Annals of Operations Research 124, 35–48, 2003

Solving Difficult Multicommodity Problems with a Specialized Interior-Point Algorithm

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

Abstract. Due to recent advances in the development of linear programming solvers, some of the formerly considered difficult multicommodity problems can today be solved in few minutes, even faster than with specialized methods. However, for other kind of multicommodity instances, general linear solvers can still be quite inefficient. In this paper we will give an overview of the current state-of-the-art in solving large-scale multicommodity problems, comparing an specialized interior-point algorithm with CPLEX 6.5 in the solution of difficult multicommodity problems of up to 1 million of variables and 300,000 constraints.

Keywords: interior-point methods, linear programming, multicommodity flows, network optimization

1. Introduction

Multicommodity flows are regarded among some of the most difficult linear programming problems. Until recently, general linear programming solvers, and even specialized methods, required large execution times for some of the largest multicommodity instances. This was specially true for general interior-point linear programming codes, which usually showed a very poor performance when applied to multicommodity instances. This situation changed with the introduction of the specialized interior-point multicommodity algorithm of Castro (2000a). This algorithm, when developed, was shown to be more efficient than simplex-based alternative methods (i.e., PPRN (Castro and Nabona, 1996) and CPLEX 4.0).

However, the progress made in the simplex implementations during the last years changed the scene again. For instance, CPLEX 6.5 (ILOG CPLEX, 1999) can solve large multicommodity problems one order of magnitude faster than its previous release CPLEX 5.0 (Bixby et al., 2000). Therefore, it is not clear which tool – specialized interior-point or general linear programming solvers – is currently more efficient for multicommodity flows. The purpose of this paper is thus to compare the specialized interior-point algorithm of Castro (2000a) with CPLEX 6.5, one of the most efficient simplex-based implementations. As we will comment later, both codes can be considered as good representatives of the best currently available software for multicommodity flows. In this sense, the paper will also provide an overview of the current state-of-the-art in solving large-scale multicommodity problems.

Listing an extensive bibliography for multicommodity flows is beyond the scope of the paper. (A description of the main solution strategies can be found in Ahuja, Magnanti, and Orlin (1993).) Instead, we will just focus on the recent developments achieved in the field during the last decade. These can be grouped into four classes of methods: simplex-based, decomposition, approximation and interior-point methods.

The simplex-based multicommodity codes rely on primal partitioning techniques that exploit the special structure of the basis Ahuja, Magnanti, and Orlin (1993). Two codes of this type have been developed/improved during the last years: PPRN (Castro and Nabona, 1996), for problems with either linear or nonlinear objective functions, and EMNET (McBride, 1998; Mamer and McBride, 2000) just for the linear case. PPRN and EMNET can also deal with additional linear side constraints. Although both codes share the same underlying theoretical principles, EMNET turns out to be more efficient than PPRN. The improvement is due to a better tuning of pricing and an efficient heuristic for obtaining an initial feasible point. From the computational results presented in McBride (1998) in the solution of the PDS problems with EMNET and those obtained in this paper with CPLEX 6.5 (see section 4), we can conclude that both codes have similar performances. It is thus possible to use a highly efficient general linear programming solver as a good replacement for current primal partitioning multicommodity codes.

The two main decomposition approaches applied in the previous years to multicommodity problems are based on Lagrangian relaxations. The first one relies on bundle-methods (Frangioni and Gallo, 1999) while the second applies the analytic center cutting plane method (ACCPM) (Goffin et al., 1996), both for the maximization of the non-differentiable dual function. The algorithm of Frangioni and Gallo (1999) provides excellent results for the Mnetgen instances, but this good behaviour is not observed in general for the PDS ones (these problems are described in section 4). However, and through an indirect comparison of the results of Frangioni and Gallo (1999) with those of this paper, CPLEX 6.5 seems to provide similar performances to that of the bundle-method-based algorithm. The excellent computational results of Goffin et al. (1996) are difficult to evaluate, since they are obtained with an ad-hoc nonlinear multicommodity generator, and no results are reported with either the standard Mnetgen or PDS instances.

Approximation algorithms for multicommodity flows are presented in Bienstock (1999), Goldberg et al. (1998), and Grigoriadis and Khachiyan (1995), the first of them seeming to be the most successful approach. These methods are able to provide fast ϵ-approximated solutions, where the ϵ parameter is akin to the optimality tolerance. The main drawback of these algorithms is that ϵ is required to be about 0.01 in order to be efficient. This means solutions that match the optimal objective function in the first two significant figures, whereas both CPLEX 6.5 and the specialized interior-point algorithm of Castro (2000a) achieve better precisions (full precision for CPLEX 6.5, about 6 significant figures for the interior-point code).

The last line of research in multicommodity flows comes from the interior-point field. One of the first attempts to apply an interior-point-based algorithm is described in Schultz and Meyer (1991). However, the efficiency of that algorithm is not impressive if compared to the currently available software. In Kamath, Karmarkar, and Ramakrishnan (1993) a general interior-point code was applied to the solution of many multicommodity instances. That approach, however, did not exploit the structure of the multicommodity problem. A generalization for multicommodity flows of the efficient algorithm for single-commodity flows of Resende and Veiga (1993) was presented in Portugal et al. (1997), without reporting any computational result. Finally, the specialized interior-point multicommodity algorithm of Castro (2000a) has shown to be the most efficient interior-point approach up to now. This is the algorithm to be compared with CPLEX 6.5 in this paper.

Some of the above methods were implemented and described in Chardaire and Lisser (1999, 2002) for the case of non-oriented multicommodity flow problems (arcs have no orientation). Among all the algorithms tried (primal partitioning, dual affine scaling interior-point specialization, Dantzig–Wolfe decomposition and ACCPM), the Danzig–Wolfe was shown to be the most efficient method. However, it is difficult to extrapolate these results since non-standard and proprietary real multicommodity instances from the telecommunications field were used. Since these instances are confidential and hence not available for further testing, it is not possible to comment on the behaviour of alternative solvers on these non-oriented problems.

The paper is organized as follows. Section 2 presents the node-arc formulation of the multicommodity problem. Section 3 sketches out the specialized interior-point algorithm of Castro (2000a). Finally section 4 shows the computational comparison between CPLEX 6.5 and the multicommodity interior-point code.

2. The multicommodity network flow problem

Multicommodity network flow models provide optimization problems whose solution gives the best routing of a set of k different types of flows (the commodities) through the arcs of a network. This kind of problems arise naturally when modelling applications in, e.g., routing, telecommunications networks and transportation problems.

We will deal with a fairly general formulation, where arcs are considered to have a capacity for each commodity, and a mutual capacity for all the commodities. The node-arc formulation of the problem is

$$\min \sum_{i=1}^{k} (c^i)^T x^i \tag{1}$$

$$\text{s.t.} \quad \begin{bmatrix} N & 0 & \dots & 0 & 0 \\ 0 & N & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & N & 0 \\ \mathbb{1} & \mathbb{1} & \dots & \mathbb{1} & \mathbb{1} \end{bmatrix} \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^k \\ x^0 \end{bmatrix} = \begin{bmatrix} b^1 \\ b^2 \\ \vdots \\ b^k \\ u \end{bmatrix}, \tag{2}$$

$$\begin{aligned} &0 \leqslant x^i \leqslant u^i, \quad i = 1, \dots, k, \\ &0 \leqslant x^0 \leqslant u. \end{aligned} \tag{3}$$

Vectors $x^i \in \mathbb{R}^n$ are the flow arrays 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 directed graph, while $\mathbb{1}$ denotes the $n \times n$ identity matrix. Clearly, m and n are the number of nodes and arcs of the network, respectively. We shall assume that N is a full row-rank matrix: this can always be guaranteed by removing any of the redundant node balance constraints. $c^i \in \mathbb{R}^n$ and $u^i \in \mathbb{R}^n$ are respectively the flow cost vector and the individual capacity vector for commodity i, while $u \in \mathbb{R}^n$ is the vector of the mutual capacities. Finally, $b^i \in \mathbb{R}^m$ is the vector of supplies/demands for commodity i at the nodes of the network.

The multicommodity flow problem is a linear program with $\widetilde{m} = km+n$ constraints and $\widetilde{n} = (k+1)n$ variables. In some real-world models, k can be very large (e.g., $k = n^2$). For instance, in many telecommunication problems (Chardaire and Lisser, 1999) we have a commodity for the flow of data/voice to be sent between each each pair of nodes of the network. Thus, the resulting linear program can be huge even for graphs of moderate size.

3. The specialized interior-point algorithm

The specialized interior-point method considered in this paper was introduced in Castro (2000a), and it turned out to be an efficient and promising tool for the solution of large and difficult multicommodity problems. The purpose of this paper is not to describe that algorithm; instead, we will only list its main characteristics. A comprehensive description can be found in Castro (2000a). For details about a parallel implementation of the algorithm see (Castro, 2000b; Castro and Frangioni, 2001).

The main features of the algorithm are:

(i) It is based on a primal-dual infeasible path-following algorithm; hence it globally converges superlinearly to the optimum (Wright, 1996).

(ii) At each interior-point iteration it solves the positive definite system

$$A\Theta A^T \Delta y = \bar{b}, \tag{4}$$

where $A \in \mathbb{R}^{\widetilde{m}\times\widetilde{n}}$ is the constraints matrix of (2), $\Theta \in \mathbb{R}^{\widetilde{n}\times\widetilde{n}}$ is a diagonal matrix, $\Delta y \in \mathbb{R}^{\widetilde{m}}$ is the direction for the dual variables, and $\bar{b} \in \mathbb{R}^{\widetilde{m}}$. The solution of (4) is obtained through a scheme that combines direct factorizations and preconditioned conjugate gradients (PCG) (see (Golub and Van Loan, 1996, chapter 10) for a description of PCG). Indeed, exploiting the structure of A, see (2), and Θ – which is made of $k+1$ $n \times n$ diagonal blocks related to the flows for the k commodities and the slacks $\Theta = \text{diag}(\Theta^1, \ldots, \Theta^k, \Theta^0)$, and partitioning accordingly Δy and $\bar{b}$, the solution of (4) is reduced to

$$\left(\sum_{i=0}^{k} \Theta^i - \sum_{i=1}^{k} \Theta^i N^T \left(N\Theta^i N^T\right)^{-1} N\Theta^i\right) \Delta y^0$$

$$= \bar{b}^0 - \sum_{i=1}^{k} \Theta^i N^T \left(N\Theta^i N^T\right)^{-1} \bar{b}^i, \tag{5}$$

$$\left(N\Theta^i N^T\right)\Delta y^i = \left(\bar{b}^i - N\Theta^i \Delta y^0\right), \quad i = 1, \ldots, k. \tag{6}$$

The system (5) is solved through a PCG, while k Cholesky factorizations are performed for (6). The matrix of system (5) let's denote it by H is known as the Schur complement of (4). See (Castro, 2000a) for more details.

(iii) An ad-hoc preconditioner for multicommodity flows is applied in the PCG. In Castro (2000a) it is proved that the inverse of the Schur complement is given by

$$H^{-1} = \left(\sum_{i=0}^{\infty} \left(D^{-1}Q\right)^i\right) D^{-1}, \tag{7}$$

where

$$D = \sum_{i=0}^{k} \Theta^i \quad \text{and} \quad Q = \sum_{i=1}^{k} \Theta^i N^T \left(N\Theta^i N^T\right)^{-1} N\Theta^i.$$

A preconditioner for (5) is obtained by truncating (7) at the hth term. The best computational results are usually obtained with $h = 0$ or $h = 1$ (see (Castro, 2000a)).

(iv) Heuristic procedures for computing directions (as the Mehrotra's predictor–corrector (Mehrotra, 1992) or multiple centrality corrections schemes (Gondzio, 1996) are not applied, since they would mean solving at least twice the PCG system at each interior-point iteration. The extensive set of tests performed with the Mehrotra's predictor–corrector method showed that, although the number of interior-point iterations was significantly reduced, the overall performance of the algorithm decreased (Castro, 2000a).

(v) Inactive mutual capacity constraints are detected and removed during the optimization process, reducing the dimension of the system to be solved by the PCG. The heuristic implementing is based on Gondzio and Makowski (1995).

(vi) This algorithm was implemented in the IPM code. There also exists a parallel version denoted as pIPM (Castro, 2000b; Castro and Frangioni, 2001). Both codes can be freely obtained for research purposes from `http://www-eio.upc.es/~jcastro`, at the software entry.

4. Computational results

4.1. *Test cases*

Three sets of multicommodity instances were used for the computational comparison between CPLEX 6.5 and IPM. The first is made up of 24 problems obtained

Table 1
Dimensions and optimal objective values for the Mnetgen problems.

Problem	m	n	k	$\widetilde{n}$	$\widetilde{m}$	f^*
M_{64-4}	64	524	4	2620	780	192400.1
M_{64-8}	64	532	8	4788	1044	394051.1
M_{64-16}	64	497	16	8449	1521	1071474.9
M_{64-32}	64	509	32	16797	2557	2146944.1
M_{64-64}	64	511	64	33215	4607	4623138.5
M_{128-4}	128	997	4	4985	1509	919643.2
M_{128-8}	128	1089	8	9801	2113	1924133.9
M_{128-16}	128	1114	16	18938	3162	4145079.4
M_{128-32}	128	1141	32	37653	5237	9785961.1
M_{128-64}	128	1171	64	76115	9363	19269824.2
$M_{128-128}$	128	1204	128	155316	17588	40143200.8
M_{256-4}	256	2023	4	10115	3047	5026132.3
M_{256-8}	256	2165	8	19485	4213	9919483.2
M_{256-16}	256	2308	16	39236	6404	20692883.7
M_{256-32}	256	2314	32	76362	10506	45671076.1
M_{256-64}	256	2320	64	150800	18704	92249381.1
$M_{256-128}$	256	2358	128	304182	35126	190137259.9
$M_{256-256}$	256	2204	256	566428	67740	397882591.3
M_{512-4}	512	4077	4	20385	6125	21324851.2
M_{512-8}	512	4373	8	39357	8469	46339269.9
M_{512-16}	512	4620	16	78540	12812	96992237.2
M_{512-32}	512	4646	32	153318	21030	192941834.8
M_{512-64}	512	4768	64	309920	37536	412943158.7
$M_{512-128}$	512	4786	128	617394	70322	828013599.8

with Ali and Kennington's Mnetgen generator (Ali and Kennington, 1977). Table 1 shows the dimensions and optimal solutions of the Mnetgen problems. The parameters used to generate the instances can be found in Frangioni (1997), and can be retrieved from `http://www.di.unipi.it/di/groups/optimize/Data/MMCF.html\#MNetGen`. Columns "m", "n", and "k" show the number of nodes, arcs, and commodities. Columns "$\widetilde{n}$" and "$\widetilde{m}$" give the number of variables and constraints of the linear problem. Finally, column "f^*" gives the exact optimal objective function value, as reported by CPLEX 6.5.

The second set consists of ten of the PDS (Patient Distribution System) problems (Carolan et al., 1990). These problems arise from a logistic model for evacuating patients from a place of military conflict. Each instance depends on a parameter t which denotes the planning horizon under study (in number of days). The size of the network increases with t, while the number of commodities is always 11. Problems obtained with this generator are denoted as PDSt, where t is the number of days considered. The PDS generator can be retrieved from `http://www.di.unipi.it/di/groups/optimize/Data/MMCF.html\#Pds`. The dimensions and optimal objective functions of these instances can be found in table 2. The meaning of the columns is the same as in table 1. Until recently, these problems were considered very difficult, and

Table 2
Dimensions and optimal objective values for the PDS problems.

Problem	m	n	k	$\tilde{n}$	$\tilde{m}$	f^*
PDS1	126	372	11	4464	1758	29083930523.0
PDS10	1399	4792	11	57504	20181	26727094976.0
PDS20	2857	10858	11	130296	42285	23821658640.0
PDS30	4223	16148	11	193776	62601	21385445736.0
PDS40	5652	22059	11	264708	84231	18855198824.0
PDS50	7031	27668	11	332016	105009	16603525724.0
PDS60	8423	33388	11	400656	126041	14265904407.0
PDS70	9750	38396	11	460752	145646	12241162812.0
PDS80	10989	42472	11	509664	163351	11469077462.0
PDS90	12186	46161	11	553932	180207	11087561635.0

Table 3
Dimensions and optimal objective values for the Tripart and Mnetgen problems.

Problem	m	n	k	$\tilde{n}$	$\tilde{m}$	f^*
Tripart1	192	2096	16	35632	5168	63478798.6
Tripart2	768	8432	16	143344	20720	387162296.6
Tripart3	1200	16380	20	343980	40380	269568993.0
Tripart4	1050	24815	35	893340	61565	17774676.0
Gridgen1	1025	3072	320	986112	331072	$\approx$ 1622326915454.6[a]

[a] Approximate solution obtained with CPLEX 6.5 after 35 days of execution.

could not be solved with a high degree of accuracy. As we shall see later, this has radically changed, and CPLEX 6.5 can solve the largest instances in less than an hour on a midrange workstation.

The last set is made of the four Tripart problems and of the Gridgen1 problem. These instances were obtained with the Tripartite generator and with a variation for multicommodity flows of the Gridgen generator. These generators were designed to produce difficult multicommodity instances for approximation algorithms (Bienstock, 1999). The structure of these instances is similar to that of real-world telecommunications problems: they have one source and one destination node per commodity. These can be considered very difficult multicommodity instances, as shown in section 4.2. This is the only of the three sets that was not used in Castro (2000a). Table 3 shows the dimensions and optimal objective functions for these problems. For problem Gridgen1, the f^* column shows an approximate optimal solution provided by CPLEX 6.5 after 35 days of execution.

4.2. *Results*

Tables 4–6 show the results obtained with CPLEX 6.5 and IPM in the solution of the three sets of problems. We also include the results obtained in Castro (2000a) with CPLEX 4.0 for the Mnetgen and PDS instances. The results with this old version of

Table 4
Results for the Mnetgen problems.

Problem	CPLEX 4.0	CPLEX 6.5	IPM	
	CPU	CPU	$(f^* - f^*_{\text{IPM}})/(1+f^*)$	CPU
M_{64-4}	0.2	0.2	−6e−7	0.5
M_{64-8}	0.4	0.4	4e−6	0.5
M_{64-16}	1.9	1.4	1e−5	5.4
M_{64-32}	12.8	5.9	1e−5	11.8
M_{64-64}	141.3	29.2	8e−6	91.3
M_{128-4}	0.4	0.5	1e−6	2.4
M_{128-8}	1.3	1.5	−6e−7	7.6
M_{128-16}	13.1	5.3	6e−6	24.8
M_{128-32}	214.4	33.1	6e−6	154.8
M_{128-64}	1646.8	223.3	−3e−6	484.9
$\text{M}_{128-128}$	7880.1	377.4	9e−6	549.4
M_{256-4}	1.4	1.3	1e−5	12.0
M_{256-8}	12.4	4.0	−2e−6	40.0
M_{256-16}	157.9	21.3	6e−6	145.6
M_{256-32}	1663.9	142.4	−1e−6	465.3
M_{256-64}	9134.5	281.9	−1e−6	1039.6
$\text{M}_{256-128}$	45990.1	747.0	−7e−6	3741.5
$\text{M}_{256-256}$	181700.8	1419.3	−1e−6	9186.7
M_{512-4}	7.2	3.4	−7e−5	99.1
M_{512-8}	100.5	10.1	1e−5	190.2
M_{512-16}	1456.5	32.3	−4e−6	1582.3
M_{512-32}	8301.8	173.0	−7e−7	2644.3
M_{512-64}	55027.5	512.4	8e−8	7411.2
$\text{M}_{512-128}$	280541.4	1217.1	−1e−6	21262.6

CPLEX are useful to understand both the evolution of general simplex solvers for multicommodity problems, and that IPM is no longer the best tool for any class of multicommodity instances, but just for some of them. Columns "$(f^* - f^*_{\text{IPM}})/(1 + f^*)$" show the relative error of the approximate solution obtained with IPM. Columns "CPU" show the execution times (in seconds) obtained for CPLEX 4.0, CPLEX 6.5 and IPM. These times were obtained on a Sun Ultra2 2200 workstation with 200 MHz clock, 1 Gb of main memory, $\approx$ 68 Mflops Linpack, 14.7 SPECfp95 and 7.8 SPECint95. For the Mnetgen and PDS instances we used the network + dual solver of both CPLEX 4.0 and CPLEX 6.5; this is the most efficient combination for most multicommodity problems. For the Tripart and Gridgen problems both the network + dual and the barrier solvers of CPLEX 6.5 were run, as shown in table 6. The two results that appear in this table for Gridgen1 and IPM correspond to executions with different optimality tolerances (10^{-4} and 10^{-5}, respectively).

The Mnetgen and PDS instances have been used for many years as a standard test for multicommodity and linear programming solvers, and were regarded as difficult problems. As shown by tables 4 and 5, in Castro (2000a) IPM was from one to two orders of magnitude faster than CPLEX 4.0 for the largest instances, providing accurate enough

Table 5
Results for the PDS problems.

Problem	CPLEX 4.0	CPLEX 6.5	IPM	
	CPU	CPU	$(f^* - f^*_{\mathrm{IPM}})/(1+f^*)$	CPU
PDS1	0.4	0.4	−2e−6	1.2
PDS10	59.5	27.5	−8e−6	88.0
PDS20	1830.4	208.0	−7e−5	386.3
PDS30	24904.9	336.2	1e−6	1324.9
PDS40	95063.7	771.7	−1e−4	1494.1
PDS50	85839.5	1035.4	−3e−5	4165.7
PDS60	387577.3	1461.9	−2e−6	6761.3
PDS70	540606.3	1654.5	−2e−5	12209.9
PDS80	Not executed	2349.2	−3e−5	13004.9
PDS90	Not executed	2899.5	−1e−5	21781.4

Table 6
Results for the Tripart and Gridgen problems.

Problem	CPLEX 6.5 CPU		IPM	
	Net + Dual	Barrier	$(f^* - f^*_{\mathrm{IPM}})/(1+f^*)$	CPU
Tripart1	18.2	76.3	−5e−5	39.9
Tripart2	3263.8	672.8	−4e−5	248.9
Tripart3	13577.8	3046.8	−1e−4	1584.2
Tripart4	89501.2	34371.9	−4e−5	4982.6
Gridgen1	> 3.1e+6	–[a]	9e−5[b]	61807.8
			9e−6[b]	126008.3

[a] Not enough memory to run this problem.
[b] Relative length of the objective optimal interval provided by the primal and dual solutions.

solutions – the relative error in the objective was never greater than 1.0e–5. Moreover, the efficiency of IPM increased with the dimension of the instances. This is clearly shown by figures 1 and 2, that plot the ratio between the running times of CPLEX 4.0 and IPM in relation to the dimension of the Mnetgen and PDS problems, respectively.

However, the Mnetgen and PDS instances can no longer be considered as difficult problems. Indeed, as shown by column “CPLEX 6.5 CPU” of tables 4 and 5, all the Mnetgen and PDS instances could be solved by CPLEX 6.5 in less than an hour. This means that, unlike its fourth release, CPLEX 6.5 is now outperforming IPM by one order of magnitude. The ratios between the running times of IPM and CPLEX 6.5 for the Mnetgen and PDS problems are plotted in figures 3 and 4. For the largest Mnetgen and PDS instances CPLEX 6.5 was up to 20 and 10 times faster than IPM, respectively. These results clearly show that the recent improvements to the simplex implementations (Bixby et al., 2000) turned out state-of-the-art general linear solvers an efficient tool for the solution of multicommodity problems.

Despite these advances in simplex-based implementations, some classes of multicommodity problems are still difficult for them. Among these we find the third set of

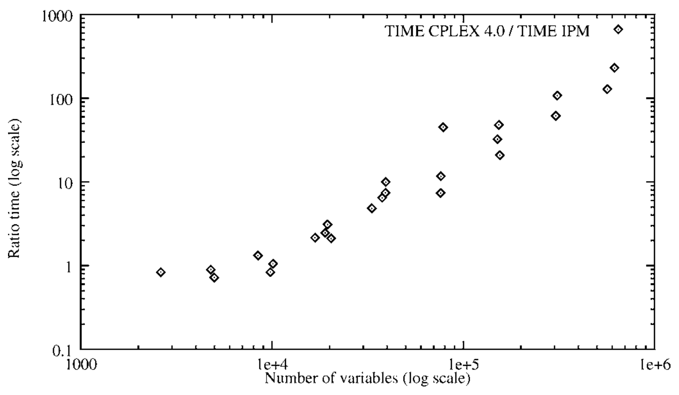

Figure 1. Ratio time between CPLEX 4.0 and IPM for the Mnetgen problems.

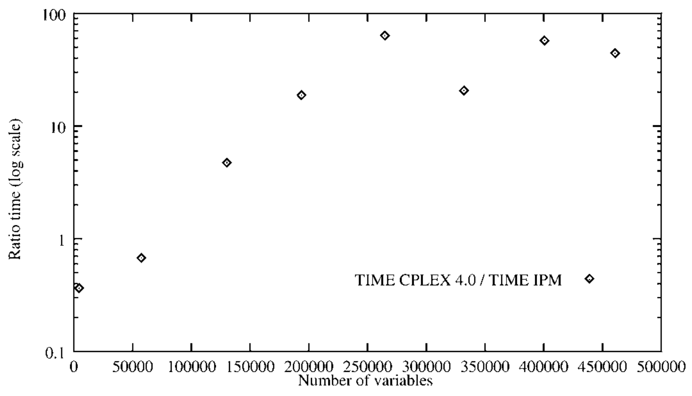

Figure 2. Ratio time between CPLEX 4.0 and IPM for the PDS problems.

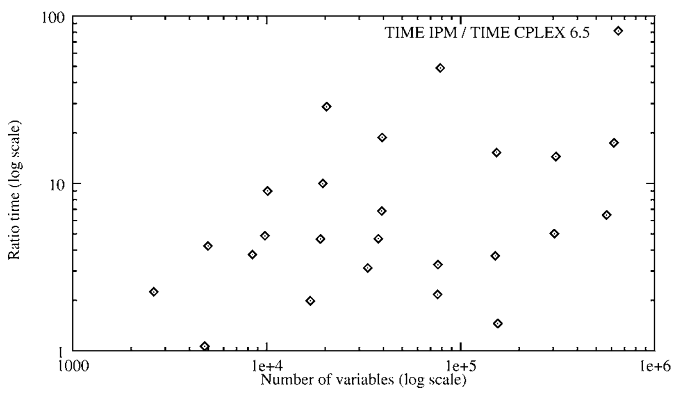

Figure 3. Ratio time between IPM and CPLEX 6.5 for the Mnetgen problems.

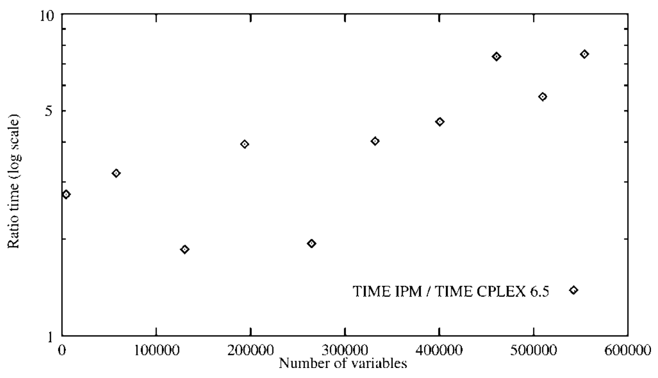

Figure 4. Ratio time between IPM and CPLEX 6.5 for the PDS problems.

Table 7
Memory requirements (in MB) of each solver for the Tripart and Gridgen problems.

Problem	CPLEX 6.5		IPM
	Net + Dual	Barrier	
Tripart1	17	22	6
Tripart2	58	93	17
Tripart3	130	239	37
Tripart4	350	743	92
Gridgen1	553	> 1000	147

Tripart and Gridgen instances. For these problems the barrier algorithm of CPLEX 6.5 turned out to be more efficient than the network + dual solver for large enough instances, as shown by table 6, while the specialized interior-point algorithm of IPM in its turn outperformed the general one of CPLEX 6.5. For instance, IPM was about seven times faster for problem Tripart4. Problem Gridgen1 could not be solved with the barrier CPLEX 6.5 option, due to memory limitations, and only an almost optimal solution was computed with the network + dual solver after 35 days of execution. For Gridgen1, IPM was 50.1 or 24.6 times faster than CPLEX 6.5, depending on the optimality tolerance we used in IPM. In fact, for this problem the objective function value provided by CPLEX 6.5 after 26 days of execution – $1.62228 \cdot 10^{12}$ – still was not inside the optimal interval $[f^*_{\text{dual}}, f^*_{\text{primal}}]$ provided by IPM with the more accurate optimality tolerance – $[1.62232 \cdot 10^{12}, 1.62234 \cdot 10^{12}]$. These results are specially relevant if we consider that CPLEX 6.5 includes highly tuned sparse linear algebra routines (Bixby et al., 2000), whereas IPM uses the standard library of Ng and Peyton (1993).

It is also worth to note the memory requirements of each solver for the Tripart and Gridgen problems, which are reported in table 7. The specialized interior-point algorithm needed by far much less memory than the network + dual CPLEX 6.5 option. In fact, all the IPM runs could have been performed on a standard PC. From table 7 it can also be concluded that general interior-point algorithms require large amount of

Table 8
Results for the Tripart and Gridgen problems with pIPM.

Problem	p	T_p	S_p	$\overline{S_p}$
Tripart1	1	34.9	1.0	1.0
	4	21.3	1.6	3.4
	8	17.9	1.9	5.5
	16	19.6	1.8	8.2
Tripart2	1	156.6	1.0	1.0
	4	71.6	2.2	3.2
	8	55.4	2.8	5.1
	16	60.3	2.6	7.2
Tripart3	1	1140.7	1.0	1.0
	4	408.4	2.8	3.5
	10	300.5	3.8	6.9
	20	304.8	3.7	10.2
Tripart4	1	3273.2	1.0	1.0
	5	893.7	3.7	4.3
	7	721.5	4.5	5.5
	35	601.1	5.4	14.0
Gridgen1	1	37234.9	1.0	1.0
	8	10533.2	3.5	7.7
	16	7678.7	4.8	14.9
	32	4426.5	8.4	27.7
	64	3248.6	11.5	48.7

memory when applied to multicommodity problems. As noted in Castro (2000a), this is due to the fill-in during the Cholesky factorization of the $A\Theta A^T$ matrix.

Problems Tripart and Gridgen can even be solved more efficiently using the parallel version of IPM, denoted as pIPM (Castro, 2000b; Castro and Frangioni, 2001). Table 8 shows the results obtained running pIPM on a Silicon Graphics Origin2000 server, with 64 MIPS R10000 processors running at 250 Mhz, each of them credited of 24.5 SPECfp95 and 14.7 SPECint95. A total of 8 Gb of memory is distributed among these processing elements. It is worth mentioning that the coarse-grained parallelization scheme of pIPM could be ported to a less sophisticated parallel computing platform without substantially affecting the quality of these results. Column "p" gives the number of processors used in the execution. "T_p" denotes the execution (wall-clock) time. Columns "S_p" and "$\overline{S_p}$" give respectively the observed and the ideal speedups. S_p is defined as T_1/T_p, while $\overline{S_p}$ depends on the fraction f of the time spent in the parallel region in the sequential execution, and is computed as $1/(f/p + (1 - f))$. Although there is a gap between the observed speedups and their maximum values, the overall execution time is reduced with the number of processors (except for problems Tripart 1, 2 and 3 with 16 processors). For instance, the difficult Gridgen1 problem could be solved in less than one hour with 64 processors.

From the above results it is clear that the behaviour of each algorithm mainly depends on the problem structure rather than its size. As a rule of thumb, it could be stated that problems with sparse networks – leading to sparse $N\Theta^i N^T$ matrices – can be efficiently solved with interior-point methods. Simplex-based solvers should be chosen for dense networks. However, it is difficult to stablish a definite decision rule, since other problem characteristics, as the number of binding mutual capacity constraints, the number of degenerate steps performed by simplex-based algorithms, etc., play an important role. The behaviour of the algorithms tested with problem Gridgen1 clearly illustrates this fact.

5. Conclusions

From the computational experience in the solution of a fairly large set of instances, it can be stated that there seems not to be a definite algorithm for multicommodity flows. Simplex-based solvers can outperform specialized interior-point codes for some particular problems, while for others the interior-point approach is a more effective tool. The final recommendation to an end-user would be to check which solution method (simplex-based, interior-point, decomposition methods) is the most appropriate for her/his particular type of problems. If the accuracy of the solution is not important, approximation methods should also be considered. Combining all these strategies in a single solver, or developing a decision rule to know in advance the best solver for each particular problem is still part of the further work to be done in the field of multicommodity flows.

Acknowledgments

This work has been supported by the CICYT Project TAP99-1075-C02-02. The author is indebted to Robert Bixby for communicating about the existence of the difficult Gridgen and Tripart instances, and to Daniel Bienstock for providing him with them.

References

Ahuja, R.K., T.L. Magnanti, and J.B. Orlin. (1993). *Network Flows*. New York: Prentice Hall.

Ali, A. and J.L. Kennington. (1977). "Mnetgen Program Documentation." Technical Report 77003, Department of Ind. Eng. and Operations Research, Southern Methodist University, Dallas.

Bienstock, D. (1999). "Approximately Solving Large-Scale Linear Programs. I: Strengthening Lower Bounds and Accelerating Convergence." CORC Report 1999–1, Columbia University, New York.

Bixby, R.E., M. Fenelon, Z. Gu, E. Rothberg, and R. Wunderling. (2000). "MIP: Theory and Practice – Closing the Gap." In M.J.D. Powell and S. Scholtes (eds.), *System Modelling and Optimization. Methods, Theory and Applications*. Dordrecht: Kluwer, pp. 19–49.

Castro, J. (2000a). "A Specialized Interior-Point Algorithm for Multicommodity Network Flows." *SIAM Journal on Optimization* 10(3), 852–877.

Castro, J. (2000b). "Computational Experience with a Parallel Implementation of an Interior-Point Algorithm for Multicommodity Flows." In M.J.D. Powell and S. Scholtes (eds.), *System Modelling and Optimization. Methods, Theory and Applications*. Dordrecht: Kluwer, pp. 79–95.

Castro, J. and A. Frangioni. (2001). "A Parallel Implementation of an Interior-Point Algorithm for Multicommodity Network Flows." In J.M. Palma, J. Dongarra, and V. Hernández (eds.), *Vector and Parallel Processing*, Lecture Notes in Computer Sciences, Vol. 1981. Berlin: Springer, pp. 301–315.

Castro, J. and N. Nabona. (1996). “An Implementation of Linear and Nonlinear Multicommodity Network Flows.” *European Journal of Operations Research* 92, 37–53.

Carolan, W.J., 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.

Chardaire, P. and A. Lisser. (1999). “Simplex and Interior Point Specialized Algorithms for Solving Non-Oriented Multicommodity Flow Problems.” To appear in *Operations Research*.

Chardaire, P. and A. Lisser. (2002). “Minimum Cost Multicommodity Flow.” In M. Pardalos and M. Resende (eds.), *Handbook of Applied Optimization*. Oxford: Oxford University Press, pp. 404–422.

Frangioni, A. (1997). “Dual Ascent Methods and Multicommodity Flows.” Ph.D. Thesis TD 5/97, Dip. di Informatica, Univ. di Pisa.

Frangioni, A. and G. Gallo. (1999). “A Bundle Type Dual-Ascent Approach to Linear Multicommodity Min Cost Flow Problems.” *INFORMS Journal on Computing* 11(4), 370–393.

Goffin, J.-L., J. Gondzio, R. Sarkissian, and J.-P. Vial. (1996). “Solving Nonlinear Multicommodity Flow Problems by the Analytic Center Cutting Plane Method.” *Mathematical Programming* 76, 131–154.

Goldberg, A.V., J.D. Oldham, S. Plotkin, and C. Stein. (1998). “An Implementation of a Combinatorial Approximation Algorithm for Minimum-Cost Multicommodity Flow.” In R.E. Bixby, E.A. Boyd, and R.Z. Ríos-Mercado (eds.), *Proceedings of the 6th International Integer Programming and Combinatorial Optimization Conference*. Lecture Notes in Computer Sciences, Vol. 1412. Berlin: Springer.

Golub, G.H. and C.F. Van Loan. (1996). *Matrix Computations*, 3rd. ed. Baltimore, MD: Johns Hopkins University Press.

Gondzio, J. (1996). “Multiple Centrality Corrections in a Primal-Dual Method for Linear Programming.” *Computational Optimization and Applications* 6, 137–156.

Gondzio, J. and M. Makowski. (1995). “Solving a Class of LP Problems with a Primal-Dual Logarithmic Barrier Method.” *European Journal of Operations Research* 80, 184–192.

Grigoriadis, M.D. and L.G. Khachiyan. (1995). “An Exponential-Function Reduction Method for Block-Angular Convex Programs.” *Networks* 26, 59–68.

ILOG CPLEX. (1999). ILOG CPLEX 6.5 Reference Manual Library. ILOG.

Kamath, A.P., N.K. Karmarkar, and K.G. Ramakrishnan. (1993). “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, pp. 116–122.

Mamer, J. and R. McBride. (2000). “A Decomposition Bases Pricing Procedure for Large Scale Linear Programs: An Application to the Linear Multicommodity Flow Problem.” *Management Science* 46, 693–709.

McBride, R.D. (1998). “Progress Made in Solving the Multicommodity Flow Problem.” *SIAM Journal on Optimization* 8, 947–955.

Mehrotra, S. (1992). “On the Implementation of a Primal-Dual Interior Point Method.” *SIAM Journal on Optimization* 2, 575–601.

Ng, E. and B.W. Peyton. (1993). “Block Sparse Cholesky Algorithms on Advanced Uniprocessor Computers.” *SIAM Journal on Scientific Computing* 14, 1034–1056.

Portugal, L., M.G.C. Resende, G. Veiga, G., and J. Júdice. (1997). “A Truncated Interior-Point Method for the Solution of Minimum Cost Flow Problems on an Undirected Multicommodity Flow Network.” In *Proceedings of First Portuguese National Telecommunications Conference*, pp. 381–384 (in Portuguese).

Resende, M.G.C. and G. Veiga. (1993). “An Implementation of the Dual Affine Scaling Algorithm for Minimum Cost Flow on Bipartite Uncapacitated Networks.” *SIAM Journal on Optimization* 3, 516–537.

Schultz, G.L. and R.R. Meyer. (1991). “An Interior Point Method for Block Angular Optimization.” *SIAM Journal on Optimization* 1, 583–602.

Wright, S.J. (1996). *Primal–Dual Interior-Point Methods*. Philadelphia, PA: SIAM.