Solving quadratic multicommodity problems through an interior-point algorithm

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

DR 2001-14
August 2001

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

Solving quadratic multicommodity problems through an interior-point algorithm *

Jordi Castro
Department of Statistics and Operations Research
Universitat Politècnica de Catalunya
Pau Gargallo 5, 08028 Barcelona
jcastro@eio.upc.es

Abstract

Standard interior-point algorithms usually show a poor performance when applied to multicommodity network flows problems. A recent specialized interior-point algorithm for linear multicommodity network flows overcame this drawback, and was able to efficiently solve large and difficult instances. In this work we perform a computational evaluation of an extension of that specialized algorithm for multicommodity problems with convex and separable quadratic objective functions. As in the linear case, the specialized method for convex separable quadratic problems is based on the solution of the positive definite system that appears at each interior-point iteration through a scheme that combines direct (Cholesky) and iterative (preconditioned conjugate gradient) solvers. The preconditioner considered for linear problems, which was instrumental in the performance of the method, has shown to be even more efficient for quadratic problems. The specialized interior-point algorithm is compared with the general barrier solver of CPLEX 6.5, and with the specialized codes PPRN and ACCPM, using a set of convex separable quadratic multicommodity instances of up to 500000 variables and 180000 constraints. The specialized interior-point method was, in average, about 10 times and two orders of magnitude faster than the CPLEX 6.5 barrier solver and the other two codes, respectively.

Key words: Interior-point methods, network optimization, multicommodity flows, quadratic programming, large-scale optimization.

1 Introduction

Multicommodity flows are widely used as a modeling tool in many fields as, e.g., in telecommunications and transportation problems. The multicommodity network flow problem is a generalization of the minimum cost network flow one where k different items—the commodities— have to be routed from a set of supply nodes to a set of demand nodes using the same underlying network. This

*Work supported by grant CICYT TAP99-1075-C02-02.

kind of models are usually very large and difficult linear programming problems, and there is a wide literature about specialized approaches for their efficient solution. However most of them only deal with the linear objective function case. In this work we consider a specialized interior-point algorithm for multicommodity network flow problems with convex and separable quadratic objective functions. The algorithm has been able to solve large and difficult quadratic multicommodity problems in a fraction of the time required by alternative solvers.

In the last years there has been a significant amount of research in the field of multicommodity flows, mainly for linear problems. The new solution strategies can be classified into four main categories: simplex-based methods [6, 15], decomposition methods [10, 12], approximation methods [13], and interior-point methods [4, 12]. Some of these algorithms were compared in [7] for linear problems.

The available literature for nonlinear multicommodity flows is not so extensive. For instance, of the above approaches, only the codes of [6] and [12] (named PPRN —nonlinear primal partitioning—and ACCPM—analytic center cutting plane method—, respectively) were extended to nonlinear (possibly non-quadratic) objective functions. In this work we compared the specialized interior-point algorithm with those two codes using a set of large-scale quadratic multicommodity problems. The specialized interior-point algorithm turned out to be the most efficient strategy for all the instances. A description and empirical evaluation of additional nonlinear multicommodity algorithms can be found in the survey [14].

The specialized-interior point method presented here is an extension for convex and separable quadratic objective functions of the algorithm introduced in [4] for linear multicommodity flows. The solution strategy suggested for linear problems (i.e., solving the positive definite system at each interior-point iteration through a scheme that combines direct and iterative solvers) can also be applied to convex and separable quadratic multicommodity problems. Moreover, as it will be shown in the computational results, this solution strategy turned out to be even more efficient for quadratic than for linear problems.

Up to now most applications of multicommodity flows models dealt with linear objective functions. Quadratic multicommodity problems are not usually recognized as a modeling tool, mainly due to the lack of an efficient solver for them. The specialized interior-point method can help to fill this void. The efficient solution of large and difficult quadratic multicommodity problems would open new modeling perspectives (e.g., they could be used in network design algorithms [9]).

The structure of the document is as follows. In Section 2 we formulate the quadratic multicommodity flow problem. In Section 3 we sketch the specialized interior-point algorithm for multicommodity flow problems, and show that it can also be applied to the quadratic case. Finally in Section 4 we perform an empirical evaluation of the algorithm using a set of large-scale quadratic multicommodity flow instances, and three alternative solvers (i.e., CPLEX 6.5, PPRN and ACCPM).

2 The quadratic multicommodity flow problem

Given a network of m nodes, n arcs and k commodities, the quadratic multicommodity network flow problem can be formulated as

$$\begin{aligned}
\min \quad & \sum_{i=0}^{k} ((c^i)^T x^i + (x^i)^T Q^i x^i) \\
\text{subject to} \quad & \begin{bmatrix} N & 0 & \dots & 0 & 0 \\ 0 & N & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & N & 0 \\ \boldsymbol{1} & \boldsymbol{1} & \dots & \boldsymbol{1} & \boldsymbol{1} \end{bmatrix} \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^k \\ x^0 \end{bmatrix} = \begin{bmatrix} b^1 \\ b^2 \\ \vdots \\ b^k \\ u \end{bmatrix} \qquad (1) \\
& 0 \le x^i \le u^i \quad i = 1 \dots k \\
& 0 \le x^0 \le u.
\end{aligned}$$

Vectors $x^i \in \mathbb{R}^n, i = 1 \dots k$, are the flows for each commodity, while $x^0 \in \mathbb{R}^n$ are the slacks of the mutual capacity constraints. $N \in \mathbb{R}^{m \times n}$ is the node-arc incidence matrix of the underlying network, and $\boldsymbol{1}$ denotes the $n \times n$ identity matrix. $c^i \in \mathbb{R}^n$ are the arc linear costs for each commodity and for the slacks. $u^i \in \mathbb{R}^n$ and $u \in \mathbb{R}^n$ are respectively the individual capacities for each commodity and the mutual capacity for all the commodities. $b^i \in \mathbb{R}^m$ are the supply/demand vectors at the nodes of the network for each commodity. Finally $Q^i \in \mathbb{R}^{n \times n}$ are the arc quadratic costs for each commodity and for the slacks. We will restrict to the case where Q^i is a positive semidefinite diagonal matrix, thus having a convex and separable quadratic objective function. Note that (1) is a quadratic problem with $\tilde{m} = km + n$ constraints and $\tilde{n} = (k+1)n$ variables.

Most of the applications of multicommodity flows in the literature only involve linear costs. However, quadratic costs can be useful in the following situations:

- Adding a quadratic penalty term to the occupation of a line in a transmission/transportation network. In this case we would set $Q^i = \boldsymbol{1}, i = 1 \dots k$. This would penalize saturation of lines, guaranteeing a reserve capacity to redistribute the current pattern of flows when line failures occur.
- Replacing a convex and separable nonlinear function by its quadratic approximation.
- Finding the closest pattern of flows x to the currently used $\tilde{x}$, when changes in capacities/demands are performed. In this case the quadratic term would be $(x - \tilde{x})^T (x - \tilde{x})$
- Solution of the subproblems in an augmented Lagrangian relaxation scheme for the network design problem [9, 11]

3 The specialized interior-point algorithm

The multicommodity problem (1) is a quadratic program that can be written in standard form as

$$\min \left\{ c^T x + \frac{1}{2} x^T Q x \ : \ Ax = b, \ x + s = u, \ x, s \geq 0 \right\} , \tag{2}$$

where $x, s, u \in \mathbb{R}^{\tilde{n}}$, $Q \in \mathbb{R}^{\tilde{n} \times \tilde{n}}$ and $b \in \mathbb{R}^{\tilde{m}}$. The dual of (2) is

$$\max \left\{ b^T y - \frac{1}{2} x^T Q x - w^T u \ : \ A^T y - Qx + z - w = c, \ z, w \geq 0 \right\} , \tag{3}$$

where $y \in \mathbb{R}^{\tilde{m}}$ and $z, w \in \mathbb{R}^{\tilde{n}}$. For problem (1), matrix Q is made of $k+1$ diagonal blocks; blocks $Q^i, i = 1 \ldots k$, are related to the flows for each commodity, while Q^0 are the quadratic costs of the mutual capacity slacks.

The solution of (2) and (3) by an interior-point algorithm is obtained through the following system of nonlinear equations (see [18] for details)

$$\begin{array}{ll} r_{xz} & \equiv \mu e - XZe = 0 \\ r_{sw} & \equiv \mu e - SWe = 0 \\ r_b & \equiv b - Ax = 0 \\ r_c & \equiv c - (A^T y - Qx + z - w) = 0 \\ & (x, s, z, w) \geq 0 \ , \end{array} \tag{4}$$

where e is a vector of 1's of appropriate dimension, and matrices X, Z, S, W are diagonal matrices made from vectors x, z, s, w. The set of unique solutions of (4) for each μ value is known as the *central path*, and when $\mu \to 0$ these solutions superlinearly converge to those of (2) and (3) [18]. System (4) is usually solved by a damped version of Newton's method, reducing the μ parameter at each iteration. This procedure is known as the *path-following* algorithm [18]. Figure 1 shows the main steps of the path-following algorithm for quadratic problems.

The specialized interior-point algorithm introduced in [4] for linear multicommodity problems exploited the constraints matrix structure of the problem for solving $(A\Theta A^T)\Delta y = \bar{b}$ (line 5 of Figure 1), which is by far the most computationally expensive step. Considering the structure of A in (1) and accordingly partitioning the diagonal matrix Θ defined in line 3 of Figure 1, we obtain

$$A\Theta A^T = \left[\begin{array}{c|c} B & C \\ \hline C^T & D \end{array}\right] = \left[\begin{array}{ccc|c} N\Theta^1 N^T & \ldots & 0 & N\Theta^1 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \ldots & N\Theta^k N^T & N\Theta^k \\ \hline \Theta^1 N^T & \ldots & \Theta^k N^T & \sum_{i=0}^{k} \Theta^i \end{array}\right] , \tag{5}$$

where $\Theta^i = ((X^i)^{-1} Z^i + (S^i)^{-1} W^i + Q^i)^{-1}, i = 0, 1 \ldots k$. Note that the only difference between the linear and quadratic case is term Q^i of Θ^i. Moreover, as we are assuming that Q^i is a diagonal matrix, Θ^i can be easily computed.

Using (5), and appropriately partitioning Δy and $\bar{b}$, we can write $(A\Theta A^T)\Delta y = \bar{b}$ as

$$\left[\begin{array}{c|c} B & C \\ \hline C^T & D \end{array}\right] \left[\begin{array}{c} \Delta y_1 \\ \hline \Delta y_2 \end{array}\right] = \left[\begin{array}{c} \bar{b}_1 \\ \hline \bar{b}_2 \end{array}\right] . \tag{6}$$

Figure 1: Path-following algorithm for quadratic problems.

Algorithm *Path-following*(A, Q, b, c, u):
1 Initialize $x > 0, s > 0, y, z > 0, w > 0$;
2 **while** (x, s, y, z, w) is not solution **do**
3 $\quad \Theta = (X^{-1}Z + S^{-1}W + Q)^{-1}$;
4 $\quad r = S^{-1}r_{sw} + r_c - X^{-1}r_{xz}$;
5 $\quad (A\Theta A^T)\Delta y = r_b + A\Theta r$;
6 $\quad \Delta x = \Theta(A^T\Delta y - r)$;
7 $\quad \Delta w = S^{-1}(r_{sw} + W\Delta x)$;
8 $\quad \Delta z = r_c + \Delta w + Q\Delta x - A^T\Delta y$;
9 $\quad$ Compute $\alpha_P \in (0, 1], \alpha_D \in (0, 1]$;
10 $\quad x \leftarrow x + \alpha_P\Delta x$;
11 $\quad (y, z, w) \leftarrow (y, z, w) + \alpha_D(\Delta y, \Delta z, \Delta w)$;
12 **end_while**
End_algorithm

By block multiplication, we can reduce (6) to

$$(D - C^T B^{-1} C)\Delta y_2 = (\bar{b}_2 - C^T B^{-1}\bar{b}_1) \quad (7)$$

$$B\Delta y_1 = (\bar{b}_1 - C\Delta y_2). \quad (8)$$

System (8) is solved by performing a Cholesky factorization of each diagonal block $N\Theta^i N, i = 1 \ldots k$, of B. System with matrix $H = D - C^T B^{-1} C$, the Schur complement of (5), is solved by a preconditioned conjugate gradient (PCG) method. A good preconditioner is instrumental for the performance of the method. In [4] it was proved that if

- D is positive semidefinite, an
- $D + C^T B^{-1} C$ is positive semidefinite,

then the inverse of the Schur complement can be computed as

$$H^{-1} = \left(\sum_{i=0}^{\infty} (D^{-1}(C^T B^{-1} C))^i \right) D^{-1}. \quad (9)$$

The preconditioner is thus obtained by truncating the infinite power series (9) at some term h (in practice $h = 0$ or $h = 1$; all the computational results in this work have been obtained with $h = 0$). Since

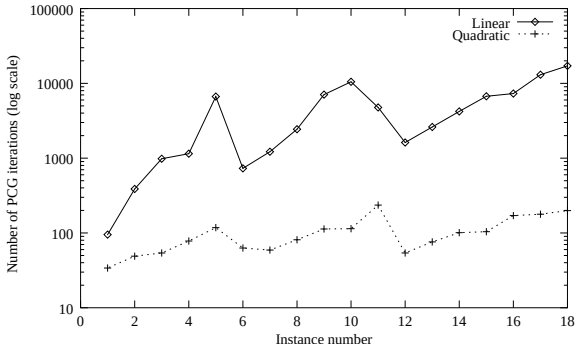
$$D_q = D_l + \sum_{i=0}^{k} (Q^i)^{-1},$$

D_q and D_l denoting the D matrix for a quadratic and linear problem respectively, it is clear that for quadratic multicommodity problems the above two conditions are also guaranteed, and then the same preconditioner can also be

applied. Moreover, since we are assuming diagonal Q^i matrices, for $h = 0$ the preconditioner is equal to $H^{-1} = D^{-1}$, which is also diagonal, as for linear multicommodity problems. This is instrumental in the overall performance of the algorithm. More details about this solution strategy can be found in [4].

The effectiveness of the preconditioner is governed by the spectral radius of $D^{-1}(C^T B^{-1} C))$, which is always in $[0, 1)$. The farthest from 1, the better the preconditioner. According to the computational results obtained, this value seems to be less for quadratic problems than for the equivalent linear problems without the quadratic term, since fewer conjugate gradient iterations are performed for solving (7). Moreover, the number of interior-point iterations also decreases in some instances. This can be observed in Figures 2, 3 and 4. Figures 2 and 3 show the overall number of PCG and IP iterations for the linear and quadratic versions of the Mnetgen problems in Table 1 of Section 4. Both versions only differ in the Q matrix. Clearly, for the quadratic problems fewer IP and PCG iterations are performed. The number of PCG iterations per IP iteration has also been observed to decrease for quadratic problems. For instance, Figure 4 shows the number of PCG iterations per IP iteration for the linear and quadratic versions of problem PDS20 in Table 2 of Section 4. We chose this instance because it can be considered a good representative of the general behavior observed and, in addition, the number of IP iterations is similar for the linear and quadratic problems. A better understanding of the relationship between the spectral radius of $D^{-1}(C^T B^{-1} C))$ for the linear and quadratic problems is part of the further work to be done.

Figure 2: Overall number of PCG iterations for the quadratic and linear Mnetgen instances.



4 Computational results

The specialized algorithm of the previous section was tested using two sets of quadratic multicommodity instances. As far as we know, there is no standard set of quadratic multicommodity problems. Thus we developed a meta-generator

Figure 3: Overall number of IP iterations for the quadratic and linear Mnetgen instances.

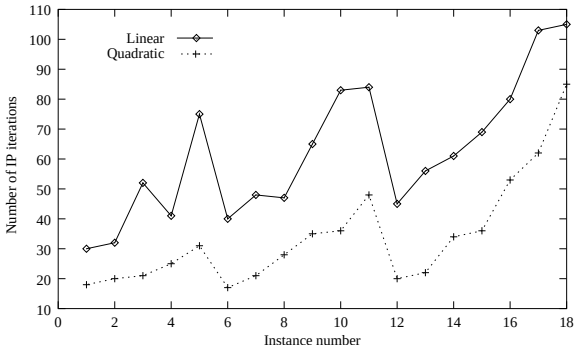
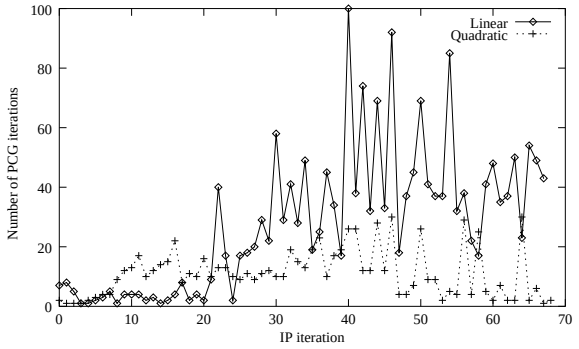


Figure 4: Number of PCG iterations per interior-point iteration, for the quadratic and linear PDS20 instance.



that adds the quadratic term

$$\sum_{i=1}^{k} \sum_{j=1}^{n} q_j^i (x_j^i)^2$$

to the objective function of a linear multicommodity problem. Coefficients q_j^i are randomly obtained from an uniform distribution $U[0, C]$, where

$$C = \sqrt{\left| \frac{\sum_{i=1}^{k} \sum_{j=1}^{n} c_j^i}{kn} \right|},$$

in an attempt to guarantee that linear and quadratic terms are of the same order.

We applied our meta-generator to two sets of linear multicommodity instances obtained with the well-known Mnetgen [1] and PDS [3] generators. Tables 1 and 2 show the dimensions of the instances. Columns "m", "n", and "k" give the number of nodes, arcs and commodities of the network. Columns '$\tilde{n}$"

Table 1: Dimensions of the quadratic Mnetgen instances.

Instance	m	n	k	$\tilde{n}$	$\tilde{m}$
M_{64-4}	64	524	4	2620	780
M_{64-8}	64	532	8	4788	1044
M_{64-16}	64	497	16	8449	1521
M_{64-32}	64	509	32	16797	2557
M_{64-64}	64	511	64	33215	4607
M_{128-4}	128	997	4	4985	1509
M_{128-8}	128	1089	8	9801	2113
M_{128-16}	128	1114	16	18938	3162
M_{128-32}	128	1141	32	37653	5237
M_{128-64}	128	1171	64	76115	9363
$M_{128-128}$	128	1204	128	155316	17588
M_{256-4}	256	2023	4	10115	3047
M_{256-8}	256	2165	8	19485	4213
M_{256-16}	256	2308	16	39236	6404
M_{256-32}	256	2314	32	76362	10506
M_{256-64}	256	2320	64	150800	18704
$M_{256-128}$	256	2358	128	304182	35126
$M_{256-256}$	256	2204	256	566428	67740

and "$\tilde{m}$" give the number of variables and constraints of the quadratic problem. The Mnetgen and PDS generators can be downloaded from

`http://www.di.unipi.it/di/groups/optimize/Data/MMCF.html`.

We solved both sets with an implementation of the specialized interior-point algorithm, referred to as IPM [4], and with CPLEX 6.5 [8], a state-of-the-art interior-point code for quadratic problems. The IPM code, as well as a parallel version [5], can be downloaded for research purposes from

`http://www-eio.upc.es/~jcastro`.

Table 2: Dimensions of the quadratic PDS instances.

Instance	m	n	k	$\tilde{n}$	$\tilde{m}$
PDS1	126	372	11	4464	1758
PDS10	1399	4792	11	57504	20181
PDS20	2857	10858	11	130296	42285
PDS30	4223	16148	11	193776	62601
PDS40	5652	22059	11	264708	84231
PDS50	7031	27668	11	332016	105009
PDS60	8423	33388	11	400656	126041
PDS70	9750	38396	11	460752	145646
PDS80	10989	42472	11	509664	163351
PDS90	12186	46161	11	553932	180207

Table 3: Results for the quadratic Mnetgen problems

	CPLEX 6.5		IPM		
Instance	CPU	n.it.	CPU	n.it	$\frac{f^*_{CPLEX}-f^*_{\text{IPM}}}{1+f^*_{CPLEX}}$
M_{64-4}	0.7	12	0.3	18	−2.0e−6
M_{64-8}	3.1	12	0.8	20	5.6e−7
M_{64-16}	10.7	15	1.6	21	−1.6e−6
M_{64-32}	20.8	16	4.3	25	1.9e−6
M_{64-64}	46.8	14	10.7	31	−1.1e-6
M_{128-4}	2.8	11	0.8	17	1.2e−7
M_{128-8}	12.6	11	2.1	21	2.5e−6
M_{128-16}	80.5	13	5.9	28	6.9e−6
M_{128-32}	153.6	14	15.7	35	2.8e−6
M_{128-64}	305.5	14	35.3	36	−1.4e−6
$M_{128-128}$	741.9	15	98.8	48	−5.7e−7
M_{256-4}	13.1	13	2.7	20	−7.9e−6
M_{256-8}	73.8	14	6.7	22	2.4e−5
M_{256-16}	634.1	15	22.5	34	2.3e−5
M_{256-32}	1105.2	16	49.9	36	2.4e−6
M_{256-64}	2102.2	16	140.0	53	4.9e−7
$M_{256-128}$	4507.3	17	327.6	62	5.0e−6
$M_{256-256}$	11761.3	24	835.3	85	7.0e−6

Table 4: Results for the quadratic PDS problems

	CPLEX 6.5		IPM		
Instance	CPU	n.it.	CPU	n.it	$\frac{f^*_{CPLEX}-f^*_{\text{IPM}}}{1+f^*_{CPLEX}}$
PDS1	1.6	23	1.3	29	−2.7e−7
PDS10	234.8	43	78.6	62	−6.6e−7
PDS20	1425.6	55	271.0	69	1.9e−6
PDS30	5309.8	76	938.3	96	−6.0e−6
PDS40	10712.3	79	1965.2	105	−4.1e−6
PDS50	14049.7	80	3163.3	114	−4.1e−7
PDS60	17133.4	71	3644.2	95	3.6e−6
PDS70	25158.3	74	5548.7	101	−1.9e−7
PDS80	26232.1	74	7029.9	100	−1.3e−6
PDS90	32412.9	77	9786.7	109	−1.2e−6

For each instance, Tables 3 and 4 give the CPU time in seconds required by IPM and CPLEX 6.5 (columns "CPU"), the number of interior-point iterations performed by IPM and CPLEX 6.5 (columns "n.it."), and the relative error $\frac{f^*_{CPLEX} - f^*_{\text{IPM}}}{1+f^*_{CPLEX}}$ of the solution obtained with IPM (assuming CPLEX 6.5 provides the exact optimum). Executions were carried out on a Sun Ultra2 2200 workstation with 200MHz, 1Gb of main memory, and ≈45 Linpack Mflops.

Figures 5–8 summarize the information of Tables 3 and 4. Figures 5 and 6 show respectively the ratio between the CPU times of CPLEX 6.5 and IPM, and the number of interior-point iterations performed by CPLEX 6.5 and IPM, with respect to the dimension of the problem (i.e., number of variables), for the Mnetgen instances. The same information is shown in Figures 7 and 8 for the PDS problems.

Figure 5: Ratio of the execution times of CPLEX 6.5 and IPM for the quadratic Mnetgen problems.

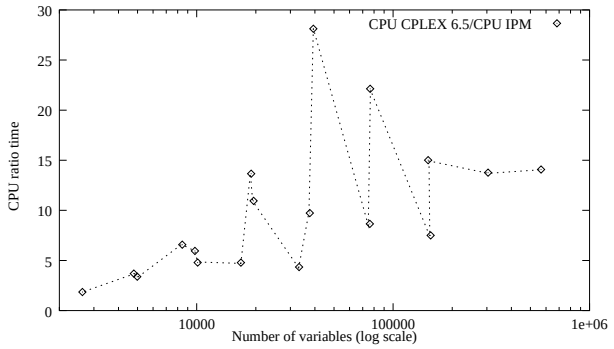
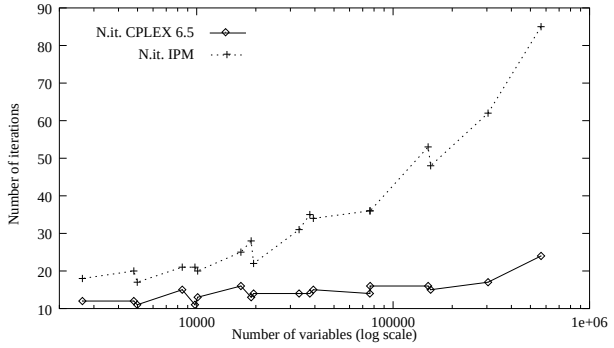


Figure 6: Number of IP iterations performed by CPLEX 6.5 and IPM for the quadratic Mnetgen problems.



From Figures 5 and 7, IPM was in all the cases more efficient than CPLEX 6.5 (the ratio time was always greater than 1.0). For some Mnetgen and PDS instances IPM was about 20 and 5 times faster, respectively. It is important to note that IPM makes use of standard Cholesky routines [16], whereas CPLEX

Figure 7: Ratio of the execution times of CPLEX 6.5 and IPM for the quadratic PDS problems.

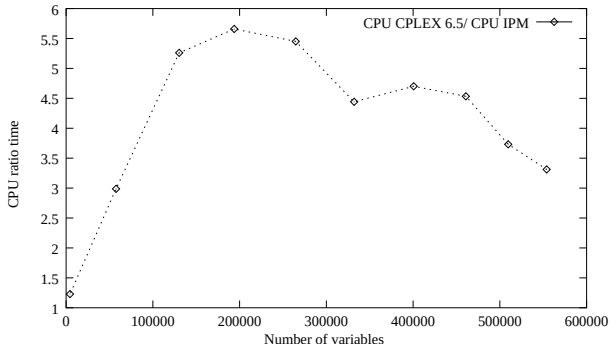
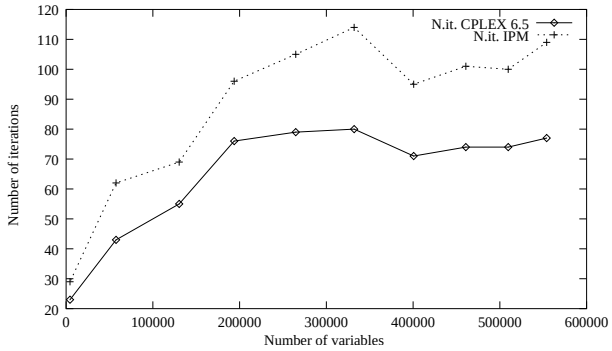


Figure 8: Number of IP iterations performed by CPLEX 6.5 and IPM for the quadratic PDS problems.



6.5 includes a highly tuned and optimized factorization code [2]. Therefore, in principle, the performance of IPM could even be improved. Looking at Figures 6 and 8 it can be seen that IPM performed many more interior-point iterations than CPLEX 6.5. This is because, unlike CPLEX 6.5, the current version of IPM does not implement Mehrotra's predictor-corrector heuristic. In [4] it was shown that Mehrotra's heuristic was not appropriate for linear multicommodity problems. However, for quadratic problems, and because of the good behavior of the preconditioner, it could be an efficient option. Adding Mehrotra's strategy to IPM is part of the additional tasks to be performed.

Finally, we compared IPM and CPLEX 6.5 with PPRN [6] and with an implementation of the ACCPM [12] that we developed using the standard ACCPM library distribution [17]. For this purpose we chose some of the smallest Mnetgen and PDS instances, whose dimensions are shown in Tables 5 and 6 (columns m, n, k, $\tilde{m}$ and $\tilde{n}$, with the same meaning as before). These Tables also give the execution time in seconds (columns "CPU") for each solver. Clearly, CPLEX 6.5 and IPM outperformed both PPRN and ACCPM. Moreover, PPRN and ACCPM seemed not to be competitive approaches for quadratic multicommod-

ity flows. (On the other hand, unlike CPLEX 6.5 and IPM, they can deal with nonlinear objective functions.)

Table 5: Dimensions and results for the small quadratic Mnetgen problems.

						CPU			
Instance	m	n	k	$\tilde{n}$	$\tilde{m}$	CPLEX	IPM	PPRN	ACCPM
M_{64-4}	64	524	4	2620	780	0.7	0.3	6.0	158.0
M_{64-8}	64	532	8	4788	1044	3.1	0.8	38.0	2116.9
M_{64-16}	64	497	16	8449	1521	10.7	1.6	184.6	5683.4
M_{64-32}	64	509	32	16797	2557	20.8	4.3	failed	15753.4
M_{64-64}	64	511	64	33215	4607	46.8	10.7	12710.1	34027.3

Table 6: Dimensions and results for the small quadratic PDS problems.

						CPU			
Instance	m	n	k	$\tilde{n}$	$\tilde{m}$	CPLEX	IPM	PPRN	ACCPM
PDS1	126	372	11	4464	1758	1.6	1.3	75.5	failed
PDS2	252	746	11	8952	3518	5.2	7.9	293.3	failed
PDS3	390	1218	11	14616	5508	10.2	10.5	903.4	failed
PDS4	541	1790	11	21480	7741	22.4	33.9	1702.8	failed
PDS5	686	2325	11	27900	9871	53.2	44.7	2631.3	failed

5 Conclusions and future tasks

From the computational experience reported, it can be stated that the specialized interior-point algorithm is a promising approach for separable quadratic multicommodity problems. Among the future tasks to be performed we find a deep study of the behavior of the spectral radius of $D^{-1}(C^T B^{-1} C)$, the addition of Mehrotra's predictor-corrector method, and using the algorithm in a network design framework.

References

[1] A. Ali and J.L. Kennington. (1977). Mnetgen program documentation, Technical Report 77003, Dept. of Ind. Eng. and Operations Research, Southern Methodist University, Dallas.

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

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

[4] J. Castro. (2000). A specialized interior-point algorithm for multicommodity network flows, *SIAM J. on Optimization*, 10(3), 852–877.

[5] J. Castro. (2000). Computational experience with a parallel implementation of an interior-point algorithm for multicommodity flows, in: *System Modelling and Optimization. Methods, Theory and Applications*, eds. M.J.D. Powell and S. Scholtes, Kluwer, 75–95.

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

[7] P. Chardaire and A. Lisser. (1999). Simplex and interior point specialized algorithms for solving non-oriented multicommodity flow problems, Operations Research (to appear).

[8] ILOG CPLEX. (1999). *ILOG CPLEX 6.5 Reference Manual Library*, ILOG.

[9] A. Frangioni. (2000). Personal communication.

[10] A. Frangioni and G. Gallo. (1999). A bundle type dual-ascent approach to linear multicommodity min cost flow problems, *INFORMS J. on Comp.*, 11(4), 370–393.

[11] B. Gendron, T.G. Crainic, A. Frangioni. (1999). Multicommodity capacitated network design, in *Telecommunications Network Planning*, B. Sansó and P. Soriano (Eds.), Kluwer Academics Publishers, 1–19.

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

[13] Andrew V. Goldberg, Jeffrey D. Oldham, Serge Plotkin and Cliff Stein. (1998). An implementation of a combinatorial approximation algorithm for minimum-cost multicommodity flow, in: *Lecture Notes in Computer Sciences. Proceedings of the 6th International Integer Programming and Combinatorial Optimization Conference*, eds. R.E. Bixby, E.A. Boyd and R.Z. Ríos-Mercado, Springer.

[14] A. Ourou, P. Mahey and J.-Ph. Vial. (2000). A survey of algorithms for convex multicommodity flow problems. *Management Science*, 46(1), 126–147.

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

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

[17] O. Péton and J.-Ph. Vial. (2000). A tutorial on ACCPM, Technical Report, HEC/Logilab, University of Geneva.

[18] S.J. Wright. (1997). *Primal-Dual Interior-Point Methods*, SIAM, Philadelphia, PA.