	15616	4079.	(a)	497.	61792270.7	–4.9e–9
S^4_{16}	976	7808	16	132736	23424	5509.	(a)	6466.	165808232.3	9.1e–5
S^4_{50}	101	606	50	30906	5656	14.	5.	4.	1409470.3	2.2e–5
S^4_{100}	101	606	100	61206	10706	39.	16.	25.	2940217.3	–1.6e–6
S^4_{150}	101	606	150	91506	15756	68.	38.	58.	4614971.4	3.2e–6
S^4_{200}	101	606	200	121806	20806	121.	126.	189.	6440385.6	2.6e–6

(a) Problem reported as infeasible by the solver.

before. (In this case, however, the number of commodities is always greater than 200.) Tables 1 and 2 show the dimensions of these problems. Column Pr. is the name of the problem. Columns m, n, and k show the number of nodes, arcs, and commodities, respectively. Columns $\tilde{n}$ and $\tilde{m}$ give the number of variables and constraints of the linear problem (where $\tilde{n} = (k+1)n$ and $\tilde{m} = km + n$). Columns IPM, CPLEX, and PPRN correspond to the CPU time, in seconds, required by each code to solve the problem. Finally, column f^* gives the optimal objective function value provided by both CPLEX and PPRN, whereas column $\frac{f^*-f^*_{\rm IPM}}{1+f^*}$ shows the relative error of the solution provided by IPM.

TABLE 2
Dimensions and results obtained for the large Dimacs2pprn problems.

Pr.	m	n	k	$\tilde{n}$	$\tilde{m}$	CPU time (seconds) IPM	CPLEX	PPRN	f^*	$\frac{f^*-f^*_{\text{IPM}}}{1+f^*}$
L^1_{200}	128	496	200	99400	26096	50.	49.	254.	43496063.1	–2.5e–6
L^1_{400}	128	496	400	198800	51696	140.	319.	2092.	89227358.5	–5.3e–7
L^1_{600}	128	496	600	298200	77296	242.	1328.	6590.	135813634.7	–1.3e–6
L^1_{800}	128	496	800	397600	102896	278.	3001.	13428.	184848693.7	–1.3e–6
L^1_{1000}	128	496	1000	497000	128496	363.	6006.	25813.	235407084.7	–1.2e–6
L^1_{1200}	128	496	1200	596400	154096	546.	11887.	43946.	287243145.4	2.7e–5
L^1_{1400}	128	496	1400	695800	179696	756.	20080.	78800.	339708251.9	7.7e–6
L^2_{200}	80	500	200	100200	16500	71.	92.	278.	1372096.3	3.1e–6
L^2_{400}	80	500	400	200400	32500	522.	2358.	4647.	7004937.6	4.6e–6
L^2_{500}	80	500	500	250500	40500	905.	5395.	10259.	11941741.3	8.1e–6
L^2_{600}	80	500	600	300600	48500	885.	10778.	20478.	17857546.4	1.6e–5
L^3_{200}	242	472	200	94600	48872	59.	71.	387.	8153455.3	9.5e–6
L^3_{400}	242	472	400	189200	97272	202.	685.	3367.	16715597.6	6.2e–6
L^3_{500}	242	472	500	236500	121472	319.	1968.	8025.	21219420.2	1.9e–6
L^3_{600}	242	472	600	283800	145672	384.	3256.	14614.	25646734.6	1.5e–5
L^4_{200}	151	1208	200	241800	31408	310.	104.	231.	1690360.3	–9.7e–7
L^4_{300}	151	1208	300	362700	46508	537.	365.	893.	2614303.6	–4.3e–6
L^4_{400}	151	1208	400	483600	61608	805.	673.	2195.	3389601.0	7.4e–7

The second kind of problems were obtained with A. Frangioni's [13] C version of Ali and Kennington's Mnetgen generator [4]. It can be freely obtained from http://www.di.unipi.it/di/groups/optimize/Data/MMCF.html. We generated 24 problems with different dimensions. They can all be considered difficult problems, since they have a "dense" network (the ratio "number of arcs/number of nodes" is 8), 80% of the arcs have a mutual capacity, 30% of the arcs have a high cost, and 90% of the arcs have individual capacities for each commodity. The parameters used for generating the instances can be found in [13]. The problems obtained with this generator will be denoted as M_i, $i = 1, \ldots, 24$. Table 3 shows the dimensions of these tests, where the columns have the same meaning as in Tables 1 and 2.

The last type of problems corresponds to the PDS instances [7]. These problems arise from a model for evacuating patients from a place of military conflict. Each instance depends on a parameter t that denotes the planning horizon under study (in number of days). The size of the network increases with t, whereas the number of commodities is always 11. Problems obtained with this generator are denoted as PDSt, where t is the number of days considered. Their dimensions are shown in Table 4. The meaning of the columns is the same as in previous tables. The largest problems were not solved with CPLEX due to the amount of time required. The PDS problems can be retrieved from http://www.di.unipi.it/di/groups/optimize/Data/MMCF.html.

Tables 1–4 show that IPM and PPRN solved all the problems, whereas CPLEX exited with an infeasibility message in three of the small Dimacs2pprn tests and was not run for the largest PDS. IPM solved most of the problems using the default initial PCG tolerance of $\epsilon_0 = 10^{-2}$. Only in three cases (problems S^3_{16}, L^2_{200}, and L^2_{500}) did this value have to be reduced to 10^{-3} to guarantee the convergence. In general

TABLE 3
Dimensions and results obtained for the Mnetgen problems.

Pr.	m	n	k	$\tilde{n}$	$\tilde{m}$	CPU time (seconds) IPM	CPLEX	PPRN	f^*	$\frac{f^*-f^*_{\mathrm{IPM}}}{1+f^*}$
M_1	64	524	4	2100	780	1.	0.	0.	192400.1	–6.3e–7
M_2	64	532	8	4264	1044	2.	1.	1.	394051.1	4.1e–6
M_3	64	497	16	7968	1521	5.	2.	4.	1071474.9	1.0 e–5
M_4	64	509	32	16320	2557	12.	13.	19.	2146944.1	1.0e–5
M_5	64	511	64	32768	4607	91.	141.	136.	4623138.5	8.1e–6
M_6	128	997	4	3992	1509	2.	0.	1.	919643.2	1.5e–6
M_7	128	1089	8	8720	2113	8.	1.	4.	1924133.9	–6.7e–7
M_8	128	1114	16	17840	3162	25.	13.	34.	4145079.4	6.0e–6
M_9	128	1141	32	36544	5237	155.	214.	478.	9785961.1	6.3e–6
M_{10}	128	1171	64	76115	9363	485.	1647.	3419.	19269824.2	–3.9e–6
M_{11}	128	1204	128	154240	17588	549.	7880.	9334.	40143200.8	9.2e–6
M_{12}	256	2023	4	8096	3047	12.	1.	7.	5026132.3	1.4e–5
M_{13}	256	2165	8	17328	4213	40.	13.	69.	9919483.2	–2.1e–6
M_{14}	256	2308	16	36944	6404	146.	158.	769.	20692883.7	6.9e–6
M_{15}	256	2314	32	74080	10506	465.	1664.	7610.	45671076.1	–1.4e–6
M_{16}	256	2320	64	148544	18704	1040.	9235.	27722.	92249381.1	–1.2e–6
M_{17}	256	2358	128	301952	35126	3742.	45990.	84066.	190137259.9	–7.8e–6
M_{18}	256	2204	256	564480	67740	9187.	181701.	169810.	397882591.3	–1.4e–6
M_{19}	512	4077	4	16312	6125	99.	7.	85.	21324851.2	-7.3e–6
M_{20}	512	4373	8	34992	8469	190.	101.	654.	46339269.9	1.6e–5
M_{21}	512	4620	16	73936	12812	1582.	1457.	7279.	96992237.2	–4.5e–6
M_{22}	512	4646	32	148704	21030	2644.	8302.	73439.	192941834.8	-7.0e–7
M_{23}	512	4768	64	305216	37536	7411.	55028.	178188.	412943158.7	8.9e–8
M_{24}	512	4786	128	612736	70322	21263.	289541.	947790.	828013599.8	–1.3e–6

IPM required no more than 100 iterations to achieve a point with a dual relative gap less than 10^{-6}—the default optimality tolerance. We can also see that the solution provided by IPM can be considered good enough: the relative error in the objective function (last column of Tables 1–4) fluctuates between 10^{-5} and 10^{-7} (the worst case corresponds to problem PDS40, with a relative error of $1.5 \cdot 10^{-4}$). These results are more accurate (two more exact figures in the objective function) than those provided in [17] and [31] for the largest PDS instances. For instance, Table 5 summarizes the results presented by Grigoriadis and Khachiyan in [17] and by Schultz and Meyer in [31] for the largest PDS problems they solved using their ϵ-approximation and barrier decomposition methods, respectively. Columns $f^*_{\epsilon\mathrm{A}}$ and f^*_{BD} give the optimal objective function provided by each method, whereas columns $\frac{f^*-f^*_{\epsilon\mathrm{A}}}{1+f^*}$ and $\frac{f^*-f^*_{\mathrm{BD}}}{1+f^*}$ show the relative errors of the solutions obtained. Looking at Tables 4 and 5, we see that IPM provided two and three more significant figures in all the problems. Indeed, as stated by the authors in [17], ϵ-approximation methods are practical for computing fast approximations to large instances, whereas the solutions provided by IPM can be considered almost optimal.

Figures 10, 11, 12, and 13 show the ratio of the CPU time of CPLEX and PPRN to IPM (i.e., "CPLEX CPU time/IPM CPU time" and "PPRN CPU time/IPM CPU time") for the problems in Tables 1–4. The executions are ordered by the number

TABLE 4
Dimensions and results obtained for the PDSt problems.

$t^{(a)}$	m	n	$\tilde{n}$	$\tilde{m}$	CPU time (seconds) IPM	CPLEX	PPRN	f^*	$\frac{f^*-f^*_{\text{IPM}}}{1+f^*}$
1	126	372	4464	1758	1.	0.	0.	29083930523.	2.8e–6
2	252	746	8952	3518	4.	1.	2.	28857862010.	–5.9e–6
3	390	1218	14616	5508	8.	2.	5.	28597374145.	–4.9e–6
4	541	1790	21480	7741	16.	4.	10.	28341928581.	–1.5e–6
5	686	2325	27900	9871	29.	8.	19.	28054052607.	–6.6e–6
6	835	2827	33924	12012	47.	13.	35.	27761037600.	1.3e–5
7	971	3241	38892	13922	46.	20.	52.	27510377013.	2.1e–5
8	1104	3629	43548	15773	45.	31.	75.	27239627210.	2.1e–5
9	1253	4205	50460	17988	94.	51.	84.	26974586241.	1.5e–5
10	1399	4792	57504	20181	88.	56.	136.	26727094976.	8.4e–6
11	1541	5342	64101	22293	144.	98.	178.	26418289612.	1.3e–5
12	1692	5965	71580	24577	113.	143.	188.	26103493922.	7.4e–5
13	1837	6571	78852	26778	137.	160.	328.	25825886804.	4.6e–5
14	1981	7151	85812	28942	180.	270.	342.	25529159469.	3.4e–5
15	2125	7756	93072	31131	236.	425.	564.	25177923601.	3.3e–5
18	2558	9589	115068	37727	396.	864.	1227.	24332411902.	6.4e–6
20	2857	10858	130296	42285	386.	1830.	2138.	23821658640.	7.0e–5
21	2996	11401	136812	44357	529.	1912.	2322.	23576150674.	2.6e–5
24	3419	13065	156780	50674	963.	4393.	3411.	22856729593.	5.1e–7
27	3823	14611	175332	56664	1010.	7178.	4810.	22133391961.	1.9e–5
30	4223	16148	193776	62601	1325.	24905.	6827.	21385445736.	–1.7e–6
33	4643	17840	214080	68913	1750.	35397.	9154.	20589962883.	1.4e–5
36	5081	19673	236076	75564	1346.	44144.	12704.	19857712721.	4.4e–5
40	5652	22059	264708	84231	1494.	95064.	16779.	18855198824.	1.5e–4
50	7031	27668	332016	105009	4166.	85840.	46664.	16603525724.	3.5e–5
60	8423	33388	400656	126041	6761.	387577.	75880.	14265904407.	2.4e–6
70	9750	38396	460752	145646	12210.	540606.	112310.	12241162812.	2.0e–5
80	10989	42472	509664	163351	13005.	—	125770.	11469077462.	3.0e–5
90	12186	46161	553932	180207	21781.	—	178248.	11087561635.	1.8e–5
100	13366	49742	596904	196768	17222.	—	214961.	10928229968.	8.8e–5

[a] $k = 11$ for all t.

of variables of the problem. The dashed line of the figures separates the executions according to whether IPM was outperformed or not. For the small Dimacs2pprn problems (Figure 10) both CPLEX and PPRN provided better times than IPM, particularly in the smaller instances. In some cases they were 50 and 33 times faster than IPM, respectively. However, for the large Dimacs2pprn cases (Figure 11), IPM provided the best executions and was up to 26 and 100 times more efficient than CPLEX and PPRN. However, the Dimacs2pprn problems are not very complicated, in spite of the large number of variables. This explains the moderate CPU times required by the three codes in their solution. On the other hand, the Mnetgen and PDS instances (Figures 12 and 13) can be considered to be difficult. It is in these situations that IPM clearly outperforms both CPLEX and PPRN. For the Mnetgen problems it was, on average, 4 times faster than CPLEX (20 in the best case) and

TABLE 5
Results reported in [17] *and* [31] *for some of the largest PDSt problems.*

	ϵ-approximation		barrier decomposition	
t	$f^*_{\epsilon A}$	$\frac{f^*-f^*_{\epsilon A}}{1+f^*}$	f^*_{BD}	$\frac{f^*-f^*_{BD}}{1+f^*}$
50	$1.66257 \cdot 10^{10}$	-1.3e–3	$1.6625 \cdot 10^{10}$	-1.3e–3
60	$1.42914 \cdot 10^{10}$	-1.8e–3	$1.4462 \cdot 10^{10}$	-1.4e–2
70	$1.22640 \cdot 10^{10}$	-1.9e–3	$1.2311 \cdot 10^{10}$	-5.7e–3
80	$1.15047 \cdot 10^{10}$	-3.1e–3	—	—

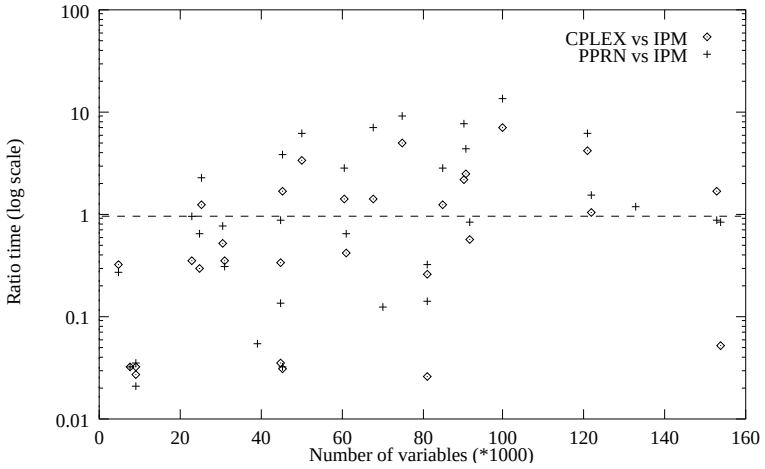


FIG. 10. *Ratio time of CPLEX and PPRN to IPM for the small Dimacs2pprn problems.*

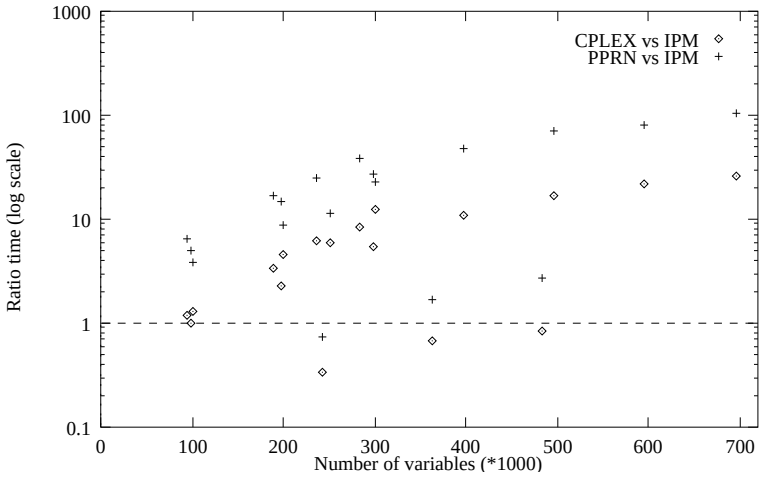


FIG. 11. *Ratio time of CPLEX and PPRN to IPM for the large Dimacs2pprn problems.*

10 times faster than PPRN (45 in the best run). These average figures were 11 for CPLEX and 4 for PPRN when solving the PDS problems (the maximum ratios were of 67 and 12, respectively). It should be pointed out that IPM performed best for the large problems, as indicated by the positive slope of the points in Figures 12 and 13 (note that a log scale is used for the vertical axis). This is especially true for the big

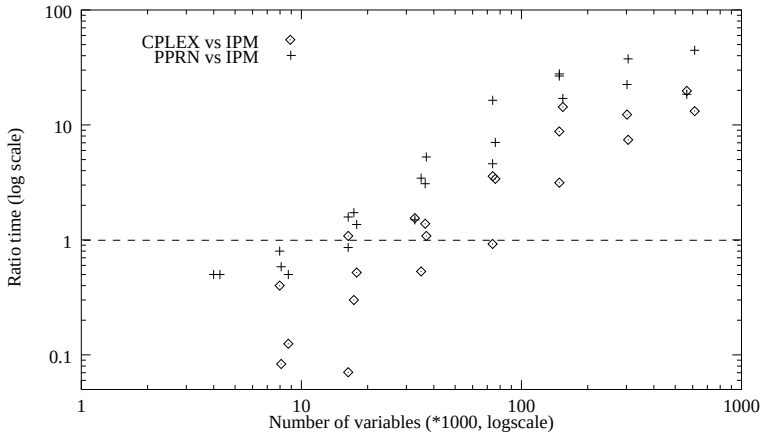


FIG. 12. *Ratio time of CPLEX and PPRN to IPM for the Mnetgen problems.*

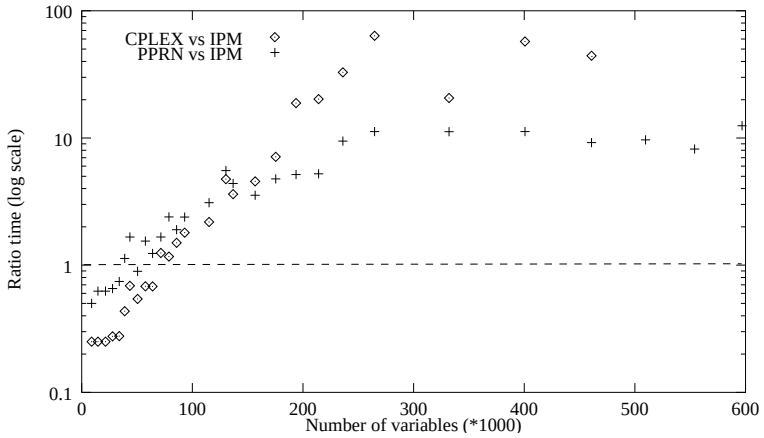


FIG. 13. *Ratio time of CPLEX and PPRN to IPM for the PDS problems.*

Mnetgen problems, where IPM was consistently faster than both CPLEX and PPRN. For the large PDS tests (e.g., $t > 30$) PPRN behaved very well and was only about 11 times slower than IPM, whereas CPLEX provided poorer performances.

Finally, we compared IPM with the CPLEX barrier solver, a state-of-the-art implementation of a general interior-point algorithm. For the comparison, we solved the small Dimacs2pprn problems S_k^i of Table 1. The remaining problems were not executed due to the excessive CPU time the CPLEX barrier solver would take. Table 6 shows the results. The last column gives the ratio time between the CPLEX barrier solver and IPM. The runs not reported correspond to cases where either the system memory was insufficient or the program was stopped because of an excessive execution time. Figure 14 shows the ratio times for the number of variables of the problem. It can be seen that only in some of the smaller problems did the general interior-point code slightly outperform the specialized one. As the size of the problem increases, IPM performs better and is up to 800 times faster in the best case (i.e., S_{100}^4).

7. Conclusions. From the computational experiments reported, it can be stated that the specialized interior-point algorithm is an efficient and promising tool for the

TABLE 6
Performance comparison of IPM and CPLEX (barrier solver).

Prob.	CPU time (seconds) IPM	CPLEX	Ratio time
S^1_1	36.6	35.6	0.97
S^1_4	273.3	2865.7	10.49
S^1_8	1057.8	29445.6	27.84
S^1_{16}	1377.1	—	—
S^1_{50}	17.2	995.5	57.91
S^1_{100}	76.3	3147.7	41.27
S^1_{150}	136.8	6684.8	48.86
S^1_{200}	173.9	13777.1	79.22
S^2_1	28.1	28.7	1.02
S^2_4	622.6	2467.0	3.96
S^2_8	2498.7	23289.5	9.32
S^2_{16}	7550.3	—	—
S^2_{50}	27.6	1195.0	43.22
S^2_{100}	77.8	4263.7	54.78
S^2_{150}	262.6	7584.8	28.89
S^2_{200}	565.2	6095.8	10.79

Prob.	CPU time (seconds) IPM	CPLEX	Ratio time
S^3_1	2.0	2.6	1.28
S^3_4	183.6	64.7	0.35
S^3_8	246.7	548.6	2.22
S^3_{16}	956.1	16290.0	17.04
S^3_{50}	59.8	213.0	3.56
S^3_{100}	172.7	462.9	2.68
S^3_{150}	103.6	623.0	6.01
S^3_{200}	246.9	1254.3	5.08
S^4_1	24.8	23.3	0.94
S^4_4	747.3	1697.4	2.27
S^4_8	4078.8	17400.7	4.27
S^4_{16}	5508.8	—	—
S^4_{50}	14.1	5409.8	383.67
S^4_{100}	39.0	32760.9	839.81
S^4_{150}	68.3	—	—
S^4_{200}	120.7	—	—

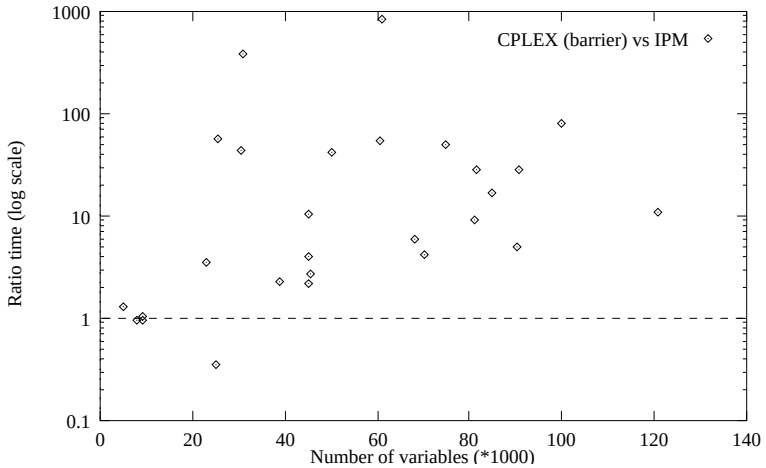


FIG. 14. *Ratio time of CPLEX (barrier solver) to IPM for the small Dimacs2pprn problems.*

solution of large and difficult multicommodity problems. However, the algorithm can still be improved with additional refinements. These include optimal face detection and crossover procedures, improvement in the accuracy of the solution provided by the PCG, and a more appropriate Cholesky factorization solver to reduce the time spent by the forward and backward substitutions. A coarse-grain parallel implementation of the algorithm should also be developed in the future.

Acknowledgments. The author is indebted to N. Nabona, A. Frangioni, and one of the referees for helpful comments and suggestions. The author also thanks Cs. Mészaros for providing a version of BPMPD that reports sparsity pattern information, E. Ng for delivering the Ng–Peyton sparse Cholesky package, and, again, A. Frangioni for the PDS and Mnetgen test instances.

REFERENCES

[1] I. Adler, M.G.C. Resende, and G. Veiga, *An implementation of Karmarkar's algorithm for linear programming*, Math. Programming, 44 (1989), pp. 297–335.

[2] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network Flows*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[3] A. Ali, R.V. Helgason, J.L. Kennington, and H. Lall, *Computational comparison among three multicommodity network flow algorithms*, Oper. Res., 28 (1980), pp. 995–1000.

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

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

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

[7] 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), pp. 240–248.

[8] P. Chardaire and A. Lisser, *Simplex and interior point specialized algorithms for solving non-oriented multicommodity flow problems*, Oper. Res., accepted subject to revision.

[9] I.C. Choi and D. Goldfarb, *Solving multicommodity network flow problems by an interior point method*, in Large-Scale Numerical Optimization, T.F. Coleman and Y. Li, eds., SIAM, Philadelphia, PA, 1990, pp. 58–69.

[10] CPLEX Optimization Inc., *Using the CPLEX Callable Library*, Incline Village, NV, 1995.

[11] DIMACS, *The First DIMACS International Algorithm Implementation Challenge: The Benchmark Experiments*, Technical Report, DIMACS, New Brunswick, NJ, 1991.

[12] A. Frangioni, *personal communication*, Department of Computer Science, Universitá di Pisa, Pisa, Italy, 1998.

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

[14] J.A. George and J.W. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[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. Programming, 76 (1996), pp. 131–154.

[16] J. Gondzio and M. Makowski, *Solving a class of LP problems with a primal-dual logarithmic barrier method*, European J. Oper. Res., 80 (1995), pp. 184–192.

[17] M.D. Grigoriadis and L.G. Khachiyan, *An exponential-function reduction method for block-angular convex programs*, Networks, 26 (1995), pp. 59–68.

[18] A.P. Kamath, N.K. Karmarkar, and K.G. Ramakrishnan, *Computational and Complexity Results for an Interior Point Algorithm on Multicommodity Flow Problems*, Technical Report TR-21/93, Dip. di Informatica, Univ. di Pisa, Italy, 1993, pp. 116–122.

[19] A.P. Kamath and O. Palmon, *Improved Interior Point Algorithms for Exact and Approximate Solutions of Multicommodity Flow Problems*, Technical Report, Dept. of Computer Sciences, Stanford University, Stanford, CA, 1994.

[20] S. Kapoor and P.M. Vaidya, *Speeding up Karmarkar's algorithm for multicommodity flows*, Math. Programming, 73 (1996), pp. 111–127.

[21] J.L. Kennington and R.V. Helgason, *Algorithms for Network Programming*, Wiley, New York, 1980.

[22] I.J. Lustig, R.E. Marsten, and D.F. Shanno, *On implementing Mehrotra's predictor-corrector interior-point method for linear programming*, SIAM J. Optim., 2 (1992), pp. 435–449.

[23] Cs. Mészáros, *The Efficient Implementation of Interior Point Methods for Linear Programming and Their Applications*, Ph.D. Thesis, Eötvös Loránd University of Sciences, Budapest, Hungary, 1996.

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

[25] J.M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, New York, 1988.

[26] L. Portugal, M.G.C. Resende, G. Veiga, G., and J. Júdice, *A truncated primal-infeasible*

dual-feasible network interior point method, Networks, 35 (2000), pp. 91–108.

[27] L. Portugal, M.G.C. Resende, G. Veiga, G., and J. Júdice, *A truncated interior-point method for the solution of minimum cost flow problems on an undirected multicommodity flow network*, in Proceedings of the First Portuguese National Telecommunications Conference, Aveiro, Portugal, 1997, pp. 381–384 (in Portuguese).

[28] M.G.C. Resende and P. Pardalos, *Interior point algorithms for network flow problems*, in Advances in Linear and Integer Programming, J.E. Beasley, ed., Oxford University Press, New York, 1996, pp. 149–189.

[29] M.G.C. Resende and G. Veiga, *An implementation of the dual affine scaling algorithm for minimum-cost flow on bipartite uncapacitated networks*, SIAM J. Optim., 3 (1993), pp. 516–537.

[30] M.G.C. Resende, T. Tsuchiya, and G. Veiga, *Identifying the optimal face of a network linear program with a globally convergent interior point method*, in Large Scale Optimization: State of the Art, W. Hager, D. Hearn, and P. Pardalos, eds., Kluwer Academic Publishers, Dordrecht, The Netherlands, 1994, pp. 362–387.

[31] G.L. Schultz and R.R. Meyer, *An interior point method for block angular optimization*, SIAM J. Optim., 1 (1991), pp. 583–602.

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

[33] S. Zenios, *A smooth penalty function algorithm for network-structured problems*, European J. Oper. Res., 83 (1995), pp. 220–236.