A Bundle Type Dual-Ascent Approach to Linear Multicommodity Min-Cost Flow Problems

ANTONIO FRANGIONI / *Department of Computer Science, University of Pisa, Corso Italia 40, 56100 Pisa – Italy, Email: frangio@di.unipi.it*

GIORGIO GALLO/ *Department of Computer Science, University of Pisa, Corso Italia 40, 56100 Pisa – Italy, Email: gallo@di.unipi.it*

Abstract: We present a Cost Decomposition approach for the linear Multicommodity Min-Cost Flow problem, where the mutual capacity constraints are dualized and the resulting Lagrangean Dual is solved with a dual-ascent algorithm belonging to the class of Bundle methods. Although decomposition approaches to block-structured Linear Programs have been reported not to be competitive with general-purpose software, our extensive computational comparison shows that, when carefully implemented, a decomposition algorithm can outperform several other approaches, especially on problems where the number of commodities is “large” with respect to the size of the graph. Our specialized Bundle algorithm is characterized by a new heuristic for the trust region parameter handling, and embeds a specialized Quadratic Program solver that allows the efficient implementation of strategies for reducing the number of active Lagrangean variables. We also exploit the structural properties of the single-commodity Min-Cost Flow subproblems to reduce the overall computational cost. The proposed approach can be easily extended to handle variants of the problem.

Keywords: Multicommodity Flows, Bundle methods, Decomposition Methods, Lagrangean Dual

0. Introduction

The Multicommodity Min-Cost Flow problem (*MMCF*), i.e. the problem of shipping flows of different nature (*commodities*) at minimal cost on a network, where different commodities compete for the resources represented by the arc capacities, has been widely addressed in the literature since it models a wide variety of transportation and scheduling problems[28, 58, 68, 3, 2, 6, 11]. *MMCF* is a structured Linear Program (*LP*), but the instances arising from practical applications are often huge and the usual solution techniques are not efficient enough: this is especially true if the solution is required to be integral, since then the problem is $\mathcal{NP}$-hard and most of the solution techniques (Branch & Bound, Branch & Cut …) rely on the repeated solution of the continuous version.

From an algorithmic viewpoint, *MMCF* has motivated many important ideas that have later found broader application: examples are the column generation approach[22] and the Dantzig-Wolfe decomposition algorithm[21]. This “pushing” effect is still continuing, as demonstrated by a number of interesting recent developments[59, 31, 32, 33].

MMCF problems also arise in finding approximate solutions to several hard graph problems[10, 43]. Recently, some ε-approximation approaches have been developed for the problem[36, 60, 64], making *MMCF* one of the few *LP*s for which approximations algorithms of practical interest are known[52, 34].

In this work, we present a wide computational experience with a Cost Decomposition algorithm for *MMCF*. Decomposition methods have been around for 40 years, and several proposals for the "coordination phase"[22, 23, 51, 54, 61, 20] can be found in the literature. Our aim is to assess the effectiveness of a Cost Decomposition approach based on a NonDifferentiable Optimization (*NDO*) algorithm belonging to the class of "Bundle methods"[63, 39]. To the best of our knowledge, only one attempt[54] has previously been made to use Bundle methods in this context, and just making use of pre-existent *NDO* software: here we use a specialized Bundle code for *MMCF*. Innovative features of our code include a new heuristic for setting the trust region parameter, an efficient specialized Quadratic Program (*QP*) solver[24] and a Lagrangean variables generation strategy. Perhaps, the main contribution of the paper is the extensive set of computational experiences performed: we have tested our implementation together with several other approaches on a large set of test problems of various size and structure. Our experience shows that the Cost Decomposition approach can be competitive, especially on problems where the number of commodities is "large" with respect to the size of the graph.

1. Formulation and Approaches

Given a directed graph $G(N, A)$, with $n = |N|$ nodes and $m = |A|$ arcs, and a set of k *commodities,* the linear Multicommodity Min-Cost Flow problem can be formulated as follows:

$$(MMCF) \quad \begin{cases} \min \sum_h \sum_{ij} c^h_{ij} x^h_{ij} & \\ \sum_j x^h_{ij} - \sum_j x^h_{ji} = b^h_i & \forall\, i,\, h \qquad (a) \\ \sum_h x^h_{ij} \le u_{ij} & \forall\, i,\, j,\, h \qquad (b) \\ 0 \le x^h_{ij} \le u^h_{ij} & \forall\, i,\, j \qquad (c) \end{cases}$$

where, for each arc (i, j), x^h_{ij} is the flow of commodity h, u^h_{ij} and c^h_{ij} are respectively the *individual capacity* and unit cost of x^h_{ij}, and u_{ij} is the *mutual capacity* which bounds the total quantity of flow on (i, j). Constraints (a) and (b) are the flow conservation and individual capacity constraints for each commodity, respectively, while (c) represents the mutual capacity constraints. In matrix notation, using the node-arc incidence matrix $\boldsymbol{E}$ of G, *MMCF* becomes

$$(MMCF) \quad \begin{cases} \min \sum_h \boldsymbol{c}^h \boldsymbol{x}^h \\ \begin{bmatrix} \boldsymbol{E} & \cdots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{E} \\ \boldsymbol{I} & \cdots & \boldsymbol{I} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{x}^1 \\ \vdots \\ \boldsymbol{x}^k \end{bmatrix} \begin{matrix} = \\ \le \end{matrix} \begin{bmatrix} \boldsymbol{b}^1 \\ \vdots \\ \boldsymbol{b}^k \\ \boldsymbol{u} \end{bmatrix} \\ \boldsymbol{0} \le \boldsymbol{x}^h \le \boldsymbol{u}^h \qquad \forall h \end{cases}$$

This formulation highlights the block-structured nature of the problem.

MMCF instances may have different characteristics, that make them more or less suited to be solved by a given approach: for instance, the number of commodities can be small, as in many distribution problems, or as large as is the number of all the possible Origin / Destination pairs, as the case of traffic and telecommunication problems.

MMCF is the prototype of many block-structured problems[33], such as Fractional Packing[59] and Resource Sharing problems[31]. Common variants of *MMCF* are the *Nonhomogeneous MMCF*[4] where (c) is replaced by $\boldsymbol{A}[\ \boldsymbol{x}^1 \ \ldots\ \boldsymbol{x}^k\ ] \leq \boldsymbol{u}$, the *Nonsimultaneous MMCF*[56] where the commodities are partitioned in p "blocks" and only commodities within the same block compete for the capacity, and the *Equal Flow problem*[3] where the flow on a certain set of pairs of arcs is constrained to be identical.

Several algorithmic approaches have been proposed for *MMCF*: among them, we just name *Column Generation*[13, 57, 8], *Resource Decomposition*[28, 50, 49], *Primal Partitioning*[23, 18], *specialized Interior Point*[62, 48, 70, 19], *Cost Decomposition*[23, 51, 54, 61, 20, 69], *Primal-Dual*[67] and *ε-approximation*[52, 34] methods. Unfortunately, the available surveys are either old[5, 45] or concentrated on just the "classical" approaches[4. - Chap. 17]: hence, we refer the interested reader to[25], where the relevant literature on the subject is surveyed. Information about *parallel* approaches to *MMCF* can also be found there or in[14] (these papers can be downloaded from http://www.di.unipi.it/~frangio/).

Our approach belongs to the class of *Cost Decomposition* methods: to solve *MMCF*, we consider its Lagrangean Relaxation with respect to the "complicating" constraints (c), i.e.

$$(RMG_\lambda) \qquad \varphi(\lambda) = \Sigma_h \ \min\{\ (\boldsymbol{c}^h + \lambda)\boldsymbol{x}^h : \boldsymbol{E}\boldsymbol{x}^h = \boldsymbol{b}^h\ ,\ \boldsymbol{0} \leq \boldsymbol{x}^h \leq \boldsymbol{u}^h\ \} - \lambda\boldsymbol{u}$$

and solve the corresponding Lagrangean Dual

$$(DMMCF) \qquad \max\{\ \varphi(\lambda) : \lambda \geq \boldsymbol{0}\ \}.$$

The advantage of this approach is that the calculation of $\varphi(\lambda)$ requires the solution of k independent single-commodity Min-Cost Flow problems (*MCF*), for which several efficient algorithms exist. The drawback is that maximization of the *nondifferentiable* function φ is required.

Many *NDO* algorithms can be used to maximize φ; among them: the (many variants of the) *Subgradient* method[40, 53], the *Cutting Plane* approach[44] (that is the dual form of the *Dantzig-Wolfe decomposition*[21, 23, 42]), the *Analytic Center Cutting Plane* approach[30, 55] and various *Bundle methods*[39, 51, 54]. The latter can also be shown[26] to be intimately related to seemingly different approaches such as *smooth Penalty Function*[61] and *Augmented Lagrangean*[20] algorithms, where φ is approximated with a smooth function. Most of these methods only require at each step a *subgradient* $\boldsymbol{g}(\lambda) = \boldsymbol{u} - \Sigma_h\ \boldsymbol{x}^h(\lambda)$ of φ, where $(\ \boldsymbol{x}^1(\lambda) \ldots \boldsymbol{x}^k(\lambda)\ )$ is any optimal solution of (RMG_λ).

2. The Cost Decomposition code

It is outside the scope of the present work to provide a full description of the theory of Bundle methods: the interested reader is referred to [39]. In [25, 26], the use of Bundle methods for Lagrangean optimization and their relations with other approaches (Dantzig-Wolfe decomposition, Augmented Lagrangean ...) are discusses in detail.

Bundle methods are iterative *NDO* algorithms that visit a sequence of points $\{\ \lambda_i\ \}$ and, at each step, use (a subset of) the first-order information $\beta = \{\ \langle\ \varphi(\lambda_i)\ ,\ \boldsymbol{g}(\lambda_i)\ \rangle\ \}$ (the *Bundle*) gathered so far to compute a tentative ascent direction $\boldsymbol{d}$. In the "classical" Bundle method, $\boldsymbol{d}$ is the solution of

$$(\Delta_{\beta t}) \qquad \min_\theta \{\ 1/2 \|\ \Sigma_{i \in \beta}\ \boldsymbol{g}_i\theta_i\ \|^2 + (1/t)\ \alpha_\beta\theta : \Sigma_{i \in \beta}\ \theta_i = 1\ ,\ \theta \geq \boldsymbol{0}\ \}$$

where $\boldsymbol{g}_i = \boldsymbol{g}(\lambda_i)$, $\alpha_i = \varphi(\lambda_i) + \boldsymbol{g}_i(\bar\lambda - \lambda_i) - \varphi(\bar\lambda)$ is the *linearization error* of $\boldsymbol{g}_i$ with respect to the *current point* $\bar\lambda$ and $t > 0$ is the *trust region parameter*. From the "dual viewpoint", $(\Delta_{\beta t})$ can be considered as an approximation of the steepest ε-ascent direction finding problem; from the "primal viewpoint", $(\Delta_{\beta t})$ can be regarded to as a "least-square version" of the Master Problem of Dantzig-Wolfe decomposition. The Quadratic Dual of $(\Delta_{\beta t})$ is

$$(\Pi_{\beta t}) \qquad \max_{\boldsymbol{d}}\{\ \varphi_\beta(\boldsymbol{d}) - {}^{1}\!/_{2t}\,\|\boldsymbol{d}\|^2\ \}$$

where $\varphi_\beta(\boldsymbol{d}) = \min_{i\in\beta}\{\ \alpha_i + \boldsymbol{g}_i\boldsymbol{d}\ \}$ is the *Cutting Plane model*, i.e. the polyhedral upper approximation of φ built up with the first order information collected so far. A *penalty term* (sometimes called *stabilizing term*), weighted with t, penalizes "far away" points in which φ_β is presumably a "bad" approximation of φ (Figure 1): hence, t measures our "trust" in the *Cutting Plane* model as we move farther from the current point, playing a role similar to that of the *trust radius* in Trust Region methods. Once $\boldsymbol{d}$ has been found, the value of $\varphi(\bar\lambda + \boldsymbol{d})$ and the relative subgradient are used to adjust t[47, 63]. The current point $\bar\lambda$ is moved (a *Serious Step*) only if a "sufficient" increase in the value of φ has been attained: otherwise (a *Null Step*), $\bar\lambda$ is not changed and the newly obtained first-order information is used to enrich the *Cutting Plane* model. Actually, when solving *MMCF* the "$\lambda \geq \mathbf{0}$" constraints must be taken into account: this leads to the extended problem

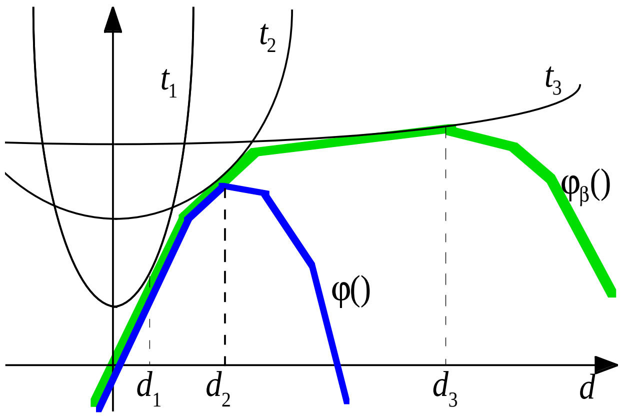

Figure 1: effect of the stabilizing term; for $t_1 < t_2 < t_3$, d_i are the optimal solutions

$$(\Pi'_{\beta t}) \qquad \max_{\boldsymbol{d}}\{\ \varphi_\beta(\boldsymbol{d}) - {}^{1}\!/_{2t}\,\|\boldsymbol{d}\|^2 : \boldsymbol{d} \geq -\bar\lambda\ \}$$

that guarantees feasibility of the tentative point, and can still be efficiently solved[24].

Our Bundle algorithm is described in Figure 2. At each step, the *predicted increase* $v = \varphi_\beta(\boldsymbol{d})$ is compared with the *obtained increase* $\Delta\varphi = \varphi(\lambda) - \varphi(\bar\lambda)$, and a *Serious Step* is performed only if Δφ is large enough relative: in this case, t can also be increased. Otherwise, the "reliability" of the newly obtained subgradient is tested by means of the "average linearization error" σ: if $\boldsymbol{g}$ is not believed to improve the "accuracy" of φ_β, t is decreased. The IncreaseT() and DecreaseT() functions are implemented as shown in Figure 3: both the formulas are based on the idea of constructing a quadratic function that interpolates the restriction of φ along $\boldsymbol{d}$ passing through $(\lambda, \varphi(\lambda))$ and $(\bar\lambda, \varphi(\bar\lambda))$, and choosing t as its maximizer. The latter formula is obtained by assigning the value of the derivative in $\bar\lambda$[47], while the former one is rather obtained by assigning the value of the derivative in λ[25]. The IncreaseT() heuristic guarantees that t will actually increase $\Leftrightarrow$ $\boldsymbol{dg} > 0$, which is exactly the (SS.ii) condition in [63]. Using two different heuristics has proven to be usually better than using only one of them.

```
                              MMCFB

⟨ choose $m_1 \in [0, 1]$ , $0 < m_2$ , $t_0 > 0$ , $t^* > 0$ and $\varepsilon > 0$ ⟩;
$\bar{\boldsymbol{y}} = \mathbf{0}$; $t = t_0$; ⟨ calculate $\varphi(\bar{\boldsymbol{y}})$ and $\boldsymbol{g}(\bar{\boldsymbol{y}})$ ⟩; $\beta = \{ ( \boldsymbol{g}(\bar{\boldsymbol{y}}) , 0 ) \}$;
repeat
    ⟨ β-strategy: eliminate outdated subgradients ⟩;
    ⟨ solve $(\Delta'_{\beta t})$ and $(\Pi'_{\beta t})$ for $\theta$ and $\boldsymbol{d}$ ⟩; $\sigma = \alpha_\beta\theta$;
    if( $t^* \| \boldsymbol{d} \|^2 + \sigma \le \varepsilon\varphi(\bar{\boldsymbol{y}})$ )
        then STOP;
        else $\boldsymbol{y} = \bar{\boldsymbol{y}} + \boldsymbol{d}$;
             ⟨ calculate $\varphi(\boldsymbol{y})$, $\boldsymbol{g}(\boldsymbol{y})$ and the corresponding $\alpha$ ⟩;
             ⟨ add ( $\boldsymbol{g}$ , $\alpha$ ) to $\beta$ ⟩;
             if( $\Delta\varphi = \varphi(\boldsymbol{y}) - \varphi(\bar{\boldsymbol{y}}) \ge m_1\varphi_\beta(\boldsymbol{d})$ )
                 then $\bar{\boldsymbol{y}} = \boldsymbol{y}$; $\alpha = \alpha + \boldsymbol{dG}$; $t$ = IncreaseT();
                 else if( $\alpha > m_2\sigma$ )
                          then $t$ = DecreaseT();
until( STOP );
```

Figure 2: the “main” of the MMCFB code

```
⟨ let $M > 1$ , $m < 1$ and $t_M \ge t_0 \ge t_m$ be fixed ⟩;

IncreaseT()  =  max{ $t$ , min{ $t_M$ , $Mt$ , $2tv( v - \Delta\varphi )$ } }

DecreaseT()  =  min{ $t$ , max{ $t_m$ , $mt$ , $( \alpha + \Delta\varphi ) / 2\alpha$ } }
```

Figure 3: the heuristic *t*-strategies

The m_2 parameter, controlling the decrease of t, should be chosen “large” (≈ 3): the result is that t is changed almost only in the early stages of the algorithm’s operations. Smaller values of m_2 cause t to decrease too quickly, so yelding a very long sequence of “short” steps. This setting is remarkably different from the one usually suggested in the literature, i.e. ≈ 0.9[47, 63]: the reason might be that Lagrangean functions of Linear Programs *are* qualitatively different from other classes of nondifferentiable functions, having an enormous number of facets which makes them extremely “kinky”. The choice of the other parameters is far less critical: m_1 seems to be properly fixed to 0.1, the (absolute and relative) safeguards on t almost never enter into play and the heuristics for increasing and decreasing t are capable of correcting blatantly wrong initial estimates t_0. If t^* has been chosen sufficiently large, upon termination the current point $\bar{\lambda}$ is guaranteed to be an ε-optimal solution of *DMMCF*: although selecting t^* is in principle difficult, in practice it can be easily guessed by looking at a few test executions of the code.

An innovative feature of our code is the *Lagrangean Variables Generation* (LVG) strategy: rather than always working with the full vector of Lagrangean multipliers, we maintain a (hopefully small) set of “active” variables and only solve the restricted $(\Pi'_{\beta t})$. Every p_1 (≈ 10) iterations, we add to the active set all “inactive” variables having the corresponding entry of $\boldsymbol{d}$ strictly positive; this is also done for the first p_2 (≈ 30) iterations, and when the algorithm would terminate because the STOP condition is verified. Usually, almost all the active variables reveal themselves in the first few iterations, and the active set is very stable. Note that, from the primal viewpoint, the LVG strategy is a row generation scheme, where a complicating constraint is added only if it is violated by the primal unfeasible solution corresponding to $\boldsymbol{d}$[16].

The LVG strategy has a dramatic impact on the time spent in solving ($\Pi'_{\beta t}$), which is reduced by up to a factor of 5 even on small instances. Clearly, the LVG strategy can be beneficial only if the *QP* solver efficiently supports the online creation and destruction of variables of subproblem ($\Pi'_{\beta t}$): this is the case of the specialized solver used in the code[24], which employs a *two-level active set* strategy for handling the $\boldsymbol{d} \geq -\bar{\lambda}$ constraints. The availability of the specialized solver has been crucial for developing an efficient Bundle code: other than supporting the LVG strategy, it is much faster than non-specialized *QP* codes in solving sequences of ($\Pi'_{\beta t}$) problems. This has been shown in [24] by comparing it with two standard *QP* codes (interestingly enough, one of them being exactly the *QP* solver used for the previous Bundle approach to *MMCF*[54]), which have been outperformed by up to two orders of magnitude; since the *coordination cost* ranges from 1% to 20% of the total running time, a Bundle algorithm using a non-specialized *QP* solver would rapidly become impractical as the instances size increases.

To keep the size of the Bundle low, we delete all the subgradients that have $\theta_i = 0$ for more than a fixed number (≈ 20) of consecutive iterations. We do not perform subgradients aggregation, since it usually has a negative impact on the total number of iterations.

In most cases, and especially if k is large, most of the time is spent in solving the *MCF* subproblems; hence, it is important to use a fast *MCF* solver with efficient reoptimizing capabilities. However, in many classes of instances the *MCF* subproblems have a special structure that can be exploited: the typical case is when all individual capacities u^h_{ij} are either 0 or $+\infty$ and there is only one source node for each commodity, that is when the subproblems are Shortest Path Tree problems (*SPT*). In our C++ code, Object-Oriented Programming techniques have been used to accomplish this: *MCF* subproblems are solved with an implementation of the Relaxation algorithm[7], while *SPT* subproblems are solved with classical Shortest Path algorithms[35]. Even on small problems with few commodities, this gives a speedup of up to *four*.

Object-Oriented Programming techniques have also been used in order to make it easy to develop specialized or extended versions. In fact, we already developed a parallel version of the code[14, 25], as well as extended versions for computing lower bounds for the *Fixed Charge MMCF*[25, 15] and for the *Reserve Problem*[25]. Only minor changes would be required for solving the *Nonhomogeneous MMCF*, while the *Nonlinear commodity-separable MMCF* would only require the availability of a suitable nonlinear *MCF* solver.

Finally, we remark that, unlike other dual algorithms, our code is a “complete” *MMCF* solver, in that upon termination it not only gives a dual optimal solution $\bar{\lambda}$, but also a primal optimal solution

$$\bar{\boldsymbol{x}} = [\ \bar{\boldsymbol{x}}^1\ \dots\ \bar{\boldsymbol{x}}^k\ ] = \Sigma_{i \in \beta}\ [\ \boldsymbol{x}^1_i\ \dots\ \boldsymbol{x}^k_i\ ]\theta_i$$

where $\boldsymbol{x}^h_i$ is the solution of the h-th *MCF* with costs $\boldsymbol{c}^h + \lambda_i$ and θ is the solution of the latest ($\Delta'_{\beta t}$). Obtaining $\bar{\boldsymbol{x}}$ requires keeping the flow solutions of the k *MCF*s relative to any subgradient in β, hence it may considerably increase the memory requirements of the code; therefore, this feature has been made optional in the code, and in our experiments we have not used it. The impact on the running times, however, is negligible (always well under 5%).

3. The other codes

One of the objectives of this research was to understand where a modern Cost Decomposition code stands, in terms of efficiency, among the solution methods proposed in the literature: hence, an effort has been made to compare MMCFB with recent alternative *MMCF* solvers. The efficiency of Cost Decomposition methods is controversial: although they have been used with success in some applications, many researchers[37, 38] would agree that "... the folklore is that generally such schemes take a long time to converge so that they're slower than just solving the model as a whole, although research continues. For now my advice, unless [...] your model is so huge that a good solver can't fit it in memory, is to not bother decomposing it. It's probably more cost effective to upgrade your solver, if the algorithm is limiting you ..."[27]. However, if Cost Decomposition approaches are of any value, *MMCF* is probably the application where such a value shows up, as demonstrated by our experiments.

3.1 A general-purpose LP solver

In principle, any *LP* solver can be used to solve *MMCF*: actually, our experience shows that (according to above statement) commercial general-purpose *LP* solvers can be the *best* choice for solving even quite large instances. This should not be surprising, since these codes are the result of an impressive amount of work and experience obtained during many years of development, and exploit the state of the art in matrix factorization algorithms, pricing rules, sophisticated data structures, preprocessing and software technology. CPLEX 3.0[17] is one of the best commercial *LP* solvers on the market: it offers Primal and Dual simplex algorithms, a Network simplex algorithm for exploiting the embedded network structure of the problems and a Barrier (Interior Point) algorithm.

Size	PP	PD	HP	HD
5125	9.02	2.72	4.73	1.36
16000	50.90	23.47	21.37	6.07
25000	81.05	14.15	22.65	5.41
47850	504.04	124.33	25.48	10.06
68225	1393.63	42.24	18.96	8.10
61275	1646.35	254.13	185.04	78.94
207733	16707.66	492.33	244.89	44.67
257700	13301.88	556.39	165.56	48.55

Table 1: different simplex options for CPLEX

Size	HD1	HD2	HD3	HD4
5125	4.08	1.36	2.11	1.34
16000	19.65	5.99	11.00	6.08
25000	18.24	6.16	19.71	5.39
47850	13.35	11.95	51.03	10.00
68225	8.05	10.76	71.59	8.06
61275	1000*	54.07	152.36	79.16
207733	1000*	88.38	681.27	44.88
257700	57.53	92.58	846.65	48.48

Table 2: different pricing rules for CPLEX

In our experimentation, to make the comparisons meaningful, efforts have been made to use CPLEX in its full power by repeatedly solving subsets of the test problems with different combinations of the optimization parameters, in order to find a good setting. CPLEX can identify the network structure embedded in a *LP*, and efficiently solve the restricted problem to obtain a good starting base for the "Pure" Primal and Dual (*PP* and *PD*) simplex algorithms: the performances of the resulting "Hybrid" Primal and Dual algorithms (*HP* and *HD*) are reported in Table 1, where $Size = mk$ is the number of variables of the *LP* and figures are the average running times over groups of several problems of the same size. From these results, we see that a Dual approach can be 30 times faster than a Primal one, and that the "warm start" speeds up the solution by up to a factor of 70, so that the Hybrid Dual version can be *400 times* faster than the Pure Primal one. Hence, it is not unfair to regard CPLEX as a *semi-specialized* algorithm for network-structured problems. Other details, such as the pricing strategy, can have a dramatic impact on the number of iterations and hence on the performance: Table 2 reports a comparison between the four available pricing rules for the Dual simplex, the standard dual pricing *HD1* and three variants of the steepest-edge rule *HD2-4*. Performance improvements of up to a factor of 15 (and more, since the instances marked with "*" had taken much longer than 1000 seconds to terminate) can be obtained by selecting the appropriate rule.

3.2 A Primal Partitioning code

PPRN 1.0[12] is an available (in .a format at ftp://ftp-eio.upc.es/pub/onl/codes/pprn/libpprn) recent implementation of the Reduced Gradient algorithm for the Nonlinear *MMCF*, i.e. a Primal Partitioning approach when applied to the linear case. In their computational experience[18], the authors show that it generally outperforms the known Primal Partitioning code MCNF85[46]. In our experience, PPRN has never shown to be dramatically faster then CPLEX: however, this should not lead to the conclusion that it is inefficient. In fact, PPRN is a primal code, and it is definitely much faster than the primal simplex of CPLEX: hence, it can be regarded as a good representative of its class of methods.

Again, an appropriate tuning of its (several) parameters can significantly enhance the performances. The most important choice concerns the *Phase 0* strategy, i.e. whether the starting base is obtained from any feasible solution of the *k MCF* subproblems rather than from their optimal solutions (as in the "warm start" of CPLEX). As shown in Table 3, exploiting the optimal solutions decreases the running times by up to a factor of 20 (*Opt* vs. *Feas* columns): moreover, appropriate selection of the pricing rule[12] can result in a speedup of 3, as shown in Table 4.

Size	Opt	Feas
7125	2.21	6.89
18250	7.87	29.62
28750	7.76	53.68
50783	9.43	43.24
63850	32.33	145.39
91250	11.24	271.73
207867	55.62	425.68
268200	29.28	626.73

Table 3: Phase 0 options for PPRN

Size	P1	P3
7125	21.99	64.02
18250	8.40	41.41
28750	30.13	32.84
50783	8.98	25.99
63850	4.67	5.88
91250	42.06	53.50
207867	31.85	36.33
268200	8.90	16.38

Table 4: pricing rules for PPRN

3.3 Interior Point codes

A beta version of IPM, a specialized Interior Point code for *MMCF*[19], has been made available by the author for comparison purposes. Like most Interior Point methods, IPM must solve at each step a linear system with matrix $\boldsymbol{A}\Theta^{-2}\boldsymbol{A}^T$, where $\boldsymbol{A}$ is the full $nk \times m(k + 1)$ coefficient matrix of the *LP*. Rather than calculating a Cholesky factorization of such a matrix, however, IPM uses a Preconditioned Conjugate Gradient method where, at each step, k independent $m \times m$ subsystems are solved. The code does not currently handle zero individual capacities ($u_{ij}^h = 0$), that were present in all but one of the classes of instances that we tested; the obvious workaround, i.e. setting u_{ij}^h to some very small number and/or c^h_{ij} to some very large number, might result in numerical problems, yet the code was able to solve (almost) all the instances.

Since only indirect comparisons were possible for the other specialized Interior Point methods (Cf. §3.5), we also tested two general-purpose Interior Point *LP* solvers: CPLEX 3.0 Barrier (that will be referred to as IPCPLEX) and L0Q0 2.21[66, 65], a recent implementation of the *primal-dual, path following, predictor-corrector* approach. Also in this case, the parameters were tuned in order to get the best performances: the most important choice is the one between "myopic" (Minimum Local FillIn) and "global" (Priority Minimum Degree modified Cholesky and Multiple Minimum Degree respectively) heuristics for columns ordering when computing the LL^T factorization of $\boldsymbol{A}\Theta^{-2}\boldsymbol{A}^T$. For IPCPLEX, the myopic heuristic is more efficient; this is shown in Table 5, where for both of *MMD* and *MLF*, the two columns with "*-L*" and "*-R*" report the time obtained with Leftward and Rightward Cholesky factorization (another possible degree of freedom). The table also shows that the total number of iterations is very small, and that it is not always possible to find the "best" setting for a given parameter (MMD-L is better than MMD-D on small problems and worse on larger ones). For L0Q0, the global heuristic is uniformly better, but does not dramatically improve the performances; this difference is not surprising, since the two codes use different (*normal equations* and *reduced-KKT* respectively) approaches to matrix factorization.

k	*n*	*m*	*Size*	*Iter*	MMD-L	MMD-R	MLF-L	MLF-R
10	50	625	6250	12	37.70	30.97	33.24	27.84
10	50	625	6250	12	37.35	30.98	33.13	27.60
100	30	518	51800	12	3709.12	3344.74	1789.23	1829.65
100	30	519	51900	13	4036.83	3382.39	1927.01	1935.82

Table 5: some examples from CPLEX-Barrier optimization parameters tuning

3.4 A Column Generation code

A limited comparison was also possible with BCG, a modern Column Generation code[9], developed for repeatedly solving *MMCF* subproblems in a Branch & Price approach to the *Integer MMCF*, and using CPLEX for solving the "Master" problem at each iteration. The BCG code was not directly available; however, the authors kindly shared the test instances and the results (even some not reported in the article), allowing an indirect comparison. It should be noted that, in its present version, BCG only solves the *undirected MMCF*, although only minor modifications would be needed to extend it to the directed case. Hence, the comparison is not completely satisfactory. However, the efficiency of a Column Generation code depends on many important implementation details, and it would not have been fair to compare MMCFB with some (most likely naïve) implementation of ours. Hence, we preferred to keep this limited comparison with a sophisticated code such as BCG. The relative performances of *LP*-based Cost Decomposition methods like the Column Generation approach (but see also[25, 26]) and *QP*-based ones like MMCFB are currently under research, and will be the subject of a future article.

3.5 Other codes

The above codes do not exhaust all the proposed computational approaches to *MMCF*: indirect comparison have been made also with other Interior Point methods[70, 62], different Cost Decomposition approaches[61, 69] and ε-approximation algorithms[33]. The comparison concerns the PDS problems (Cf. §4.3), that are difficult *MMCF*s with large graphs but few commodities and *MCF* subproblems: hence, the results do not necessarily provide useful information on other classes of instances, such as those arising in telecommunications, which are characterized by a large number of commodities. Furthermore, the codes were ran on a heterogeneous set of computers, ranging from (vector or massively parallel) supercomputers to (clusters of) workstations, making it difficult to extract meaningful information from the running times reported.

No comparison at all was possible with Resource Decomposition approaches[49] or Primal-Dual methods[67]: however, the available codes are at least sufficient to draw an initial picture of the current scenery of computational approaches to *MMCF*. Hopefully, other tests will come to further enhance our understanding.

4. The test problems

The other fundamental ingredient for obtaining a meaningful computational comparison is an adequate set of test problems. Some data sets and random generators of *MMCF*s have been used in the literature to test some of the proposed approaches: however, many of these test problems are small, and even the larger ones have a small number of commodities. This is probably due to the fact that, with most of the methods, the cost for solving an M*MCF* grows rapidly with *k*, so that problems with hundreds of commodities have for long time been out of reach. For our experiments, some known data sets have been gathered that might be considered representative of classes of instances arising in practice: when no satisfactory test problems were available, different random generators have been used to produce data sets with controlled characteristics. All these instances, and some others, can be retrieved at http://www.di.unipi.it/di/groups/optimize/Data/MMCF.html together with an instance-by-instance output for all the codes listed in §3. We hope that this database will grow and will form the basis for more accurate computational comparisons in the future.

4.1 The Canad problems

The first data set is made of 96 problems, generated with two random generators (one for bipartite and the other for general graphs) developed to test algorithms for the *Fixed Charge MMCF* problem[15]. A parameter, called *capacity ratio*, is available as a "knob" to produce lightly or heavily capacitated instances. As *MMCF*s, these instances have proven a posteriori to be quite "easy"; on the contrary, the corresponding Fixed Charge problems are known to be "hard", and hence the efficient solution of "easy" *MMCF*s can still have an interest. Furthermore, this data set is the only one having instances where the number of commodities is considerably larger that the number of nodes (up to 400 vs. 30), as in some real applications. The instances are divided into the three groups:

– 32 bipartite problems (group A) with multiple sources and sinks for each commodity;

– 32 nonbipartite problems (group B) with single source/sink for each commodity;

– 32 nonbipartite problems (group C) with multiple sources and sinks for each commodity.

The characteristics of the instances are shown in Table 6 (§5.1): for each choice of (k, n, m), two "easy" and two "hard" instances have been generated.

4.2 The Mnetgen generator

Mnetgen[1] is a well-known random generator. An improved version has been developed for our tests, where the internal random number generator has been replaced with the standard `drand48()` routine, the output file format has been "standardized"[41] and a minor bug has been fixed. 216 problems have been generated with the following rules: the problems are divided in 18 groups of 12 problems each, each group being characterized by a pair (n, k) for n in $\{64, 128, 256\}$ and k in $\{4, 8, \ldots, n\}$ (as Mnetgen cannot generate problems with $k > n$). Within each group, 6 problems are "sparse" and 6 are "dense", with m / n respectively equal to about 3 and 8. In both these subgroups, 3 problems are "easy" and 3 are "hard", where an easy problem has mutual capacity constraints on 40% of the arcs and 10% of the arcs with an "high" cost, while these figures are 80% and 30% respectively for a hard problem. Finally, the 3 subproblems characterized by the 4-tuple (n, k, S, D) $(S \in \{\text{sparse, dense}\}, D \in \{\text{easy, hard}\})$ have respectively 30%, 60% and 90% of arcs with individual capacity constraints. Randomly generated instances are believed to be generally "easier" than real-world problems; however, the "hard" Mnetgen instances, especially if "dense", have proven to be significantly harder to solve (by all the codes) than both the "easy" ones and instances of equivalent size taken from the other sets of test problems.

4.3 The PDS problems

The PDS (Patient Distribution System) instances derive from the problem of evacuating patients from a place of military conflict. The available data represents only one basic problem, but the model is parametric in the *planning horizon t* (the number of days): the problems have a time-space network whose size grows about linearly with t. This characteristic is often encountered in *MMCF* models of transportation and logistic problems. However, in other types of applications k also grows with the planning horizon, while in the PDS problems k is a constant (11); hence, the larger instances have a very large m / k ratio.

4.4 The JLF data set

This set of problems has been used in [42] to test the DW approach to *MMCF*: it contains various (small) groups of small real-world problems with different structures, often with *SPT* subproblems and with never more than 15 commodities.

4.5 The dimacs2pprn "meta" generator

Several *MMCF*s, especially those arising in telecommunications, have large graphs, many commodities and *SPT* subproblems; all these characteristics are separately present in some of the previous data sets, but never in the same instance. To generate such instances, we developed a slightly enhanced version of the *dimacs2pprn* “meta” generator, originally proposed in [18], that allows a great degree of freedom in the choice of the problem structure. *Dimacs2pprn* inputs an *MCF* in DIMACS standard format (ftp://dimacs.rutgers.edu/pub/netflow), i.e. a graph G with arc costs $\bar{\boldsymbol{c}}$, node deficits $\bar{\boldsymbol{b}}$, arc capacities $\bar{\boldsymbol{u}}$ and three parameters k, r and f, and constructs an *MMCF* with k commodities on the same graph G. Deficits and capacities of the i-th subproblem are a “scaling” of those of the original *MCF*, i.e. k numbers $\{ r_1 .. r_k \}$ are uniformly drawn at random from $[1 .. r]$ and the deficits and individual capacities are chosen as $\boldsymbol{b}^h = r_h\bar{\boldsymbol{b}}$ and $\boldsymbol{u}^h = r_h\bar{\boldsymbol{u}}$; costs c^h_{ij} are independently and uniformly drawn randomly from $[0 .. \bar{c}_{ij}]$. The mutual capacities $\boldsymbol{u}$ are initially fixed to $f\bar{\boldsymbol{u}}$, and eventually increased to accommodate the multiflow obtained by solving the k *MCF*s separately (with random costs uncorrelated with the c^h_{ij}), in order to ensure the feasibility of the instance. At the user’s choice, the individual capacities can then be set to $+\infty$. This is a “meta” generator since most of the structure of the resulting *MMCF* is not “hard-wired” into the generator, like in Mnetgen, but depends on the initial *MCF*; in turn, this initial *MCF* can be obtained with a random generator. For our tests, we used the well-known Rmfgen[29], which, given two integer values a and b, produces a graph made of b squared grids of side a, with one source and one sink at the “far ends” of the graph. For all 6 combinations of a in $\{4, 8\}$ and b in $\{4, 8, 16\}$, an “easy” and a “hard” *MCF* have been generated with maximum arc capacities respectively equal to 1/2 and 1/10 of the total flow to be shipped. Then, for each of these 12 *MCF*s, 4 *MMCF*s have been generated, with k in $\{4, 16, 64, 256\}$, $r = 16$ and $f = 2k$ (so that the mutual capacity constraints were four times “stricter” than the original capacities $\bar{\boldsymbol{u}}$), yielding a total of 48 instances with up to one million of variables.

4.6 The BHV problems

Two sets of randomly generated *undirected MMCF*s have been used to test the Column Generation approach of [9]. In both sets, the commodities are intended to be Origin / Destination pairs, but the commodities with the same origin can be aggregated in a much smaller number of single origin - many destinations flows. The 12 “q*x*” problems all have 50 nodes, 130 arcs (of which 96 are capacitated) and 585 O/D pairs that can be aggregated into 48 commodities, yielding a total *LP* size of $6.2 \cdot 10^3$; a posteriori, they have proven to be quite easy. The 10 “r10-*x*-*y*” problems all have 301 nodes, 497 arcs (of which 297 are capacitated) and 1320 O/D pairs that can be aggregated into 270 commodities, yelding a total *LP* size of $1.3 \cdot 10^5$. The group is divided into 5 “r10-5-*y*” and 5 “r10-55-*y*” instances that are similar, but for the fact that the arc capacities in the first group are integral while those in the second one are highly fractional: the two subgroups have proven to be of a surprisingly different “difficulty”.

5. Computational results and conclusions

In this paragraph, we report on the comparison of MMCFB with the codes described in §3 on the test problems described in §4. The approaches being very different, it appears that the only performance measure which allows a meaningful comparison is the running time. In the following, we report CPU time in seconds needed to solve the different problems; the loading and preprocessing times have not been included. Information like the number of iterations (with different meanings for each code) or the time spent in different parts of the optimization process (e.g. for computing φ and solving ($\Delta_{\beta t}$)) has not been reported in the tables for reasons of space and clarity: all these data are however available at the above mentioned Web page.

Unless explicitly stated, the times have been obtained on a HP9000/712-80 workstation: for the codes run on different machines, the original figures will be reported and a qualitative estimate of the relative speed has been obtained—if possible—by comparing the SPECint92 and SPECfp92 figures (97.1 and 123.3 respectively for the HP9000/712-80). For MMCFB, CPLEX (both simplex and Interior Point) and L0Q0, the required relative precision was set to 10^{-6}: in fact, the reported values of the optimal solution always agree in the first 6 digits. PPRN do not allow such a choice, but it always obtained solutions at least as accurate as the previous codes. Upon suggestion of the authors, the relative precision required for IPM was instead set to 10^{-5}, as smaller values could cause numerical difficulties to the PCG: in fact, the values obtained were often different in the sixth digit. The precision of the BCG code is not known to us, but should be comparable: the other codes will be explicitly discussed later.

In all the following tables, b ($< m$), if present, is the number of arcs that have a mutual capacity constraint, and, for large data sets, each entry is averaged from a set of instances with similar *size* $= mk$. The columns with the name of a code contain the relative running time; “*” means that some of the the instances of that group could not be solved due to memory problems, “**” means that the code aborted due to numerical problems and “***” indicates that the computation had to be interrupted because the workstation was “thrashing”, i.e. spending almost all the time in paging virtual memory faults.

5.1 The Canad problems

The results of the three groups of instances A, B and C are summarized in Table 6: they are also visualized in Figure 4, Figure 5 and Figure 6 for the three groups separately, where the running time of each algorithm is plotted as a function of the *size*.

If compared to the faster code, the general-purpose Interior Point solvers are 2 to 3 orders of magnitude slower on groups B and C and 1 to 2 on group A; furthermore, they were not able to solve the largest instances due to memory or numerical problems. The specialized IPM is (approximately) 3 to 20 times faster on group A, 20 to 70 times faster on group B and 10 to 40 times faster on group C: however, it is almost never competitive with any of MMCFB, CPLEX and PPRN. Among those latter solvers, MMCFB is 10 to 40 times faster than CPLEX and 5 to 20 times faster than PPRN on all instances; an exception is the first half of group A, where MMCFB is less than 2 times slower. These are “difficult” small instances with few commodities, where MMCFB requires considerably more φ evaluations than for all the other problems. This may lead to the conjecture that MMCFB is not competitive on “difficult” instances, but the following experiences will show that this is true only if the number of commodities is also small.

	k	*n*	*m*	*size*	MMCFB	Cplex	PPRN	IPM	L0Q0	IPCplex
	10	50	400	4000	2.37	1.27	2.13	3.38	13.59	11.76
	10	50	625	6250	3.40	1.99	2.18	9.13	49.83	34.82
	10	100	1600	16000	3.01	3.66	4.66	17.87	100.09	* *
A	10	100	2500	25000	3.04	4.24	6.13	31.42	387.88	* *
	100	50	400	40000	0.76	10.31	10.23	18.53	472.95	358.47
	100	50	625	62500	2.43	89.96	29.79	42.49	2609.69	1729.01
	100	100	1600	160000	1.39	28.95	28.00	103.04	*	*
	100	100	2500	250000	1.46	36.51	27.03	174.03	*	*
	40	20	230	9200	0.06	0.90	0.26	6.11	193.53	100.64
	40	20	289	11560	0.06	1.14	0.28	3.71	154.13	176.97
	200	20	229	45800	0.24	5.56	2.02	26.61	645.80	988.65
B	100	30	517	51700	0.45	6.16	3.93	29.25	9895.41	1988.24
	200	20	287	57400	0.36	7.54	3.52	32.45	888.71	1343.36
	100	30	669	66900	0.25	7.07	1.64	32.11	14016.83	2915.02
	400	30	519	207600	1.03	30.18	24.96	145.15	*	* * *
	400	30	688	275200	1.19	38.32	23.32	170.14	*	* * *
	40	20	230	9200	0.23	1.02	0.37	5.64	223.79	108.88
	40	20	289	11560	0.15	1.22	0.31	5.79	182.43	189.19
	200	20	229	45800	0.90	7.86	6.23	22.92	727.55	1044.92
C	100	30	517	51700	1.10	8.05	6.51	25.99	12126.88	2156.04
	200	20	287	57400	0.51	6.93	4.28	25.85	947.17	1429.17
	100	30	669	66900	0.79	8.76	4.24	32.18	16093.50	3392.77
	400	30	519	207600	3.26	48.21	38.53	129.10	*	* * *
	400	30	688	275200	3.62	60.87	42.69	167.04	*	* * *

Table 6: results for groups A, B and C of the Canad problems

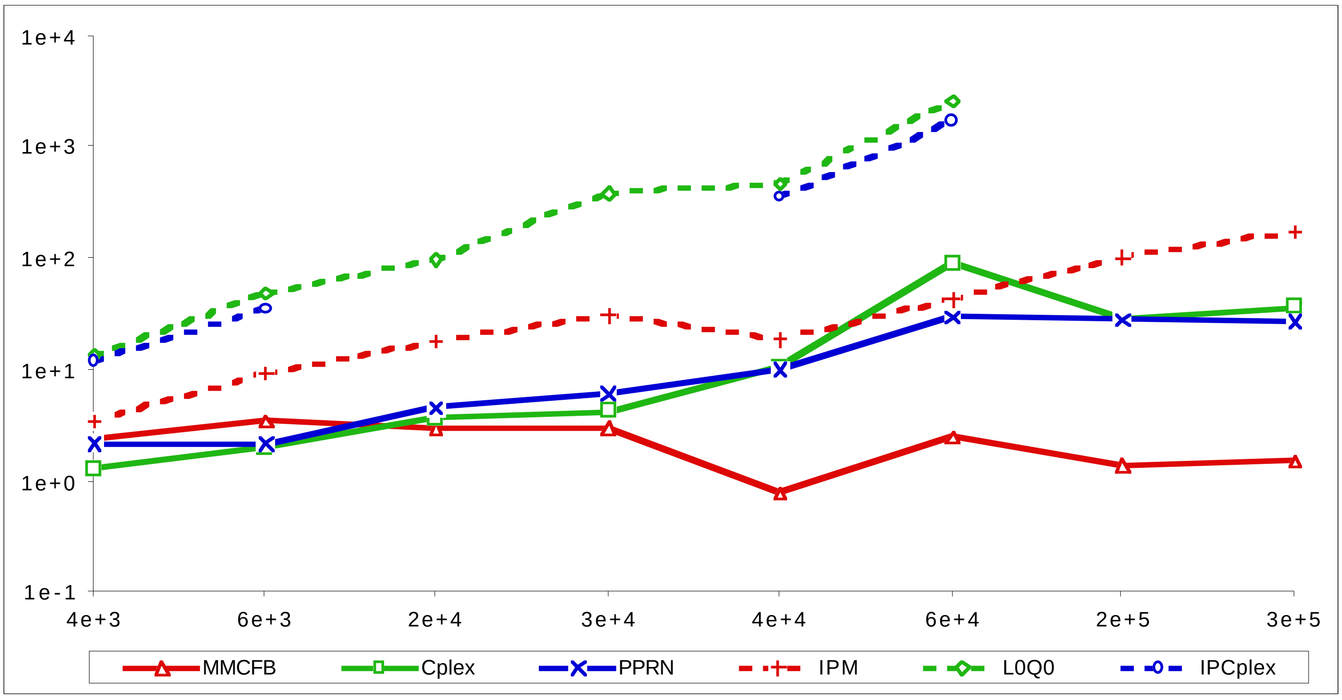

Figure 4: Canad problems, bipartite graphs (group A)

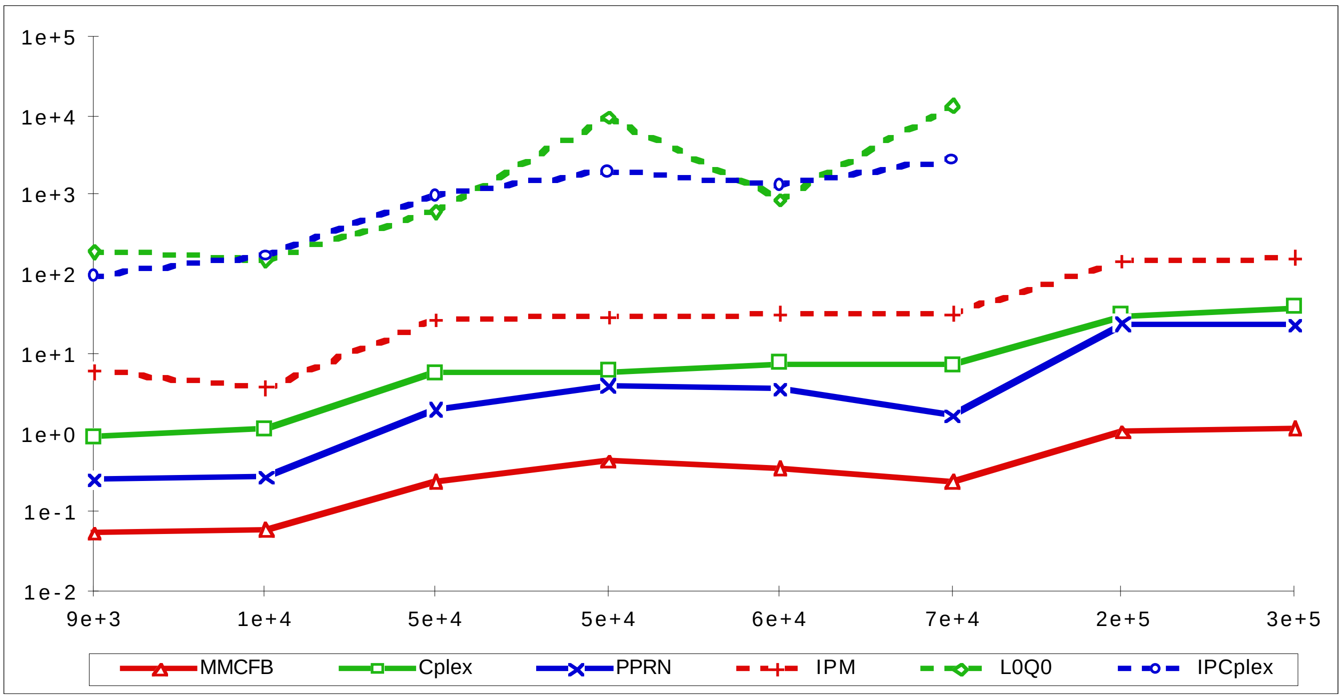

Figure 5: Canad problems, generic graphs (group B)

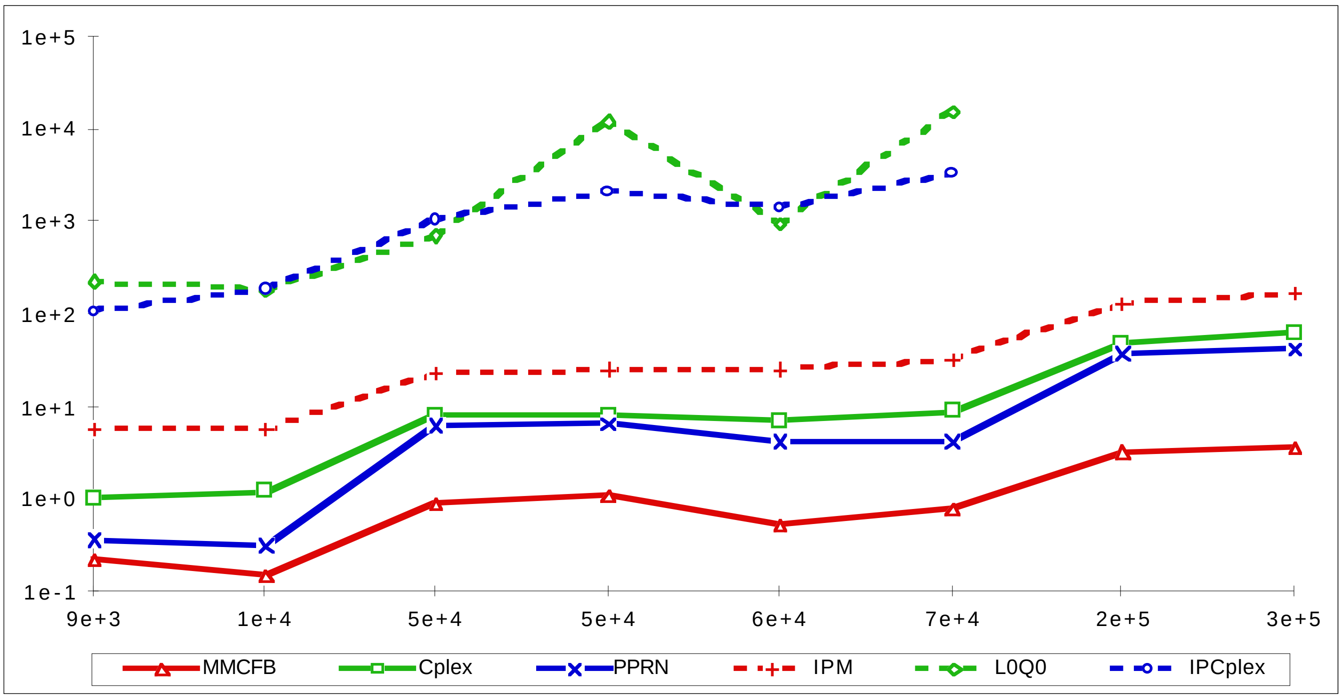

Figure 6: Canad problems, generic graphs (group C)

5.2 The Mnetgen problems

The Mnetgen problems have a wide range of different characteristics, from small to very large sizes (more than $5 \cdot 10^5$ variables), and from small to very large m / k ratios. Table 7 and Figure 7 summarize the results we have obtained; the times are the averages taken over 12 problems with the same (n, k).

k	n	m	b	*Size*	MMCFB	Cplex	PPRN	IPM	LOQO	IPCplex
4	64	362	148	1.4e+3	0.07	0.22	0.13	1.44	1.73	1.45
8	64	371	183	3.0e+3	0.26	0.50	0.52	4.26	9.22	7.56
16	64	356	191	5.7e+3	1.08	2.01	3.41	16.03	58.96	40.14
32	64	362	208	1.2e+4	3.42	12.99	22.04	43.27	190.76	98.73
64	64	361	213	2.3e+4	8.53	115.99	147.10	114.19	244.66	216.44
4	128	694	293	2.8e+3	0.58	0.54	0.85	6.45	7.18	6.49
8	128	735	363	5.9e+3	2.57	1.81	4.79	26.32	66.65	50.56
16	128	766	424	1.2e+4	11.30	17.31	40.57	116.26	683.47	394.19
32	128	779	445	2.5e+4	27.72	212.09	503.48	346.91	*	*
64	128	784	469	5.0e+4	44.04	1137.05	2215.48	719.69	*	*
128	128	808	485	1.0e+5	52.15	5816.54	6521.94	1546.91	*	*
4	256	1401	570	5.6e+3	7.54	2.38	9.88	51.00	50.70	40.37
8	256	1486	743	1.2e+4	25.09	15.48	105.89	208.10	568.02	377.87
16	256	1553	854	2.5e+4	60.85	180.06	955.20	844.09	*	*
32	256	1572	907	5.0e+4	107.54	1339.46	6605.45	1782.47	*	*
64	256	1573	931	1.0e+5	144.75	7463.14	18467.73	3441.62	*	*
128	256	1581	932	2.0e+5	223.13	35891.37	61522.94	9074.31	*	*
256	256	1503	902	3.8e+5	445.81	110897+	187156+	17279.00	*	*

Table 7: aggregated results on the Mnetgen problems

	4	8	16	32	64	128	256
64	2.93	1.90	1.87	3.80	13.60		
128	0.93	0.70	1.53	7.65	25.82	111.53	
256	0.32	0.62	2.96	12.46	51.56	160.85	251*

Table 8: ratio between CPLEX and MMCFB times - problems in the same row have fixed *n* and varying *k*

	4	8	16	32	64	128	256
64	1.73	1.98	3.16	6.45	17.25		
128	1.47	1.86	3.59	18.17	50.30	125.06	
256	1.31	4.22	15.70	61.42	127.58	275.72	424*

Table 9: ratio between PPRN and MMCFB times - problems in the same row have fixed *n* and varying *k*

	4	8	16	32	64	128	256
64	19.69	16.24	14.85	12.66	13.39		
128	11.14	10.23	10.29	12.52	16.34	29.66	
256	6.77	8.29	13.87	16.57	23.78	40.67	38.76

Table 10: ratio between IPM and MMCFB times - problems in the same row have fixed *n* and varying *k*

Again, the general-purpose Interior Point codes are not competitive, but this time the specialized IPM is better than both CPLEX and PPRN on the instances with large *k* (greater than 32): the speedup also increases with both *m* and *k*, reaching over an order of magnitude in the (256, 256) instances. However, MMCFB is even faster, outperforming the simplex-based codes by about three orders of magnitude on the largest instances. Furthermore, the hard and dense (256, 256) problems could be solved by neither CPLEX nor PPRN in reasonable time: they were stopped after having run for several days without produceing any result. Therefore, the corresponding entries in Table 7, marked with a "+", are actually (mild) estimates.

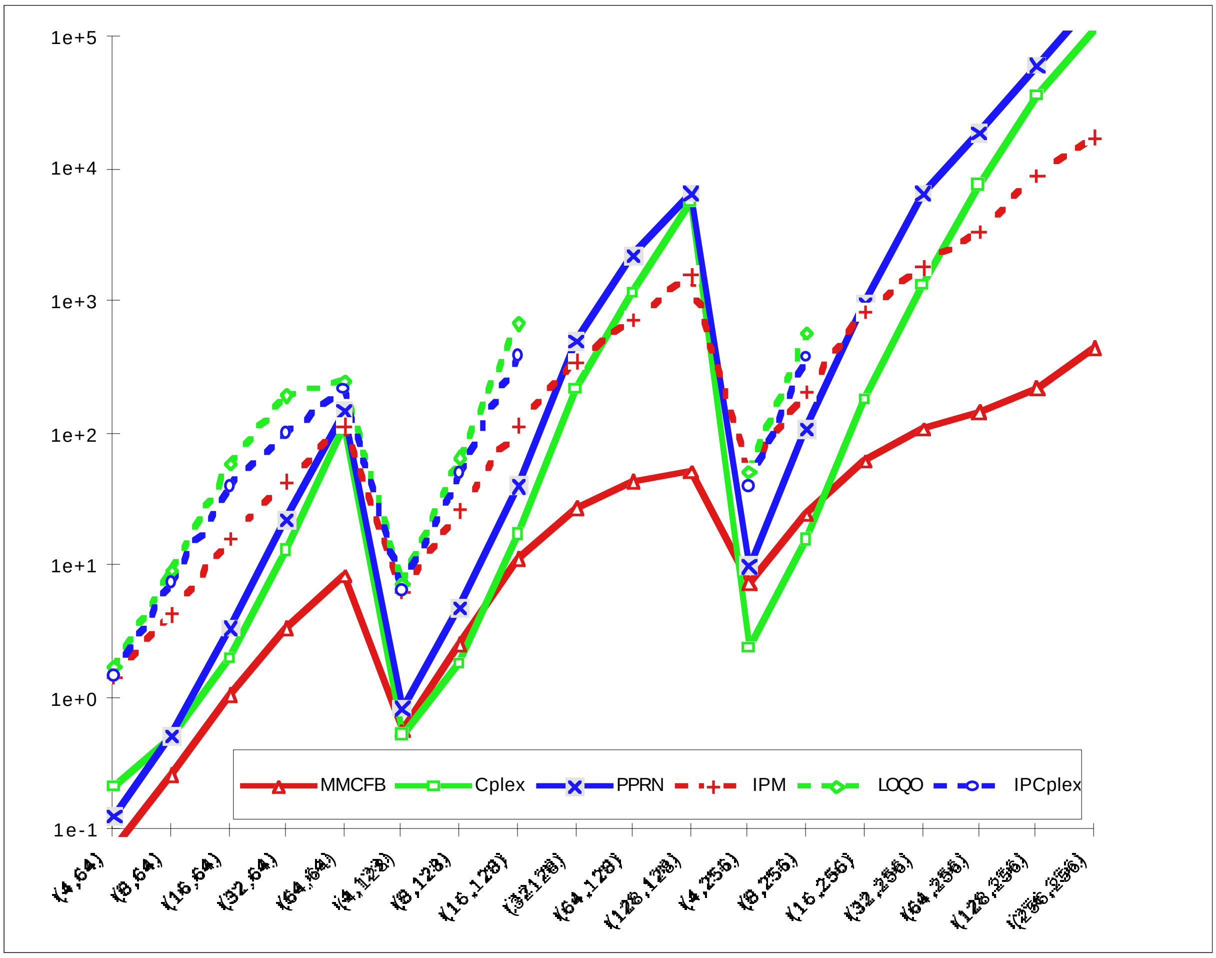

Figure 7: aggregated results on the Mnetgen problems

However, a closer examination of the results enlightens the weakness of MMCFB on large instances with few commodities. This is better seen in Table 8, Table 9 and Table 10, and the corresponding Figure 8, Figure 9 and Figure 10, where the ratio between the running times of CPLEX (respectively PPRN and IPM) and MMCFB is reported for different values of (n, k)—the entries marked with a "*" are those based on estimates, and should actually be larger. MMCFB is slower than CPLEX by up to a factor of 4 for instances with a large m / k ratio: more disaggregated data about some "critical" 256-nodes problems are reported in Table 11 for a better understanding of the phenomenon. Noticeably, "hard" problems (h) are really harder to solve, for all the codes, than "easy" ones (e) of the same size; moreover, they are much harder than, e.g., Canad problems of comparable size, since a $1.6 \cdot 10^5$ variables (100, 100, 1600) Canad instance can be solved in 1.5 seconds, while a hard $1.5 \cdot 10^5$ variables (64, 256, 2300) Mnetgen instance requires over 350 seconds.

On problems with few commodities, MMCFB is about 2 times slower than CPLEX when the instances are easy, and about 4 times slower in the case of hard ones. Conversely, as the number of commodities increases, the *relative efficiency* (the *Cplex%* and *PPRN%* columns) of MMCFB is larger on hard instances than on easy ones, and it increases with the problem size. This is probably due to the fact that the number of φ evaluations depends on the "hardness" of the instance, but much less on k, while the number of simplex iterations tends to increase under the combined effect of m and

k. This is true for both (the dual) CPLEX and (the primal) PPRN, possibly indicating that decomposition methods may *inherently* perform better on many-commodities problems.

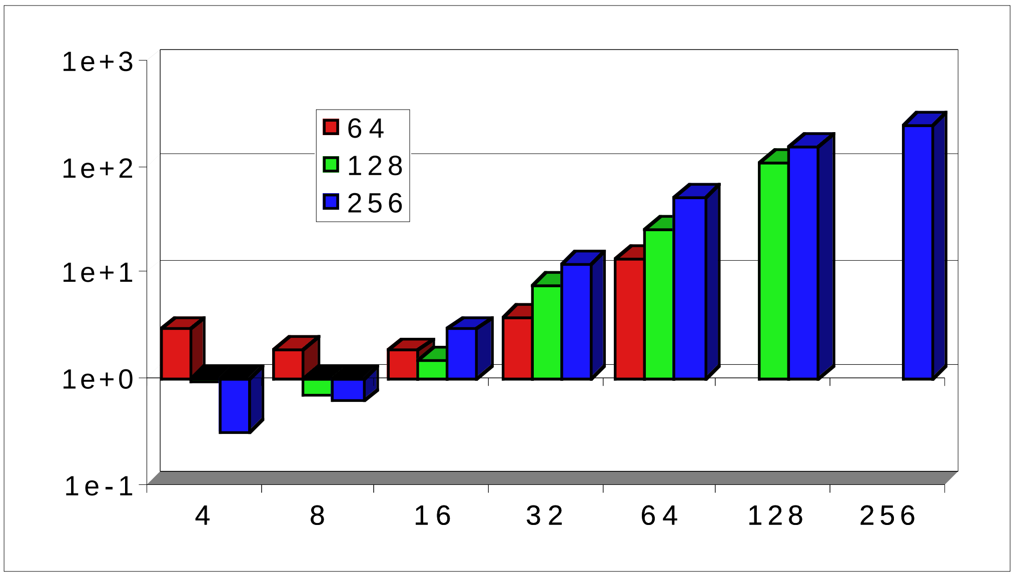

Figure 8: ratio between CPLEX and MMCFB times - for each k, the columns correspond to different n

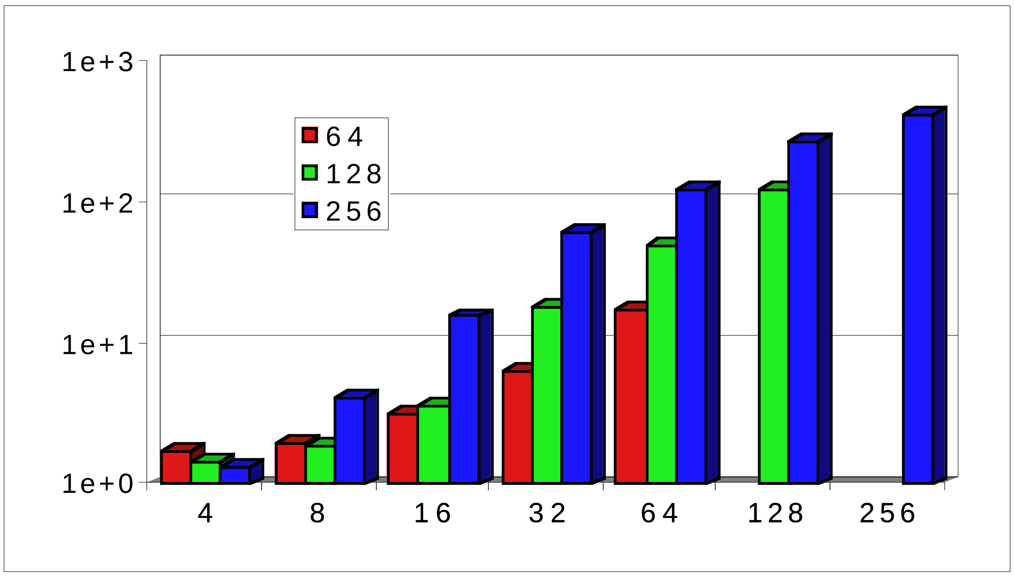

Figure 9: ratio between PPRN and MMCFB times - for each k, the columns correspond to different n

As we expected, IPM was less affected by the "difficulty" of the instance than all the other codes: in fact, the relative efficiency of MMCFB (the *IPM%* column) is lower on hard instances, confirming that the Interior Point approach is especially promising on difficult problems. However, since MMCFB is more and more competitive with IPM as k increases, IPM is likely to be competitive only on hard instances with small k: this is confirmed by the next data set, and by noting that (see Figure 10) the relative efficiency of MMCFB decreases as n increases for small k, but increases with n for large k. Yet, these results must be taken with some care: in all the Mnetgen instances there are arcs with $u^k_{ij} = 0$, that have been given a tiny capacity and a huge cost, and this workaround may have an adverse effect on the performance of IPM that is currently impossible to estimate.

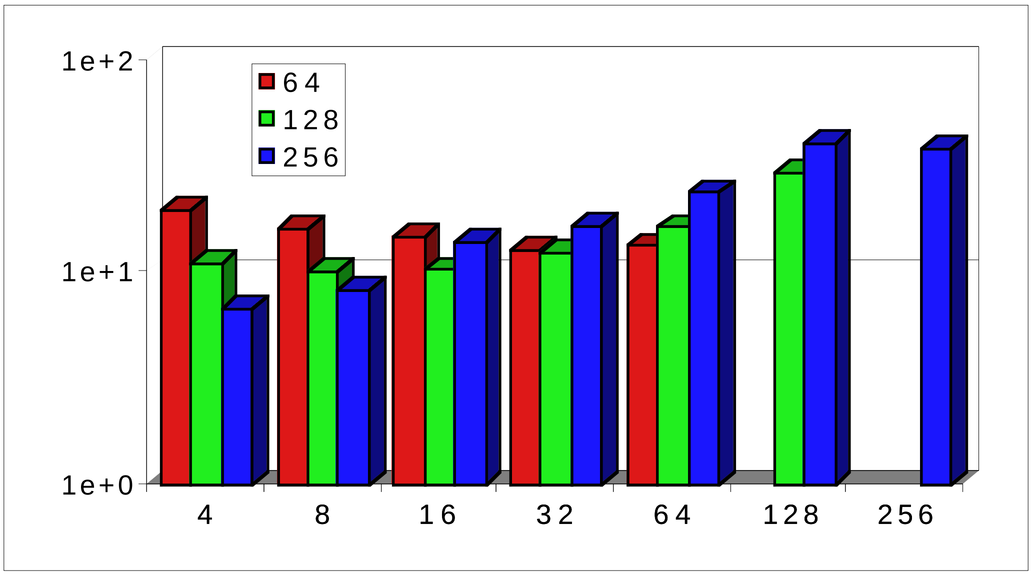

Figure 10: ratio between IPM and MMCFB times - for each *k*, the columns correspond to different *n*

k	*m*	type	MMCFB	Cplex	PPRN	IPM	Cplex%	PPRN%	IPM%
4	779	s / e	0.99	0.68	1.47	8.97	0.69	1.49	9.06
4	781	s / h	7.30	1.73	5.93	26.59	0.24	0.81	3.64
4	2021	d / e	3.21	1.76	5.88	42.06	0.55	1.83	13.09
4	2021	d / h	18.65	5.36	26.22	126.38	0.29	1.41	6.78
8	795	s / e	1.98	1.85	5.74	26.55	0.93	2.90	13.41
8	798	s / h	12.62	4.52	15.67	83.88	0.36	1.24	6.65
8	2171	d / e	21.32	10.09	55.33	181.09	0.47	2.60	8.49
8	2177	d / h	64.44	45.46	346.82	540.88	0.71	5.38	8.39
16	799	s / e	5.56	6.15	28.16	65.87	1.11	5.07	11.85
16	831	s / h	21.76	19.97	77.01	208.20	0.92	3.54	9.57
16	2280	d / e	45.74	54.53	272.50	735.33	1.19	5.96	16.08
16	2303	d / h	170.33	639.59	3443.10	2366.96	3.76	20.21	13.90
32	816	s / e	11.01	20.52	100.64	197.00	1.86	9.14	17.90
32	820	s / h	47.16	164.15	488.98	451.94	3.48	10.37	9.58
32	2331	d / e	79.03	464.85	1205.80	1391.59	5.88	15.26	17.61
32	2321	d / h	292.98	4708.31	24626.37	5089.33	16.07	84.06	17.37
64	825	s / e	20.20	116.21	522.45	437.24	5.75	25.86	21.64
64	801	s / h	54.73	362.61	1197.35	699.22	6.63	21.88	12.78
64	2337	d / e	146.11	5239.85	7298.90	3818.54	35.86	49.95	26.13
64	2330	d / h	357.97	24133.90	64852.23	8811.50	67.42	181.17	24.62

Table 11: disaggregated data about some 256-nodes problems

5.3 The PDS problems

As shown in Table 12 and Figure 11, several PDS instances were solved up to t = 40: the *LP* sizes range from $4.1\cdot10^3$ to $2.4\cdot10^5$, but k is always 11, so that the m / k ratio for PDS40 is about $2\cdot10^3$.

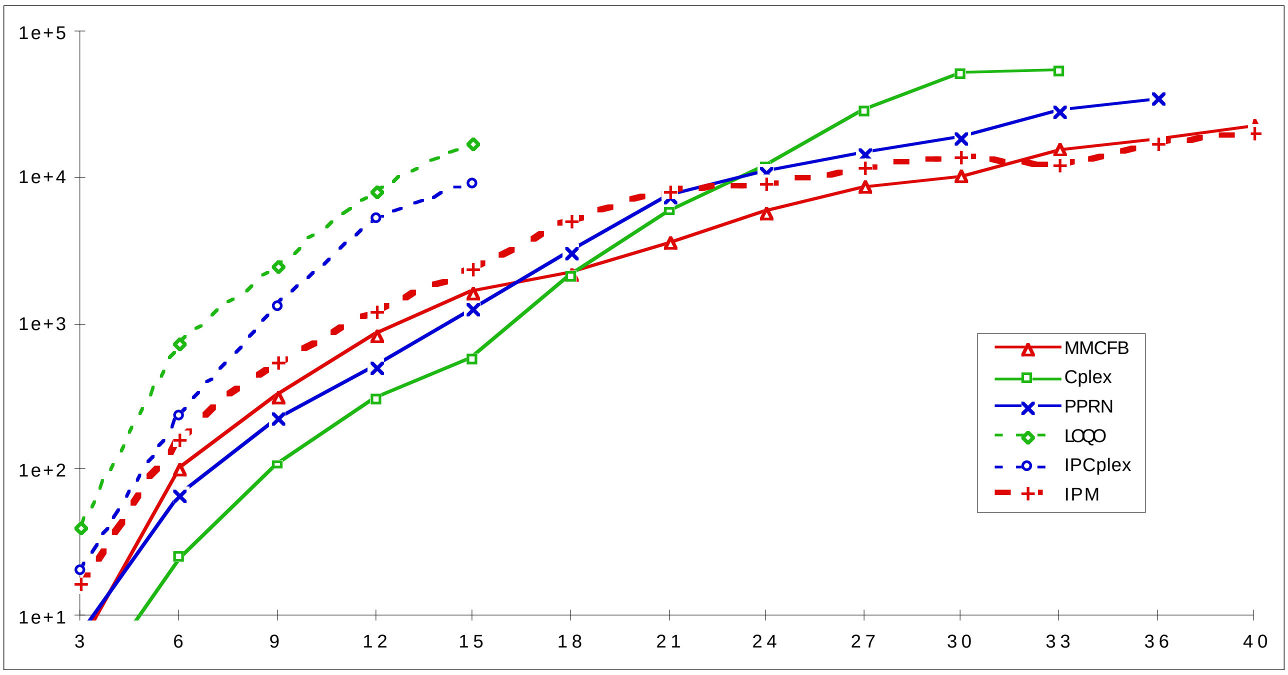

Figure 11: results for the PDS problems

t	*n*	*m*	*b*	*Size*	MMCFB	Cplex	PPRN	IPM	LOQO	IPCplex
1	126	372	87	4.1e+3	0.70	0.81	1.12	3.80	7.38	3.76
2	252	746	181	8.2e+3	5.66	2.12	5.60	13.38	23.57	14.52
3	390	1218	303	1.3e+4	13.10	5.32	15.88	32.22	93.91	42.93
4	541	1790	421	2.0e+4	50.18	12.87	33.74	73.98	316.32	102.07
5	686	2325	553	2.6e+4	127.03	19.54	60.16	120.34	799.49	215.09
6	835	2827	696	3.1e+4	129.92	41.97	112.3	296.30	1118.41	387.11
7	971	3241	804	3.6e+4	241.13	79.31	171.64	536.86	1860.80	635.53
8	1104	3629	908	4.0e+4	287.03	96.61	221.46	429.26	2171.42	1039.43
9	1253	4205	1048	4.6e+4	424.59	147.93	272.28	667.90	3523.90	2281.92
10	1399	4792	1169	5.3e+4	928.14	207.64	467.49	823.81	5437.82	3485.19
11	1541	5342	1295	5.9e+4	813.08	324.78	499.58	1167.99	7820.92	5961.47
12	1692	5965	1430	6.6e+4	828.58	373.35	560.25	1625.69	11785.1	6522.1
13	1837	6571	1556	7.2e+4	1033.7	315.61	945.13	2060.11	15922.9	8281.6
14	1981	7151	1684	7.9e+4	2198.5	524.05	1325.3	2172.29	19033.1	10276.9
15	2125	7756	1812	8.5e+4	1666.6	885.88	1431.1	3054.57	*	*
18	2558	9589	2184	1.1e+5	2237.47	2132.57	3093.51	5182.26	*	*
20	2857	10858	2447	1.2e+5	3571.58	3766.72	5213.73	8910.87	*	*
21	2996	11401	2553	1.3e+5	3541.34	5823.80	7475.30	8237.08	*	*
24	3419	13065	2893	1.4e+5	5796.18	11936.5	10817.8	9151.14	*	*
27	3823	14611	3201	1.6e+5	8761.89	28790.4	14670.2	11687.2	*	*
30	4223	16148	3491	1.8e+5	10343.2	51011.5	18995.0	13935.1	*	*
33	4643	17840	3829	2.0e+5	15459.3	54351.3	28869.6	12537.9	*	*
36	5081	19673	4193	2.2e+5	17689.2	*	34453.8	17519.3	*	*
40	5652	22059	4672	2.4e+5	22888.5	*	*	20384.2	*	*

Table 12: results for the PDS problems

The two general-purpose Interior Point codes were not able to solve instances larger than PDS15 due to memory problems, but even for the smaller instances they were always one order of magnitude slower than the simplex-based codes. For $t \leq 18$ CPLEX outperforms both PPRN and MMCFB, but as the size increases the gap lessens, and for the largest instances MMCFB outperforms CPLEX by a factor of four; also, for $t = 24$ PPRN is competitive with CPLEX, and for larger instances the specialized simplex is better than the generic one of a factor of two. However, neither CPLEX nor PPRN were able to solve PDS40 due to memory problems, while MMCFB solved the problem, although it required more than 6 hours. The only other code that was capable of solving all the instances is IPM; for $t \geq 33$, it has analogous—but slightly better—performance than MMCFB.

Since PDS problems have $m / n \approx 3$, it is possible (and instructive) to compare the results with that on hard (*.h*) and easy (*.e*) "sparse" Mnetgen instances: Table 13 shows that MMCFB solves a hard Mnetgen instance much faster than a PDS instance of similar size. The running times of the other methods for these pairs of Mnetgen and PDS instances are far less different: however, the difference tends to increase when $k >> 11$ in the Mnetgen instance. The PDS problems are considered difficult, but the "measure of difficulty" has never been explicitly stated: by taking it as the solution time required by a simplex method, it would be possible to assert that the Mnetgen instances are about as hard, but they can be efficiently solved by MMCFB. This statement can be criticized, however it is reasonable that a Cost Decomposition method performs better on problems where the overall "difficulty" is given by a large *k* rather than by a large *m*.

	k	*n*	*Size*	MMCFB	Cplex	PPRN	IPM
16.256.e	16	256	12789	5.56	6.15	28.16	65.87
16.256.h	16	256	13301	21.76	19.97	77.01	208.20
PDS3	11	390	13398	13.10	5.32	15.88	32.22
32.256.e	32	256	26123	11.01	20.52	100.64	197.00
32.256.h	32	256	26229	47.16	164.15	488.98	451.94
PDS5	11	686	25575	127.03	19.54	60.16	120.34
64.256.e	64	256	52779	20.20	116.21	522.45	437.24
64.256.h	64	256	51264	54.73	362.61	1197.35	699.22
PDS10	11	1399	52712	928.14	207.64	467.49	823.81
256.256.e	256	256	210859	183.37	4173.05	16150.89	3346.51
256.256.h	256	256	212480	316.59	16222.97	27625.40	6735.61
PDS33	11	4643	196240	15459.30	54351.30	28869.60	12537.90

Table 13: Mnetgen and PDS problems of the same size

	Cplex	MMCFB	ZaA	PZ	GK	SM	ZaG
1	1	1			7	840	
10	208	928	65	232	123	1920	45
20	3767	3572	497	1225	372	8220	253
30	51012	10343	557	5536	765	19380	654
40		22889	858		1448	38160	1434

Table 14: comparison with approaches from the literature for the PDS problems

On the PDS problems, it is also possible to try a comparison with some of the other approaches proposed in the literature. This is done in Table 14 with PDS1, 10, 20, 30 and 40.
The other approaches considered are:

– the parallel Nonlinear Jacobi Algorithm (ZaA) applied to the Augmented Lagrangean of [69], that was run on a Thinking Machine CM-5 with 64 processors, each one (apparently) capable of about 25 SPECfp92;

- the linear-quadratic exact penalty function method (PZ) of [61], that was run on a Cray Y-MP vector supercomputer, whose performances with respect to scalar architectures are very hard to assess;
- the ε-approximation algorithm (GK) of [33], that was run on a IBM RS6000-550 workstation (83.3 SPECfp92, 40.7 SPECint92); the largest instances were solved to the precision of 4-5 significant digits;
- the specialized Interior Point method (SM) of [62], that was run on a DECstation 3100 (whose SPEC figures are unknown to us) and that obtained solutions with 8 digits accuracy;
- a modification and parallelization of the above (ZaG) that was run on a cluster of SPARCstation 20 workstations [70]; unfortunately, the exact model is not specified, so that the (SPECfp92, SPECint92) can be anything between (80.1, 76.9) and (208.2, 169.4).

Unfortunately, extracting significant information from the figures is very difficult due to the different machine used. However, it is clear that, for these instances, the ε-approximation algorithm provides good solutions in a very low time: in Table 15, optimal objective function values (obtained with CPLEX), lower bounds obtained by MMCFB and upper bounds obtained by GK are compared. The reported *Gap*s for GK are directly drawn from [33], except for the last two problems for which an estimate of the exact optimal objective function value had been used in [33].

	Cplex	MMCFB		GK	
n	O.F. Value	O.F. Value	Gap	O.F. Value	Gap
1	29083930523	29083906658	-8e-07	2.90839E+10	3e-09
2	28857862010	28857838002	-8e-07	2.88579E+10	6e-08
3	28597374145	28597348614	-9e-07	2.85974E+10	1e-08
4	28341928581	28341903396	-9e-07	2.83419E+10	2e-07
5	28054052607	28054035803	-6e-07	2.80541E+10	3e-07
6	27761037600	27761015915	-8e-07	2.77611E+10	5e-07
7	27510377013	27510253762	-4e-06	2.75107E+10	1e-05
8	27239627210	27239603634	-9e-07	2.72399E+10	9e-06
9	26974586241	26974456167	-5e-06	2.69749E+10	1e-05
10	26727094976	26727038830	-2e-06	2.67280E+10	3e-05
20	23821658640	23821637841	-9e-07	2.38232E+10	7e-05
30	21385445736	21385443262	-1e-07	2.13888E+10	2e-04
40	18855198824	18855181456	-9e-07	1.88595E+10	2e-04

Table 15: comparison of the relative gap of MMCFB and GK

Among the other methods, the two parallel ones seem to provide good results, since MMCFB could not surely achieve a speedup larger than 11 (on the same machines): however, the comparison is very difficult, and it is perhaps better not to attempt any comment. A remark can instead be done about the concept of “solution” that is underneath some of the figures in the table: many of the codes—for instance (GK), (SM) and (ZaG)—use heuristic stopping criteria, essentially performing a fixed number of iterations before stopping. A posteriori, the solutions are found to be accurate, but the algorithm is not capable of “certifying” it by means of appropriate primal-dual estimates of the gap.

5.4 The JLF problems

	k	*n*	*m*	*b*	*Size*	MMCFB	Cplex	PPRN	IPM	LOQO	IPCplex
10term	10	190	510	146	5100	0.27	0.84	1.07	9.16	26.17	10.97
10term.0	10	190	507	143	5070	0.23	0.78	0.70	7.04	28.29	13.10
10term.50	10	190	498	134	4980	2.00	0.82	1.22	7.97	18.05	11.60
10term.100	10	190	491	127	4910	2.40	0.86	1.10	8.69	19.21	10.58
15term	15	285	796	253	11940	4.81	3.39	11.57	41.35	210.94	106.98
15term.0	15	285	745	202	11175	1.94	2.72	10.23	30.35	123.21	47.15
Chen0	4	26	117	43	468	0.30	0.25	0.32	0.25	0.44	0.38
Chen1	5	36	174	65	870	0.98	0.64	0.73	0.41	0.99	0.77
Chen2	7	41	358	155	2506	5.98	4.89	3.77	1.59	4.17	3.91
Chen3	15	31	149	56	2235	1.22	5.18	1.96	1.10	3.12	2.62
Chen4	15	55	420	176	6300	25.49	83.39	20.05	5.92	46.92	32.75
Chen5	10	65	569	242	5690	52.58	48.74	48.12	5.24	21.18	15.53
assad1.5k	3	47	98	98	294	0.03	0.11	0.09	0.36	0.39	0.22
assad1.6k	3	47	98	98	294	0.01	0.08	0.06	0.39	0.37	0.22
assad3.4k	6	85	204	95	1224	0.08	0.33	0.55	1.41	4.26	1.44
assad3.7k	6	85	204	95	1224	0.10	0.29	0.56	1.57	4.22	1.41

Table 16: results for some of the JLF problems

Table 16 shows the results obtained on some JLF problems, those with *SPT* subproblems: despite the small number of commodities, MMCFB is still competitive on the "*x*term.*y*" and "Assad*x*.*y*" sets. The "Chen*x*" problems are more "difficult" than the others, since they require an order of magnitude more time to be solved than the "*x*term.*y*" instances of comparable size. For this data set, the Interior Point codes are competitive with, and IPM largely outperforms, all the others, confirming the potential advantage of the Interior Point technology on "hard" instances. However, such small *MMCF*s are well in the range of general-purpose *LP* solvers, so that it makes little sense to use ad-hoc technology. In this case, other issues such as availability, ease-of-use and reoptimization should be taken into account.

5.5 The dimacs2pprn problems

Due to the previous results, the general-purpose Interior Point codes were not run on this data set, that contains the largest instances of all. As described in §4.5, for any given dimension a *hard* and an *easy* instance have been constructed: the results are separately reported in Table 17 and Table 18 respectively (note the difference in the number of mutual capacity constraints, *b*). As in the Mnetgen case, CPLEX ran out of memory on the largest instances, and PPRN was sometimes stopped after having run for a very long time without producing a result (the "+" entries): IPM also experienced memory problems on the largest instances, with 1.2 millions of variables. Note that a $2.5 \cdot 10^5$ variables *MMCF* requires about 11, 20 and 129 megabytes of memory to be solved by MMCFB, PPRN and CPLEX respectively: MMCFB requires an order of magnitude less memory than CPLEX.

The results obtained from these instances confirm those obtained from the previous data sets: MMCFB is faster than the two simplex-based codes on all but the smallest instances, IPM is competitive with both CPLEX and PPRN on instances with large *k* but MMCFB is even more efficient—for *k* = 256 it is always more than an order of magnitude faster than the other codes, approaching the three orders of magnitude as *m* increases. Yet, especially on large problems, MMCFB spends most of the time in the *SPT* solver, which suggests that substantial improvements should be obtained by making use of reoptimization techniques in the *SPT* algorithms.

The results are also illustrated by Figure 12, Figure 13, Figure 14, Figure 15, Figure 16 and Figure 17, where the ratio of the running times of CPLEX, PPRN and IPM vs. MMCFB is reported for "hard" and "easy" instances separately. It is interesting to note that the relative efficiency of MMCFB with respect to the simplex-based codes for $k = 4$ is lower on easy problems than on hard ones, while the converse is true for the relative efficiency of MMCFB with respect to IPM.

k	*n*	*m*	*size*	*b*	MMCFB	Cplex	PPRN	IPM
4	64	240	9.6e+2	240	0.46	0.32	0.37	1.23
4	128	496	2.0e+3	112	1.30	1.05	2.36	2.65
4	256	1008	4.0e+3	1007	21.50	8.85	14.82	23.13
4	256	1088	4.4e+3	192	0.36	1.38	2.22	8.86
4	512	2240	9.0e+3	447	5.97	14.39	32.26	63.44
4	1024	4544	1.8e+4	960	39.76	70.41	178.06	266.79
16	64	240	3.8e+3	240	0.44	1.23	1.68	7.58
16	128	496	7.9e+3	112	4.31	19.72	29.75	24.74
16	256	1008	1.6e+4	240	64.79	230.82	550.11	437.41
16	256	1088	1.7e+4	192	1.08	5.38	22.14	39.70
16	512	2240	3.6e+4	448	14.15	218.55	479.96	540.90
16	1024	4544	7.3e+4	960	69.59	1079.63	2796.85	2363.02
64	64	240	1.5e+4	240	2.89	50.70	92.77	51.55
64	128	496	3.2e+4	112	9.23	222.65	562.71	410.42
64	256	1008	6.5e+4	1008	94.07	3256.52	10011.30	4556.57
64	256	1088	7.0e+4	192	4.37	93.14	535.47	283.19
64	512	2240	1.4e+5	448	37.69	2362.37	7146.33	2703.51
64	1024	4544	2.9e+5	960	215.70	18763.9	52787.6	22899.4
256	64	240	6.1e+4	240	17.52	668.27	1415.55	258.87
256	128	496	1.3e+5	112	48.72	3525.27	11443.50	501.80
256	256	1008	2.6e+5	1008	458.36	56756.9	200000+	6069.46
256	256	1088	2.8e+5	192	15.30	821.86	7832.38	1475.91
256	512	2240	5.7e+5	448	218.84	*	138225.0	22266.8
256	1024	4544	1.2e+6	960	898.51	*	400000+	*

Table 17: results of the "hard" dimacs2pprn instances

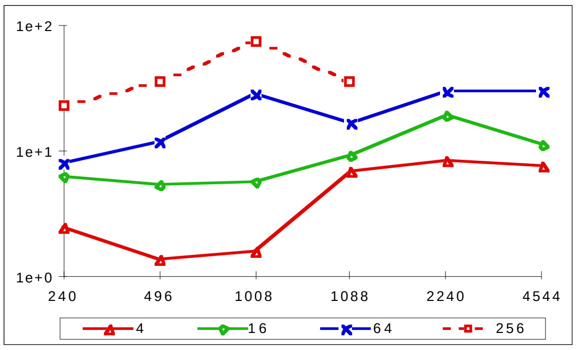

Figure 12: CPLEX / MMCFB ratio for easy problems

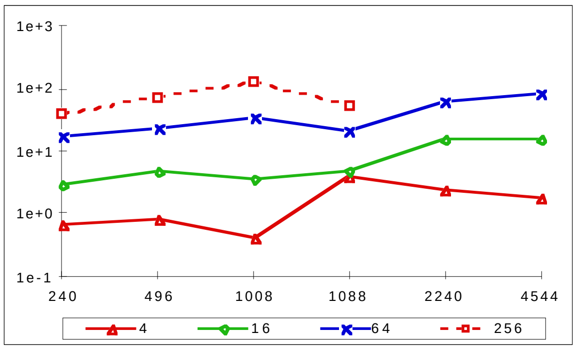

Figure 13: CPLEX / MMCFB ratio for hard problems

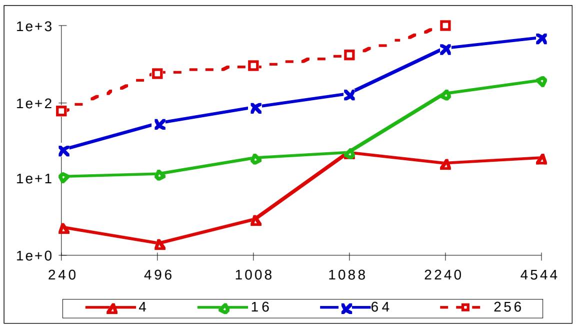

Figure 14: PPRN / MMCFB ratio for easy problems

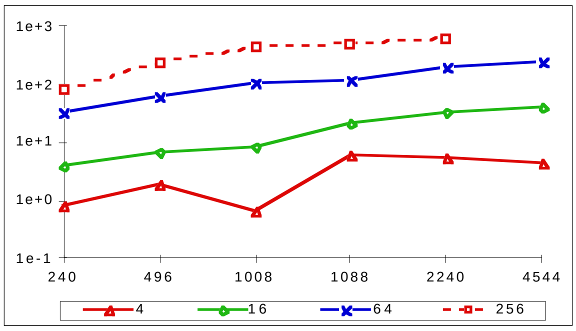

Figure 15: PPRN / MMCFB ratio for hard problems

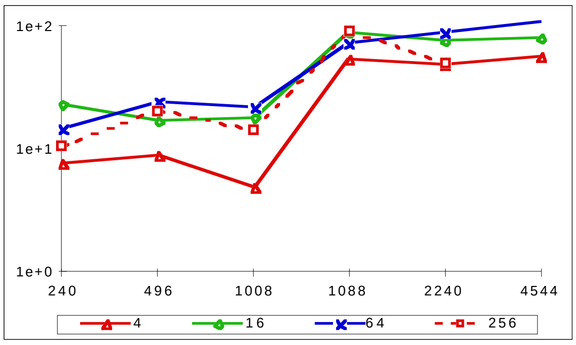

Figure 16: IPM / MMCFB ratio for easy problems

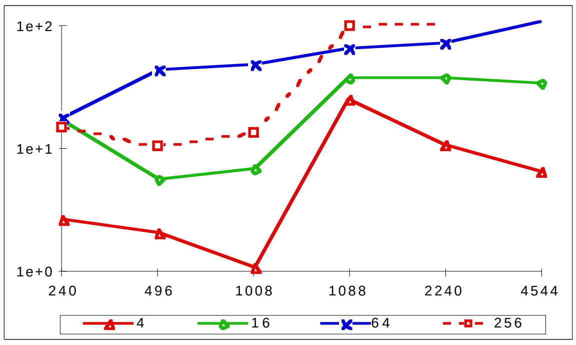

Figure 17: IPM / MMCFB ratio for hard problems

k	*n*	*m*	*size*	*b*	MMCFB	Cplex	PPRN	IPM
4	64	240	9.6e+2	48	0.07	0.17	0.16	0.54
4	128	496	2.0e+3	112	0.18	0.24	0.26	1.61
4	256	1008	4.0e+3	239	1.12	1.81	3.40	5.37
4	256	1088	4.4e+3	191	0.08	0.54	1.80	4.17
4	512	2240	9.0e+3	447	0.75	6.41	12.08	35.58
4	1024	4544	1.8e+4	960	1.84	14.02	34.84	101.49
16	64	240	3.8e+3	48	0.14	0.88	1.51	3.19
16	128	496	7.9e+3	112	0.66	3.50	7.67	10.88
16	256	1008	1.6e+4	240	3.53	20.27	65.31	63.66
16	256	1088	1.7e+4	192	0.37	3.42	8.05	31.60
16	512	2240	3.6e+4	448	2.24	43.30	275.89	166.14
16	1024	4544	7.3e+4	960	6.64	74.82	1286.26	514.74
64	64	240	1.5e+4	48	1.37	11.03	34.00	19.72
64	128	496	3.2e+4	112	2.12	25.80	109.97	51.61
64	256	1008	6.5e+4	240	15.75	445.28	1395.74	333.13
64	256	1088	7.0e+4	192	2.75	47.05	339.72	195.77
64	512	2240	1.4e+5	448	8.83	261.62	4492.95	754.25
64	1024	4544	2.9e+5	960	32.18	978.31	22511.3	3411.4
256	64	240	6.1e+4	48	9.21	208.90	694.94	94.94
256	128	496	1.3e+5	112	11.36	401.02	2662.57	228.16
256	256	1008	2.6e+5	240	96.34	7324.71	28435.0	1340.8
256	256	1088	2.8e+5	192	13.33	474.34	5270.80	1178.85
256	512	2240	5.7e+5	448	85.17	*	81419.1	4236.8
256	1024	4544	1.2e+6	960	159.33	*	400000+	*

Table 18: results of the “easy” dimacs2pprn instances

5.6 The BHV problems

	MMCFB		Cplex	BCG	
	Iter	*Time*	*Time*	*Iter*	*Time*
q1	42	0.63	7.93	4	5.06
q2	34	0.51	9.03	5	4.15
q3	42	0.63	12.45	5	5.15
q4	55	0.83	16.08	6	6.78
q5	85	1.33	20.3	6	6.78
q6	100	1.55	26.75	7	7.78
q7	100	1.58	31.07	7	8.69
q8	156	2.66	37.41	8	10.66
q9	221	3.76	44.96	8	10.60
q10	172	2.92	51.95	9	12.88
q11	196	3.37	61.58	8	12.55
q12	270	4.75	64.96	9	15.95

Table 19: results for the “p*x*” problems

	MMCFB		Cplex	BCG	
	Iter	*Time*	*Time*	*Iter*	*Time*
r10-5-1	25	8.73	6570.19	4391	662
r10-5-2	26	9.01	9249.35	4208	607
r10-5-3	38	13.02	6598.73	3237	398
r10-5-4	20	7.14	8481.97	4149	501
r10-5-5	23	8.16	9262.55	3633	420
r10-55-1	877	311.01	10843.10	2455	162
r10-55-2	934	330.21	12978.80	2690	199
r10-55-3	1243	445.27	13849.00	2694	224
r10-55-4	999	349.20	11581.60	2511	218
r10-55-5	877	318.00	*	2706	234

Table 20: results for the “r10-*x*-*y*” problems

Among the available specialized solvers, only MMCFB is capable of solving undirected instances; the three general-purpose *LP* solvers can be adapted for the task, but, due to the very poor results obtained by the IP codes, only CPLEX was actually run on these data sets. The times for the Column Generation algorithm (*BCG*) were obtained on an IBM RS6000-590 workstation, credited for 121.6 SPECint92 and 259.7 SPECfp92, i.e. essentially twice as fast as the HP9000/712-80: Table 19 shows that on the easy “p*x*” problems MMCFB is 10 times faster than BCG. The results on the other data set, shown in Table 20, are quite surprising: the two methods are essentially equivalent on the (highly fractional) “r10-55-*y*” instances, but MMCFB is able to solve the similar (integral) “r10-5-*y*” instances 30 times faster, while BCG needs about twice the time. A rationale for the behavior of MMCFB exists: a smaller number of trees is likely to be necessary to form the optimal flow as a convex combination in the integral data case. However, it is unclear why the same should not be true for the CG code: should it be proved that this is due to the Bundle-type approach, the result could be rather interesting.

5.7 Conclusions

The above computational experience shows that MMCFB can be a valuable tool for solving *MMCF*s with many commodities: to stress one figure, MMCFB solved the hard, dense (256, 256) Mnetgen instances in less than 20 *minutes*, but more than 6 and 10 *days* were needed by CPLEX and PPRN respectively, while IPM required a little less than 5 *hours*. As the experience with the dimacs2pprn data set shows, *MMCF*s with over one million of variables can be solved in less than 15 minutes of CPU time by a low-end computer, provided that the subproblems have a *SPT* structure. In a companion paper[14], we also show that a somewhat “naïve” parallelization of MMCFB obtains very good parallel efficiencies on a massively-parallel computer for the same instances for which the sequential code is more competitive, i.e. when *k* is large. Conversely, MMCFB does not appear to be a good choice for problems with a small number of commodities, unless the instances are very large: the PDS problems show that MMCFB can be competitive even on problems with m / k ratio larger than 1000, but the speedup with respect to simplex-based codes is far smaller than on many-commodities problem.

The general-purpose Interior Point solvers are not competitive with the other approaches, and they are not able to solve even moderately-sized instances due to excessive memory requirements. However, the specialized one dramatically improves on them and it is competitive on difficult problems with few commodities: it is also competitive with the simplex-based methods, although not with MMCFB, for large values of k. Among the latters, CPLEX is (sometimes consistently) faster than PPRN on the Mnetgen and dimacs2pprn problems, while the converse is true for the Canad ones: however, on the largest (difficult) PDS problems PPRN neatly outperforms CPLEX.

The Column Generation code was found (on a limited set of problems) to be less effective than MMCFB, even though in some cases not dramatically. For the subset of the problems where the speedup is really impressive, it is still not clear which features of the codes determine such difference, and further investigation is needed. Finally, the indirect comparison with methods from the literature seems to indicate that alternative approaches can be at least comparably efficient, or even much more efficient for rapidly finding approximate solutions, at least on some classes of problems. Even more clearly, however, they show the need for further computational studies that help in assessing the effectiveness of the various approaches on different classes of problems: we hope that this work proves to be useful as a step in this direction.

Acknowledgements

The authors wish to acknowledge all the people that, in various ways, have contributed to the research described in the present paper. C. Barnhart, B. Gendron, J. Castro Perez, P. Vance, R. Sarkissian, J.A. Shapiro, and R.J. Vanderbei have made available to us either their codes or their data or both. Without their valuable help, our experiments would have been much more limited. Finally, we wish to acknowledge Claudio Gentile for his excellent work in coding the largest part of the *MCF* solver used in our code.
This research has been supported by the Italian National Research Council (CNR) within the national program on transportation “Progetto Finalizzato Trasporti 2”.

References

1. A.I. ALI , J. KENNINGTON, 1977. MNETGEN program documentation *Technical Report IEOR* **77003**, Department of Industrial Engineering and Operations Research, Southern Methodist University.
2. A.I. ALI, J.L. KENNINGTON AND T.T. LIANG, 1993. Assignment with En-Route Training of Navy Personnel *Nav. Res. Log.* **40**, 581-592.
3. A.I. ALI , J. KENNINGTON AND B. SHETTY, 1988. The Equal Flow Problem *EJOR* **36**, 107-115.
4. R.K. AHUJA, T.L. MAGNANTI AND J.B. ORLIN, 1993. *Network Flows* Prentice Hall.
5. A.A. ASSAD, 1978. Multicommodity Network Flows—A Survey *Networks* **8**, 37-91.
6. C. BARNHART, 1993. Dual Ascent Methods for Large-Scale Multicommodity Flow Problems *Nav. Res. Log.* **40**(3), 305-324.
7. D.P. BERTSEKAS, 1991 *Linear Network Optimization* The MIT Press.
8. C. BARNHART, C.A. HANE, E.L. JHONSON AND G. SIGISMONDI, 1995. A Column Generation and Partitioning Approach for Multicommodity Flow Problems *Telecommunication Systems* **3**, 239-258.
9. C. BARNHART, C.A. HANE AND P. VANCE, 1996 Integer Multicommodity Flow Problems *Center for*

Transportation Studies Working Paper, Center for Transportation Studies, MIT.

10. S.N. BHATT, T. LEIGHTON , 1984. A Framework for Solving VLSI Layout Problems *J. Comp. and Syst. Sci.* **28**(2), 300-343.
11. C. BARNHART, Y. SHEFFY, 1993. A Network-Based Primal-Dual Heuristic for Multicommodity Network Flow Problems *Trans. Science* **27**(2), 102-117.
12. J. CASTRO, 1994 PPRN 1.0 User's Guide *DR* **94/06**, Statistics & Operations Research Dept., Univeristat Politècnica de Catalunya.
13. H. CHEN, C.G. DEWALD, 1974. A Generalized Chain Labelling Algorithm for Solving Multicommodity Flow Problems *Comput. & Op. Res.* **4**, 437-465.
14. P. CAPPANERA, A. FRANGIONI, 1996 Symmetric and Asymmetric Parallelization of a Cost-Decomposition Algorithm for MultiCommodity Flow Problems *Technical Report* **TR 36/96**, Dipartimento di Informatica, Università di Pisa (submitted to *INFORMS JOC*).
15. T.G. CRAINIC, A. FRANGIONI AND B. GENDRON, 1998 Bundle-based Relaxation Methods for Multicommodity Capacitated Fixed Charge Network Design Problems *Publication* **CRT-98-45**, Centre de Recherche sur les Transports, Université de Montreal (submitted to *Discrete Applied Mathematics*).
16. P. CARRARESI, A. FRANGIONI AND M. NONATO, 1996. Applying Bundle Methods to Optimization of Polyhedral Functions: An Applications-Oriented Development *Ricerca Operativa* Vol. **25** n. 74, 5-49.
17. CPLEX OPTIMIZATION, 1994. Using the CPLEX® Callable Library *Version* **3.0**, CPLEX Optimization Inc.
18. J. CASTRO, N. NABONA, 1996. An Implementation of Linear and Nonlinear Multicommodity Network Flows *EJOR* **92**, 37-53.
19. J. CASTRO, 1999. A Specialized Interior Point Algorithm for Multicommodity Network Flows *to appear on SIAM J. on Optimization*.
20. P. DHARMA, J.A. SHAPIRO, 1995. Augmented Lagrangeans for Linearly Constrained Optimization Using AMPL *CIV* **518** Final Report.
21. G.B. DANTZIG, P. WOLFE, 1960 The Decomposition Principle for Linear Programs *Op. Res.* **8**, 101-111.
22. L.R. FORD, D.R. FULKERSON, 1958 A Suggested Computation for Maximal Multicommodity Network Flows *Mgmt. Sci.* **5**, 79-101.
23. J.M. FARVOLDEN, W.B. POWELL AND I.J. LUSTIG, 1993. A Primal Partitioning Solution for the Arc-Chain Formulation of a Multicommodity Network Flow Problem *Op. Res.* **41**(4), 669-693.
24. A. FRANGIONI, 1996. Solving Semidefinite Quadratic Problems Within Nonsmooth Optimization Algorithms *Computers & O.R.* **23**(11), 1099-1118.
25. A. FRANGIONI, 1997 Dual-Ascent Methods and Multicommodity Flow Problems *Ph.D. Thesis* **TD 5/97**, Dipartimento di Informatica, Università di Pisa.
26. A. FRANGIONI, 1998 Generalized Bundle Methods *TR* **04/98**, Dip. di Informatica, Univ. di Pisa (submitted to *SIAM J. on Optimization*).
27. J.W. GREGORY, R. FOURIER, 1998. Linear Programming FAQ http://www.mcs.anl.gov/otc/Guide/faq/linear-programming-faq.html
28. A.M. GEOFFRION, G.W. GRAVES, 1971. Multicommodity Distribution System by Benders Decomposition *Mgmt. Sci.* **20**(5), 822-844.
29. D. GOLDFARB, M.D. GRIGORIADIS, 1988 A Computational Comparison of the Dinic and Network Simplex Methods for Maximum Flow *Annals of O.R.* **13**, 83-128.
30. J.-L. GOFFIN, J. GONDZIO, R. SARKISSIAN AND J.-P. VIAL, 1997. Solving NonLinear Multicommodity Flow Problems by the Analytic Center Cutting Plane Method *Math. Prog.* **76**(1), 131-154.
31. M.D. GRIGORIADIS, L.G. KAHCHIYAN, 1994. Fast Approximation Schemes for Convex Programs with Many Blocks and Coupling Constraints *SIAM J. on Optimization* **4**, 86-107.
32. M.D. GRIGORIADIS, L.G. KAHCHIYAN, 1996. Coordination Complexity of Parallel Price-Directive Decomposition *Math. of O.R.* **21**(2), 321-340.
33. M.D. GRIGORIADIS, L.G. KHACHIYAN, 1995. An Exponential-Function Reduction Method for Block-Angular Convex Programs *Networks* **26**(2), 59-68.
34. M.D. GRIGORIADIS, L.G. KAHCHIYAN, 1996. Approximate Minimum-Cost Multicommodity Flows in

$\tilde{O}(\varepsilon^{-2}KNM)$ time *Math. Prog.* **75**(3), 477-482.
35. G. GALLO, S. PALLOTTINO, 1988. Shortest Path Algorithms *Annals of O.R.* **13**, 3-79.
36. N. GARG, V.V. VAZIRANI AND M. YANNAKAKIS, 1993. Approximate Max-Flow Min-(Multi)Cut Theorems and their Applications *Proc. 25th STOC*, 698-707.
37. J.K. HO, E. LOUTE, 1981. An advanced implementation of the Dantzig Wolfe decomposition algorithm for linear programming *Math. Prog.* **20**(1), 303-326.
38. J.K. HO, E. LOUTE, 1983. Computational Experience with Advanced Implementation of Decomposition Algorithms for Linear Programming *Math. Prog.* **27**(3), 283-290.
39. J-B. HIRIART-URRUTY, C. LEMARÉCHAL, 1993. *Convex Analysis and Minimization Algorithms* A Series of Comprehensive Studies in Mathematics, vol. **306**, Springer-Verlag.
40. M. HELD, P. WOLFE AND H. CROWDER, 1974. Validation of Subgradient Optimization *Math. Prog.* **6**, 62-88.
41. K.L. JONES , I.J. LUSTIG, 1992. Input Format for Multicommodity Flow Problems *Program in Statistics and Operations Research, Department of Civil Engineering and Operations Research, Princeton University*.
42. K.L. JONES , I.J. LUSTIG, J.M. FARVOLDEN AND W.B. POWELL, 1993. Multicommodity Network Flows: The Impact of Formulation on Decomposition *Math. Prog.* **62**, 95-117.
43. P. KLEIN, A. AGRAWAL, R. RAVI AND S. RAO, 1990. Approximation Through Multicommodity Flow *Proc. 31th FOCS*, 726-727.
44. J.E. KELLEY, 1960. The Cutting-Plane Method for Solving Convex Programs *Journal of the SIAM* **8**, 703-712.
45. J.L. KENNINGTON, 1978. A Survey of Linear Cost Multicommodity Network Flows *Op. Res.* **26**(2), 209-236.
46. J.L. KENNINGTON, 1978. A Primal Partitioning Code for Solving Multicommodity Flow Problems *Technical Report* **79008**, Department of Industrial Engineering and Operations Research, Southern Methodist University.
47. K.C. KIWIEL, 1990. Proximity Control in Bundle Methods for Convex Nondifferentiable Optimization *Math. Prog.* **46**, 05-122.
48. A.P. KAMATH, N.K. KARMARKAR AND K.G. RAMAKRISHNAN , 1994. Computational and Complexity Results for an Interior Point Algorithm on Multicommodity Flow Problems *unpublished manuscript*.
49. S.A. KONTOGIORGIS, 1994. Alternating Directions Methods for the Parallel Solution of Large-Scale Block-Structured Optimization Problems *Technical Report* **MP-TR-94-13** (PhD thesis), University of Wisconsin-Madison.
50. J. KENNINGTON, M. SHALABY, 1977. An Effective Subgradient Procedure for Minimal Cost Multicommodity Fow Problems *Mgmt. Sci.* **23**, 994-1004.
51. R. DE LEONE, M. GAUDIOSO AND M.F. MONACO, 1993. Nonsmooth Optimization Methods for Parallel Decomposition of Multicommodity Flow Problems *Annals of O.R.* **44**, 299-311.
52. T. LEONG, P. SHOR AND C. STEIN, 1993 Implementation of a Combinatorial Multicommodity Flow Algorithm *Proc. DIMACS Conference, New Brunswick*.
53. T. LARSSON, M. PATRIKSSON AND A-B. STRÖMBERG, 1996. Conditional Subgradient Optimization—Theory and Applications*"* *EJOR* **88**, 382-403.
54. D. MEDHI, 1994. Bundle-Based Decomposition for Large-Scale Convex Optimization: Error Estimate and Application to Block-Angular Linear Programs *Math. Prog.* **66A**(1), 79-102.
55. O. DU MERLE, J.-L. GOFFIN AND J.-P. VIAL, 1996. On the Comparative Behaviour of Kelley's Cutting Plane Method and the Analytic Center Cutting Plane Method *Technical Report*, HEC, Université de Genève.
56. M. MINOUX, 1989. Network Synthesis and Optimum Network Design Problems: Models, Solution Methods and Applications *Networks* **19**, 313-360.
57. G.G. POLAK, 1992. On a Parametric Shortest Path Problem from Primal-Dual Multicommodity Network Optimization *Networks* **22**, 283-295.
58. W.B. POWELL, Y. SHEFFI, 1983. The Load Planning Problem of Motor Carriers: Problem Description and a Proposed Solution Approach *Trans. Res.* **17A**, 471-480.
59. S. PLOTKIN, D.B. SHMOYS AND É. TARDOS, 1991. Fast Approximation Algorithms for Fractional Packing and Covering Problems *Proc. 32th FOCS*, 495-505.
60. S.A. PLOTKIN, É. TARDOS, 1993. Improved Bounds on the Max-Flow Min-Cut Ratio for Multicommodity Flows *Proc. 25th STOC*, 690-697.

61. M.C. PINAR, S.A. ZENIOS, 1994. On Smoothing Exact Penalty Functions for Convex Constrained Optimization *SIAM J. Optimization* **4**(3), 486-511.

62. G.L. SHULTZ, R.R. MEYER, 1991. An Interior Point Method for Block-Angular Optimization *SIAM J. on Optimization* **1**(4).

63. H. SCHRAMM, J. ZOWE, 1992. A Version of the Bundle Idea for Minimizing a Nonsmooth Function: Conceptual Idea, Convergence Analysis, Numerical Results *SIAM J. Optimization* **2**(1), 121-152.

64. É. TARDOS, 1994. Approximate Min-Max Theorems and Fast Approximation Algoritms for Multicommodity Flow Problems *unpublished manuscript*.

65. R.J. VANDERBEI, 1994. LOQO: An Interior Point Code for Quadratic Programming *SOR* **94-15**, Princeton University.

66. R.J. VANDERBEI, T.J. CARPENTER, 1993. Symmetric Indefinite Systems for Interior Point Methods *Math. Prog.* **58**, 1-32.

67. S.A. ZENIOS, 1991. On the Fine-Grain Decomposition of Multicommodity Transportation Problems *SIAM J. on Optimization* **1**(4), 643-669.

68. A. ZAHORIK, L.J. THOMAS AND W.W. TRIGEIRO, 1984. Network Programming Models for Production Scheduling in Multi-Stage, Multi-Item Capacitated Systems *Mgmt. Sci*. **30**, 308-325.

69. A.A. ZAKARIAN, 1995. Nonlinear Jacobi and ε-Relaxation Methods for Parallel Network Optimization *Technical Report* **MP-TR-95-17**, (PhD thesis), University of Wisconsin-Madison.

70. G. ZAKERI, 1995. Multi-Coordination Methods for Parallel Solution of Block-Angular Programs *Technical Report* **MP-TR-95-08** (PhD thesis), University of Wisconsin-Madison.