Part II: Multicommodity Flows

II.1 Formulations, Applications and Approaches

Multicommodity Flow problems generalize the well-known single-commodity flow problems in that flows of different nature (*commodities*) co-exist in a given network: no commodity ever transforms into another commodity, so that each one has its own flow conservation equations, but using the same network they compete for the resources represented by the arc capacities. Multicommodity Flow problems share many characteristics of their single-commodity variants, most typically of being on the boundary between continuous and combinatorial problems, with the possibility of borrowing theoretical and algorithmic tools from both the sides: however, they do *not* inherit the integrality of the optimal solutions. Hence, most of the solution techniques developed over the years for *continuous* Multicommodity Flow problems are essentially based on their representation as (large-scale) linear or nonlinear problems, while *Integer* Multicommodity Flow problems, that retain most of the "combinatorial" flavour but are considered among the most difficult $\mathcal{NP}$-hard problems, call for the machinery of enumerative or polyhedral approaches.

Multicommodity Flow problems have attracted the researchers for several different reasons: from an applicative viewpoint, the Multicommodity Min-Cost Flow problem and its variants are (sub)models of a wide variety of transportation and scheduling problems, and their (repeated) solution is required within approaches to the original problem. Moreover, Multicommodity versions of the Max-Flow/Min-Cut relation are intimately related with $\mathcal{NP}$-hard graph partitioning problems, and have both theoretical and practical applications in the design of approximation algorithms for several problems arising e.g. in VLSI design. Finally, (MMCF) is the "archetype" of block-structured problems with "complicating" constraints, and has motivated many of the results for this broad and important class of optimization problems: for instance, the celebrated Dantzig-Wolfe decomposition approach was inspired by the work of Ford and Fulkerson on Multicommodity Max-Flow problems, but other (combinatorial or not) ε-approximation approaches to block-structured problems have been recently developed starting from (MMCF)-specific results.

In §II.1.1 and §II.1.2, the Min-Cost and Max-Flow formulations of Multicommodity Flow problems are discussed in details, with a stress on the characteristics that are especially relevant for the performances of the solution approaches; the other two paragraphs present bibliographic reviews respectively of some of the most important practical applications (§II.1.3) and of the main algorithmic approaches proposed so far (§II.1.4).

II.1.1 Multicommodity Min-Cost Flow formulations

Given a directed graph $G(\mathcal{N}, \mathcal{A})$, with n nodes and m arcs, and the set $\mathcal{K} = \{ 1 \dots k \}$, a *multicommodity flow* (or *multiflow*) on G is a vector $\boldsymbol{x} = [\, \boldsymbol{x}^1 \dots \boldsymbol{x}^k \,]$ of k distinct flow vectors $\boldsymbol{x}^h : \mathcal{A} \to \mathcal{R}^m$; with a little abuse of notation, "commodity" is used for indicating both $\boldsymbol{x}^h$ and its "name" h. The focus of Part II of the thesis is on the Linear Multicommodity Min-Cost Flow problem

$$\text{(MMCF)} \quad \begin{cases} \min \sum_{h \in \mathcal{K}} \sum_{(i,j) \in \mathcal{A}} c_{ij}^h x_{ij}^h & \\ \sum_{j:(i,j) \in \mathcal{A}} x_{ij}^h - \sum_{j:(j,i) \in \mathcal{A}} x_{ji}^h = b_i^h & \forall\ i \in \mathcal{N},\ \forall\ h \in \mathcal{K} \\ 0 \le x_{ij}^h \le u_{ij}^h & \forall\ (i,j) \in \mathcal{A},\ \forall\ h \in \mathcal{K} \\ \sum_{h \in \mathcal{K}} x_{ij}^h \le u_{ij} & \forall\ (i,j) \in \mathcal{A} \end{cases}$$

that requires to route the k commodities on G at minimal *total* (linear) *cost*, respecting the usual *node balance* constraints and *individual* (or *single-commodity*) capacity constraints $0 \le x_{ij}^h \le u_{ij}^h$ as well as *mutual* or *aggregate capacity* constraints $\Sigma_h x_{ij}^h \le u_{ij}$. When stated in matrix notation, using the *node-arc incidence matrix* $\boldsymbol{E}$ of G (Figure 1), the *node-arc formulation*

$$\text{(MMCF)} \quad \begin{cases} \min \Sigma_{h \in \mathcal{K}} \boldsymbol{c}^h \boldsymbol{x}^h & \\ \begin{bmatrix} \boldsymbol{E} & \cdots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{E} \\ \boldsymbol{I} & \cdots & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^1 \\ \vdots \\ \boldsymbol{x}^k \end{bmatrix} \le \begin{bmatrix} \boldsymbol{b}^1 \\ \vdots \\ \boldsymbol{b}^k \\ \boldsymbol{u} \end{bmatrix} & \\ \boldsymbol{0} \le \boldsymbol{x}^h \le \boldsymbol{u}^h & \forall\ h \in \mathcal{K} \end{cases}$$

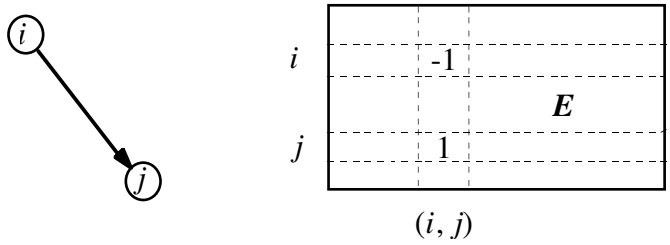

Figure 1: the node-arc incidence matrix

highlights the *block-structured* nature of the problem, i.e. that (MMCF) decomposes into k independent Min-Cost Flow subproblems if the mutual capacity (*linking*) constraints are relaxed: in this form, (MMCF) is a Linear Program with km variables and $kn + m$ (+ km) constraints. Although very sparse (each column has no more than three nonzeroes), such a (LP) becomes extremely large, even for small graphs, as the number of commodities increases; furthermore, the nice "combinatorial structure" of the (MCF) subproblems is in a large part destroyed by the mutual capacity constraints, since the optimal solution of the problem is *not* in general integral even though the data are. This is shown by the 3-commodities example of Figure 2, where one unit of commodity h has to be shipped from s_h to t_h on a (planar) graph with unit mutual capacities only in the "inner triangle": clearly, the optimal solution consists in sending 1/2 unit of each commodity along the path formed by three arcs with cost 1, and the other 1/2 unit along the single directed arc with cost 4.

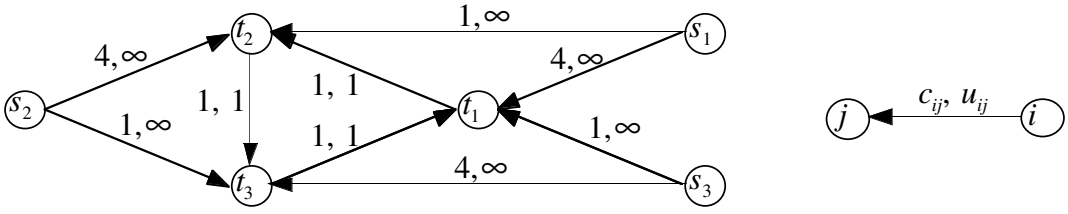

Figure 2: an instance with integral data and fractional optimal solution

Integrality is not the only property that is lost by passing from single-commodity to Multicommodity Flows: most notably, the celebrated Max-Flow/Min-Cut theorem is true only in some very special cases, as discussed in more details in the next paragraphs.

Note that, in Figure 2, no arc has an individual upper bound and, for each commodity $h \in \mathcal{K}$, $d_h > 0$ units of flow have to be shipped from the unique source s_h to the unique sink t_h: in such a case, the k independent (MCF)s obtained by relaxing the mutual capacity constraints are Shortest Path problems. This suggests a "path-based" formulation of the problem, since, being $P(h)$ the set of all the simple path from s_h to t_h, it is possible to define the function $h = \mathcal{K}(p)$ that maps each path p in $P = \bigcup_{h \in \mathcal{K}} P(h)$ to a commodity $h \in \mathcal{K}$ and write the problem as

$$\text{(MMCF)} \qquad \begin{cases} \min \sum_{p \in P} c_p x_p & \\ \sum_{p \in P(h)} x_f = d_h & \forall\ h \in \mathcal{K} \\ \sum_{p:\,(i,j) \in f} x_p \le u_{ij} & \forall\ (i,j) \in \mathcal{A} \\ x_p \ge 0 & \forall\ p \in P \end{cases}$$

where the variables x_p are now associated to (simple) paths rather than to arcs, and c_p is the cost of sending one unit of flow from $s_{\mathcal{K}(p)}$ to $t_{\mathcal{K}(p)}$ along p. Note that the vector of the variables can still be partitioned according to the commodity, i.e. $\boldsymbol{x} = [\, \boldsymbol{x}^1 \ \ldots\ \boldsymbol{x}^k \,]$, so that the *arc-path* formulation can be restated in matrix form as

$$\text{(MMCF)} \qquad \begin{cases} \min \boldsymbol{c}\boldsymbol{x} & \\ \begin{bmatrix} \boldsymbol{e}^T & & \boldsymbol{0} \\ & \ddots & \\ \boldsymbol{0} & & \boldsymbol{e}^T \\ \multicolumn{3}{c}{\boldsymbol{F}} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^1 \\ \vdots \\ \boldsymbol{x}^k \end{bmatrix} = \begin{bmatrix} d^1 \\ \vdots \\ d^k \\ \multicolumn{1}{c}{\boldsymbol{u}} \end{bmatrix} & \\ \boldsymbol{x} \ge \boldsymbol{0} \end{cases}$$

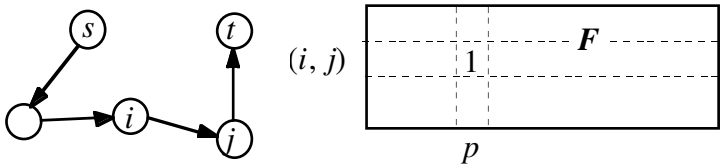

Figure 3: the arc-path incidence matrix

where $\boldsymbol{F}$ is the *arc-path* (or arc-chain) *incidence matrix* of G (Figure 3): this (LP) has exponentially many variables, but it has only $m + k$ constraints. This implies that an optimal solution exists (if any) that has at most $m + k$ strictly nonzero variables, i.e. no more than $m + k$ different paths are needed to deliver all the requested flow: hence, only a very small fraction of the variables will be needed in the optimal solution, clearly suggesting the use of column generation techniques.

The two formulations are essentially equivalent, but there are some differences that make one or the other more convenient for different purposes:

- the arc-path formulation is typically used when no individual upper bounds on the arcs are present: such bounds could be introduced as

  $$\sum_{p \in P(h):\,(i,j) \in p} x_p \le u_{ij}^h$$

  but this would increase the number of rows, possibly cancelling out the main advantage of this formulation; however, the special case $u_{ij}^h = 0$ is naturally accommodated (also in practice) within the arc-path formulation by deleting (i, j) from the graph $G^h(\mathcal{N}, \mathcal{A}^h)$ for commodity h;
- the arc-path formulation requires that all commodities have a single Origin/Destination pair, but it

can be extended as long as each path can be univocally assigned to a commodity: for instance, if all the source nodes are different but there is more than one sink, there will simply be one node balance constraint for each; in general, any problem (regardless of the structure of the demand vectors $\boldsymbol{b}^h$) can be transformed to one suitable for the arc-path formulation by augmenting the graph with at most k super-sources and k super-sinks (see Figure 4), but this requires having individual capacity b_j^h on the arc connecting any sink node j with the super-sink t_h, and $-b_i^h$ on (s_h, i) for any source node i;

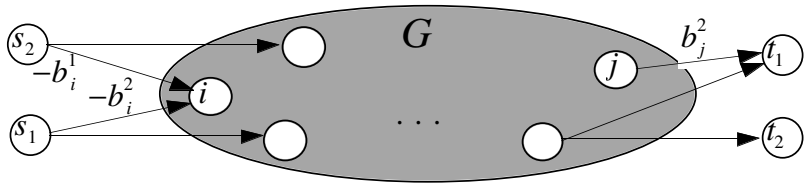

Figure 4: converting a general problem to one with $\mathcal{K} \subseteq \mathcal{N} \times \mathcal{N}$

- for each node-arc formulation, there exists an equivalent "extended" arc-path formulation where the cost of a path p is

$$c_p = \sum_{(i,j) \in P(\mathcal{K}(p))} c_{ij}^{\mathcal{K}(p)}$$

but the converse is true only if the cost of any path can be expressed as the sum of the costs of the arcs composing it: clearly, the arc-path formulation can have a more general structure of path costs, but, on the other hand, in the case of nonlinear separable arc cost functions $c_{ij}^h(x_{ij}^h)$ it has a nonseparable cost function;

- the arc-path formulation does not change if G is an *undirected* graph: the node-arc formulation can be adapted to this case by constructing both the arcs (i, j) and (j, i) relative to each *edge* $\{i, j\}$, and replacing each x_{ij}^h in the mutual capacity constraints with $x_{ij}^h + x_{ji}^h$, hence doubling the number of variables (but not the number of linking rows) in the resulting (LP).

An important, although apparently trivial, remark to be done is that the (MMCF) instances that arise from different applications can be very different under many respects:

- the number k of different commodities ranges from just a few, as in many distribution problems, to the number of all the possible O/D pairs and over, as in traffic models and routing applications: it is shown in Chapter II.2 that such a difference may influence the relative efficiency of alternative solution approaches, i.e. that efficient methods for small k may become less attractive when k grows and vice-versa;
- the mutual capacity constraints may be everything between very "loose" (a few percent being binding in the optimal solution) and very "strict" (30-50% and over): although tightly coupled (MMCF)s are generally harder to solve for all solution methods, the influence can be different for alternative approaches, as e.g. Cost Decomposition algorithms are likely to suffer on tightly coupled instances much more than Interior Point methods;
- the structure of the demand vectors has some influence on the difficulty of the problem, and in fact most of the existing theoretical results about Multicommodity Flows require the single source/sink assumption (if not stronger hypotheses on the demand pattern, such as that all pairs of nodes be source/sink for a commodity): for some solution methods this is also true in practice, as for

instance multiple sources *and* sinks for each commodity impair the possibility of using Shortest Path subproblems instead of Min-Cost Flow ones;

- analogously, individual capacities (that are needed in many real applications) seem to make the problem more "difficult" both in theory, invalidating many theoretical results, and in practice, since again the subproblems cease to be Shortest Paths: on the other hand, "tight" individual capacities may make many of the mutual capacity constraints redundant, diminishing the "coupling" of the problem - using an optimal solution $\boldsymbol{x}^*$ as arc capacities would completely decouple the blocks;
- *undirected* Multicommodity Flow problems are "easier" than *directed* ones, as many theoretical results show: for instance, the maximum inherent gap in the Max-Flow/Min-Cut relation is O(log n) for undirected graphs, but it is O(n) for directed ones (Cf. Sec. II.1.4.6).

Finally, the linear (MMCF) is only a special case - or a relaxation - of more complex problems that have a theoretical and practical interest of their own, such as

- the *Integer Multicommodity Min-Cost Flow* problem, where the integrality of the flow vectors $\boldsymbol{x}^h$ is required: being an $\mathcal{NP}$-hard problem, (IMMCF) calls for the entirely different machinery of enumerative, polyhedral or heuristic methods, that in turn may require the (repeated) solution of (MMCF)-like problems and/or the efficient computation of lower bounds to the integral optimal objective value; this is also true for other (MMCF)-structured $\mathcal{NP}$-hard problems such as the Fixed Charge [Integer] Multicommodity Min-Cost Flow problem discussed in §II.5.2;
- the *Generalised Multicommodity Flow problem*, where the mutual capacity constraints are replaced by a block of general linear constraints $\boldsymbol{D}[\, \boldsymbol{x}^1 \,..\, \boldsymbol{x}^k \,] = \boldsymbol{d}$: in turn, special cases of (GMF) are the *Nonsimultaneous Multicommodity problem* (Cf. §II.5.1), where the commodities are partitioned in p "blocks" $B(\,1\,)\,..\,B(\,p\,)$ and mutual capacity constraints are "restricted" within a block, i.e.

$$\sum_{h \in B(q)} x_{ij}^h \le u_{ij}^q \qquad q = 1 \,..\, p,$$

  and the *Equal Flow problem*, where there is only one commodity but constraints $x_{ij} = x_{uw}$ are imposed on a certain set $\{\, (\,(\,i\,,j\,)\,,\,(\,u\,,w\,)\,)\,\}$ of pairs of the arcs;
- various forms of the *Nonlinear* [Generalised] *Multicommodity Min-Cost Flow* problem, ranging from the "easy" ones with strictly convex separable cost functions $c_{ij}^h(\, x_{ij}^h\,)$ to the very "hard" ones with nonseparable and/or nonconvex functions and/or integrality constraints on the variables;
- the generic block-structured program

$$\text{(BS)} \qquad \min\Big\{ \sum_{h \in \mathcal{K}} c^h(\,\boldsymbol{x}^h\,) : \boldsymbol{x}^h \in X^h \;\forall\; h \in \mathcal{K}\,,\; \sum_{h \in \mathcal{K}} g^h(\,\boldsymbol{x}^h\,) \le \boldsymbol{u} \Big\}$$

  where optimization over a single block

$$\text{(B}_{\boldsymbol{y}}^h\text{)} \qquad \varphi^h(\,\boldsymbol{y}\,) = \min\Big\{\, c^h(\,\boldsymbol{x}^h\,) + \boldsymbol{y} g^h(\,\boldsymbol{x}^h\,) : \boldsymbol{x}^h \in X^h \Big\}$$

  is an "easy" problem for any choice of the dual vector $\boldsymbol{y} \ge \boldsymbol{0}$.

However, less general forms of Multicommodity Flow problems have also received attention in the past: the most important ones are (variants of) the Multicommodity Max-Flow problems discussed in the next paragraph.

II.1.2 Multicommodity Max-Flow formulations

Two different forms of Multicommodity Max-Flow problems have been studied in the literature: for the sake of exposition, their arc-path formulation will be presented first, therefore assuming no individual capacities and an unique source/sink pair (s_h, t_h) for each commodity h. The first version is the *Max-Sum Multicommodity Flow* problem

$$\text{(MSMF)} \qquad \begin{cases} \max \sum_{h \in \mathcal{K}} \sum_{p \in P(h)} x_p \\ \sum_{p:\,(i,j) \in p} x_p \le u_{ij} & \forall\ (i,j) \in \mathcal{A} \\ x_p \ge 0 & \forall\ p \in P \end{cases}$$

which is the direct generalization of the single-commodity Max-Flow problem, since $\Sigma_{p \in P(h)} x_p$ is the total quantity of commodity h (leaving s_h and reaching t_h) in the graph: obviously, for such a problem the uniqueness of the source/sink pair for each commodity can be assumed without loss of generality. The second form, known as the *Maximum Concurrent Multicommodity Flow* problem, is

$$\text{(MCMF)} \qquad \begin{cases} \max \lambda \\ \sum_{p \in P(h)} x_p = \lambda d_h & \forall\ h \in \mathcal{K} \\ \sum_{p:\,(i,j) \in p} x_p \le u_{ij} & \forall\ (i,j) \in \mathcal{A} \\ x_p \ge 0 & \forall\ p \in P \end{cases}$$

and it explicitly considers the demand d_h for each commodity h, trying to maximize the *throughput* λ, i.e. the common fraction of satisfied demand, subject to the capacity constraints: clearly, a multiflow satisfying all the demands exists if and only if the optimal throughput λ^* is at least 1. Equivalently, (MCMF) may be viewed as the search for the largest λ such that a feasible multiflow exists with respect to capacities $(1 / \lambda)\boldsymbol{u}$. Note that (MSMF) can be written in an analogous form by maximizing $\Sigma_{h \in \mathcal{K}} \lambda_h$ subject to

$$\sum_{p \in P(h)} x_p = \lambda_h \qquad \forall\ h \in \mathcal{K}$$

i.e. assuming $d_h = 1\ \forall\ h \in \mathcal{K}$ but allowing a different throughput for each commodity h: this shows that (MCMF) is the restriction of (MSMF) with the additional constraint $\lambda_1 = \lambda_2 = \ldots = \lambda_k$.

The duals are respectively

$$\text{(DMCMF)} \qquad \begin{cases} \min \sum_{(i,j) \in \mathcal{A}} y_{ij} u_{ij} \\ \sum_{(i,j) \in p} y_{ij} \ge z_h & \forall\ p \in P(h),\ \forall\ h \in \mathcal{K} \\ \sum_{h \in \mathcal{K}} z_h d_h \ge 1 \\ y_{ij} \ge 0 & \forall\ (i,j) \in \mathcal{A} \end{cases}$$

and

$$\text{(DMSMF)} \begin{cases} \min \sum_{(i,j) \in \mathcal{A}} y_{ij} u_{ij} \\ \sum_{(i,j) \in p} y_{ij} \geq 1 & \forall\ p \in P(h),\ \forall\ h \in \mathcal{K} \\ y_{ij} \geq 0 & \forall\ (i,j) \in \mathcal{A} \end{cases}$$

and for $k = 1$, $d = 1$ (the single-commodity Max-Flow) they are clearly equivalent: in this case, at least one optimal integral solution $\boldsymbol{y}^* \in \{0, 1\}^m$ exists that can be interpreted as the characteristic vector of a (*s*/*t*-)*cut* - a subset of arcs whose removal disconnects the source from the sink - of minimal capacity. Analogously, any feasible integral solution of (DMSMF) can be interpreted as a *multicut*, i.e. a set of arcs whose removal separates *all* source/sink pairs: therefore, (DMSMF) is the continuous relaxation of the *Minimum Capacity Multicut* problem, and a feasible fractional $\boldsymbol{y}$ is a *fractional multicut* (a convex combination of multicuts) that "covers" all the commodities. Hence, the capacity of *any* multicut provides an upper bound to z(MSMF), but since the integrality property does not hold, a nonzero gap can be expected. Similar relations can be derived for (MCMF): given a *cutset* $\mathcal{S} \subseteq \mathcal{N}$ and the set $\mathcal{K}(\mathcal{S}) = \{ h \in \mathcal{K}: s_h \in \mathcal{S} \,\&\, t_h \notin \mathcal{S} \}$ of the commodities whose source and sink are separated by $\mathcal{S}$, the *capacity* of the induced cut and the *separated demand* are respectively

$$c(\mathcal{S}) = \sum_{i \in \mathcal{S}, j \notin \mathcal{S}} u_{ij} \qquad \text{and} \qquad d(\mathcal{S}) = \sum_{h \in \mathcal{K}(\mathcal{S})} d_h$$

and $c(\mathcal{S}) / d(\mathcal{S})$ is an upper bound on the maximum throughput λ^*. Hence, the "most restraining" cut is the one with the minimum *capacity-to-separated-demand* ratio: this can be shown with simple combinatorial arguments, but it is instructive to derive it from the (LP) formulations. The coefficient of 1 in the objective function of (MCMF), and therefore the right hand side in the constraint $\Sigma_{h \in \mathcal{K}} z_h d_h \geq 1$ of (DMCMF), can be turned to any $\alpha > 0$ with the only effect of multiplying by α the optimal objective value: then, for any feasible $(\bar{\boldsymbol{z}}, \bar{\boldsymbol{y}}) \in \{0, 1\}^{m+k}$, $\bar{\boldsymbol{z}}$ is the characteristic vector of the set of commodities that are separated by the multicut induced by $\bar{\boldsymbol{y}}$, since for any h such that $\bar{z}_h = 1$ the constraints $\Sigma_{(i,j) \in p} \bar{y}_{ij} \geq 1\ \forall\ p \in P(p)$ ensure that at least one arc on any s_h-t_h path is selected. Since $(\bar{\boldsymbol{z}}, \bar{\boldsymbol{y}})$ is still feasible, by setting α equal to the "separated demand" $\bar{\boldsymbol{z}}\boldsymbol{d}$ of the cut $\bar{\boldsymbol{y}}$, the bound $\bar{\boldsymbol{y}}\boldsymbol{u} / \bar{\boldsymbol{z}}\boldsymbol{d} \geq \lambda^*$ is easily established: again, the bound is tight for $k = 1$ but a gap can exist for larger k, an example being the $K_{2,3}$ (undirected) graph depicted in Figure 5 with unitary capacities over all edges and unitary demands between all pair of vertices not directly connected by an edge, that has a minimum capacity-to-separated-demands ratio of 1 (two cuts achieving such a ratio are shown) but for which clearly no feasible multiflow exists. In fact, a stronger path-based upper bound can be obtained by observing that, since each $p \in P(h)$ is an s_h-t_h path, the quantity $\delta_p(\boldsymbol{y}) = \Sigma_{(i,j) \in p} y_{ij}$ is the distance between s_h and t_h along path p if y_{ij} is the length of arc (i, j): for any fixed $\bar{\boldsymbol{y}}$, the "optimal" variables $\boldsymbol{z}(\bar{\boldsymbol{y}})$ are clearly

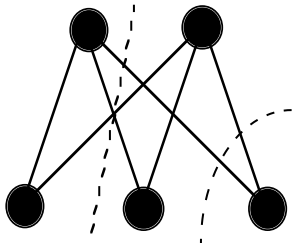

Figure 5: example of Max-Flow / Min-Cut gap

$$\varphi^h(\bar{\boldsymbol{y}}) = z(\bar{\boldsymbol{y}})_h = \min\{ \delta_p(\bar{\boldsymbol{y}}) : p \in P(h) \}$$

i.e. the length of the *shortest path* between s_h and t_h w.r.t. $\bar{\boldsymbol{y}}$, and (DMCMF) can be restated as the problem of finding arc lengths $\boldsymbol{y}$ minimizing $\boldsymbol{y}\boldsymbol{u}$ and such that the weighted (minimal) distance from all the source/sink pairs be at least 1. Analogously, (DMCMF) can be interpreted as the problem of finding arc lengths $\boldsymbol{y}$ minimizing $\boldsymbol{y}\boldsymbol{u}$ and such that the distance between each s_h and t_h is at least 1:

hence, the bound

$$\lambda^* \le \frac{\bar{\boldsymbol{y}}\,\boldsymbol{u}}{\sum_{h \in \mathcal{K}} d_h \varphi^h(\,\bar{\boldsymbol{y}}\,)}$$

can be easily obtained as the previous one for any (non necessarily integral) choice of the distances $\bar{\boldsymbol{y}}$.

The node-arc formulations of the Multicommodity Max-Flow problems are respectively

$$\text{(MSMF)} \quad \begin{cases} \max \sum_{h \in \mathcal{K}} \lambda_h \\ \begin{bmatrix} \boldsymbol{E} & \cdots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{E} \\ \boldsymbol{I} & \cdots & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^1 \\ \vdots \\ \boldsymbol{x}^k \end{bmatrix} - \begin{bmatrix} \boldsymbol{b}^1 & \cdots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{b}^k \\ \boldsymbol{0} & \cdots & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_k \end{bmatrix} = \le \begin{bmatrix} \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \\ \boldsymbol{u} \end{bmatrix} \\ \boldsymbol{0} \le \boldsymbol{x}^h \le \boldsymbol{u}^h \quad \forall\, h \in \mathcal{K} \end{cases} \qquad \text{where } \boldsymbol{b}^h = \begin{array}{|c|c|} \hline & \\ -1 & s_h \\ & \\ & \\ +1 & t_h \\ & \\ \hline \end{array}$$

$$\text{(MCMF)} \quad \begin{cases} \max \lambda \\ \begin{bmatrix} \boldsymbol{E} & \cdots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{E} \\ \boldsymbol{I} & \cdots & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^1 \\ \vdots \\ \boldsymbol{x}^k \end{bmatrix} - \lambda \begin{bmatrix} \boldsymbol{b}^1 \\ \vdots \\ \boldsymbol{b}^k \\ \boldsymbol{0} \end{bmatrix} = \le \begin{bmatrix} \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \\ \boldsymbol{u} \end{bmatrix} \\ \boldsymbol{0} \le \boldsymbol{x}^h \le \boldsymbol{u}^h \quad \forall\, h \in \mathcal{K} \end{cases} \qquad \text{where } \boldsymbol{b}^h = \begin{array}{|c|c|} \hline & \\ -d_h & s_h \\ & \\ & \\ +d_h & t_h \\ & \\ \hline \end{array}\ .$$

The first formulation is analogous to that of the single-commodity Max-Flow problem, each variable λ_h being the flow of the artificial "return arc" (t_h, s_h) (Figure 6); the second is different in that the unique variable λ can be interpreted as the flow of an unique "hyperarc with gains" (Figure 7), where each commodity has a different gain d_h, connecting all the sinks to all the sources (the mutual capacity constraints are not represented in the Figures).

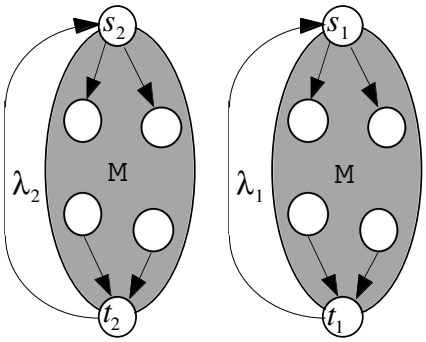

Figure 6: the Max-Sum problem

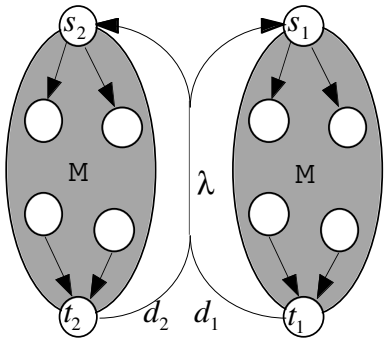

Figure 7: the Concurrent problem

As for (MMCF), the two different Multicommodity Max-Flow problems are special cases of more general classes of problems: (MSMF) is a *Fractional Packing* problem [PST93]

(FP) $\qquad \exists?\, \boldsymbol{x} : \boldsymbol{x} \in X \;\&\; \boldsymbol{A}\boldsymbol{x} \le \boldsymbol{u}$

where $\boldsymbol{A} \ge \boldsymbol{0}$ and optimization over X is "easy", that is $\min\{\ (\boldsymbol{y}\boldsymbol{A})\boldsymbol{x} : \boldsymbol{x} \in X\ \}$ can be efficiently solved for any $\boldsymbol{y} \ge \boldsymbol{0}$ (*Fractional Covering* problems have $\boldsymbol{A}\boldsymbol{x} \ge \boldsymbol{u}$ instead). The idea behind this definition is that X is a polyhedron given by an "oracle" and one wants to find a feasible fractional packing (covering) of its extreme points: note that in (MSMF) *flows* are packed, while in its dual *cuts*

are used to cover. (MCMF) is a *Resource Sharing* problem [GK94a]

$$\text{(RS)} \qquad \min\left\{ \lambda : \boldsymbol{x}^h \in X^h \ \forall\ h \in \mathcal{K},\ \sum_{h \in \mathcal{K}} \boldsymbol{g}^h(\boldsymbol{x}^h) \le \lambda \boldsymbol{u} \right\}$$

where each X^h is a nonempty convex compact set, each $\boldsymbol{g}^h()$ is a *nonnegative* continuous convex (vector) function and optimization over a single block - the (B_y^h) problem defined at the end of §II.1.2 - is "easy": here, the idea is that the $\boldsymbol{g}^h()$ functions measure the consumption of some resources shared among the (otherwise independent) blocks X^h, and the problem is to find a point $\boldsymbol{x} = [\ \boldsymbol{x}^1\ ..\ \boldsymbol{x}^k\ ]$ minimizing the total resource consumption. Some efficient ε-approximation algorithms have been recently devised for both (FP) and (RS) on the basis of results initially obtained for Multicommodity Max-Flow problems, as discussed in more details in §II.1.4.

II.1.3 Applications

Multicommodity Flow problems have a wide area of application, far beyond the "obvious" one as models for transportation problems: however, this usage alone would justify the interest in efficient solution methods, since many practical problems can be formulated as (Generalised) Multicommodity Flow Probles [To71] [GG71] [Go75] [PS83] [CG84] [ZTT84] [AKS88] [AKL93] [BG95] [CGN95] [EGM95] [HBL95] (and many more). In these models, a commodity may represent different things, such as skills of personnel to be assigned to new jobs [AKS88], depots to which vehicles belong [CG84], data to be exchanged between antennas [BG95] or types of aircrafts to be assigned to scheduled flights [HBL95]: however, in some relevant models a commodity is just the type of some sort of merchandise flowing from its source to its destination along the network. An interesting representative of such models is the Freight Transportation Planning problem [FPL93], that can be briefly described as follows: on a physical transportation network (highway, railway …), represented by a *graph of terminals* $G(\mathcal{N}, \mathcal{A})$, a fixed set of *trips* (of trains, trucks ...) have already been scheduled within a given *planning horizon* $T = \{\ t_1\ ..\ t_p\ \}$. Demands for transporting different goods along the network are given, each one with its origin and destination terminals and the (desired) time window, that must be satisfied optimally with the available *services*, i.e. trips. The problem can be formulated as a (MMCF) on the *time-space graph* $G'(\mathcal{N} \times T, \mathcal{A}')$ (Figure 8), where each node I_t represents the terminal I at time instant t, there is the *service* arc $(I_t, J_s) \in \mathcal{A}'$ if and only if a service starts from terminal I at time instant t and ends at terminal J at time instant s and there is the *holding* arc $(I_t, I_{t+1}) \in \mathcal{A}'$ whenever goods are allowed to wait at the terminal I from t to $t + 1$.

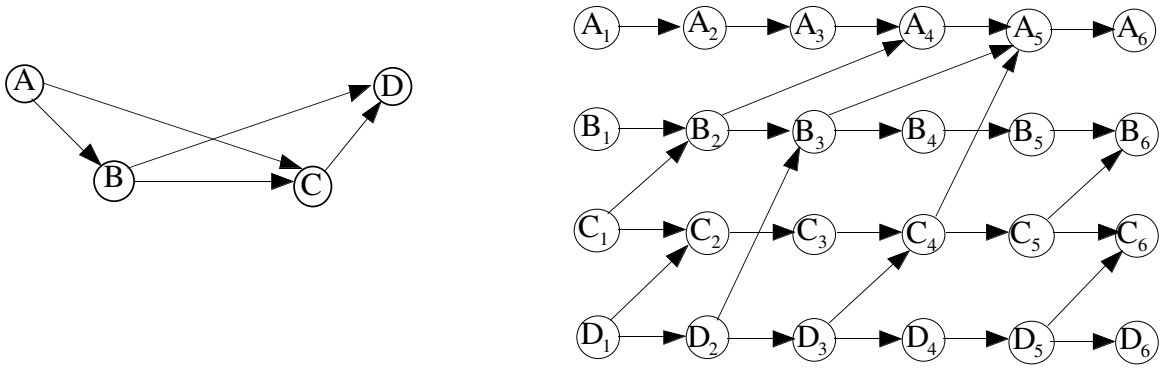

Figure 8: a graph with four terminals and the corresponding space-time graph for six time periods

Transport costs are associated to service arcs, while inventory costs are associated to holding arcs: using the technique of node splitting, costs can be assigned also to nodes for representing operations to be performed in terminals, such as rearranging the cars of a train. Typically, real-world models have hundreds of commodities, medium-sized graphs and up to thousands of time instants, giving rise to (MMCF) of very large size: furthermore, (FTP) is actually only a subproblem of the Freight Services Scheduling problem where the optimal set of services is requested, so that reasonable solution times are necessary to allow a reasonable number of alternatives be tested, either in an automated optimzation routine or in an interactive mode.

In many practical cases, "good" integer solutions are required: being $\mathcal{NP}$-hard, the Integer (MMCF) calls for the machinery of heuristic methods [BS93] [Ba93], implicit enumeration methods (Branch & Bound, Branch & Price) [BHV96] or polyhedral methods (Cutting Plane, Branch & Cut) [BCF95]. However, all the exact approaches require the solution of a sequence of continuous (MMCF)s - Generalized (MMCF)s in a Branch & Cut - in order to obtain an integer primal solution; furthermore, efficient heuristics may benefit of dual (cost) information, that can be obtained by solving the continuous relaxation or some Lagrangean Dual.

An important variant of the Integer (MMCF) is the *Fixed Charge Multicommodity* (*Min-Cost*) *Flow* problem [GC95], belonging to the broad class of Network Design problems [KBY94] [Mi89], in which the arcs must be constructed - at a fixed cost - before the flow of any commodity can be routed along them: (FCMF) is known to be among the most difficult (mixed-)integer problems (the single-commodity version is already $\mathcal{NP}$-hard [HS89]), but it has a large number of practical and theoretical applications. In fact, other than being a model for various problems in telecommunications [LSW95] [PR96], (FCMF) is used as a tool for reformulating and approaching a large class of difficult problems such as the Steiner Tree problem [NW88 §II.6.4], the Constrained Minimal Spanning Tree problem [Ka94] and a number of Production Scheduling and Lot Sizing problems [ZTT84] [PW95]: for all such problems, (FCMF) provides both "stronger" (LP) formulations and some large classes of strong valid inequalities [NW88 §II.2.4].

An interesting application of (possibly nonlinear) Multicommodity Flow problems is the design of routing algorithms: in fact, they provide a general analytic model for describing a variety of basic routing problems [RH89], such as to satisfy communication demands subject to the network capacity, to minimize communication delays or to maximize network utilization in presence of congestion. In the "basic" version of these models, commodities are just communicating pairs of nodes with (estimated) demands, i.e. $k \approx O(n^2)$, but they can also represent different classes of traffic, such as video/voice, flow control and standard data packets: multi-class traffic control will be one of the major characteristics of the next generation of routing protocols, especially after the advent of ATM communication networks. In this framework, Multicommodity Flows can provide (off-line) "optimal" routings against with to evaluate the effectiveness of actual routing protocols [CWH92]; furthermore, any (exact or approximate) solution method for Multicommodity Flow problems that is suitable for fine-grain, node-based distributed implementation (such as those of Sec. II.1.4.5) can be "interpreted" as an on-line routing protocol, and vice-versa [FGK73] [Ga77] [St77] [CC91] [RE92].

Finally, Multicommodity Max-Flow problems have been used as a tool for approximating a variety of $\mathcal{NP}$-hard graph problems arising in VLSI circuits layout [KAR90] [Tr91], including the Minimum

Feedback Arc Set, the Minimum Cut Linear Arrangement and the Minimum Area Layout: the idea is that undirected (MCMF) with unit demands and a commodity for each node pair (the *All-Pair Unit Demands* problem) provide a mean for (approximately) constructing cuts with "low" capacity and "balanced" sides. In fact, the cutset $\mathcal{S}^*$ achieving the minimum capacity-to-separated-demand ratio solves

$$\mathcal{S}^* = \text{argmin} \{ c(\mathcal{S}) / (|\mathcal{S}| \cdot |\mathcal{N} - \mathcal{S}|) : \mathcal{S} \subseteq \mathcal{N} \}$$

and the denominator $|\mathcal{S}| \cdot |\mathcal{N} - \mathcal{S}|$ is large when both sides have about the same size: in the case of unitary capacities, the cut induced by $\mathcal{S}^*$ is referred to as the *sparsest cut* and it is clearly a good way of evenly dividing the node set $\mathcal{N}$ by cancelling the minimal number of arcs. In [LR88], the first $O(\log n)$-approximation algorithm for (APUD) is presented and recursively used to construct an $O(\log^2 n)$-optimal solution to the Minimal Capacity 1/3 - 2/3 Separator problem, i.e.

$$\text{argmin} \{ c(\mathcal{S}) : \mathcal{S} \subseteq \mathcal{N}, n / 3 \leq |\mathcal{S}| \leq 2n / 3 \}.$$

In turn, such a Minimal Capacity "balanced" Cut can be successfully used to drive a Divide and Conquer strategy [Le83] [BL84], where two smaller subproblems are (recursively) solved on the two sides of the (approximatedly) optimal cut and the two partial solutions are pasted together: the almost balanced sides ensure at most a logarithmic number of recursion steps, while the small capacity limits the amount of work required to paste the two partial solutions. Interestingly enough, the original algorithm in [LR88] uses the *dual* optimal solution of (APUD), and it was also the seminal work for the ε-approximation algorithms for Multicommodity Max-Flow problems discussed in Sec. II.1.4.6.

II.1.4 Algorithmic approaches

Historically, (MMCF) has motivated many important algorithmic developments that have later found broader applications: while its arc-path formulation inspired the first column-generation approaches [FF58], the node-arc formulation is the archetype of the block-structured Linear Program, and the celebrated Dantzig-Wolfe decomposition algorithm [DW60] was originally devised for solving Multicommodity Flow problems. As more broadly discussed below, this "pushing" effect is lasting until today, and a number of interesting recent developments have been motivated by the need of solving, at least approximately, (MMCF)s of very large size.

Several algorithmic approaches have been proposed for (MMCF), but unfortunately the available surveys [As78] [Ke78] [AMO93 Chapter 17] only present the more "classical" techniques, sometimes dating back to the sixties, and this is also true for computational studies [As76] [AHK79]: the only recent surveys are limited to the fast-growing areas of approximation algorithms [AL93] [Ta94] and parallel approaches [Ze93] [DKM94]. The main algorithmic approaches to (MMCF) will be presented in the following, with a particular stress on Cost Decomposition methods: the focus will be on *sequential* approaches, since the issues relative to *parallel* algorithms will be discussed in §II.3.1.

II.1.4.1 Basis Partitioning methods

Basis Partitioning methods [HL72] [Ke77] [CN95] are specialized versions of the (usually Primal) Simplex method for Linear Programming that exploit the network structure of the basis for reducing the cost of the primal/dual basic solutions calculation and of pivoting: matrix factorizations or

inversions cannot be completely avoided (as it is the case for the single-commodity case), but they can be limited to a working matrix of a much smaller dimension. Although the details of the method are rather involved, the idea is quite simple: in each basis of (MMCF), the arcs associated with basic variables for a commodity can be partitioned into two sets, one forming a spanning tree of the graph G and the other being that of "extra" arcs corresponding to the mutual capacity constraints. By introducing slack variables $\boldsymbol{s}$ in the mutual capacity constraints, i.e. $\sum_{h \in \mathcal{K}} \boldsymbol{x}^h + \boldsymbol{s} = \boldsymbol{u}$, the problem can be turned into a standard equality form and every basis $\boldsymbol{B}$ can be partitioned as

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{L}_1 & \boldsymbol{R}_1 & \boldsymbol{0} \\ \boldsymbol{L}_2 & \boldsymbol{R}_2 & \boldsymbol{0} \\ \boldsymbol{L}_3 & \boldsymbol{R}_3 & \boldsymbol{I} \end{bmatrix} \qquad \text{where} \qquad \boldsymbol{L}_1 = \begin{bmatrix} \boldsymbol{B}_1 & \boldsymbol{0} & \mathtt{L} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{B}_2 & \mathtt{L} & \boldsymbol{0} \\ \mathtt{M} & \mathtt{M} & \mathtt{O} & \mathtt{M} \\ \boldsymbol{0} & \boldsymbol{0} & \mathtt{L} & \boldsymbol{B}_k \end{bmatrix}$$

and each $\boldsymbol{B}_h$ is the node-arc incidence matrix of a spanning tree T_h: the second and third set of rows correspond to active mutual capacity constraints and active slacks (i.e. inactive mutual capacity constraints) respectively, and the second set of columns corresponds to non-tree basic arcs. The solution of linear systems on $\boldsymbol{B}$ can be performed by first efficiently solving k smaller systems on the submatrices $\boldsymbol{B}_h$ (visiting T_h) and then solving a system on the *working matrix* $\boldsymbol{W} = \boldsymbol{R}_2 - \boldsymbol{L}_2\boldsymbol{L}_1^{-1}\boldsymbol{R}_1$ that can be efficiently computed and updated (after pivots) by exploiting the structural properties of $\boldsymbol{L}_1$: Primal Partitioning implementations [Ke78b] [Ca94] usually outperform "standard" simplex codes due to the greatly reduced cost of the core matrix operations. Moreover, the structure of the problem can be further exploited by performing "advanced starts": a typical way is to solve the k (MCF) subproblems independently to obtain a primal unfeasible (but "good") base, then achieve feasibility by solving an appropriately modified problem and finally use the resulting primal feasible base as a starting point - the computational results in §II.2.4 will show the importance of such details.

These techniques can also be used to efficiently implement *Reduced Gradient* methods for the Nonlinear (nonseparable) (MMCF), analogously to the single-commodity case [ZM88]; in general, many algorithms for solving the equality-constrained Nonlinear Problem

$$\min \{ f(\boldsymbol{x}) : \boldsymbol{E}\boldsymbol{x} = \boldsymbol{b}\ ,\ \boldsymbol{0} \le \boldsymbol{x} \le \boldsymbol{u} \}$$

are based on the *active set* philosophy, where only a subset of the variables (the *active* ones) are allowed to vary at any given iteration: the Reduced Gradient methods further subdivide the active variables into *Basic* and *Superbasic*, where the superbasic variables are the independent ones while the basic variables are adjusted in order to preserve the feasibility. Assuming for simplicity that the matrix $\boldsymbol{E}$ has full row rank, and that the upper bounds $\boldsymbol{u}$ are not present, $\boldsymbol{E}$ and $\boldsymbol{x}$ are partitioned as

$$\boldsymbol{E} = [\, \boldsymbol{E}_B \boldsymbol{E}_S \boldsymbol{E}_N \,] \qquad \text{and} \qquad \boldsymbol{x} = [\, \boldsymbol{x}_B \boldsymbol{x}_S \boldsymbol{x}_N \,]$$

where $\boldsymbol{E}_B$ is a nonsingular square matrix and $\boldsymbol{x}_N$ (the Nonbasic variables) are fixed to $\boldsymbol{0}$: the Superbasic variables $\boldsymbol{x}_S$ are "free", while the Basic variables $\boldsymbol{x}_B$ are set to $\boldsymbol{E}_B^{-1}(\, \boldsymbol{b} - \boldsymbol{E}_S\boldsymbol{x}_S \,)$ to satisfy the equality constraints, so that the *Reduced Gradient*

$$\nabla f_R = \nabla f(\, \boldsymbol{x} \,)_S - \boldsymbol{E}_S^T(\, \boldsymbol{E}_B^{-1} \nabla f(\, \boldsymbol{x} \,)_B \,)$$

gives a descent direction in the space of the $\boldsymbol{x}_S$; furthermore, a properly defined *Reduced Hessian* can be used to extract suitable second-order information, allowing the implementation of rapidly

convergent Newton or Quasi-Newton methods. Some fundamental details cannot be described here, such as the use of dual information (Lagrangean multipliers) for determining whether the current active set $B \cup S$ has to be updated by inserting some nonbasic variable (pricing) or eliminating some redundant active one: in the linear case, for instance, the only Superbasic variable is the Nonbasic one entering at the beginning of any iteration, and it is transformed into Basic to replace the leaving one. However, it is not hard to see that the necessary tasks can be efficiently dealt with by exploiting the structural properties of $\boldsymbol{E}_B$: hence, as in the linear case, Basis Partitioning techniques greatly reduce the cost of many of the core operations of standard optimization algorithms. However, while the complexity of each step is about $\max\{ O(k), O(m^2) \}$, the number of steps can be expected to increase with km, so that such methods may not be very attractive as k increases.

II.1.4.2 Cost Decomposition methods

Among the approaches to (MMCF), Cost Decomposition methods are perhaps the most intriguing, as demonstrated by the constant stream of work dedicated to this class of algorithms: a natural derivation of such methods can be obtained from the dual of the arc-path formulation of (MMCF)

$$\text{(DMMCF)} \quad \begin{cases} \max \sum_{h \in \mathcal{K}} z_h d_h - \sum_{(i,j) \in \mathcal{A}} y_{ij} u_{ij} \\ c_p + \sum_{(i,j) \in p} y_{ij} \geq z_h \qquad \forall\ p \in P(h),\ \forall\ h \in \mathcal{K} \\ y_{ij} \geq 0 \qquad\qquad \forall\ (i,j) \in \mathcal{A} \end{cases} .$$

As for the Maximum Concurrent Multicommodity Flow problem, $c_p + \sum_{(i,j) \in p} y_{ij} = \sum_{(i,j) \in p} (c^h_{ij} + y_{ij}) = \delta_p(\boldsymbol{c}^h + \boldsymbol{y})$ is the distance between s_h and t_h along path p with arc lengths $\boldsymbol{c}^h + \boldsymbol{y}$, hence at optimality z_h is the length of the *shortest path* between s_h and t_h and the problem can be rewritten as

$$\text{(DMMCF)} \qquad \max \left\{ \sum_{h \in \mathcal{K}} \varphi^h(\boldsymbol{y}) - \boldsymbol{y}\boldsymbol{u} : \boldsymbol{y} \geq \boldsymbol{0} \right\}$$

where $\varphi^h(\boldsymbol{y}) = d_h \min\{ \delta_p(\boldsymbol{c}^h + \boldsymbol{y}) : p \in P(h) \}$ is the cost of sending d_h units of flow from s_h to t_h along the shortest path: hence, the dual of (MMCF) is naturally viewed as the maximization of a function involving Shortest Path computations w.r.t. the modified costs $\boldsymbol{c}^h + \boldsymbol{y}$.

More in general, (MMCF) is a special case of the block-structured program (BS) introduced at the end of §II.1.1, and it is well-known [Ge69] that, without essentially any assumption over the sets X^h and the functions $c^h()$ and $\boldsymbol{g}^h()$, the *dual function*

$$\varphi(\boldsymbol{y}) = \sum_{h \in \mathcal{K}} \varphi^h(\boldsymbol{y}) - \boldsymbol{y}\boldsymbol{u}$$

is concave; moreover, if all the blocks X^h are closed convex sets, and all the $c^h()$ and $\boldsymbol{g}^h()$ functions are convex and finite everywhere over X^h, then the *dual* (concave, max-min) *problem*

$$\text{(DBL)} \qquad \max\{ \varphi(\boldsymbol{y}) : \boldsymbol{y} \geq \boldsymbol{0} \}$$

is equivalent to (BL) in the sense that

- each feasible solution of (DBL) provides a lower bound to z(BL), and each feasible solution of (BL) provides an upper bound to z(DBL): therefore, if (DBL) is nonempty then (BL) is either empty or bounded, and if (BL) is nonempty then (DBL) is either empty or bounded;

- under mild conditions, implied by all the known constraint qualifications, (BL) and (DBL) have the same optimal objective function value whenever (BL) is nonempty and bounded; in particular, such conditions are always true when X^h are polyhedra;
- the *value function* of (BL)

$$v(\boldsymbol{z}) = \min\left\{ \sum_{h \in \mathcal{K}} c^h(\boldsymbol{x}^h) : \boldsymbol{x}^h \in X^h \ \forall\ h \in \mathcal{K}, \sum_{h \in \mathcal{K}} g^h(\boldsymbol{x}^h) \le \boldsymbol{z} \right\}$$

  is a convex function over the (convex) set of all the *feasible* right hand sides $\boldsymbol{z}$;
- for any optimal solution $\bar{\boldsymbol{y}}$ of (DBL), $-\bar{\boldsymbol{y}}$ is a subgradient of $v()$ in $\boldsymbol{z} = \boldsymbol{u}$; moreover, *any single* $\bar{\boldsymbol{y}}$ characterises the set of *all* optimal solutions of (BL), in that $\bar{\boldsymbol{x}}$ is optimal for (BL) if and only if

$$\bar{\boldsymbol{x}}^h = \operatorname{argmin}\left\{ c^h(\boldsymbol{x}^h) + \bar{\boldsymbol{y}} g^h(\boldsymbol{x}^h) : \boldsymbol{x}^h \in X^h \right\} \qquad \forall\ h \in \mathcal{K}$$

$$\sum_{h \in \mathcal{K}} g^h(\bar{\boldsymbol{x}}^h) \le \boldsymbol{u}$$

$$\bar{\boldsymbol{y}}\left(\sum_{h \in \mathcal{K}} g^h(\bar{\boldsymbol{x}}^h) - \boldsymbol{u} \right) = 0 \qquad \text{(} \textit{complementary slackness} \text{).}$$

Hence, duality provides an attractive way of solving (BL) by attacking its dual (DBL), i.e. the constrained maximization of a concave nondifferentiable function $\varphi(\boldsymbol{y})$: for any given $\boldsymbol{y}$, the computation of $\varphi(\boldsymbol{y})$ and of a subgradient requires the solution of k independent block problems (B_y^h), that for (MMCF) are just

$$(\mathrm{RMF}_y^h) \qquad \min\left\{ (\boldsymbol{c}^h + \boldsymbol{y})\boldsymbol{x}^h : \boldsymbol{E}\boldsymbol{x}^h = \boldsymbol{b}^h,\ \boldsymbol{0} \le \boldsymbol{x}^h \le \boldsymbol{u}^h \right\}$$

i.e. single-commodity Min-Cost Flow (or Shortest Path) subproblem with the modified costs $\boldsymbol{c}^h + \boldsymbol{y}$, for which several efficient algorithms exist. Hence, the solution of (DBL) can be tackled with the methods discussed in Part I: surprisingly, the first attempts in this direction are recent [Me90] [Me94] [FG96], maybe because of the lack of widely available general-purpose Bundle codes. However, a number of alternative approaches have been proposed: most of these methods could essentially work with any (BL) problem, even though they have been mostly applied, or even initially proposed, in the context of Multicommodity Flows. This is the case of the *Dantzig-Wolfe decomposition method* for Linear Problems [DW60], that is well-known [Le91] to be equivalent to a *Cutting Plane* approach [Ke60] to the maximization of the dual function $\varphi(\boldsymbol{y})$: an extensive description of the method and its relations with NDO algorithms is given in §I.1.2. Since its proposal, about 40 years ago, many sophisticated implementations, both sequential [HL81] [HL83] [JLF93] and parallel [HLS88] [GH93] [DKM94], have been developed: however, the results have not always been encouraging, motivating the substantially negative opinion of many experts that "... the folklore is that generally such [Cost Decomposition] 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, than to invest your time ..."◊. However, if Cost Decomposition approaches are of any value, Multicommodity Flow problems are probably the application where such a value could show up, and in fact some of the recent results appear to be more encouraging, despite the enormous progresses that standard (LP)

◊ John W. Gregory, in the "Linear Programming FAQ" monthly posted to the UseNet newsgroup sci.op-research.

codes have attained in the last few years. The most significant recent findings concern the effect of the number and the "quality" of the proposals (new vertices) that are sent by the block solvers to the Master Problem: providing multiple solutions at each iteration can help in reducing the convergence problems that are often experienced by Decomposition methods, although it seems to be important that the vertices be at least near-optimal for the current subproblems [HLS88]. For the Multicommodity Flow case, a somewhat analogous effect has been observed by analysing the (relevant) impact that the formulation has on the performances of the DW method [JLF93]: path-based formulations, where commodities are origin/destination pairs, seem to converge much faster than tree-based formulations where commodities are aggregated by common origin. This is not surprising, since at any iteration a tree-based formulation produces (for each commodity) a tree with a single associated multiplier, whereas a path-based formulation produces an equivalent set of paths, each one having its independent multiplier: hence, the "disaggregated" Master Problem is much more "flexible", and fewer points are likely to be needed in order to generate the optimal solution; in other words, the number of different trees of a graph is much larger than the number of different paths. Furthermore, the running time of the Master Problem does not grow too much, despite the greatly increased dimensions: this is both due to the reduced number of overall iterations and to the higher "sparsity" of the columns of the Master Problem w.r.t. those of the tree-based formulation.
Such an implementation of the DW method is essentially a *Column Generation* approach applied to the arc-chain formulation of (MMCF), with "multiple pricing" at each iteration: actually, the general DW method was inspired by the previous work of Ford and Fulkerson [FF58] on CG approaches to Multicommodity (Max-)Flow problems, that have been [Je66b] [CD74] and continue to be [Po92] [BHJ95] an active area of research. One interesting recent result is the Primal Partitioning technique for the CG method proposed in [FPL93]: it is based on the observation that (Cf. §II.1.1) there are at most $m + k$ "active" paths along which a nonzero flow is sent, so that, if $k > m$, there are at least $k - m$ commodities sent along an unique path. Then, every basis $\boldsymbol{B}$ can be partitioned as

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{I} & \mathbf{0} & \mathbf{0} \\ \Delta_0 & \boldsymbol{P} & \mathbf{0} \\ \Delta_1 & \Delta_2 & \boldsymbol{I} \end{bmatrix}$$

where the first set of rows corresponds to *unsplit* commodities, the second to *split* commodities and saturated arcs and the third to unsaturated arcs (active slack variables): the three sets of columns correspond respectively to path variables for unsplit commodities, path variables for split commodities and basic slack variables. Hence, any system involving $\boldsymbol{B}$ can be efficiently solved as long as the inverse or a factorization of the submatrix $\boldsymbol{P}$ is available: as for the "standard" Primal Partitioning methods, the fine details of the data structures update are rather involved (there are 12 different types of pivot steps according to the type of the entering and leaving variable, out of which only 9 are feasible), but it is clear that such an implementation can contribute in reducing the computational cost of the Column Generation method.

Seen from the dual viewpoint, however, all those methods are still Cutting Plane approaches to the maximization of $\varphi()$: Bundle methods are not the only "modern" alternative to CP, others being the *Analytic Centre Cutting Plane Methods* [GHV92] [GHV93] described in some detail in §I.1.1. Ad-hoc implementations of ACCPM for (MMCF) [GGS94] have shown to be effective for the solution

of large instances: however, the "standard" ACCPM can be significantly slower than a carefully implemented CP algorithm. This behaviour is inherent in the "conservative" viewpoint of Analytic Centre Methods, that have been designed for obtaining good complexity bounds and therefore does not make use of valuable information, such as the optimum of the CP model, to avoid the case where it may be misleading: in fact, these methods are resistant to "difficult" instances, where the CP fails, but can be slower on the "easy" ones where the CP is efficient. Hence, a promising alternative is that of "hybrid" methods [MGV96], that follow the standard CP strategy as long as the speed of convergence is sufficient, and turn to the ACCPM strategy when the going gets tough; alternatively, it is possible to evaluate φ() twice per iteration, one in the (approximate) Analytic Centre and one in the (approximate) maximizer of the CP-model.

The major drawback of Cost Decomposition methods is certainly the need of maximizing a NonDifferentiable function, where the efficient smooth optimization machinery cannot be used: an attractive alternative is therefore to replace φ() with a "nicer" (smooth) function, such as the *Augmented Lagrangean function*

$$(\mathrm{AL}_{y,t}) \quad \tilde{\varphi}_t(\boldsymbol{y}) = \begin{cases} \min \sum_{h \in \mathcal{K}} \boldsymbol{c}^h \boldsymbol{x}^h + \boldsymbol{y}\left(\sum_{h \in \mathcal{K}} \boldsymbol{x}^h - \boldsymbol{u} \right) + t \left\| \left[\sum_{h \in \mathcal{K}} \boldsymbol{x}^h - \boldsymbol{u} \right]_+ \right\|_2^2 \\ \boldsymbol{E}\boldsymbol{x}^h = \boldsymbol{b}^h \,,\ \boldsymbol{0} \le \boldsymbol{x}^h \le \boldsymbol{u}^h \qquad \forall\ h \in \mathcal{K} \end{cases}$$

where $[\cdot]_+$ is the projection operator over the nonnegative hortant of $\mathbb{R}^m$. It is well-known that $\tilde{\varphi}_t()$ is differentiable, and that there exists a sufficiently large $\bar{t} < \infty$ such that

$$(\mathrm{DAL}_{\bar{t}}) \qquad \max\{\ \tilde{\varphi}_{\bar{t}}(\boldsymbol{y}) : \boldsymbol{y} \ge \boldsymbol{0}\ \}$$

is equivalent (DMMCF); although the treatment is restricted here to Multicommodity Flows, the results are valid for a much broader class of block-structured problems. Being a smooth problem, $(\mathrm{DAL}_{\bar{t}})$ can be efficiently solved by means of very simple methods: in fact, the customary *Method Of Multipliers* amounts at computing $\tilde{\varphi}_t(\bar{\boldsymbol{y}})$ and its gradient $\nabla\tilde{\varphi}_t(\bar{\boldsymbol{y}}) = t\left(\sum_{h \in \mathcal{K}} \boldsymbol{x}(\bar{\boldsymbol{y}}, t)^h - \boldsymbol{u} \right)$, where $\boldsymbol{x}(\boldsymbol{y}, t)$ is the (unique) primal optimal solution of $(\mathrm{AL}_{y,t})$, and update the multipliers as

$$\bar{\boldsymbol{y}} := \left[\bar{\boldsymbol{y}} + 2t\nabla\tilde{\varphi}_t(\bar{\boldsymbol{y}}) \right]_+$$

therefore being just a (projected) gradient method. Clearly, the *penalty parameter t* - that can also be replaced by a vector of parameters, one for each constraint - must be appropriately managed, and second-order methods can be used to accelerate the speed of convergence: however, the MOM in its simplicity often reveals to be efficient enough. The drawback of AL-based methods, at least in the context of Multicommodity Flows, is that $(\mathrm{AL}_{y,t})$ is a large nonseparable (QP): hence, it may be exceedingly costly to solve it with standard packages [DP95], even though the number of MOM iterations is sufficiently small. However, $(\mathrm{AL}_{\bar{y},t})$ can be solved in a *block Gauss-Seidel* fashion, i.e. (approximately) solving

$$\min\left\{ \left(\boldsymbol{c}^{\bar{h}} + \boldsymbol{y} \right)\boldsymbol{x}^{\bar{h}} + t \left\| \left[\sum_{h \in \mathcal{K}} \boldsymbol{x}^h - \boldsymbol{u} \right]_+ \right\|_2^2 : \boldsymbol{E}\boldsymbol{x}^{\bar{h}} = \boldsymbol{b}^{\bar{h}} \,,\ \boldsymbol{0} \le \boldsymbol{x}^{\bar{h}} \le \boldsymbol{u}^{\bar{h}} \right\}$$

where all the commodities but $\bar{h}$ are fixed, and then iterating $\bar{h}$ over $\mathcal{K}$ in a round-robin way: in this case, each subproblem is a Nonlinear separable (MCF), where the cost of arc (i, j) is linear if it is *nonsaturated* $\left(\sum_{h \in \mathcal{K}} x_{ij}^h < u_{ij}\right)$ and quadratic otherwise, for which efficient algorithms are available, such as the specialized ε-relaxation method of [DMZ95]. The method can also be parallelized in a

block Jacobi fashion, using either simple or complex coordination steps [ZaA95], as an application of the Parallel Variable Distribution paradigm [FM94]. Another possibility is the Nonlinear *Simplicial Decomposition* method [VHo77], where the feasible region of $(AL_{\bar{y},t})$ is approximated by the convex combination of some of its vertices, a restricted (quadratic) subproblem is solved giving a tentative optimal solution $\bar{\boldsymbol{x}}$, and k independent Linear (MCF)s using the gradient (i.e. a linearization) of the objective function in $\bar{\boldsymbol{x}}$ as costs are solved to obtain a new vertex: with this implementation, the Augmented Lagrangean method is a variant of the Bundle approach, where the subproblem $(\Delta_{\bar{y},t})$ on the *full* polyhedron $\bar{X}$ (Cf. §I.2.2) is solved to optimality rather than performing a Serious Step whenever possible (possibly indicating a potential advantage of the Bundle method).

A different approach to the "smoothing" of φ() [DPZ91] [PZ92] [PZ94] uses the linear-quadratic (univariate) *Smooth Penalty Function*

$$\sigma_\varepsilon(x) = \begin{cases} 0 & x \le 0 \\ x^2/2\varepsilon & 0 \le x \le \varepsilon \\ x - \varepsilon/2 & \varepsilon \le x \end{cases}$$

so that the "master" problem

$$(M_{\varepsilon,t}) \qquad \begin{cases} \min \sum_{h \in \mathcal{K}} \boldsymbol{c}^h \boldsymbol{x}^h + t\left(\sum_{(i,j) \in \mathcal{A}} \sigma_\varepsilon\left(\sum_{h \in \mathcal{K}} x_{ij}^h - u_{ij} \right) \right) \\ \quad \boldsymbol{E}\boldsymbol{x}^h = \boldsymbol{b}^h \,,\ \boldsymbol{0} \le \boldsymbol{x}^h \le \boldsymbol{u}^h \qquad \forall\ h \in \mathcal{K} \end{cases}$$

has a differentiable objective function: once again, the problem is *not* separable into k single-commodity (MCF)s, requiring e.g. a Simplicial Decomposition method for its solution. Such an implementation of the *Smooth Penalty Function* method is similar to that of the Bundle and AL approaches, but the "restricted" subproblem is *not* a (QP), and therefore calls for more general approaches: for instance, a Truncated Newton Method is used in [PZ92]. A possible advantage of the SPF method over both the AL and the Bundle approaches is that $(M_{\varepsilon,t})$ has only a parameter ε instead of the m-vector $\boldsymbol{y}$: however, it is usually beneficial to use modified Augmented Lagrangeans with vector of multipliers $\boldsymbol{t}$, so that it may conversely be better to have a different ε_i for each constraint.

Once a "basic" technique (ACCPM, AL, Bundle, CG, DW, SPF ...) is chosen, the actual performances depend on many details at the implementation level, such as how the "master problem" and the "subproblems" are solved: hence, it is difficult to assess the relative merits of the different ideas, also because extensive computational comparisons of the different implementations are *not* available. In our opinion, a comparison of "uniform" implementations of all the above methods would be a valuable piece of work.

The Cost Decomposition idea is also behind the recent ε-approximate algorithms, where constraints are dualized by means of an exponential penalty (or a logarithmic barrier) function, discussed in Sec. II.1.4.6; furthermore, these methods give rise to - more or less obvious - opportunities for parallelization, that will be discussed in §II.3.1.

II.1.4.3 Resource Decomposition methods

The basic idea of Resource Decomposition methods stems from a simple observation: if an optimal

solution $\bar{\boldsymbol{x}}$ of (MMCF) would be known, optimal dual variables could be retrieved by allocating *a priori* the capacity (resource) between the commodities, i.e. setting $u_{ij}^h = \bar{x}_{ij}^h$. More formally, the Resource Allocation problem

$$\text{(RD)} \qquad \min\{\ \psi(\boldsymbol{r}) : \Sigma_{h \in \mathcal{K}} \boldsymbol{r}^h \le \boldsymbol{u}\ ,\ \boldsymbol{r} \ge \boldsymbol{0}\ \}$$

with the (polyhedral and convex) function $\psi() : \mathcal{R}^{m \times k} \to \mathcal{R} \cup \{ +\infty \}$

$$\psi(\boldsymbol{r}) = \Sigma_{h \in \mathcal{K}} \min\{\ \boldsymbol{c}^h \boldsymbol{x}^h : \boldsymbol{E}\boldsymbol{x}^h = \boldsymbol{b}^h\ ,\ \boldsymbol{0} \le \boldsymbol{x}^h \le \boldsymbol{r}^h\ \}$$

is equivalent to (MMCF) since any optimal primal solution $\bar{\boldsymbol{x}}$ of (MMCF) also solves (RD) and vice-versa: (RD) is a constrained NonDifferentiable Optimization problem, and a subgradient of $\psi()$ in $\boldsymbol{r}$ is

$$g(\boldsymbol{r})_{ij}^h = \begin{cases} 0 & \text{if } x(\boldsymbol{r})_{ij}^h < r_{ij}^h \\ -\bar{c}_{ij}^h & \text{if } x(\boldsymbol{r})_{ij}^h = r_{ij}^h \end{cases}$$

where $\boldsymbol{x}(\boldsymbol{r})^h$ and $\bar{\boldsymbol{c}}(\boldsymbol{r})^h$ are respectively an optimal solution and *reduced costs* of the h-th (MCF), so that it can be solved by any suitable NDO algorithm, such as a subgradient method [Gr72] [KS77]. (RD) is similar to (DMMCF), but differs in a number of significant details:

- it has mk variables, while (DMMCF) has only m variables;
- its feasible region is more "complicated" than just the nonnegative orthant;
- the function $\psi()$ have value $+\infty$ if the capacities $\boldsymbol{r}$ make one of the (MCF)s unfeasible: furthermore, finding an initial feasible allocation $\bar{\boldsymbol{r}}$, such that $\psi(\bar{\boldsymbol{r}}) < +\infty$, is (in principle) a nontrivial task.

Nevertheless, computational experiences [AHK79] seem to show that $\psi()$ is an "easier" function than $\varphi()$, at least for the purpose of rapidly obtaining lower bounds with a subgradient algorithm: this can perhaps justify the continuing interest in this approach. For instance, an "inexact" Bundle method has been recently proposed [DGM93] where it is not necessary to compute $\psi()$ exactly, but no computational results are still available for that approach; alternatively, in [Ko94] some variants of (RD) are solved by means of the *Alternating Direction Method Of Multipliers*. ADMOM [PR55] [DR56] [GM75] [KM95] is a general method for the solution of problems in the form

$$\min\{\ f'(\boldsymbol{x}') + f''(\boldsymbol{x}'') : \boldsymbol{A}'\boldsymbol{x}' + \boldsymbol{b} = \boldsymbol{A}''\boldsymbol{x}''\ ,\ \boldsymbol{x}' \in \mathcal{R}^p\ ,\ \boldsymbol{x}'' \in \mathcal{R}^q\ \}$$

where $f'()$ and $f''()$ are both proper convex extended-valued functions: the main idea under ADMOM is to form the Augmented Lagrangean of the problem

$$\lambda_t(\boldsymbol{y}) = \min_{x',x''}\ \{\ f'(\boldsymbol{x}') + f''(\boldsymbol{x}'') + \boldsymbol{y}(\boldsymbol{A}'\boldsymbol{x}' + \boldsymbol{b} - \boldsymbol{A}''\boldsymbol{x}'') + t\left\| \boldsymbol{A}'\boldsymbol{x}' + \boldsymbol{b} - \boldsymbol{A}''\boldsymbol{x}'' \right\|_2^2\ \}$$

and to approach its maximization with a variant of the Method of Multipliers where, rather than calculating $\lambda_t(\boldsymbol{y})$ exactly, the minimization is performed on the $\boldsymbol{x}'$ and on the $\boldsymbol{x}''$ separately. That is, given a current point $(\bar{\boldsymbol{x}}', \bar{\boldsymbol{x}}'', \bar{\boldsymbol{y}})$ the algorithm sets

$$\bar{\boldsymbol{x}}' := \operatorname{argmin}_{x'}\ \{\ f'(\boldsymbol{x}') + (\bar{\boldsymbol{y}}\boldsymbol{A}')\boldsymbol{x}' + t\left\| \boldsymbol{A}'\boldsymbol{x}' + \boldsymbol{b} - \boldsymbol{A}''\bar{\boldsymbol{x}}'' \right\|_2^2\ \}$$

$$\bar{\boldsymbol{x}}'' := \operatorname{argmin}_{x''}\ \{\ f''(\boldsymbol{x}'') + (\bar{\boldsymbol{y}}\boldsymbol{A}'')\boldsymbol{x}'' + t\left\| \boldsymbol{A}'\bar{\boldsymbol{x}}' + \boldsymbol{b} - \boldsymbol{A}''\boldsymbol{x}'' \right\|_2^2\ \}$$

and then updates the dual multipliers with the usual formula

$$\bar{\boldsymbol{y}} := \bar{\boldsymbol{y}} + 2t(\boldsymbol{A}'\bar{\boldsymbol{x}}' + \boldsymbol{b} - \boldsymbol{A}''\bar{\boldsymbol{x}}'');$$

assuming that the two separate subproblems in $\boldsymbol{x}'$ and $\boldsymbol{x}''$ always have a solution, the method can be shown to converge (linearly, if *strong* convexity and a Lipschitz condition also hold) to an optimal solution. ADMOM is typically used in the *Lagrangean Decomposition* framework, where $\boldsymbol{x}'$ and $\boldsymbol{x}''$ are two *copies* of the same set of variables $\boldsymbol{x}$ and the linking constraints are $\boldsymbol{x}' = \boldsymbol{x}''$: obviously, this is sensible if the constraints set of the original problem can be partitioned in two parts such that separate optimization over each part is much easier than over their intersection. In the context of Resource Decomposition, different choices are possible, such as the Resource Proximization Splitting

$$f'(\boldsymbol{x}) = \sum_{h \in \mathcal{K}} \begin{cases} \boldsymbol{c}^h \boldsymbol{x}^h & \text{if } \boldsymbol{E}\boldsymbol{x}^h = \boldsymbol{b}^h,\ \boldsymbol{0} \le \boldsymbol{x}^h \le \boldsymbol{u}^h \\ +\infty & \text{otherwise} \end{cases} \qquad f''(\boldsymbol{r}) = \begin{cases} 0 & \text{if } \sum_{h \in \mathcal{K}} \boldsymbol{r}^h \le \boldsymbol{u} \\ +\infty & \text{otherwise} \end{cases}$$

with constraints $\boldsymbol{x} = \boldsymbol{r}$, or the more general Activity-and-Resource Proximization Splitting

$$f'(\boldsymbol{x}', \boldsymbol{r}') = \sum_{h \in \mathcal{K}} \begin{cases} \boldsymbol{c}^h \boldsymbol{x}'_h & \text{if } \boldsymbol{E}\boldsymbol{x}'_h = \boldsymbol{b}^h,\ \boldsymbol{0} \le \boldsymbol{x}^h \le \min\{\boldsymbol{u}^h, \boldsymbol{r}'_h\} \\ +\infty & \text{otherwise} \end{cases}$$

$$f''(\boldsymbol{x}'', \boldsymbol{r}'') = \begin{cases} 0 & \text{if } \sum_{h \in \mathcal{K}} \boldsymbol{r}''_h = \boldsymbol{u} \\ +\infty & \text{otherwise} \end{cases}$$

with constraints $\boldsymbol{x}' = \boldsymbol{x}''$, $\boldsymbol{r}' = \boldsymbol{r}''$: each choice has its advantages and drawbacks, concerning the number of variables, the convergence properties and the actual computational behaviour, but in all cases the computation of the functions can be decomposed per commodity, and hence parallelized. As in the Augmented Lagrangean, the subproblems to be solved are Quadratic (MCF)s, for which efficient codes have been devised. A number of improvements of the basic ADMOM scheme can be devised, such as a *variable penalty term* t and/or a more general matrix-induced norm $\|\cdot\|_H^2$ with a *variable penalty matrix* $\boldsymbol{H}$ in the Augmented Lagrangean term: it is also usually benefical to use *over-relaxation steps*, that is (for the case $\boldsymbol{x}' = \boldsymbol{x}''$)

$$\tilde{\boldsymbol{x}}'' := \operatorname{argmin}_{\boldsymbol{x}''} \left\{ f''(\boldsymbol{x}'') + \bar{\boldsymbol{y}}\boldsymbol{x}'' + t \left\| \omega\bar{\boldsymbol{x}}' + (1 - \omega)\bar{\boldsymbol{x}}'' - \boldsymbol{x}'' \right\|_2^2 \right\}$$

$$\bar{\boldsymbol{y}} := \bar{\boldsymbol{y}} + 2t(\omega\bar{\boldsymbol{x}}' + (1 - \omega)\bar{\boldsymbol{x}}'' - \tilde{\boldsymbol{x}}'')$$

$$\bar{\boldsymbol{x}}'' := \tilde{\boldsymbol{x}}''$$

where $\omega \in (0, 2)$. Only a few comparisons with alternative approaches are known, and the existing results [LMK94] are not very encouraging: intuitively, RD methods should be competitive on tightly coupled problems with few commodities, but it is not clear whether this is really the case.

II.1.4.4 Interior Point methods

Interior Point algorithms have grown a favourable reputation for the solution of very large, sparse Linear Programs: such methods require, at the generic i-th iteration, the computationally expensive solution of a linear system on $\boldsymbol{A}\Theta_i^{-2}\boldsymbol{A}^T$, where $\boldsymbol{A}$ is the constraint matrix of the (LP) and $\Theta_i > 0$ is a diagonal matrix, but, on the other hand, the number of iterations needed to reach convergence is usually very small, achieving a good overall complexity. In its node-arc formulation, (MMCF) is a large (LP) with no more than three nonzeroes per columns, i.e. apparently a good candidate for IP methods: however, direct application of the "standard" techniques requires forming the $(nk + m)$ matrix $\boldsymbol{A}\Theta_i^{-2}\boldsymbol{A}^T$, an expensive task in both memory and time due to the fill-in effect. To circumvent

these difficulties, specialized IP methods have been proposed [CG90] [KKR94] [AS95] [CN96] where, in the spirit of Basis Partitioning methods for the simplex (Cf. Sec. II.1.4.1), the computationally intensive operations are specialized in order to exploit the structural properties of the problem at hand: in this case, one has

$$\boldsymbol{A}\Theta_i^{-1} = \begin{bmatrix} \boldsymbol{B}' \\ \boldsymbol{D}' \end{bmatrix}\Theta_i^{-1} = \begin{bmatrix} \boldsymbol{B} \\ \boldsymbol{D} \end{bmatrix} \quad \text{where} \quad \boldsymbol{B}' = \begin{bmatrix} \boldsymbol{E} & & \boldsymbol{0} \\ & \ddots & \\ \boldsymbol{0} & & \boldsymbol{E} \end{bmatrix} \quad \text{so that} \quad \boldsymbol{A}\Theta_i^{-2}\boldsymbol{A}^T = \begin{bmatrix} \boldsymbol{B}\boldsymbol{B}^T & \boldsymbol{B}\boldsymbol{D}^T \\ \boldsymbol{D}^T\boldsymbol{B} & \boldsymbol{D}^T\boldsymbol{D} \end{bmatrix}$$

and $\boldsymbol{B}\boldsymbol{B}^T$ is a block-diagonal matrix with k blocks of dimension n. Hence, systems on $\boldsymbol{B}\boldsymbol{B}^T$ can be (concurrently) decomposed in the k much smaller blocks, and this can be exploited in several different ways: the overall system can be solved by means of "structured" Cholesky decomposition [CG90] [VC93] [AS95], with Preconditioned Conjugated Gradient methods [KKR94] or by mixtures of the two, as in [CN96] where a Cholesky decomposition of the k blocks is kept and, to avoid forming $\boldsymbol{B}\boldsymbol{D}^T(\boldsymbol{B}\boldsymbol{B}^T)^{-1}$, the "small" $m \times m$ system is attacked with a PCG where at each step a system on $\boldsymbol{B}\boldsymbol{B}^T$ is solved. These approaches easily extend to structured (LP)s with "complicating" variables [JYZ94], such as the Recourse Linear Program in stochastic programming, and even to (LP)s with both "linking" columns and rows [AS95]; also, specialized IP methods currently attain the best polynomial bound for the *exact* solution of (MMCF), $O(k^{3.5}n^3m^{1/2}\log(nBU))$ where B and U are the maximum demand and capacity respectively [KV86], even though this is a theoretical result only because it requires the fast matrix multiplication algorithms.

A different IP approach is developed in [SM90a] [SM90b] [MS91], where separability is exploited in a Cost Decomposition fashion: the method is based on a *shifted logarithmic barrier function*, i.e. the *relaxed* constraints $\Sigma_{h \in \mathcal{K}} \boldsymbol{x}^h \le \boldsymbol{r}$ (with $\boldsymbol{r} \ge \boldsymbol{u}$) are moved to the objective function with a classical logarithmic barrier term, yielding

$$(\mathrm{BP}_{\boldsymbol{r},t}) \qquad \min\{ f_{\boldsymbol{r},t}(\boldsymbol{x}) : \boldsymbol{E}\boldsymbol{x}^h = \boldsymbol{b}^h , \boldsymbol{0} \le \boldsymbol{x}^h \le \boldsymbol{u}^h \ \forall\, h \in \mathcal{K} \}$$

$$\text{where} \qquad f_{\boldsymbol{r},t}(\boldsymbol{x}) = \Sigma_{h \in \mathcal{K}} \boldsymbol{c}^h\boldsymbol{x}^h - t(\Sigma_{(i,j) \in \mathcal{A}} \ln(r_{ij} - \Sigma_{h \in \mathcal{K}} x_{ij}^h)).$$

The shifted barrier is used to make any initial point $\bar{\boldsymbol{x}}$, satisfying the flow conservation and individual capacity constraints, interior w.r.t some $\boldsymbol{r} > \Sigma_{h \in \mathcal{K}} \bar{\boldsymbol{x}}^h$: then, $\bar{\boldsymbol{x}}$ is used as a starting point for an ad-hoc IP method, that *approximately* solves $(\mathrm{BP}_{\boldsymbol{r},t})$. The method computes a search direction $\boldsymbol{d}$ as the solution of the k (MCF)s

$$\min\{ \nabla f_{\boldsymbol{r},t}(\bar{\boldsymbol{x}})^h \boldsymbol{d}^h : \boldsymbol{E}(\boldsymbol{d}^h + \bar{\boldsymbol{x}}^h) = \boldsymbol{b}^h , \bar{\boldsymbol{l}}^h \le \boldsymbol{d}^h \le \bar{\boldsymbol{u}}^h \}$$

where the bounds $\bar{\boldsymbol{l}}^h$ and $\bar{\boldsymbol{u}}^h$ define a "trust region" around $\bar{\boldsymbol{x}}$, and are chosen to enforce the feasibility; then, $\boldsymbol{d}$ can either be used in a standard line search, yielding an extremely fast "coordination phase", or in the more complex k-dimensional search

$$\min\{ f_{\boldsymbol{r},t}([\bar{\boldsymbol{x}}^1 + \delta^1\boldsymbol{d}^1 \,..\, \bar{\boldsymbol{x}}^k + \delta^k\boldsymbol{d}^k]) : \boldsymbol{0} \le \bar{\boldsymbol{x}}^h + \delta^h\boldsymbol{d}^h \le \boldsymbol{u}^h \ \forall\, h \in \mathcal{K} \}$$

that results in faster overall convergence [ZaG95]: this basic step is repeated until a prescribed optimality criteria is satisfied. The solution $\tilde{\boldsymbol{x}}$ may not be feasible w.r.t. the original capacities $\boldsymbol{u}$, but it will be such that $\Sigma_{h \in \mathcal{K}} \tilde{\boldsymbol{x}}^h < \boldsymbol{r}$ so that $\boldsymbol{r}$ can be "squeezed" towards $\boldsymbol{u}$ while keeping $\tilde{\boldsymbol{x}}$ in the interior: the process is then repeated until feasibility ($\boldsymbol{r} = \boldsymbol{u}$) is attained. Then, an optimality phase starts where the barrier parameter t is gradually decreased until an optimal point is reached: the same algorithm is

used for the solution of ($BP_{r,t}$), with the only difference that t is decreased after each iteration instead of waiting for an optimal solution to be achieved.

II.1.4.5 Primal-Dual methods

Primal-Dual methods are a well-known class of algorithms for single-commodity (MCF) problems: a particular variant, the Relaxation approach, is among the most effective approaches to (MCF) in practice. A Primal-Dual method solves the general equality-constrained, separable convex problem

(ESSC) $\min_{\boldsymbol{x}} \{ \sum_{a \in \mathcal{A}} c_a(x_a) : \boldsymbol{E}\boldsymbol{x} = \boldsymbol{b} , \boldsymbol{0} \le \boldsymbol{x} \le \boldsymbol{u} \}$

(where now $\mathcal{A}$ is just the set of variables) by relaxing in a Lagrangean fashion the equality constraints

(PR_π) $\psi(\pi) = \min_{\boldsymbol{x}} \{ \sum_{a \in \mathcal{A}} c_a(x_a) + (\pi\boldsymbol{E})\boldsymbol{x} : \boldsymbol{0} \le \boldsymbol{x} \le \boldsymbol{u} \} - \pi\boldsymbol{b}$

and approaching the corresponding Lagrangean Dual

(PD) $\max_{\pi} \{ \psi(\pi) : \pi \in \mathbb{R}^n \}$;

in the (MCF) case, π is usually referred to as the vector of *node potentials*. $\psi()$ is a concave function, and its subdifferential is (Cf. §I.1.2)

$$\partial\psi(\pi) = \{ \boldsymbol{E}\boldsymbol{x} - \boldsymbol{b} : \boldsymbol{x} \in X(\pi) \}$$

where $X(\pi)$ is the set of *all* optimal solutions of (PR_π): if the cost functions $c_{ij}()$ are *strictly* convex, then (PR_π) has *unique* optimal solution $\boldsymbol{x}(\pi)$ and $\psi()$ is a smooth function, since $\partial\psi(\pi) = \{ \boldsymbol{E}\boldsymbol{x}(\pi) - \boldsymbol{b} \} = \{ \nabla\psi(\pi) \}$. Hence, it is possible to solve (PD) with any smooth optimization approach: Relaxation methods [TB91] choose the *Coordinate Ascent* one, i.e. maximize one π_i variable at a time until the optimum is reached. It is easy to see that the (univariate) maximization problem on π_i is at its optimum if and only if the i-th constraint $\boldsymbol{E}_i\boldsymbol{x}(\pi) = \boldsymbol{b}_i$ is satisfied: hence, Relaxation methods satisfy one violated row of the dualized constraints at a time, and properly set the primal variables by adjusting the corresponding Lagrangean variable, that is used as a "knob".

Relaxation methods can be directly applied to (MMCF) with strictly convex costs [NFT90]: in this case, the vector of potentials is $\Pi = [\pi^1 , .. , \pi^k , \boldsymbol{y}]$, where π^h are the node potentials for the h-th commodity while $\boldsymbol{y}$ are the m Lagrangean variables associated to the mutual capacity constraints (turned into equations by adding slacks). If the cost functions are *Bregman Distances*, such as the quadratic function x^2 and the *entropic* function $x\ln(x)$, the strictly related *Row Action Method* [CL81] can also be used: RAM works on the different Lagrangean Dual

$$\max_{\Pi,\lambda,\mu} \{ \min_{\boldsymbol{x}} \{ \sum_{(i,j) \in \mathcal{A}} c_{ij}(x_{ij}) + (\pi\boldsymbol{E})\boldsymbol{x} - \lambda\boldsymbol{x} + \mu\boldsymbol{x} \} - \pi\boldsymbol{b} - \mu\boldsymbol{u} \}$$

where also the bound constraints $\boldsymbol{0} \le \boldsymbol{x} \le \boldsymbol{u}$ have been relaxed, and can obtain the optimal value of the chosen dual variable (π_i , λ_i or μ_i) and the corresponding primal solution $\boldsymbol{x}(\Pi , \lambda , \mu)$ with a closed form expression, whereas Relaxation methods need an iterative procedure.

These methods can be streamlined to work with "local" data only, that in the (MCF) case corresponds to the set of arcs incident in each node, so that they are well suited for fine-grain parallelization: Relaxation methods admit both synchronous (data-parallel) and asynchronous (control-parallel) implementations [BT89], while RAM can only be implemented in a synchronous fashion [CCZ91] [Ze91] [Ze93].

For the *non strictly* convec case, matters are complicated by the nondifferentiability of $\psi()$, i.e. by the fact that $X(\ \Pi\ )$ is no more necessarily a singleton: for the single-commodity (MCF), however, Primal-Dual methods can still be constructed [Be91], as described in some detail in §II.4.3. Briefly, these algorithms alternate a *primal phase*, where a Max-Flow problem is (implicitly) solved on the *reduced graph*, and a *dual phase* where the (characteristic vector of the) optimal Min-Cut(set) is used as an *elementary* dual ascent direction to change π. Unfortunately, no equivalent primal phase (i.e. no combinatorial Multicommodity Max-Flow algorithm) is known for (MMCF): although Relaxation methods can be generalized to Linear Programs [TB87], and even to the more general class of *Monotropic Programs* [Ro84], the result uses of the *Painted Index Algorithm* (for constructing the equivalent of the *augmenting paths*) that lacks a clear interpretation in terms of Multicommodity Flows. A specialized Relaxation method for block-structured (LP) have been proposed in [Ts91], with the explicit aim of providing an efficient approach to (MMCF): for the block-structured case, the Painted Index Algorithm can be streamlined and implemented by combinatorial operations on properly defined graph structures, but the resulting algorithms (from a private communication by the author) do not obtain particularly encouraging computational results.

II.1.4.6 ε-approximation methods

Starting with the seminal work [LR88] on the All-Pair Unit Demands problem (Cf. §II.1.2), a number of ε-approximation algorithms have been recently developed for Multicommodity Max-Flow problems: other than quickly providing good solutions, they can also be used to prove "approximate" Max-Flow / Min-Cut theorems. For instance, it can be shown [KRA90] [PT93] that, if the minimal capacity-to-separated-demand ratio is at least $O(\log^2 k)$, than a feasible multiflow can be *constructed*, while the required ratio is reduced to $O(\log k)$ when considering multicuts [GVY93] (i.e. Max-Sum rather than Concurrent problems): both these results can be improved if the graph has special properties, such as planarity [KPR93] [TV94]. Much weaker results hold for directed graphs: this can be seen by considering the n-commodity problem on a circle with arcs $(i, i + 1)$ of capacity u and one unit of commodity i to be sent from $i + 1$ to i. Here, the capacity-to-separated-demand of each cut is u, but it must be $u \geq n - 1$ in order for a feasible flow to exist, since each commodity must traverse $n - 1$ arcs to reach its destination: hence, the Max-Flow / Min-Cut gap can be in general as bad as $O(k)$, while an $O(\log^3 k)$ bound can be proved if the *demands* are symmetric [KPT93], i.e. if for any commodity having source i and sink j there exists another one having source j and sink i.

The ε-approximation algorithms are all based on the idea, introduced in [SMa90] for the All-Pair Unit Demands problem, of rerouting a(n entire) commodity at a time from "highly congested" to "lightly congested" arcs: in [LMP91], for instance, this is done by defining arc lengths $l_{ij} = e^{\alpha\lambda(i,j)} / u_{ij}$, where $\lambda(i, j) = \sum_{h \in \mathcal{K}} x_{ij}^h / u_{ij}$ is the *congestion* of arc (i, j), and rerouting some flow from a "long" path to a "short" one. These algorithms admit both deterministic and randomized versions [Go92] with lower (expected) complexity: actual implementations [LPS93] can be practical for finding solutions as long as the required accuracy is not too high, e.g. ε not smaller than 10^{-3}.

Building on this, ε-approximation algorithms have been developed for the more general Fractional Packing [PST91] and Resource Sharing problems [GK94a] (Cf. §I.1.2): although different in the details and the analysis, they are all based on an exponential penalty function analogous to the above. In extreme synthesis, given - with the notation of (FP) - an unfeasible point $\bar{\boldsymbol{x}} \in X$, the basic steps

$$\bar{y}_i = \frac{e^{\alpha(\boldsymbol{A}_i \bar{\boldsymbol{x}} / u_i)}}{u_i} \qquad \forall\, i$$

$$\tilde{\boldsymbol{x}} = \operatorname{argmin}_{\boldsymbol{x}} \{ (\bar{\boldsymbol{y}}\boldsymbol{A})\boldsymbol{x} : \boldsymbol{x} \in X \}$$

$$\bar{\boldsymbol{x}} = \omega\bar{\boldsymbol{x}} + (1 - \omega)\tilde{\boldsymbol{x}}$$

are repeated, together with suitable adjustments of the parameters, until an (approximately) feasible solution is obtained: as in Cost Decomposition methods, the gradient of the penalty function "direct" the subproblem solver in the search for a better point. For the block-structured case, the convex combination may be limited to the only blocks that have produced "good" new proposals $\tilde{\boldsymbol{x}}^h$: in the randomized variants, only one randomly chosen block is solved and modified at each iteration. Analogous techniques can be devised for (RS), that can in turn be used for solving the Block Structured convex program of §I.1.1 [GK95a] (hence (MMCF)): as in [MS91], the idea is a two-phase approach where first a feasible point is obtained, and then the objective function is turned into a constraint for the actual optimization. These methods require $\tilde{O}(\varepsilon^{-2} k \log(m))$ iterations, where $\tilde{O}(\cdot)$ means "up to polylog factors", to achieve an accuracy of ε, yielding a $\tilde{O}(\varepsilon^{-2} knm)$ bound [GK95b] for the ε-approximate solution of (MMCF): if ε is not too small, this compares favourably with the $O(k^{2.5} n^3 m^{1/2} \log(\varepsilon^{-1} nBU))$ bound of Interior Point methods [KV86]. Related algorithms also exist [VG96], based on a *logarithmic* barrier function, that achieve an accuracy of ε in $\tilde{O}(\varepsilon^{-2} m)$ iterations, therefore being more effective when $k >> m$: the combination of logarithmic *and* exponential barrier methods gives an almost optimal algorithm in terms of *coordination complexity* [GK94b] (the cost of the coordination phase), possibly indicating that those methods are well-suited for parallelization.

II.1.4.7 Special cases and theoretical results

Due to space limitations, a number of further results about Multicommodity Flows can only be rapidly sketched here: they are just intended to make it apparent that this class of problems have attracted many researchers from many different viewpoints.

It is known from more than 40 years than the special case of $k = 2$ admits polynomial approaches that do not generalize even to $k = 3$ [RW66a] [RW66b] [Sa66] [Hu63] [Hu64] [Hu69]: research on two-commodity flows has continued [Ra80] [Ra94], and results have been obtained about two-commodity flows in planar graphs [KP93] and the relationships with the integrality property [GP92].
Interesting theoretical results can be obtained if the underlying graph G has some special properties [Ev78] [EM78] [NI89] [NI90] [EC95]: for instance, elegant theoretical results [BGJ70] [EJD77] [Se81] [AD91] [CH92] have been obtained that describe the polyhedral properties of *undirected* multiflows and the so-called "Cut Cone", or connections with the theory of matroids. Also, polynomial algorithms [Lo85] [Ka94] have been found for generalizations of the undirected All-Pair problem; minor results have been obtained for the directed case also [Je66a] [JT72].
Finally, sequential [Pa81] [Pa84] [GMY92] and parallel [NK89] approaches have been devised for Multicommodity Flow problems with *equilibrium* constraints: although these problems have a relevant interest in fields like economic planning and transportation systems analysis and design, the required techniques are out of the scope of the present work.

II.2 A Bundle Approach

In this Chapter, extensive computational experiences with a Bundle-type Cost Decomposition approach to (MMCF) are presented: to the best of our knowledge, only one attempt [Me90] [Me94] had already been done to use Bundle methods in this context, and just making use of pre-existent (and partly outdated) NDO software. A specialized Bundle code for (MMCF) has been developed, as a part of the larger research described in Part I, whose main characteristics are

- a simple "main" logic, based on a new heuristic for the t parameter handling;
- use of the efficient BTT (QP) solver, that allows a Lagrangean Variables Generation strategy;
- a "polymorphic" Lagrangean Relaxation solver, using specialized (and efficient) codes for the solution of the different types of single-commodity Min-Cost Flow subproblems.

The code was developed for being subsequently used - if found efficient - within approaches to more complex problems requiring the solution of (MMCF) as a subroutine: however, many interesting issues related to this usage, such as efficiency of the reoptimization in case of cost / capacity changes, have not yet been faced. The relevant details of the implementation are presented in §II.2.1, while §II.2.2 describes the other (specialized or not) codes against which it has been compared on the data sets described in §II.2.3: the computational results are presented and discussed in §II.2.4.

II.2.1 Implementation details

The MMCFB code follows the general scheme for Bundle algorithms presented in Part I, using a simple implementation that is described in Figure 9. At each iteration, the *predicted increase* $v = \varphi_\beta(\boldsymbol{d})$ is compared with the *obtained increase* $\Delta\varphi = \varphi(\boldsymbol{y}) - \varphi(\bar{\boldsymbol{y}})$, and the current point is moved to $\boldsymbol{y}$ only if a sufficient (relative) increase has been obtained. Otherwise, the "aggregated linearization error" σ is used to test the "reliability" of the newly obtained information: if α is not too large than σ, then the addition of $\boldsymbol{g}$ to β is expected to consistently improve the "accuracy" of $\varphi_\beta()$, otherwise t is decreased, forcing the next tentative point to be "closer" to $\bar{\boldsymbol{y}}$, where $\varphi_\beta()$ is (hopefully) accurate enough. The IncreaseT() and DecreaseT() functions are described in Figure 10, and are respectively based on the *(b)* and *(a)* formulae of §I.5.1: exploiting the (directional) derivative in $\boldsymbol{y}$ rather than in $\bar{\boldsymbol{y}}$ seems to be more reasonable when a step to $\boldsymbol{y}$ has already been decided, and the IncreaseT() heuristic guarantees that t will be increased (if and) only if $\boldsymbol{dg} > 0$, as in the ~SS(*ii*) condition of BT (Cf. §I.5.2). On the contrary, using "near" information might be better if $\varphi_\beta()$ is

already known to be a "poor" approximation of $\varphi()$ at $\boldsymbol{y}$: using the two different heuristics seems to slightly reduce, on the average, the iterations count.

MMCFB

⟨ choose $m_1 \in [0, 1]$, $0 < m_2$, $t_0 > 0$, $t^* > 0$ and $\varepsilon > 0$ ⟩;
$\bar{\boldsymbol{y}} = \boldsymbol{0}$; $t = t_0$; ⟨ 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 $(\Delta'_{\beta r})$ and $(\Pi'_{\beta r})$ for θ 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 α ⟩;
      ⟨ add $(\boldsymbol{g}, \alpha)$ to β ⟩;
      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 9: 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 10: the heuristic t-strategies

Since no "Long Term" t-strategy is implemented, MMCFB is prone to the risk of the "small t" phenomenon (Cf. §I.5.3): however, this is avoided in practice by setting $m_2 \approx 3$, a much larger value than the .9 suggested in [Ki89] and [SZ92]. As a result, the value of t never changes after the first few iterations: therefore, a "good" choice of t_0 is important, altough the heuristics usually suffice to remedy to blatantly wrong settings. In fact, a good guess for t_0 is usually just any value in the order of magnitude of the t's showing up in the first few iterations: this is also the way in which t^* can be chosen, say an order of magnitude larger than t_0. The choice of the other parameters is far less critical: m_1 seems to be properly fixed to 0.1, the absolute safeguards on t should never enter to play (this is easily obtained in the same way as for t_0 and t^*) and the increasing / decreasing factors seem to be properly fixed to 10 and .1 respectively.

An important characteristic of MMCFB is the *Lagrangean Variables Generation strategy*: only the Lagrangean variables that are really necessary, i.e. those that at some iteration have a strictly positive entry of the direction $\boldsymbol{d}$, are explicitly "created" in the (QP) solver. This approach can be seen from a primal viewpoint as a row generation strategy, where a constraint is added only when it is violated by the aggregate primal solution corresponding to $\boldsymbol{d}$: clearly, this is effective only thanks to the *two-level*

active set strategy of BTT, that efficiently supports the on-line *creation and destruction of variables* of ($\Pi_{\beta t}'$). Especially for "easy" problems, few Lagrangean variables ever become nonzero, so that the size of ($\Pi_{\beta t}'$) can be kept much smaller than m: furthermore, the set of active Lagrangean variables is very stable after the first few iterations, so that "pricing" - determining whether new variables have to be generated - can be done with a low frequency.

Some other details of the Bundle-specific part of the implementation might be of interest:

- the β-strategy consists in eliminating the subgradients that have zero θ_i multiplier for more than a fixed number (≈ 20) of consecutive iterations: no aggregation is ever performed, hence no limitations are imposed (in principle) on the Bundle size, but the β-strategy suffices to keep it small enough, and the computational experience showed that severe limitations on the size of β have a negative effect on the convergence speed;
- "pricing" in the LVG strategy is done for all the first p_1 (≈ 30) iterations, and then once every p_2 (≈ 10) iterations: it is not computationally convenient to eliminate an "active" variable y_i once it has been generated, so that the active set is monotonically nondecreasing;
- upon termination, MMCFB can report the "full" optimal solution

  $$[\ \bar{\boldsymbol{x}}^1\ ..\ \bar{\boldsymbol{x}}^k\ ] = \Sigma_{i=1..l}\ [\ \boldsymbol{x}_i^1\ ..\ \boldsymbol{x}_i^k\ ]\theta_i\ ,$$

  where θ is the optimal solution of the last ($\Delta_{\beta t}'$), at the price of the extra memory required for storing all the flows $[\ \boldsymbol{x}_i^1\ ...\ \boldsymbol{x}_i^k\ ]$ corresponding to subgradients that are still in β: however, performances in time are essentially not affected, and furthermore the *aggregate* optimal solution $\bar{\boldsymbol{x}}_A = \Sigma_{h \in \mathcal{K}} \bar{\boldsymbol{x}}^h$ can be recovered at essentially no cost as $\boldsymbol{u} - \boldsymbol{G}_\beta\theta$.

In most cases, and especially if k is large w.r.t. m, most of the time is spent in solving the Min-Cost Flow subproblems: hence, having a fast (MCF) solver capable to efficiently reoptimize after cost changes is obviously important. However, for some instances the subproblems are in fact Shortest Path (Tree) problems, that can be solved much faster than general (MCF)s: therefore, any general-purpose Cost Decomposition (MMCF) solver should have specialized solvers that are invoked according to the properties of each subproblem. In MMCFB, the encapsulation and polymorphism features of C++ are exploited for this purpose: the *abstract base class* `McfClass` defines a standard interface for (MCF) solvers, and *k objects* of proper *derived classes* are constructed within MMCFB according to the properties of each single subproblem, making it easy to deal even with instances with nonuniform subproblems. The *public methods* of `McfClass`, properly redefined in the derived classes, are used for changing the costs, solving the problems and collecting the results: hence, it is easy to extend MMCFB to special classes of instances by just developing the appropriate subproblem solver. Moreover, MMCFB is completely independent from upgrades, or even complete changes, in the (MCF) solver: up to now, two derived classes have been coded, the `RelaxIV` class implementing an efficient Relaxation-type (MCF) solver [Be91] (Cf. also §II.4.3), and the `SPTree` class implementing four classical (SPT) algorithms (queue, dequeue, Dykstra and c-heap [GP88]).

An analogous structure has been used for MMCFB itself: an *abstract base class* `MMCFClass` has been defined, and both MMCFB all the codes described in §II.2.2 has been inserted into *derived classes*, making it possible for an application to use any of them indifferently. Moreover, a service

class `Graph` provides a unified mean to read (MMCF) instances (from files, in several different formats, or from memory) and handle them: preprocessing, changes or queries of the instance's data and memory management are all implemented within `Graph`, making it easy to use an existing (MMCF) solver by only writing small pieces of interface code.

Hence, MMCFB is composed of several independent modules, that are succintly described in Figure 12, where arrows represent functional interactions, thick lines inheritance relationships and dashed lines *virtual* methods redefined in a descendant class. This structure was really useful during the development process, since it made possible to improve the different parts independently and in different instants of time: for instance, the `RelaxIV` class was at a certain point almost entirely rewritten to follow the latest algorithmic developments, and the `SPTree` class was developed much later than `RelaxIV`, when the first instances with (SPT) subproblems had to be solved.

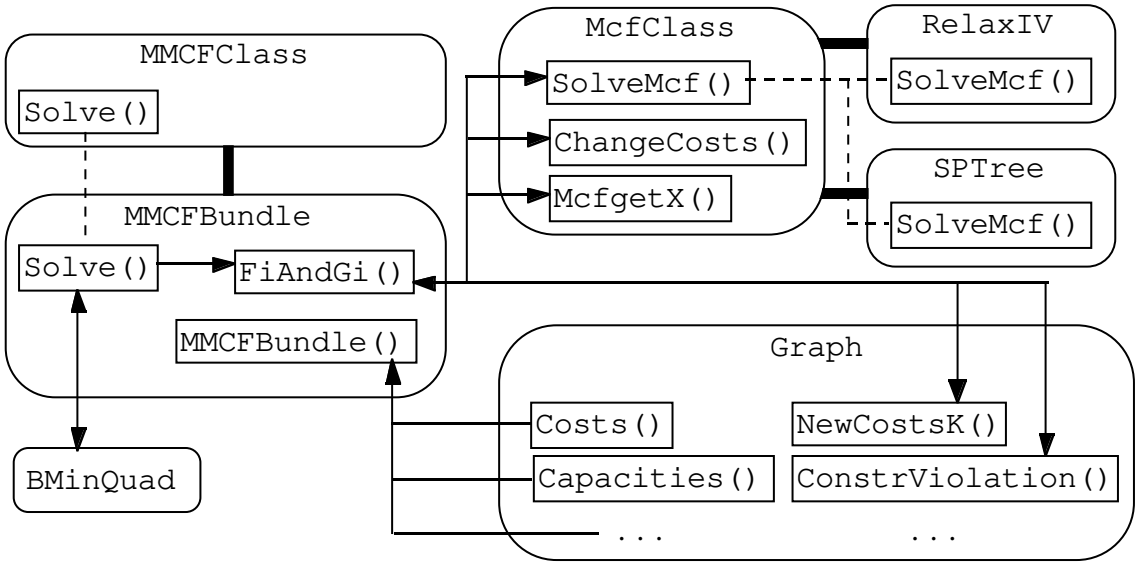

Figure 11: basic structure of the `MMCFClass`, `McfClass` and `Graph` classes

As in the `MinQuad` and `BMinQuad` classes discussed in §I.3.6, there were a number of different alternatives that was not practical to accommodate by means of the inheritance and polymorphism mechanisms, and that were made available in the "compile time switches" fashion: for instance, it is possible to choose among no Lagrangean Variables Generation at all, the standard "monotone" version where no attempt is ever made to delete an active variable and a "full" version where variables are dynamically deleted as well as created. Also, the "abstract" data type mechanism discussed in §I.3.6 was precious in this case: the two types `FNumber` and `CNumber`, designed to hold "flows" and "costs" respectively, can be uniformly changed throughout all the code by modifying a few lines in an appropriate include file, so that, knowing at compile time if flows and/or costs are integer, the (SPT) and (MCF) solver can perform some very basic operations more efficiently, such as all the "$x = 0$" tests that need not to be implemented as "$| x | \leq \varepsilon$".

Although apparently minor, many of these implementation details are critical for the overall efficiency of the code: for instance, the effects of the Lagrangean Variables Generation strategy are shown in Table 1, where the *Rel. Ratio* column shows the attained speedup in the ($\Delta_{\beta t}'$) solution. That is, the LVG strategy decreases the *syncronization cost* by up to a factor of 5, significantly reducing the *total*

execution time (the *Ratio* column): this is especially important for problems with relatively few commodities, where the ($\Delta'_{\beta t}$) cost is relatively high. On the contrary, for problems with many commodities the computation of $\varphi()$ is by far the bottleneck of the algorithm: however, Table 2 shows that, if the subproblems are (SPT)s, then relevant savings on the total execution time can be obtained by using the specialized (relatively naive) solver (*SPT* column) rather then the (sophisticated) generic (MCF) solver (*MCF* column), even on the problems with small graphs and few commodities.

k	*n*	*m*	Ratio	Rel. Ratio
10	50	400	1.47	4.88
10	50	625	1.51	2.72
10	100	1600	1.38	4.22
10	100	2500	1.68	5.18
100	50	535	1.52	2.67
100	30	518	1.72	3.08
100	30	682	1.68	3.02
100	100	1600	1.57	3.73
400	30	520	1.48	2.45
400	30	686	1.50	2.56

Table 1: effect of the variables generation strategy

k	*n*	*m*	SPT	MCF	Ratio
3	47	98	0.02	0.04	2.00
6	85	204	0.09	0.18	1.98
5	34	216	2.42	6.35	2.38
15	43	285	13.36	33.87	2.31
10	190	502	1.23	3.83	3.51
15	285	771	3.38	10.68	3.83

Table 2: effect of using a (SPT) solver instead of a (MCF) one

However, the clearest example is the specialized BTT algorithm: this is shown in the second set of tables in §I.3.7, where it is compared with a non-specialized Interior-Point (QP) using an early version of MMCFB. The solution of ($\Delta'_{\beta t}$) requires up to 20% of the total running time, and the non-specialized (QP) solver is always at least 1.5 orders of magnitude slower than BTT, clearly making a Bundle code impractical for the excessive cost of syncronization phase: note that the other (QP) code tested in §I.3.7 (on ($\Delta_{\beta t}$) only) is exactly the one used within the previous Bundle approach to (MMCF) [Me94]. Furthermore, these results are prudential, since they do not take into account the extra advantages of the BTT algorithm, such as the ability to efficiently cope with variables creation and deletion that is necessary to implement the LVG strategy.

Other than providing good performances, the modular structure of the code made it easier to enhance and to extended it to different problems: in fact, both the parallel version of Chapter II.3 and the extended version for Fixed Charge problems of Chapter II.5 have been obtained by just specialising some of the classes (`MMCFBundle` and `Graph` respectively), while keeping all the rest unchanged.

II.2.2 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 other solution methods described in §II.1.4: hence, an effort has been done to compare MMCFB with recent alternative (MMCF) solvers.

II.2.2.1 A general-purpose (LP) solver

In principle, any (LP) solver can be used to solve (MMCF) problems: actually, commercial general-purpose (LP) solvers can be the *best* choice for solving even quite large Multicommodity problems, since they are often 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 factorisation algorithms, pricing rules,

sophisticated data structures, preprocessing and software technology.

Cplex 3.0 [Cp94] 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. As for the other codes, efforts have been made to use Cplex in its full power, by repeatedly solving subsets of the test problems with various combinations of the available optimization parameters in order to find a sensible setting: this process is very important in order to obtain a fair computational comparison, as the following data will show. Cplex can identify the network structure embedded in a (LP), and efficiently solve the restricted (LP) 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 3, where *Size* = *mk* is the number of variables of the (LP) and figures are averaged within groups of several problems, and depicted in Chart 1.

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 3: 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 4: different pricing rules for Cplex

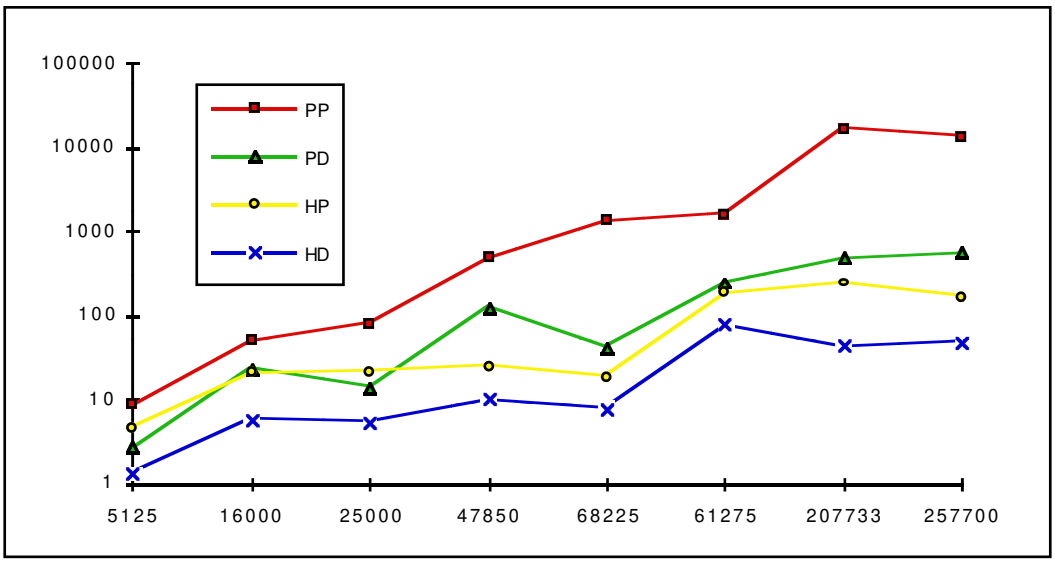

Chart 1: different simplex options for Cplex

For these instances, a Dual approach can be 30 times faster than a Primal one, and the "warm start" speeds up the the solution of 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 performances: Table 4 and Chart 2 report 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*, showing that 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 just selecting the appropriate rule.

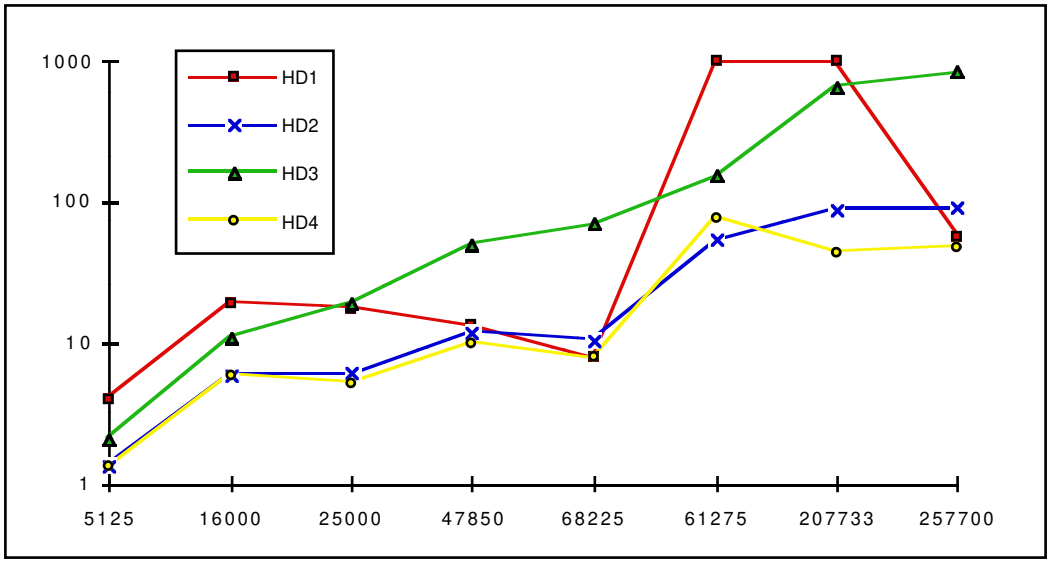

Chart 2: different pricing rules for Cplex

II.2.2.2 A Primal Partitioning code

The PPRN 1.0 [Ca94] code is an available¤ recent implementation of the Reduced Gradient algorithm for the Generalised Nonlinear (MMCF) discussed in Sec. II.1.4.1, i.e. a Primal Partitioning approach when applied to the linear case: in their computational experience [CN95], the authors show that it generally outperforms the known MCNF85 Primal Partitioning code [Ke79], so that it can be regarded as a good representative of this class of methods. As for the other codes, an appropriate tuning of its (several) optimization 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)s rather then from their optimal solutions, as in the "warm start" of Cplex. As shown in Table 5 and Chart 3, in this case also exploiting the optimal solutions decrease the running times of up to a factor of 20 (*Opt* vs *Feas* columns): moreover, appropriate selection of the pricing rule can result in a speedup of 3, as shown in Table 6 and Chart 4 (description of the pricing rules of PPRN is out of the scope of this work). Whem judging the performances of PPRN, it is necessary to take into account the fact that it is an academic code at its first final version, while all the others (except MMCFB) have a much longer history, and it has been written in FORTRAN while all the others have been coded in C (whose compilers may be more efficient): furthermore (from a personal communication by the authors) it does not have a preprocessing phase yet, and due to its input format it was impossible to exploit the preprocessing made by `Graph`.

¤ PPRN is available in a library (.a) format for some different architectures (mainly Unix workstations) at ftp://ftp-eio.upc.es/pub/onl/codes/pprn/libpprn: Jordi Castro can be contacted at jcastro@etse.urv.es

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 5: 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 6: pricing rules for PPRN

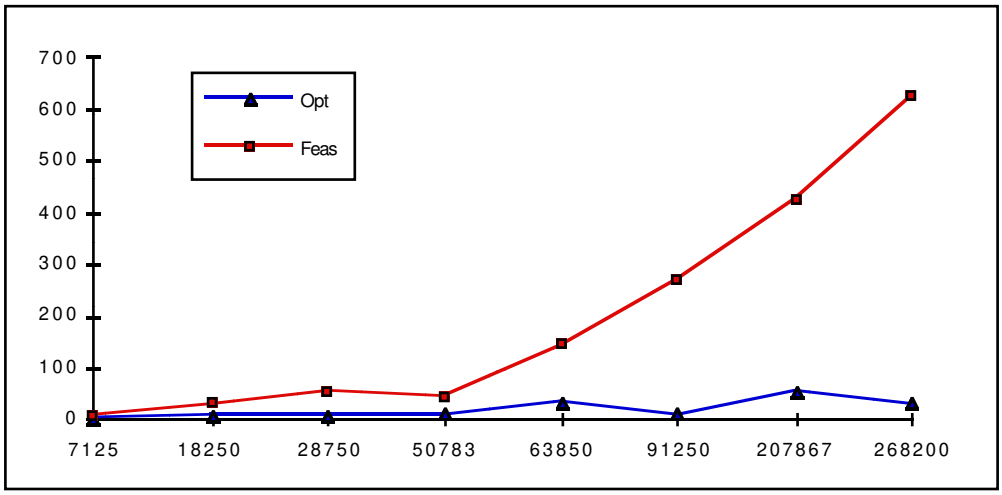

Chart 3: Phase 0 options for PPRN

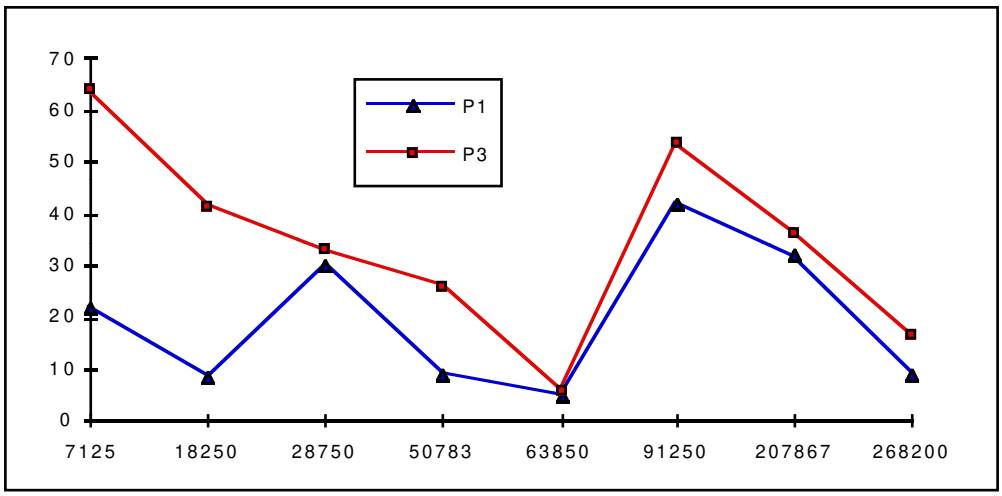

Chart 4: pricing rules for PPRN

II.2.2.3 Interior Point codes

Some preliminari versions of IPM, a specialized Interior Point code for (MMCF) [CN96] that is still evolving quite significantly, were kindly made available by the authors for comparison. The code does not currently handle zero single-commodity capacities, i.e. $u_{ij}^h = 0$, while all but one of the classes of instances have such "forbidden" arcs: the obvious workaround, i.e. setting u_{ij}^h to some very

small number and/or c_{ij}^h to some very large number, might result in numerical problems, yet the code was able to solve (almost) all the instances; on the other hand, explicitly handling this case would require separate symbolic factorization of the k different blocks of the $\boldsymbol{A}\Theta^{-2}\boldsymbol{A}^T$ matrix (Cf. Sec. II.1.4.4), which is costly both in memory and time.

For the non-matrix-based IP method of [SM91], only (indirect) comparisons can be provided, and only for one class of problems: hence, two general-purpose (LP) solvers based on the IP technology were also tested, one being the Cplex 3.0 Barrier solver (that will be referred to as IPCplex) and the other the L0Q0 2.21 code [Va92] [VC93] [Va94], a recent implementation of the *primal-dual, path following, predictor-corrector* approach. Again, some optimization parameters were tuned for improving the performances: the most important is the choice between "myopic" (Minimum Local FillIn) and "global" (Priority Minimum Degree modified Cholesky and Multiple Minimum Degree respectively) heuristics for column ordering when computing the LL^T factorisation of the (full) $\boldsymbol{A}\Theta^{-2}\boldsymbol{A}^T$ matrix. For IPCplex, the myopic heuristic is more efficient, as shown in Table 7 where for each of *MMD* and *MLF* the two columns with *L* and *R* reports the time obtained with Leftward and Rightward Cholesky factorisation (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 factorisation.

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 7: some examples from Cplex-Barrier optimization parameters tuning

Interestingly enough, the *preprocessing* done by `Graph` improves the performances of L0Q0 of over a factor of 10 (vs only a few percent of MMCFB), since it is used to diminish the size of the actual (LP) passed to the solver: this far less important for Cplex, that embeds his own efficient preprocessing phase, but some improvements can still be obtained, showing that knowledge of the (LP) structure makes it easier to locate redundant constraints, and suggesting that a smarter preprocessing phase may be profitable.

II.2.2.4 A Column Generation code

A comparison was also possible with a modern Column Generation code, developed for (repeatedly) solving (MMCF) in a Branch & Bound approach to the Integer (MMCF) [BHV96], and using Cplex for solving the "Master" problem at each iteration. The BCG code was not directly available, anyway it only solves *undirected* (MMCF)s, so that it would not have been able to solve all the other classes of instances without (minor) modifications: however, the authors shared the test instances and the results, even some not reported in the article, allowing a quite satisfactory comparison. On the other hand, only (MMCFB) and the non-specialized codes were able to deal with undirected instances: that

of being orthogonal to the type of subproblem is a seldom noticed but relevant advantage of decomposition approaches to (MMCF).

II.2.2.5 Other codes

The above codes do not exhaust all the proposed computational approaches to (MMCF): only indirect comparisons were possible with other IP methods [ZaG95] [SM91], different Cost Decomposition approaches [PZ94] [ZaA95] and ε-approximation algorithms [GK95a], while no comparison at all was possible with Resource Decomposition approaches [Ko94] or Primal-Dual methods [Ze91]. The indirect comparisons concern the PDS problems (Cf. Sec. II.2.3.3), that are difficult (MMCF) with large graphs but few commodities and (MCF) subproblems, and therefore are not representative of other classes of instances, such as those arising in telecommunications: furthermore, the codes were ran on a really etherogeneous set of computers, ranging from vector supercomputers to massively parallel supercomputers to (clusters of) workstations of different models, making it difficult to extract meaningful information from the time figures.

However, the available codes are at least sufficient to draw a picture of the current scenery of computational approaches to (MMCF), to see where Cost Decomposition methods stand: hopefully, other tests will come to further enhance our understanding.

II.2.3 The test problems

The other fundamental ingredient for obtaining a sensible computational comparison is an adequate set of test problems: several data sets and random generators of (MMCF)s are available, and have been used in the literature to test some of the proposed approaches. However, many of the "standard" 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 complexity of solving a (MMCF) 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: however, when no satisfactory test problems were available, different random generators have been used to produce data sets with controlled characteristics, in order to understand the impact of some of them, such as the number of commodities, on the relative performances of the various approaches.

II.2.3.1 The Canad problems

The first data set is a bunch of 96 problems, generated with two custom random generators (one for bipartite and the other for general graphs) developed in the context of heuristic approaches to the Fixed Charge (MMCF) problem [GC95]: the (MMCF) solved in this Chapter have been obtained by simply ignoring the "fixed charge costs", but the same instances are used again in §II.5.2 in their original setting. A parameter called *capacity ratio* was available as a "knob" to produce lightly or heavily capacitated instances: as (MMCF)s, the problems have shown a posteriori to be quite "easy", but the corresponding Fixed Charge problems are known to be "hard", showing that efficient solution of "easy" (MMCF)s still can 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;

and the characteristics of the instances are shown in Table 8 of Sec. II.2.4.1: for each choice of (k, n, m), two "easy" and two "hard" instances have been generated, whose results will always be reported in an aggregate form.

II.2.3.2 The Mnetgen generator

Mnetgen [AK77] is a well-known random generator: an improved C++ version has been developed for these tests, where the internal random number generator (that has shown not to be portable) has been replaced with the `drand48()` routine in the standard C library, the output file format has been made more "standard" [JL92] and (what appeared to be) a minor bug has been fixed. 216 problems have been generated with uniform rules: the problems are divided in 18 groups of 12 problems each, each group being characterised by a pair (n, k) for n in {64, 128, 256} and k in {4, 8, ... , 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 the 40% of the arcs and 10% of the arcs with an "high" cost, while the figures are 80% and 30% respectively for a hard problem. Finally, the 3 subproblems characterised by the touple (n, k, S, D) ($S \in$ {sparse, dense}, $D \in$ {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" also, have a posteriori shown to be significantly harder to solve (by all the codes) than both the "easy" ones and other instances of equivalent size.

II.2.3.3 The PDS problems

The PDS (Patient Distribution System) instances come from a probelm of transportation of 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) in that the problems have a time-space network whose size grows about linearly with t. This characteristic is often encountered in (MMCF) models of transpostation and logistic problems, even though in typical cases k also grows with the planning horizon, while in the PDS problems it is uniformly fixed to 11: hence, the largest instances have a very large m / k ratio. Moreover, although all the commodities have a single source and destination, the subproblems are not Shortest Paths because of the individual arc capacities: it might be possible to reformulate the model to make it more easily solvable by Cost Decomposition methods, but this raises the question of the impact of the formulation over the relative efficiency of the different approaches, that requires separate investigations.

II.2.3.4 The JLF data set

A relatively large data set was presented in [JLF93] to test the DW approach to (MMCF): it is formed of various (small) groups of small real-world problems with different structures, often with Shortest Path subproblems and with never more than 15 commodities.

II.2.3.5 The dimacs2pprn "meta" generator

Several (MMCF)s, especially those arising in telecommunications, have large graphs, many commodities and Shortest Path subproblems: all these characteristics are separately present in some of the previous data sets, but never in the same instance. Hence, a generator was needed that allows a great degree of freedom in the choice of the problem structure: the *dimacs2pprn* "meta" generator originally proposed in [CN95] was suited to the task, and a slightly enhanced C++ version was developed for these tests. The generator inputs the description of a single-commodity (MCF) in DIMACS standard format*, i.e. a graph $\bar{G}$ with arc costs $\bar{\boldsymbol{c}}$, node deficits $\bar{\boldsymbol{b}}$ and arc capacities $\bar{\boldsymbol{u}}$, and three parameters k, r and f: it outputs an (MMCF) problem with k commodities on the same graph $\bar{G}$, where deficits and capacities of the i-th subproblem are a "scaling" of those of the original (MCF). More formally, 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_{ij}^h are independently and uniformly drawn at random from $[0 .. \bar{c}_{ij}]$, while 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_{ij}^h: 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) problem is not "hard-wired" into the generator, like in Mnetgen, but depends on the initial (MCF): in turn, it can be obtained with a random generator, such as the well-known Rmfgen [GG88]. For two user-provided parameters a and b, Rmfgen 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) were 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 were generated, with k in $\{4, 16, 64, 256\}$, $r = 16$ and $f = 2k$ (so that the mutual capacity constraints are four times "stricter" than the original capacities $\bar{\boldsymbol{u}}$): the individual capacities had been set to $+\infty$, yielding a total of 48 instances with up to one milion of variables, but with (SPT) subproblems.

II.2.3.6 The BHV problems

Two sets of randomly generated *undirected* (MMCF)s have been used to test the Column Generation approach of [BHV96]: 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 capacitated) and 585 o/d pairs that can be aggregated into 48 commodities, yielding a total (LP) size of 6240: a posteriori, they have shown to be quite easy.

The 10 "r10-*x*-*y*" problems all have 301 nodes, 497 arcs (of which 297 capacitated) and 1320 o/d

* details on the format and the (MCF) generators can be found at ftp://dimacs.rutgers.edu/pub/netflow

pairs that can be aggregated into 270 commodities, yelding a total (LP) size of 134190 (note that the "disaggregated" (LP) sizes are respectively of $7.5 \cdot 10^4$ and $6.5 \cdot 10^5$ variables): 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, and the two subgroups have shown to be of a surprisingly different "difficulty".

It would be nice to be able to tell (to an unknowledgeable end-user) which solution approach is more likely to be efficient by just looking at the instances (he/she has) to solve, but at the present day this is very far from being the case: all the above data sets will be made available to the researchers, in the hope to encourage more computational comparisons to be performed and hence to gather a deeper understanding of the matter.

II.2.4 Computational results

In this paragraph, MMCFB is compared to the other codes described in §II.2.2 for the solution of the test problems described in §II.2.3: since the approaches are very different, the performance measure will be the running time in seconds taken by the solution of the problem, excluding loading and preprocessing time. Unless explicitly stated, the times have been obtained on a HP9000/712-80 workstation: for the codes ran on different machines, the original figures will be reported and a qualitative estimate of the relative speed will be obtained - if possible - by comparing the SPECint92 and SPECfp92 performance figures (97.1 and 123.3 respectively for the reference machine¶). For all the codes, the required relative precision of the optimal solution was set to 10^{-6}, i.e. about six digits, exceptions being PPRN that do not allow such a choice, IPM where values smaller than 10^{-5} could cause numerical difficulties to the PCG and some of the runs reported in the literature, that will be explicitly discussed.

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 $size = mk$; a "*" means that the code was not able to solve (all) the instance(s of that group), usually due to memory problems.

II.2.4.1 The Canad problems

The results of the three groups of instances A, B and C are summarized in Table 8, where "*" means that the instance could not even be loaded due to memory problems, "**" that the code broke down with numerical difficulties (a quite unusual outcome for Cplex) and "***" that the computation had to be interrupted because the workstation was "trashing" (i.e. spending almost all the time in paging virtual memory faults): the results are also visualized in Chart 5, Chart 6 and Chart 7 for the three groups separatedly, where the running time of each algorithm is plotted as a function of the *size*.

If compared to the faster code, the general-purpose IP solvers are 2 to 3 orders of magnitude slower on groups B and C and 1 to 2 on group A; furthermore, they are not able to solve the largest instances

¶ most of the figures used here have been obtained from the Performance Database Server at the University of Tennessee http://performance.netlib.org/performance/html/spec.html

due to memory or numerical problems.

	k	*n*	*m*	*size*	MMCFB	Cplex	PPRN	IPM	L0Q0	IPCplex
A	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	* *
	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	*	*
B	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
	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	*	* * *
C	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
	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 8: results for groups A, B and C of the Canad problems

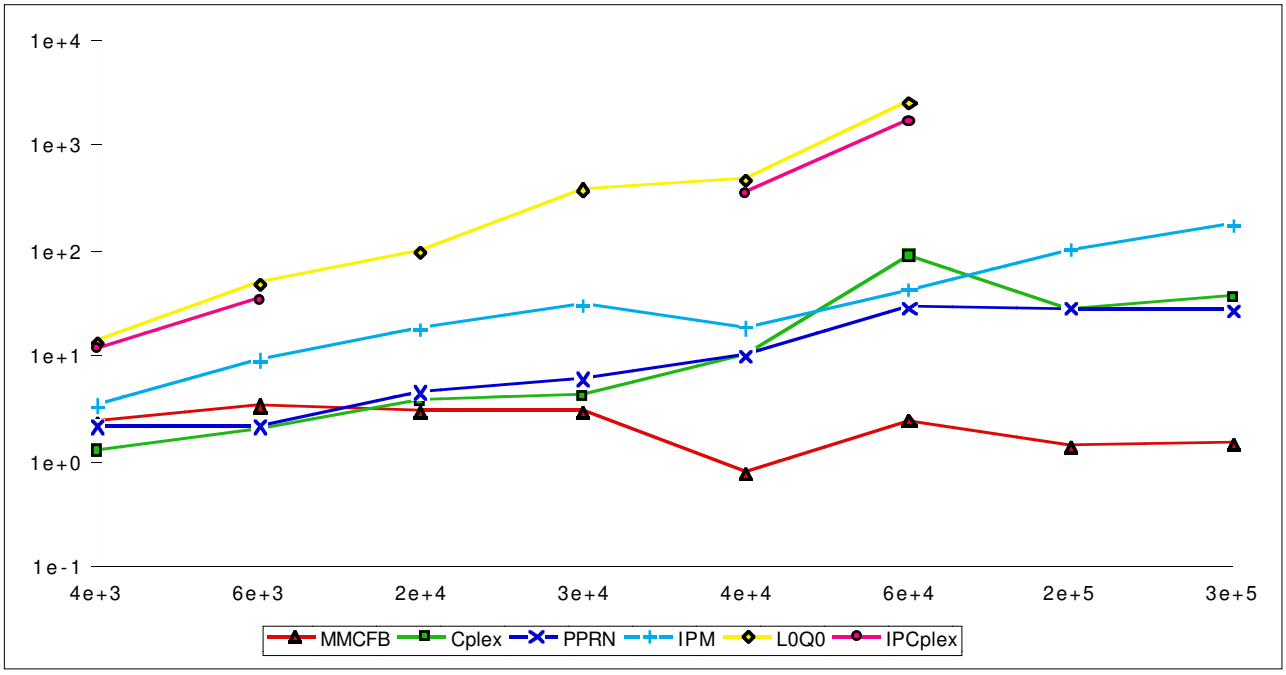

Chart 5: Canad problems, bipartite graphs (group A)

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.

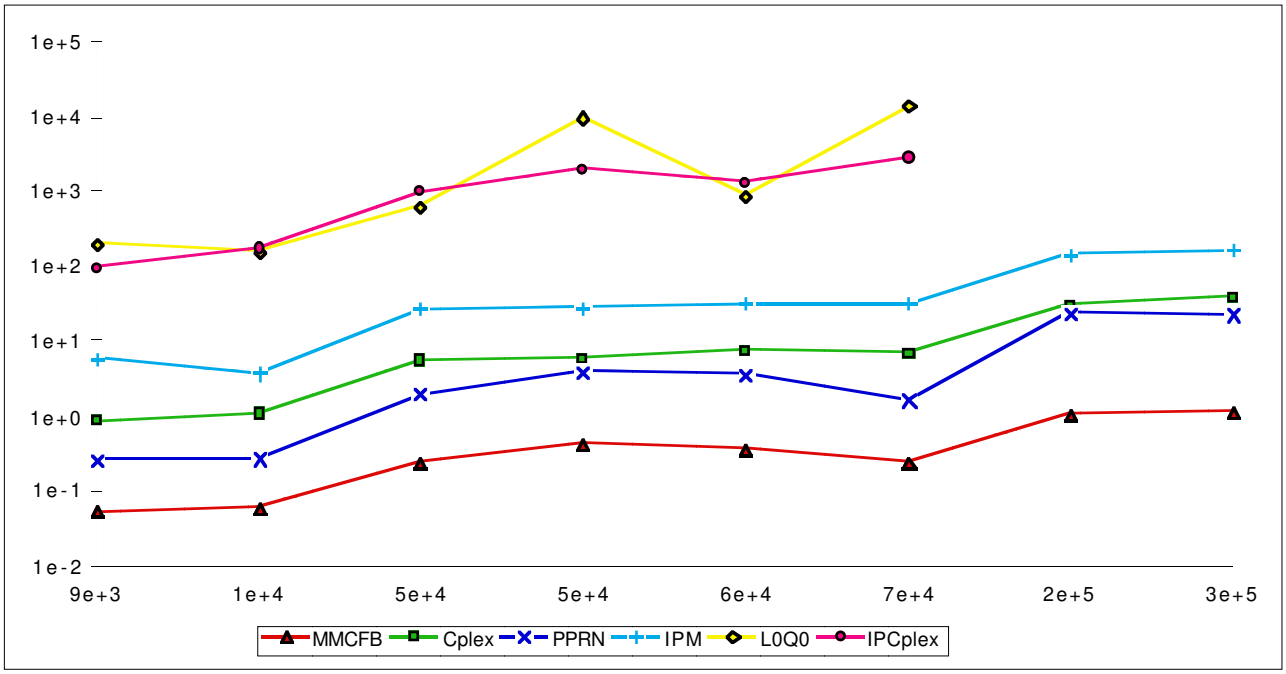

Chart 6: Canad problems, generic graphs (group B)

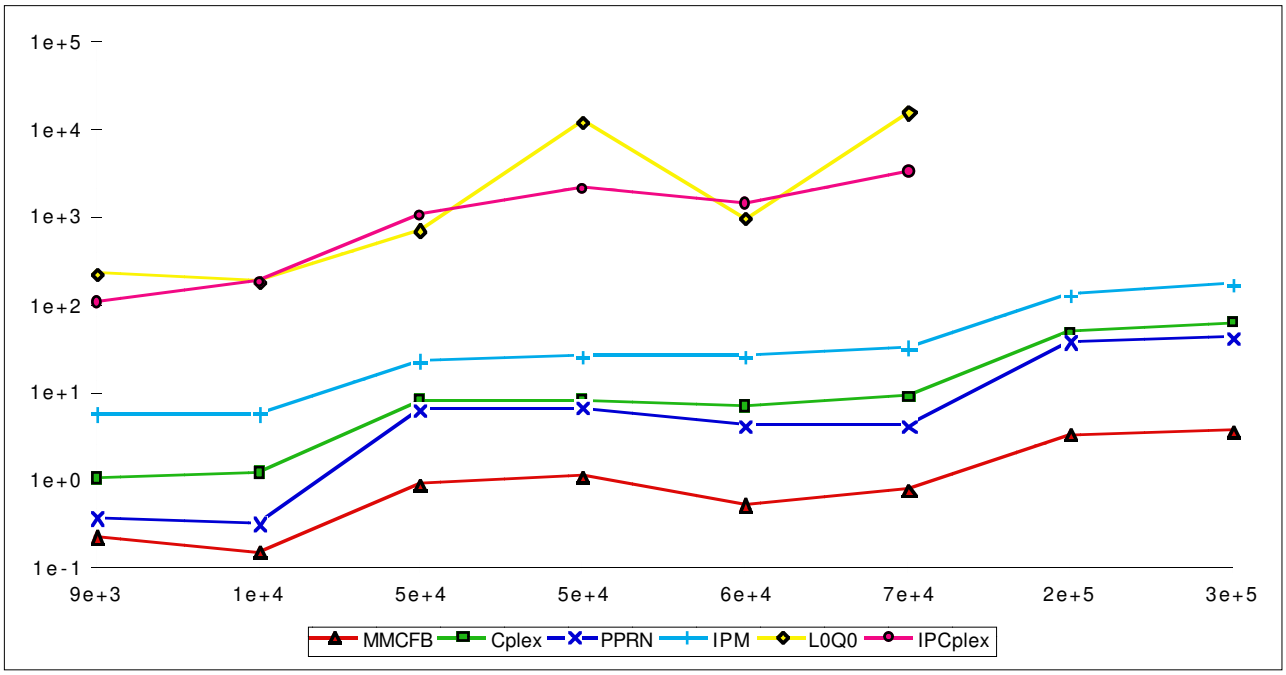

Chart 7: Canad problems, generic graphs (group C)

Among the latters, MMCFB is 10 to 40 times faster than Cplex and 5 to 20 times faster than PPRN on all instances but the first half of group A, where it is less than 2 times slower: these are “difficult” small instances with few commodities, where MMCFB requires considerably more iterations to converge than for all the other problem groups. This may lead to the conjecture that MMCFB is not competitive on “difficult” instances, but other experiences will show that this is true only if the

number of commodities is also small: since there are not difficult Canad problems with many commodities, the relative efficiency of IPM might be underestimated by these results, because, differently from the other iterative methods, Interior Point methods always terminate in a small number of steps almost regardless of the "difficulty" of the instance.

II.2.4.2 The Mnetgen problems

The Mnetgen problems have a wide range of different characteristics, from tiny to very large sizes (more than half a million of variables), and from small to very large m / k ratios: Table 9 summarises the obtained results where all the 12 problems with the same (n, k) have been grouped together. Again, the general-purpose IP codes are never 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 of three orders of magnitude on the largest instances; for example, the hard and dense (256, 256) problems were solved in about 10^3 seconds by MMCFB, while PPRN was not able to produce a result in $4 \cdot 10^6$ seconds and Cplex was trashing the workstation: the expected running times of Cplex and PPRN on those instances are, in the mild estimate used for Table 9 (the "+" entries), more than $3 \cdot 10^6$ and $6 \cdot 10^6$ seconds respectively.

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 9: 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 10: ratio between Cplex and MMCFB times - problems in the same row have fixed n and varying k

However, a closer examination of the results enlightens the weakness of MMCFB on large instances with few commodities: this is better visualised in Table 10, Table 11 and Table 12, and the

corresponding Chart 8, Chart 9 and Chart 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.

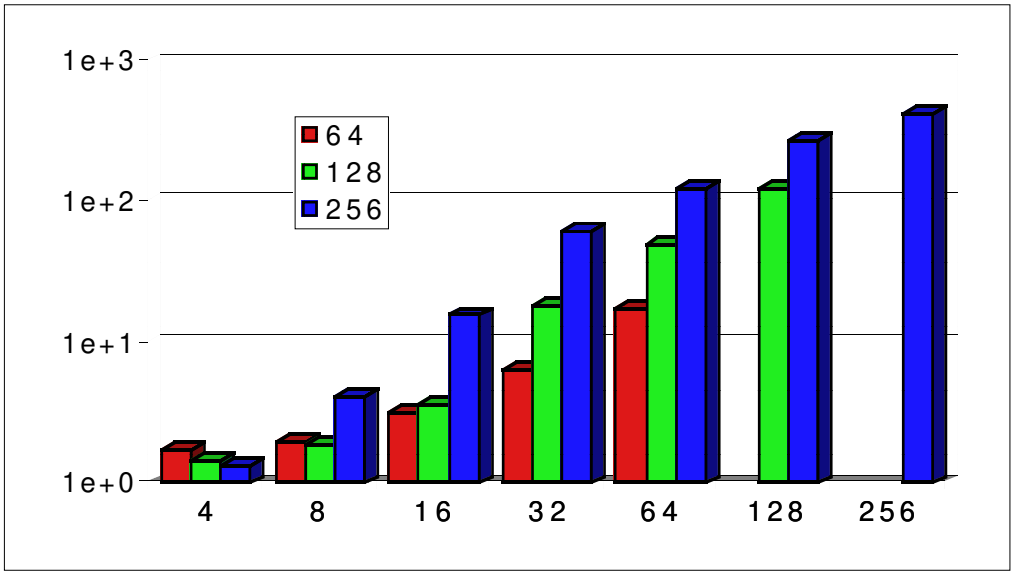

Chart 8: data of Table 10 - for each k, three columns are shown for the three values of n

	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 11 ratio between PPRN and MMCFB times - problems in the same row have fixed n and varying k

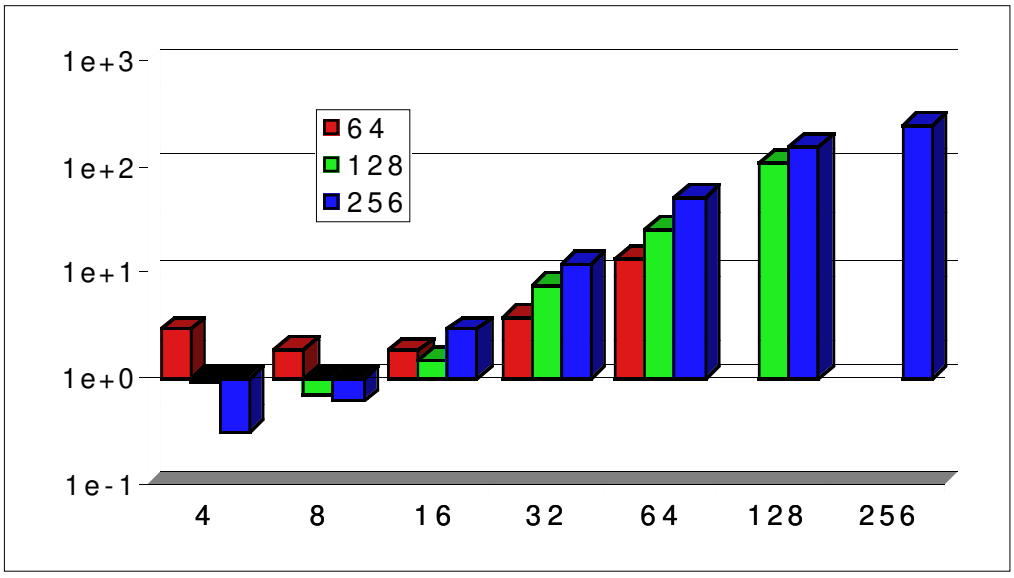

Chart 9: data of Table 11 - for each k, three columns are shown for the three values of n

As the table shows, MMCFB is slower than Cplex of up to a factor of 4 for instances with a large m / k ratio: more disaggregated data about some "critical" 256-nodes problems is reported in Table 13 for a better understanding of the phenomenon.

	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 12 ratio between IPM and MMCFB times - problems in the same row have fixed n and varying k

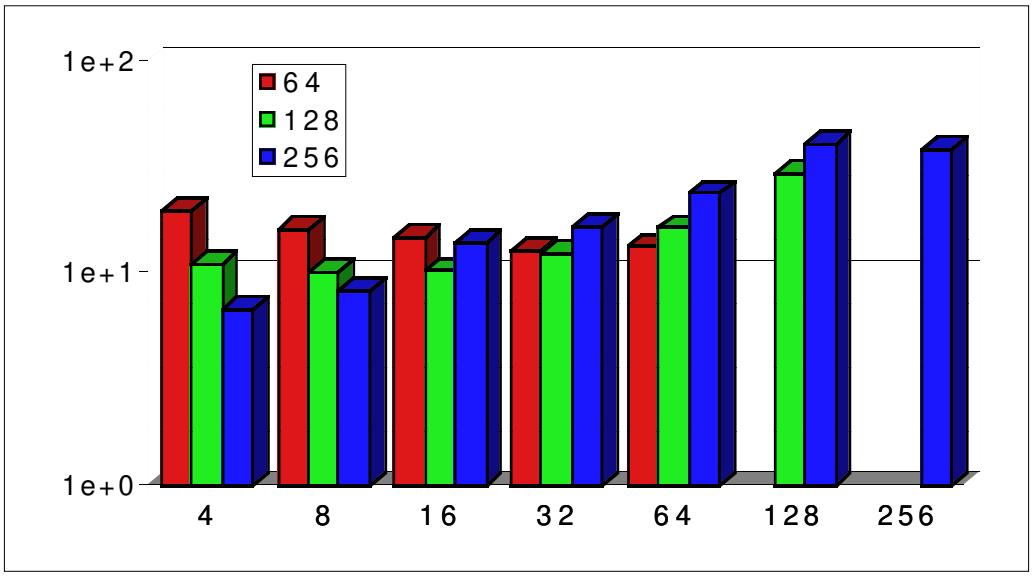

Chart 10: data of Table 12 - for each k, three columns are shown for the three values of n

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 13: disaggregated data about some 256-nodes problems

Noticeably, “hard” problems are really harder (for all the codes) to solve than “easy” ones 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 an hard $1.5 \cdot 10^5$ variables (64, 256, 2300) Mnetgen instance requires over 350 seconds. On easy instances with few commodities, MMCFB is about 2 times slower than Cplex, and about 4 times slower on hard ones: conversely, as the number of commodities increases the *relative efficiency* (the *Cplex%* and *PPRN%* columns) of MMCFB is larger on hard problems 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: the fact that two different simplex-like method have a similar behaviour seem to indicate that this may be a *structural* characteristic of this class of algorithms, i.e. that decomposition methods may *inherently* have better performances on many-commodities problems. As we expected, IPM seem to be less affected by the "difficulty" of the instance than all the other codes: in fact, the relative efficiency of MMCFB (the *IPM%* column) is on the contrary lower on "hard" problems, confirming that the IP 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 convenient only on "hard" problems with small k: this is in fact confirmed by the next data set, and by noting that, in Chart 10, the relative efficiency of MMCFB decreases as n increases for small k, but increases with n for larger values of k. It is important to note, however, that these results have to be taken with some care: in all the Mnetgen instances there are arcs with $u_{ij}^k = 0$, that have been given a tiny capacity and a very large cost, and this workaround may have an adverse effect on the performances of IPM that is currently impossible to estimate.

II.2.4.3 The PDS problems

As shown in Table 14, several PDSt instances were solved up to $t = 40$: the (LP) sizes range from $4.1 \cdot 10^3$ to about $2.4 \cdot 10^5$ variables, but k is always 11, so that the m / k ratio for PDS40 is about 2000. Starting with PDS15, the two general-purpose IP codes began trashing, but even for the smaller instances they are always one order of magnitude slower than the simplex-based codes, as depicted in Chart 11. 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 of 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 was, although it required more than 6 hours. The only other code that was capable of solving all the instances is IPM, that for $t \geq 33$ have quite analogous (but slightly better) performances than MMCFB: recalling §II.2.3, the latter may become competitive earlier on instances with analogous characteristics but (SPT) subproblems, but it seems to be unlikely that a Bundle-based Cost Decomposition method can obtain consistent speedups on these classes of instances.

Since PDS problems have $m / n \approx 3$, it is possible (and instructive) to contrast the results with that on hard (.h) and easy (.e) "sparse" Mnetgen instances: Table 15 shows that MMCFB solves an hard Mnetgen instance much faster than the corresponding PDS one, while the running times of the other methods are far less different (however, the difference increases as k goes far from 11).

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 14: results for the PDS problems

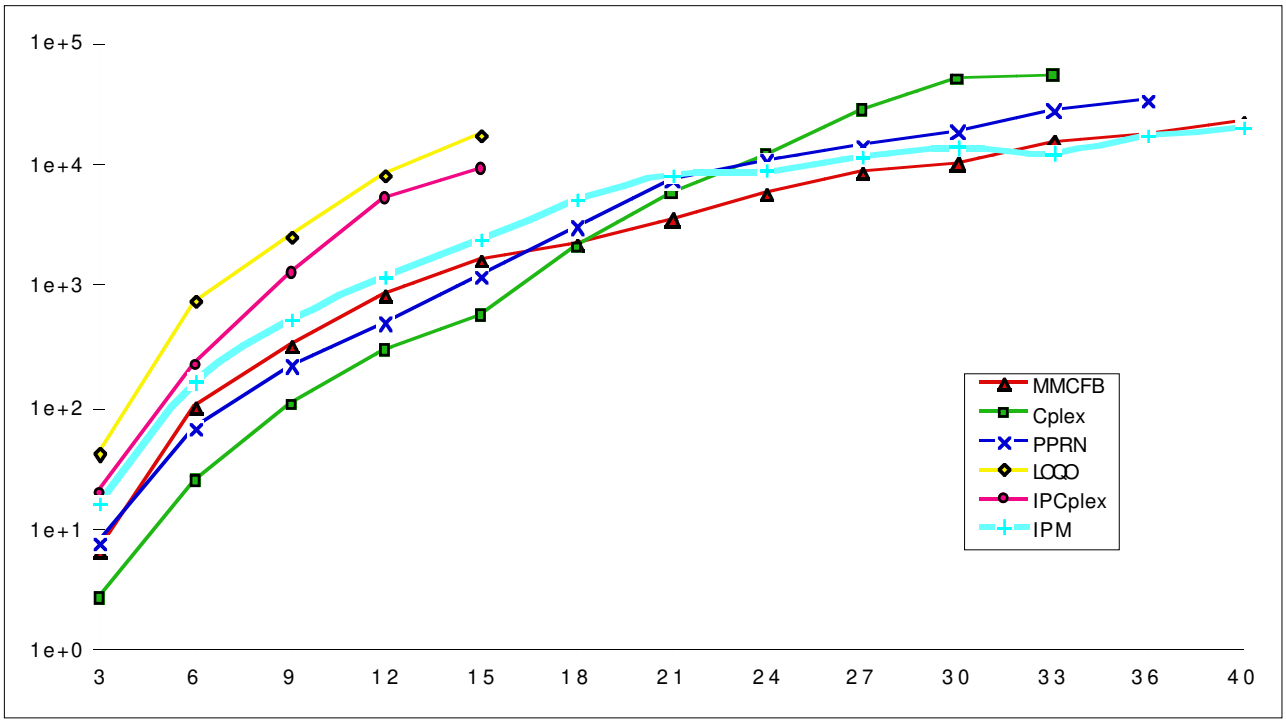

Chart 11: results for the PDS problems

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. Clearly, this statement can be criticised: 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 15: Mnetgen and PDS problems of the same size

On the PDS problems, it is also possible to compare several of the approaches proposed in the literature: in Table 16, this is attempted for PDS1, 10, 20, 30 and 40.

	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 16: comparison with approaches from the literature for the PDS problems

Unfortunately, extracting significant information from the figures is very difficult: the other approaches are

- the parallel Nonlinear Jacobi Algorithm (ZaA) applied to the Augmented Lagrangean of [ZaA95], 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 [PZ94], that was run on a Cray Y-MP vector supercomputer, whose performances w.r.t. scalar architectures are very hard to assess;
- the ε-approximation algorithm (GK) of [GK95a], that was run on a IBM RS6000-550 workstation (83.3 SPECfp92 and 40.7 SPECint92): however, the largest instances were only solved to the precision of 4-5 significant digits;
- the specialized IP method (SM) of [SM91], that was run on a DECstation 3100 (whose SPEC figures are unknown) and that obtained the solutions with the precision of 8 digits;
- a modification and parallelization of the above (ZaG) that was run on a cluster of SPARCstation 20 workstations [ZaG95]; 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).

For these instances, the ε-approximation algorithm provides reasonably good solutions in a very low time: however, the theoretical analysis of the method predicts O($k$log(m)) iterations, so that it is particularly well-suited for the PDS problems, but it may suffer large values of k. 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 machine): however, the comparison is very difficult, and it is perhaps better not to attempt any comment. An interesting remark can instead be done about the concept of "solution" that is underneath some of the figures in the table, since many of the codes 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. To illustrate the issue, in Chart 12 and Chart 13 the value of $\varphi(\bar{\boldsymbol{y}})$ and the φ* estimate ($\varphi(\bar{\boldsymbol{y}}) + t^*\|\boldsymbol{d}\|^2 + \sigma$) of MMCFB are plotted as a function of the time for PDS10 and PDS30 respectively, together with the "solution" of GK (top horizontal line) and the optimum value (middle line): the arrow indicates the (normalised) stop time of GK.

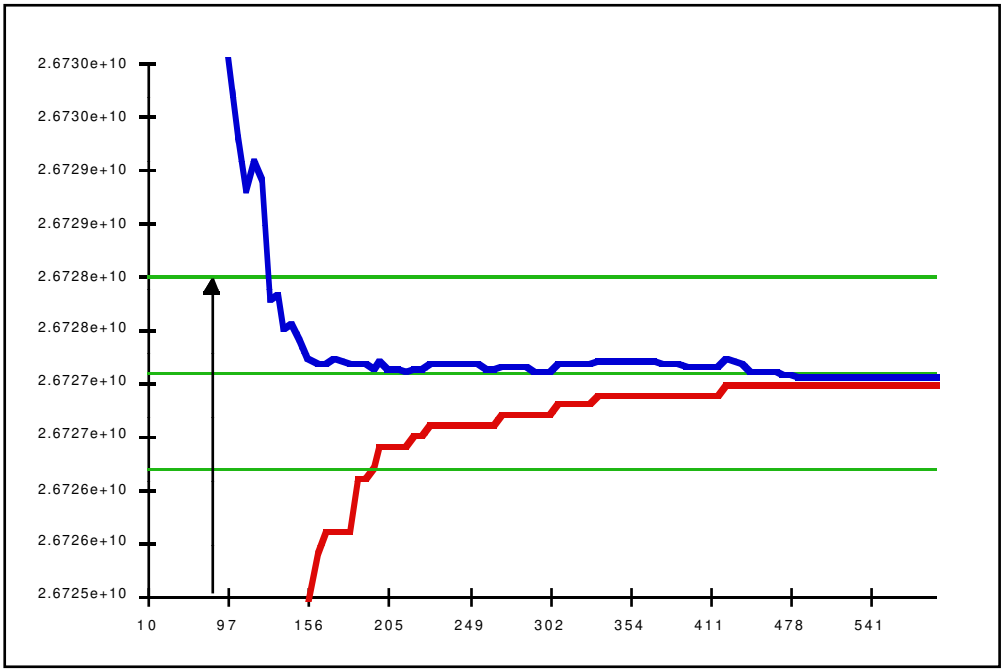

Chart 12: convergence of MMCFB on PDS10

Indeed, the GK algorithm rapidly obtains a good solution, but a comparable accuracy (bottom horizontal line) on the dual side is reached by MMCFB in a fraction of its "solution" time reported in the tables: surprisingly, the "primal" accuracy is even superior to the "dual" one, but MMCFB uses a dual method so that no primal solution corresponding to the estimate is actually available, and since the focus was on the "exact" solution no attempt has been done to obtain it; nonetheless, it may be possible to convert the aggregate unfeasible solution into a feasible (hopefully good) one, e.g. by means of "resource allocation" heuristics, turning MMCFB into an approximate solver.

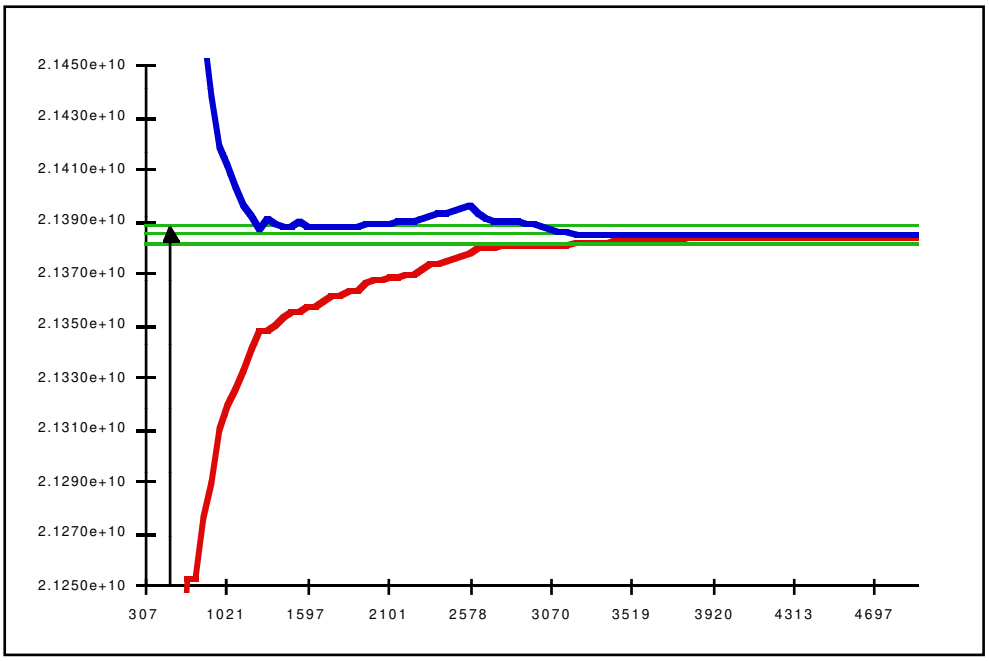

Chart 13: convergence of MMCFB on PDS30

II.2.4.4 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 17: results for some of the JLF problems

Table 17 shows the results obtained on some JLF problems, those with a Shortest Path structure: despite the small number of commodities, MMCFB is still competitive on the "*x*term.*y*" and "Assad*x*.*y*" sets. The "Chen*x*" problems seem to be more "difficult" than the others, since they require an order of magnitude more time to be solved than "*x*term.*y*" instances of comparable size: for this data set, all the IP-based codes are at least competitive, and IPM largely outperform all the others, confirming the potential advantage of the IP technology on "hard" instances. However, such small (MMCF)s are well in the range of general-purpose (LP) solvers, so that it makes a 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 instead.

II.2.4.5 The dimacs2pprn problems

Due to the previous results, the general-purpose IP codes were not run on this data set, that contains the largest instances of all: as described in Sec. II.2.3.5, for any given dimension a *hard* and an *easy* instance have been constructed, whose results are separatedly reported in Table 18 and Table 19 respectively (note the differences in the number of mutual capacity constraints b). As in the Mnetgen case, Cplex ran out of memory on the largest instances (the "*" entries), and PPRN was sometimes stopped after having run for a very long time without producing a result (the "+" entries). Noticeably, 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, so that MMCFB requires an order of magnitude less memory than Cplex, making it possible to solve large (MMCF)s even on low-end computers such as modern PCs; moreover, due to the "abstract" data types described in §II.2.2 memory consumption could also be further reduced. In this case, also IPM experienced memory problems on the largest instances with 1.2 millions of variables: hence, the specialized IP code is more demanding in memory than the specialized simplex code, although being remarkably faster as k increases.

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 18: results of the "hard" dimacs2pprn instances

The results obtained for these instances further confirm those of the previous data sets: MMCFB is faster than the two simplex-based codes on all but the smallest instances, the specialized IP code is competitive with both Cplex and PPRN on the instances with large k, but MMCFB is even more efficient - for the 256-commodities problems it is always over an order of magnitude faster than the

other codes, approaching the three orders of magnitude on the largest instances. Yet, especially on large problems, MMCFB spends most of the time in the (SPT) solver, that could probably be significantly enhanced, for instance by using more sophisticated algorithms or exploiting the solution of the previous iteration.

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 19: results of the “easy” dimacs2pprn instances

The results are also depicted in Chart 14, Chart 15, Chart 16, Chart 17, Chart 18 and Chart 19, where the ratio of the running times of Cplex, PPRN and IPM vs MMCFB is reported for “hard” and “easy” instances separatedly: it is interesting to note that the relative efficiency of MMCFB w.r.t. 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 w.r.t. IPM.

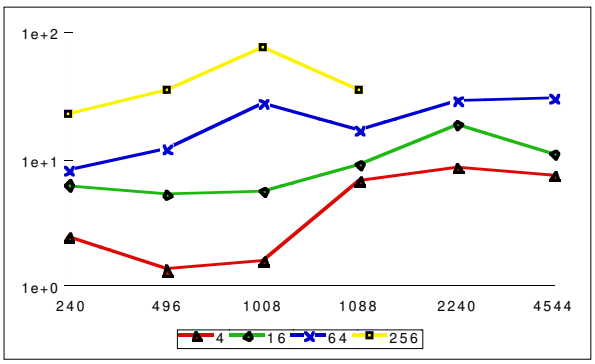

Chart 14: Cplex / MMCFB ratio for easy problems

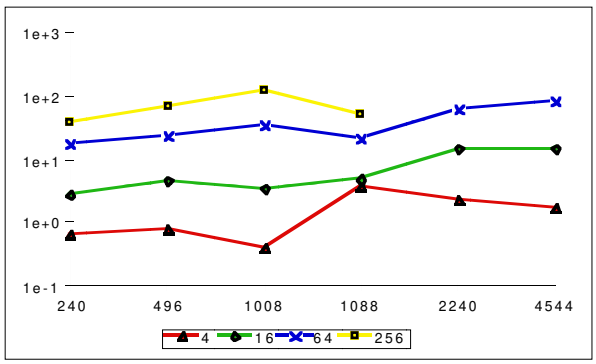

Chart 15: Cplex / MMCFB ratio for hard problems

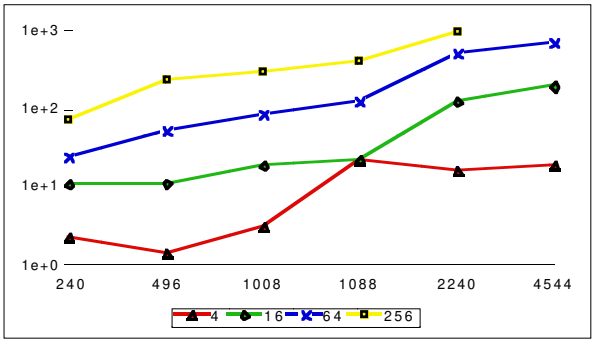

Chart 16: PPRN / MMCFB ratio for easy problems

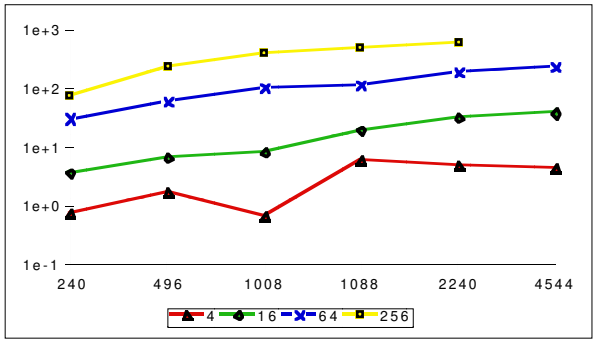

Chart 17: PPRN / MMCFB ratio for hard problems

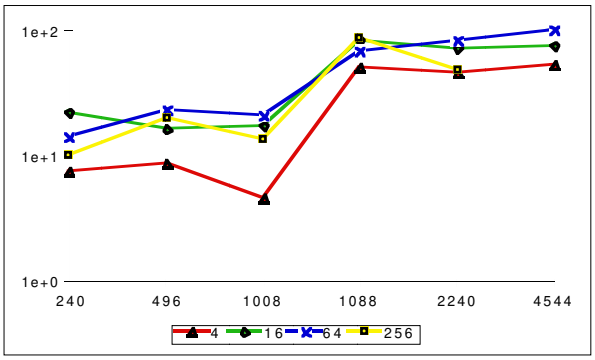

Chart 18: IPM / MMCFB ratio for easy problems

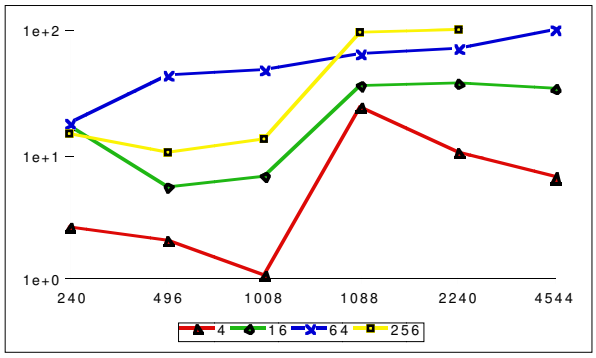

Chart 19: IPM / MMCFB ratio for hard problems

II.2.4.6 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 20: 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 21: results for the "r10-*x*-*y*" problems

Among the available solvers, only MMCFB was "naturally" capable of solving undirected instances, but the three general-purpose (LP) solvers could be adapted for the task; due to the previous results, only Cplex was actually run on these data sets. The times for the Column Generation algorithm (*BCG*) were obtained on a IBM RS6000-590 workstation, credited for 121.6 SPECint92 and 259.7 SPECfp92, i.e. essentially twice as fast as the reference machine: Table 20 shows that on the easy "p*x*" problems MMCFB is about a factor of 10 faster than the other approach. The results on the other data set, shown in Table 21, 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 needed about twice the time. A rationale for the behaviour of MMCFB exists, since a smaller number of trees is likely to be necessary to form the optimal flow as a convex combination in the integral data case, but it is really unclear why the same is not true for the CG approach: should it be proved that this is due to the Bundle-type approach, the result could well be defined encouraging.

II.2.4.7 Conclusions

The above computational experience seems to show that MMCFB can be a valuable mean of solving (MMCF) 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 by a middle-end workstation, provided that the subproblems have a (SPT) structure. Conversely, MMCFB does not appear to be a good choice for problems with a small number of commodities, unless on very large instances: the PDS problems show that MMCFB can be competitive even on problems with an m / k ratio larger than 1000, but the speedup w.r.t. simplex-based codes is far smaller than on many-commodities problem.

The general-purpose IP 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, being also competitive with the simplex-based methods (although not with MMCFB) for larger 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 investigations are needed. Finally, the indirect comparisons with methods from the literature seem 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, they show the need for further computational experiences that help in assessing the effectiveness of the various approaches on the very different class of problems that might need to be solved.

Acknowledgements

Many people have contributed to the work described in this Chapter, mainly (Cf. Acknowledgements) by providing me the codes and/or the test problems for the computational comparisons. Here, I wish to acknowledge Claudio Gentile for his excellent work in coding the largest part of the `RelaxIV` class, and for contributing to the definition of the `McfClass` class: that state-of-the-art implementation of an efficient Min-Cost Flow algorithm was clearly crucial for obtaining an overall effective Cost Decomposition code.

II.3 Parallelizing a Bundle code

(MMCF) is only one example - albeit surely among the most significant ones - of the many models in planning and scheduling that exhibit a block-angular structure, representing spatial or temporal partial decomposability: decision variables can be broken down to largely independent blocks, corresponding to first-level decisions satisfying a subset of the constraints, which may represent a time period, a geographical region or, such as in this case, a commodity. The blocks interact via coupling constraints related to second-level coordination of block decisions, such as shared resource allocation: therefore, the block-angular structure naturally leads to a number of different decomposition algorithms, whereby a globally convergent Master problem is iteratively updated by solving a set of subproblems (one to each block), until some prescribed convergence criterion is met.
In this Chapter, the parallelization of the MMCFB code is discussed: the parallel code *derives* - in a strict sense - from the sequential implementation, that had not been thought for parallelization, and it has not been substantially modified to best suit the parallel machines where it has been ported; the aim was to verify whether exploitation of the "obvious" parallelism inherent in a Bundle-type Cost Decomposition approach was enough to provide consistent speedups.
Our parallel code is only one of the several approaches known in the literature, that are presented in §II.3.1, but it has some interesting features that are analysed in §II.3.2; then, §II.3.3 and §II.3.4 are devoted respectively to the description of the computational environment in which the code was developed and of the test problems on which it was run, while §II.3.5 presents the actual results for two versions of the code and the conclusions that can be drawn.

II.3.1 Parallel approaches to (MMCF)

During the last few years, a large theoretical and practical effort has been done to develop parallel approaches to (MMCF), demonstrating the interest that the solution of large-scale Multicommodity Flow problems has in the OR community. Most parallel approaches exploit the "obvious" parallelism inherent in (MMCF), i.e. separability at the "commodity level": in a number of different ways, the problem is decomposed into k (linear or not) Min-Cost Flow subproblems that are solved in parallel, within a distributed-memory environment, by assigning a dedicated processor to each (group). The results from the subproblems are then used in a "Master" problem, that outputs modified costs and/or capacities for the next iterate: such an algorithm has the common "Master-Slave" structure, in that it alternates between a (serial) Master phase, during which the subproblem processors are idle, and a (concurrent) Slave phase during which the Master is idle. Clearly, the performance of distributed

decomposition, as measured by *speedup* or *efficiency*, is adversely affected by the magnitude of the serial workload in the implementation: thus, efficiency can be expected to be low when the amount of computational effort expended on the Master problem is significant - as might be expected, for instance, when the coupling constraints are numerous or when they are "tight" - although speedup can often be improved by scaling the problem size with the number of processors. Therefore, attempts at improving efficiency directly focus on achieving better load balance between the Master and subproblem processors: deciding which tasks the Master has to accomplish is possibly the critical point of any such parallel algorithm, since there is a trade-off between the complexity of the coordinating mechanism and the overall performance in a parallel environment.

According to [GH93] [DKM94], parallel decomposition methods can be subdivided according to the "complexity" of the Master problem (*coordinator*): *simple* coordinators, that require only summing and averaging of subproblem information, are on the surface better suited to parallel environments, since they reduce the potential serial bottleneck; on the other hand, this may lead to slower convergence relative to methods using *complex* coordinators, that require the solution of an optimization problem, and hence to worse overall performances.

Most Cost Decomposition methods are based on complex coordinators: such are the Dantzig-Wolfe algorithm [HLS88] [GH93], the Linear-Quadratic Penalty algorithm [PZ92] and the Bundle method [Me90] [CFG95] [CF96]. The experiences performed with parallel implementations of the DW method show that a number of implementative details are critical for keeping the cost of the Master problem low, such as performing the "β-strategy" (i.e. elimination of outdated information) concurrently with the φ() calculation, providing only "reliable" (i.e. almost optimal) columns to the Master but possibly providing more than one column for each subproblem at each turn: however, it is generally reported that the Master problem tends to be "dense", so that the serial bottleneck is a main limiting factor in the performances. For (MMCF), however, these results contrast with those in [JLF93] and reveal an issue that has not yet been considered in details, that is the effect of "aggregation" on the results: although disaggregated information may lead to a smaller number of iterations, the fully aggregated information used e.g. by the Bundle and LQP algorithms leads to Master problems of greatly reduced dimension. Hence, the trade-off between convergence and complexity can be different in the parallel case than in the serial one, and possibly different from one algorithm to the other due to the different nature of the Master problem (Quadratic for the Bundle and even more general for LQP): unfortunately, such trade-offs are not clear even in sequential, and more research is surely needed in this area.

Apparently, the "coordination step" of the Augmented Lagrangean method is a very light one: however, parallel implementations of AL actually decompose the computation of $\tilde{\varphi}_t()$ (Cf. Sec. II.1.4.2), so that the problem is simply moved to the "inner cycle". The idea of [ZaA95] is to apply a Nonlinear block-Jacobi algorithm to the computation of $\tilde{\varphi}_t()$: given the current point $[\, \boldsymbol{x}^1 \,..\, \boldsymbol{x}^k \,]$, a subproblem is (concurrently) solved for each commodity h where only $\boldsymbol{x}^h$ is allowed to vary, producing a new tentative point $\bar{\boldsymbol{x}}^h$. Then, the coordination phase uses the new information to select the next current point, a very simple coordinator being the one that just selects the best among the k points $[\, \boldsymbol{x}^1 \,..\, \bar{\boldsymbol{x}}^h \,..\, \boldsymbol{x}^k \,]$, but such an approach may lead to a very slow overall convergence: the complex coordinator obtained by setting $\boldsymbol{v}^h = \bar{\boldsymbol{x}}^h - \boldsymbol{x}^k$ and solving on α

$$\begin{cases} \min \Sigma_{h \in \mathcal{K}} \alpha_h (\boldsymbol{c}^h + \boldsymbol{y}) \boldsymbol{v}^h + t \left\| [\Sigma_{h \in \mathcal{K}} (\boldsymbol{x}^h + \alpha_h \boldsymbol{v}^h) - \boldsymbol{u}]_+ \right\|_2^2 \\ \mathbf{0} \leq (\boldsymbol{x}^h + \alpha_h \boldsymbol{v}^h) \leq \boldsymbol{u}^h \qquad \forall \; h \in \mathcal{K} \end{cases}$$

attains so much better performances to make it preferable to the simple one, despite the larger serial bottleneck due to the solution of a *k*-dimensional nonlinear optimization problem. This *k*-dimensional search is essentially the one that is used in the three phase IP algorithm of [Sc91] [MS91] (Cf. Sec. II.1.4.4), to which the above discussion extends almost plainly: an interesting possibility developed in [ZaG95] is that of attempting to solve the coordination problem itself in parallel by employing *p* complex coordinators of reduced size instead of a single large one. The *p* coordinating problems, each one using the information of *k* / *p* blocks, are solved in parallel to produce a proposed next iterate each, and the next current point is then chosen as the best among such proposals: this is clearly a "two-level" coordination scheme, where the second level is simple, than could be extended to other approaches as well.

A quite analogous opportunity for parallelism is present in the specialized matrix-based IP methods (Cf. Sec. II.1.4.4) [AS95]: there, the solution of the system is divided in two phases, the first concerning *k* small systems - that can obviously be parallelized - and the second using the solutions of the previous phase in a "coordinating" system of moderate size. Hence, quite straightforward implementations of such methods are possible both in shared-memory and distributed-memory environments: results obtained in the analogous case of the Recourse Linear Program in stochastic programming [JYZ94] seem to show that high efficiencies can be obtained.

Not many methods have "essentially" simple coordinators: one is the ADMOM (Cf. Sec. II.1.4.3) used in [Ko94] within a Resource Decomposition approach, and that is based on the ideas underlying the Parallel Constraint Distribution [FM91] and Parallel Variable Distribution [FM94] methods. However, results comparing this class of approaches with Cost Decomposition and (specialized) Interior Point methods [DKM94] seem to show that using complex coordinators is better.

A common feature to all the methods described so far is that the amount of parallelism that can be extracted is not only limited by the serial bottleneck of the Master problem, but also by a more "structural" factor, i.e. the number of commodities *k*: unless each of the subproblems is itself solved by means of a parallel approach (that is, however, theoretically possible in almost all cases), no better speedup than *k* can be expected. Higher levels of parallelism can (in theory) be extracted by Row Action methods [Ze91] [Ze93], where the "unit of concurrency" is the single equation of the linear system: these methods are particularly well-suited for implementation on shared-memory SIMD architectures, that however does not seem to be "leading edge" in parallel computer technology, even though they can clearly be implemented in a distributed-memory environment. A major theoretical drawback of such methods is that they are not currently suitable for the solution of the Linear (MMCF): some related issues will be discussed in Chapter II.4.

II.3.2 Parallelization of the Bundle approach

The aim of the project [CFG95] [Ca96] was to directly exploit the existing MMCFB code, instead of developing an "ad hoc" parallel approach: an advantage of this strategy is that the behaviour of

MMCFB on several classes of problems is well-known, while almost nothing is known about the actual effectiveness, on a serial computer, of many of the parallel approaches proposed in the literature, so that only "relative" measures of parallel efficiency can be obtained. Using the inheritance mechanism of C++, the parallel code (PMMCFB) has been designed in such a way that updates on the sequential code would automatically re-echo on it: for instance, early versions of MMCFB used a Line Search to find the new point ***y*** along the current direction ***d***, that was subsequently abandoned in favour of a more Trust Region oriented approach, but these changes have not required any modification to PMMCFB.

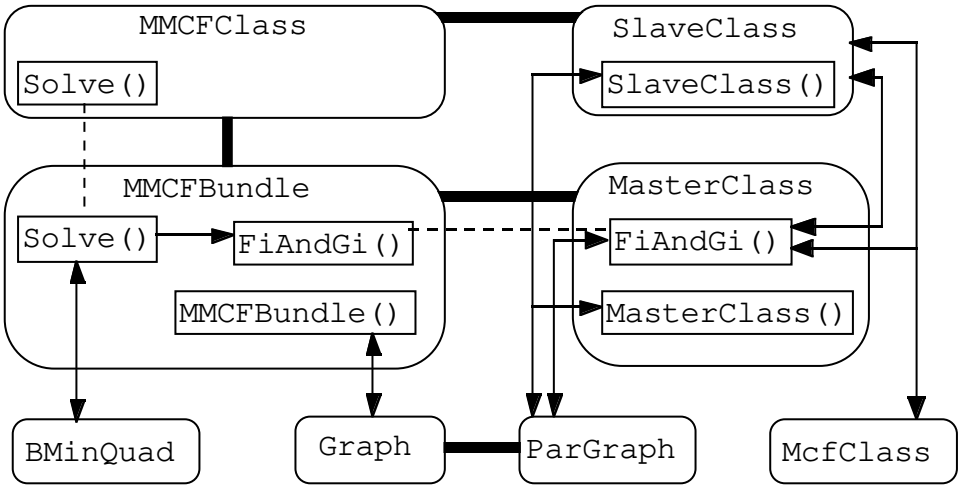

Figure 12: basic structure of the `MasterClass`, `SlaveClass` and `ParGraph` classes

In the C++ terminology, PMMCFB *derives* from MMCFB, as it consists of the three *derived* classes `MasterClass`, `SlaveClass` and `ParGraph`, interacting as shown in Figure 12 with the original classes implementing MMCFB (described in §II.2.2): as usual, arrows represent functional interactions, dashed lines the use of the virtual mechanism and thick lines inheritance relationships. The three new classes are:

- `ParGraph`, that derives from `Graph` and provides support for splitting the data of the problem among the different Processing Elements: a *constructor* of the class that reads the data from a PVM *channel* can be used to set up the data structures of each slave PE, and methods to properly send them the data are provided for being used at the Master PE, where one of the original `Graph` constructors is used instead;
- `MasterClass`, that derives from `MMCFBundle` and differs from its parent class in the constructor - that splits the data among the Slaves by using the `ParGraph` methods - and in the (*virtual*) method `FiAndGi()` only: at each iteration, the sequential code of `MMCFBundle` is executed until `FiAndGi()` is invoked, then control is passed to `MasterClass::FiAndGi()` where the new prices ***y*** are broadcasted to the Slaves, the results are gathered and the corresponding objective function and subgradient are calculated prior to returning the control to the `MMCFBundle` methods;
- `SlaveClass`, that simply implements the (MCF) solution phase - nearly identical to the one in `MMCFBundle::FiAndGi()` - for the subset of the commodities assigned to the `ParGraph` object of the corresponding PE: thanks to the inheritance mechanism and the `ParGraph` class,

extension of the code to any block-structured network problem with side constraint would not require changing a single line of both the `MasterClass` and `SlaveClass` implementations. Although the Slaves could send their solutions to the Master as soon as possible, the use of a *reduce* operation (that involves some form of synchronisation) has been preferred, since efficient implementations are given on almost any parallel machine: besides, in a Bundle approach the Master cannot proceed until all the subproblems are terminated, differently from a DW approach where the Master could begin processing proposals received from the Slaves piecemeal, but even in that case [GH93] such a strategy increases the number of cycles required for overall convergence, thereby cancelling out any benefits due to improved load balance.

Since `MMCFBundle` is in turn derived from `MMCFClass`, the *abstract base class* defining a standard interface for (MMCF) solvers, all the projects where the original sequential code has been used could benefit of a parallel environment without ever knowing that parallel techniques are being used. In this case also, the "abstract" data types mechanism has proved to be useful, making it very easy to port the code under different architectures and tailoring it to the characteristics of the target computer: for instance, in the Cray C90 described in the next paragraph there exists only one integer data type of 8 bytes, while usually integer types of 1, 2 and 4 bytes are also available. Finally, the compile-time switches mechanism has been used to overcome the limitations - or exploit the peculiarities - of each parallel architecture without having to write and maintain different implementations: some of these switches are discussed in the next paragraph, where the computational environment is described in some detail.

II.3.3 The parallel environment

PMMCFB has been mainly tested on a Cray T3D from Cray Research Inc., incorporating 64 state-of-the-art DEC Alpha microprocessors, each capable of 150 MFLOPS peak performance: a Processing Element in the system comprises the Alpha processor, 8 KB of Instruction Cache and 8 KB of Data Cache, 64 MB of local DRAM memory and custom support logic for the fast bi-directional 3-D torus network, ensuring short connection paths and high bisection bandwidth with peak interprocessor communication rates of 300 MB/s in every direction. The Cray T3D system design supports some major latency hiding mechanisms, and offers a wide range of synchronisation mechanisms to accommodate both SIMD and MIMD programming styles: in fact, memory (although physically distributed among processors) is globally addressable, but an highly-optimized implementation of PVM [Cr94a] is also provided for portability of the codes. Cray T3D is running UNICOS MAX, a distributed operating system whose functions are divided between the PEs of the CRAY T3D system and the UNICOS operating system on the Cray C90 host, that allows the user to edit, compile and link programs, and to initiate, control and terminate all T3D processes: the C90 system installed in CINECA is a C94/2148 and consists of 2 autonomous CPUs with 14 independent functional units and 8 vector registers, capable of almost 240 MFLOPS peak performance (clock cycle 4.2 ns).

PMMCFB has been designed to work under any distributed environment where PVM is available, but, via compile-time switches, it is also possible to activate architecture-specific portions of code for all the basic communication and process control primitives, so that it would be quite easy to port the

code under any other message-passing paradigm, such as MPI. As an example, the first thing a PVM task does is to find out if it is the Master process - which is, by convention, the only task that has not a *parent*, that is the task not started-up by a `pvm_spawn()` - but when operating on T3D in Stand-Alone mode the concept of parent does not apply and therefore PE_0 is chosen to be the Master. Analogously, before the Master can send messages to the Slaves, they have to acknowledge their successful start-up and join into a global group for broadcast and reduce operations: the necessary synchronisation can be implemented either explicitly, through a `pvm_barrier()`, or implicitly through message-passing or by counting the members of the group; on T3D, where all processes of the user partition are launched at the same time and a global group is pre-defined for all applications, the synchronisation point is avoided. As far as communication is concerned, PMMCFB uses `pvm_bcast()` to send common data to all PEs, `pvm_send()` for point-to-point communications and `pvm_reduce()` to calculate $\boldsymbol{g}(\boldsymbol{y})$ and $\varphi(\boldsymbol{y})$, but these calls can be substituted with other architecture-specific primitives, such as the `SHMem` (SHared Memory) routines of T3D [Cr94b]; furthermore, to face the case of `pvm_reduce()` not being available, a home-made implementation of the reduce operation, based on a binary tree, is also provided in alternative to $(p-1)$ `pvm_recv()` in the Master.

A remarkable characteristic of the available parallel environment was the side-to-side presence of two radically different architectures, the vector supercomputer Cray C90 and the massively parallel MIMD machine Cray T3D: while the latter is clearly the machine where the $\varphi(\boldsymbol{y})$ calculation has to be performed, the former is particularly efficient on the kind of vector operations that forms the computational core of the (otherwise non-parallelizable) BTT algorithm for the solution of $(\Pi'_{\beta t})$. This self-evident fact was indeed confirmed by a preliminary study, a sample of which is illustrated in Table 22 and Table 23 below: there, the time spent by MMCFB for calculating $\varphi(\boldsymbol{y})$ (*MCF*) and solving $(\Pi'_{\beta t})$ (*QP*) on the two architectures is compared for two particular groups of instances (the 128-128 and 256-128 problems, Cf. §II.3.4).

MCF		QP	
C90	T3D	C90	T3D
26.94	25.51	0.03	0.05
21.98	20.59	0.03	0.04
11.64	10.70	0.02	0.02
45.71	43.77	0.07	0.15
22.85	21.62	0.04	0.08
16.07	14.83	0.04	0.03
243.88	232.47	0.78	1.78
99.12	91.22	0.16	0.73
133.75	126.35	0.23	0.70
526.24	486.24	1.44	4.63
355.84	325.70	0.75	3.62
233.00	217.45	0.45	1.57

Table 22: times for computing $\varphi(\boldsymbol{y})$ and $(\Pi'_{\beta t})$, 128-128

MCF		QP	
C90	T3D	C90	T3D
298.03	298.60	0.25	0.45
129.05	121.07	0.09	0.12
80.51	74.58	0.11	0.12
510.35	499.00	0.45	1.28
217.54	211.27	0.17	0.55
80.74	76.40	0.13	0.25
553.34	554.80	0.92	2.55
529.95	526.59	0.79	6.27
379.24	373.49	0.42	1.82
2518.62	2546.11	6.04	26.39
1203.13	1167.85	2.05	11.54
777.06	786.76	1.23	6.63

Table 23: times for computing $\varphi(\boldsymbol{y})$ and $(\Pi'_{\beta t})$, 256-128

The "highly combinatorial" (MCF) solver used in MMCFB is not faster on the powerful C90 than on a single PE of the T3D, while BTT is up to six times faster, and the difference increases with the problem size (larger in Table 23 than in Table 22, and in the second six problems of each table than in

the first six): this is not surprising, since RelaxIV uses virtually no vector operations and is therefore almost no vectorizable, while the costly operations in BTT are all vector ones. Furthermore, all the results have been obtained by only invoking the vectorization options provided by the C90 compiler, but further performance improvements could probably be obtained by hand-made vectorization of the BTT code, that has been originally developed for Unix workstations.

These results clearly suggest the possibility of an *asymmetric version* of PMMCFB, where each of the two basic operations is executed on the machine that is suited most for the task: using the compile-time switches technique to accomodate the necessary modifications, however, allowed not to develop two separate codes. Some details of the asymmetric implementation are worth description: for instance, with PVM operating in Distributed Mode, C90-T3D communications should go via the *PVM-daemon* (running on the C90) and the slow Unix *sockets* system, and since the number of socket connections is limited, by default only the T3D PE_0 is given a connection to the daemon - additional connections can be made, but performance rapidly degrades. Therefore, in the specialized implementation the PE_0 of T3D acts as a SubMaster between the Master running on C90 and the Slaves running on the other PEs of T3D: otherwise being a "normal" Slave, i.e. solving its share of (MCF)s, only the SubMaster directly communicates with the C90, so that most of the data exchange is performed via the fast T3D interconnection network. For instance, the reduce operation is implemented by first accumulating the Slaves solutions in the SubMaster, and only then sending the results to the Master: this also served to overcome a difficulty arising with `pvm_reduce()`, i.e. that dynamic groups consisting of tasks running on both the T3D *and* the C90 cannot be formed. For Master-SubMaster communications, it is possible either to use PVM alone or to combine it with I/O operations [Cr94a]: the idea is to separate control information and actual data, by using the slower PVM (Unix sockets) for the first and faster UNICOS I/O features (read/write operation on a memory-mapped file) for the second. Some other (compile-time) choices are left to the user, such as if the Master has to solve a group of (MCF)s or not.

II.3.4 The test problems

For these tests, 78 Mnetgen instances have been used: but for the very large problems, the same data set had been previously used for testing the efficiency of MMCFB. The same rules described in §II.2.3 have been used, generating 12 different instances for each of 6 *classes* of problems characterised by a pair (n, k) with n in { 64, 128, 256 } and k in { 64, ... , n }: within each class, 6 problems (*groups* A and B) are "sparse" and 6 (groups C and D) are "dense", and both these subclasses are in turn subdivided of into 3 "easy" (A and C) and 3 "hard" (B and D) problems; usually, within the same group the difficulty of the problem decreases as the percentage of arcs with individual capacities increases, since the number of "tight" mutual constraints diminish. To test the code on larger problems, the "sparse" (512, 512) groups (A and B) were also generated, whose (LP)s have more than $8 \cdot 10^5$ variables: curiously, it was not possible to test the (512, 512) "dense" groups, or problems with more nodes, because of limitations of the *disk space* available on the server machine, while the computational results show that even much larger problems could be solved in a reasonable time. Table 24 resumes the (averaged) characteristics of the test problems: each group is denoted by *n-k-g*, where $g \in$ { A, B, C, D }, *Constr* (= $nk + b$) is the total number of constraints and

Var (= *mk*) is the total number of variables of the (LP) formulation.
Actually, preliminary experiments were conducted on a larger class of problems, typically having less than 64 commodities: lately, those problems were discarded for two reasons, the first being that it was impossible to exploit all the parallelism provided by the available computers, and the second being that they were solvable within a few seconds on a workstation, so that their parallelization was of a limited practical interest. However, many-commodities problems were a posteriori found to be "best suited" for the Bundle approach, so that the results are in some sense "biased": still, problems with such a structure are quite common in practice, for instance in telecommunications, and moreover the limited amount of CPU time available imposed a selection among many possibilities; the current set of problems seems to be able to indicate both potentials and limitations of the proposed approach.

group	*k*	*n*	*m*	*b*	*Var*	*Constr*
64-64-A	64	64	209	86	13376	4182
64-64-B	64	64	197	160	12608	4256
64-64-C	64	64	516	193	33003	4289
64-64-D	64	64	524	415	33536	4511
128-64-A	64	128	388	155	24811	8347
128-64-B	64	128	405	328	25920	8520
128-64-C	64	128	1159	458	74155	8650
128-64-D	64	128	1185	932	75819	9124
128-128-A	128	128	394	156	50475	16540
128-128-B	128	128	407	332	52053	16716
128-128-C	128	128	1219	486	156032	16870
128-128-D	128	128	1213	966	155307	17350
256-64-A	64	256	825	327	52779	16711
256-64-B	64	256	801	647	51264	17031
256-64-C	64	256	2337	922	149547	17306
256-64-D	64	256	2330	1828	149120	18212
256-128-A	128	256	814	304	104149	33072
256-128-B	128	256	812	655	103893	33423
256-128-C	128	256	2344	909	300032	33677
256-128-D	128	256	2353	1861	301184	34629
256-256-A	256	256	824	314	210859	65850
256-256-B	256	256	830	679	212480	66215
256-256-C	256	256	2168	841	554923	66377
256-256-D	256	256	2190	1772	560640	67308
512-512-A	512	512	1633	644	836096	262788
512-512-B	512	512	1635	1291	837120	263435

Table 24: characteristics of the problems

II.3.5 Computational results

In this paragraph, the computational results obtained by the *symmetric* and *asymmetric* versions of PMMCFB are presented and discussed: the results are analysed from the "parallel" viewpoint, i.e. in terms of speedup and efficiency, but the asymmetric version creates some problems with this respect that call for an initial discussion.

II.3.5.1 Performance Measures

It is nowadays widely acknowledged [HP90] that a reliable performance metric for a computer system is just *program execution time*, since it incorporates several other measures of goodness, that, when taken independently, do not portray accurately the overall performance: in a sequential computer, these measures are the quality of the instruction set, the capability of the compiler to produce code that keeps the ALU busy, the latency and bandwidth of the memory subsystem, the clock rate of the chip and the rate of execution of floating-point operations. When the object of benchmarking is a parallel system, program execution time can also mirror the communication and synchronisation costs, the

latency and bandwidth of the interconnection network and the quality of the corresponding routing scheme: hence, the *performance measures* [BT89] [Pa89] [GW89] used hereafter will be based on the execution time. Let now T_p stand for the execution time of a program on a system with p exactly alike processors and T^* be the time it takes to execute the "best" known serial algorithm for the same problem, on the same data set, on a single processor: widely accepted efficiency measures are the *absolute speedup* $s_p^* = T^* / T_p$, the *relative speedup* $s_p = T_1 / T_p$ and the *relative efficiency* $\mu_p = s_p / p$. The value of μ_p is usually less than one (i.e. $s_p \quad p$), indicating efficiency loss due to a number of factors such as overhead introduced by communication, duplication of operations and PE suspension (forced idleness) due to load imbalance or message dependencies: in fact, *linear speedup* ($s_p \quad p$) is the best one can hope for, since any algorithm running on p processors can be simulated as p serial copies running on a single processor in roughly p-fold time. However, in the literature (and in our experience) *superlinear* speedups ($s_p > p$) have sometimes been observed: this phenomenon is usually due to the effect of *memory hierarchies*, i.e. to the fact that the data of the problem is subdivided between the different PEs, possibly reducing cache misses and/or page faults and the corresponding disk I/O. However, a linear speedup is usually difficult enough to achieve, because of the limit on the amount of parallelism that can be extracted from an algorithm: this is quantitatively described by Amdahl's Law [Am67], which states that *parallel speedup cannot exceed the inverse of the fraction of the residual sequential code*, in formulas

$$\mu_p \le \mu_p^I = \frac{1}{(p-1)s+1}$$

where $s \in [0, 1]$ is the fraction of inherently sequential (non-parallelizable) code - for PMMCFB, s is essentially the time that is *not* spent in $\varphi()$ calculation.

In certain cases, the above definitions can be problematic to apply: few problems admits an uncontested "best" serial algorithm, and the actual performance may strongly depend on the particular data set and on the quality of the implementation. Moreover, the *absolute* speedup does not reflect the *degree of parallelization* of the algorithm, as measured against its serial implementation, while the *relative* speedup and efficiency do: fortunately, for the instances at hand MMCFB is one of the most efficient approaches (Cf. §II.2.4), and consequently $s_p^* \quad s_p$. A more serious problem arise with the *asymmetric* PMMCFB, where different parts of the code are run on very different machines: clearly, the above definitions cease to be sensible, and several questions on how to define an appropriate efficiency measure for this case should need to be answered. Although a number of different alternatives can be proposed, the best choice is probably not to address the question at all, presenting the data "as is" and leaving the conclusion to the reader: since the purpose of this research was essentially practical, it will be enough to show that the asymmetric implementation allows to sensibly reduce the overall solution time.

II.3.5.2 Results for the symmetric version (Cray T3D only)

The computational results obtained by the *symmetric* PMMCFB on the Cray T3D are reported in the following tables, where T_p denotes the running time, in seconds, taken by the solution of the problem (excluding loading and preprocessing time) with p processors, μ_p^I the *ideal efficiency* according to the Amdahl's Law, and μ_p the actual *relative efficiency* achieved; near to the *sequential time* T_1, s will

indicate a very close estimate of the fraction (expressed as percentage) of inherently sequential code. In Table 25, the aggregated computational results for all the problem classes are reported: the results are visualised in the corresponding Chart 20, where the obtained relative efficiency is depicted as a function of both p and the problem class, and in Chart 21, Chart 22 and Chart 23 where the ideal and actual efficiencies are compared for p equals to 4, 16 and 64 respectively.

group	T_1	*s*	T_4	T_{16}	T_{64}	μ_4^I	μ_{16}^I	μ_{64}^I	μ_4	μ_{16}	μ_{64}
64-64	21.31	1.10	5.98	2.08	1.00	0.97	0.86	0.61	0.90	0.64	0.34
128-64	123.66	1.25	35.70	13.16	7.01	0.96	0.86	0.62	0.88	0.65	0.34
128-128	159.78	0.66	42.04	12.65	4.95	0.98	0.91	0.73	0.96	0.78	0.51
256-64	466.35	1.51	129.75	44.69	21.89	0.96	0.83	0.58	0.90	0.69	0.39
256-128	718.35	0.62	188.96	57.23	22.99	0.98	0.92	0.74	0.96	0.79	0.50
256-256	1404.48	0.30	348.46	98.30	33.85	0.99	0.96	0.85	0.99	0.88	0.65
512-512	15898.89	0.22	*	1025.26	291.40	*	0.97	0.88	*	0.99	0.86

Table 25: aggregated computational results for the Cray T3D

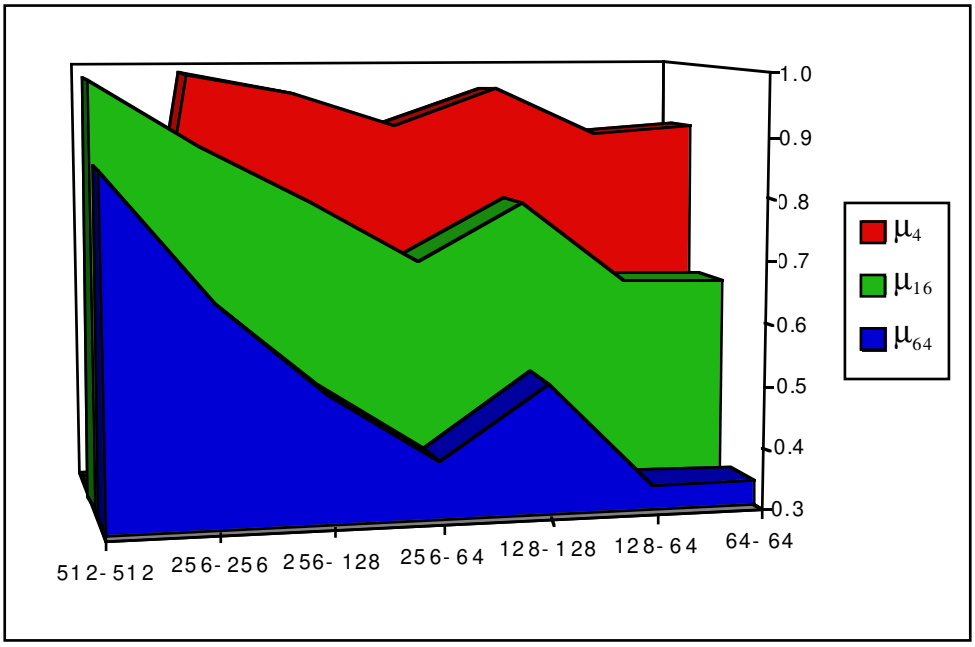

Chart 20: efficiency as a function of p and the problem group from Table 25

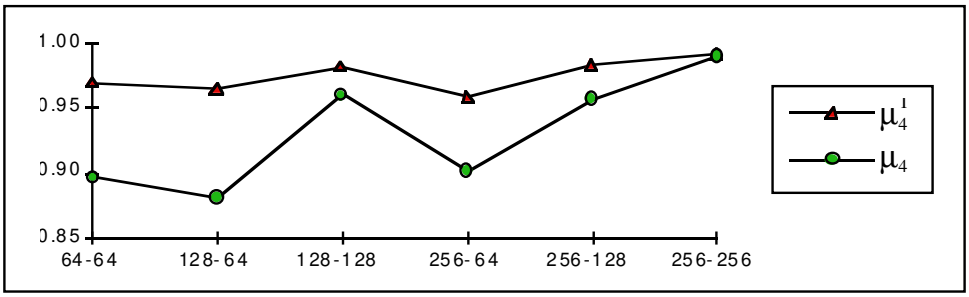

Chart 21: ideal and actual efficiency for $p = 4$

These computational results already show that satisfactory efficiencies (always over 63%) are obtained when using at most 16 PEs, that is the typical number of processors for most of the experiences reported in the literature; on the contrary, as p further increases μ_p soon degrades down to

34% on some classes of instances.

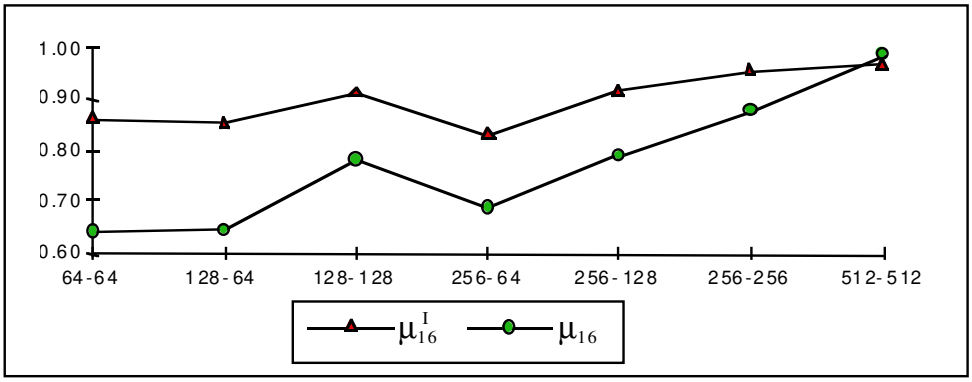

Chart 22: ideal and actual efficiency for $p = 16$

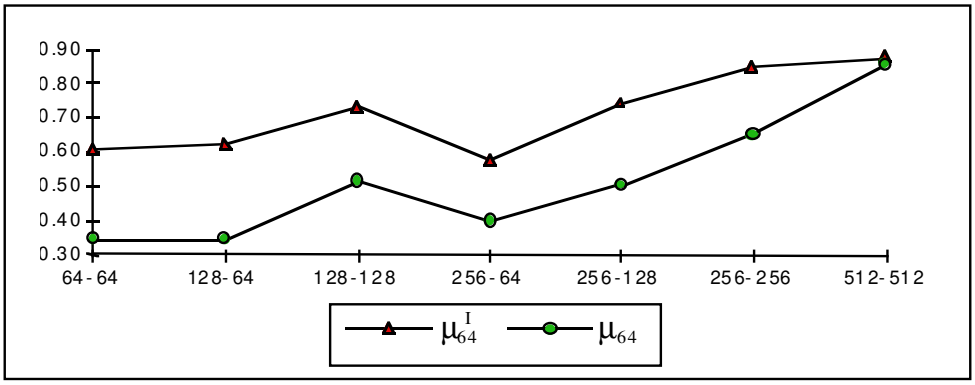

Chart 23: ideal and actual efficiency for $p = 64$

However, Chart 20 clearly shows that μ_p also depends on the size of the problem and on the number of commodities, in a way that can be summarised as follows:

- for a fixed graph size, μ_p increases as k does (Cf. the 256-64, 256-128 and 256-256 problems): in fact, when k increases the relative weight of the sequential part (measured by s) decreases;
- for a fixed m / k, μ_p increases when m does (Cf. 64-64, 128-128 and 256-256 problems): this means that, although both the communication and the (QP) solution costs increase with m, this increase is largely compensated by a corresponding decrease of s;
- for a fixed k, μ_p seems to be "stable" (Cf. 64-64, 128-64 and 256-64 problems), meaning that the (QP) solution cost does not increase faster - and apparently a little slower - than the subproblem solution cost as m increases.

This is relevant because, as shown in §II.2.4, the *sequential* MMCFB is more and more competitive (w.r.t. a number of alternative approaches) as the size of the problem increases, and for "small" values of m / k: hence, PMMCFB seems to be particularly well-suited for large instances with "many" commodities, since for such problems the code is effective in sequential and the obtained efficiencies are high even with a large number of processors. Furthermore, the gap between actual and ideal efficiency seems to follow an analogous law: although large for "small" problems, and increasing with the number of processors p, it is smaller for instances with relatively many commodities, and it tends to reduce (up to, eventually, closing) as the size of the problem increases.

For a more detailed analysis, Table 26 reports the results for each single problem group: note that it contains violations of the Amdahl's Law (128-128-C with 4 PEs) and superlinear speedups (256-256-C and D with 4 PEs, 512-512-A with 16 PEs), presumably due to the above mentioned effects of memory hierarchies. As in some cases the *observed* efficiency is exactly the same as *ideal* efficiency, communication time is *not* a significant factor, at least for these problems classes: this is confirmed by the estimates that can be done on the declared performances of the interprocessor network, and especially by the fact that the gap between real and ideal behaviour of the algorithm diminishes as m (and therefore the amount of exchanged data per iteration) increases, clearly indicating that the loss of efficiency is due to Master-Slaves synchronisation rather than to communication overhead.

group	T_1	s	T_4	T_{16}	T_{64}	μ_4^I	μ_{16}^I	μ_{64}^I	μ_4	μ_{16}	μ_{64}
64-64-A	3.27	0.73	0.90	0.31	0.15	0.95	0.79	0.48	0.89	0.64	0.34
64-64-B	8.56	0.85	2.38	0.82	0.37	0.97	0.85	0.58	0.90	0.65	0.33
64-64-C	14.52	1.06	4.04	1.42	0.70	0.98	0.89	0.65	0.90	0.63	0.34
64-64-D	58.89	1.78	16.59	5.78	2.77	0.98	0.91	0.71	0.89	0.66	0.35
128-64-A	6.97	0.55	1.97	0.65	0.30	0.98	0.92	0.74	0.89	0.67	0.36
128-64-B	26.47	0.71	7.56	2.52	1.11	0.98	0.90	0.69	0.88	0.66	0.37
128-64-C	82.37	0.77	23.03	7.73	3.67	0.98	0.90	0.68	0.89	0.67	0.36
128-64-D	378.85	2.98	110.25	41.73	22.98	0.92	0.70	0.38	0.87	0.60	0.29
128-128-A	23.91	0.21	6.06	1.94	0.68	0.99	0.97	0.88	0.99	0.77	0.54
128-128-B	30.90	0.41	8.60	2.48	0.96	0.99	0.94	0.80	0.92	0.78	0.50
128-128-C	177.67	0.86	44.80	14.01	5.63	0.97	0.89	0.65	0.99	0.79	0.50
128-128-D	406.63	1.16	108.70	32.18	12.54	0.97	0.85	0.58	0.94	0.79	0.51
256-64-A	46.28	0.80	13.32	4.23	1.77	0.98	0.89	0.67	0.88	0.70	0.41
256-64-B	100.52	0.63	28.89	9.09	3.56	0.98	0.91	0.72	0.90	0.70	0.43
256-64-C	598.23	1.60	160.19	54.88	25.92	0.95	0.82	0.55	0.94	0.71	0.40
256-64-D	1120.37	3.02	316.60	110.56	56.33	0.92	0.70	0.37	0.89	0.65	0.33
256-128-A	155.91	0.21	41.85	12.44	4.92	0.99	0.97	0.88	0.95	0.79	0.48
256-128-B	282.99	0.38	76.40	21.74	7.87	0.99	0.95	0.81	0.94	0.81	0.54
256-128-C	702.94	0.70	183.43	57.79	21.63	0.98	0.91	0.70	0.96	0.76	0.51
256-128-D	1731.54	1.18	454.18	136.93	57.55	0.97	0.85	0.58	0.97	0.80	0.48
256-256-A	431.58	0.12	115.39	31.60	10.16	1.00	0.98	0.93	0.94	0.85	0.65
256-256-B	744.50	0.22	204.28	54.91	17.16	0.99	0.97	0.88	0.92	0.85	0.68
256-256-C	1381.11	0.41	322.95	95.71	34.11	0.99	0.94	0.80	1.07	0.90	0.64
256-256-D	3060.72	0.46	751.23	210.98	73.98	0.99	0.93	0.77	1.03	0.91	0.64
512-512-A	7662.02	0.16	*	467.91	138.20	*	0.98	0.91	*	1.02	0.87
512-512-B	24135.76	0.28	*	1582.61	444.61	*	0.96	0.85	*	0.95	0.85

Table 26: disaggregated computational results for the Cray T3D

Remarkably, the largest problems in this suite, the 512-512 with over $8 \cdot 10^5$ variables, always attained very high efficiencies, as large as 85% even with 64 PEs, and a very little (or no) gap between ideal and actual efficiency: for the B group, the parallel code reduces the "wall-clock time" of over a factor of 50, from more than 6 hours to about 7 minutes.

Further insights can be obtained by examining the results for each of the 12 instances of any particular class of problems: in Table 27, this is done for the 64-64 (where the code performs significantly worse than in all the others), where the instances from 1 to 12 are ordered by increasing *loss of efficiency* $\Delta\mu = \mu_{64}^I - \mu_{64}$, and *sc* is the percentage of arcs having single-commodity constraints. For the analysis, the problems can be divided into two groups, the first 6 having $\Delta\mu$ 0.27 and the others having $\Delta\mu$ 0.3; belonging to C and D classes, the first ones are larger and hence more "difficult",

the exception being the swap 4(B)-9(C) that is still meaningful since (as remarked in §II.3.3) 4(B) is the most difficult among the small problems whereas 9(C) is the easiest among the big ones. This relation between $\Delta\mu$ and the "difficulty" of the problem is also confirmed by the sequential times, and by the fact that, on average, D problems perform better than C ones and B problems perform better than A ones.

	$\Delta\mu$	*sc*	T_1	*s*	T_4	T_{16}	T_{64}	μ_4^I	μ_{16}^I	μ_{64}^I	μ_4	μ_{16}	μ_{64}
11 (D)	0.10	60%	80.2	2.24	22.96	8.13	3.92	0.94	0.75	0.42	0.87	0.62	0.32
10 (D)	0.14	30%	60.8	1.79	16.90	5.77	2.85	0.95	0.79	0.47	0.90	0.66	0.33
12 (D)	0.19	90%	35.7	1.31	9.91	3.43	1.56	0.96	0.84	0.55	0.90	0.65	0.36
8 (C)	0.23	60%	19.6	1.35	5.38	1.92	0.97	0.96	0.83	0.54	0.91	0.64	0.31
7 (C)	0.25	30%	16.7	1.22	4.61	1.65	0.82	0.96	0.85	0.57	0.90	0.63	0.32
4 (B)	0.27	30%	18.7	0.90	5.17	1.73	0.79	0.97	0.88	0.64	0.90	0.67	0.37
1 (A)	0.30	30%	5.57	0.87	1.54	0.53	0.25	0.97	0.88	0.65	0.91	0.66	0.34
5 (B)	0.31	60%	4.89	0.83	1.34	0.52	0.22	0.98	0.89	0.66	0.91	0.59	0.34
6 (B)	0.32	90%	2.16	0.84	0.61	0.21	0.10	0.98	0.89	0.65	0.88	0.63	0.34
3 (A)	0.32	90%	2.26	0.72	0.63	0.22	0.10	0.98	0.90	0.69	0.90	0.65	0.37
9 (C)	0.35	90%	7.31	0.60	2.11	0.68	0.30	0.98	0.92	0.73	0.87	0.67	0.38
2 (A)	0.41	60%	1.96	0.59	0.55	0.19	0.10	0.98	0.92	0.73	0.89	0.65	0.32

Table 27: even more disaggregated computational results for the 64-64 problems

Using 64 PEs, each PE is assigned exactly one (MCF), and we can easily conclude that, for the small or easy problems, time spent on busy waiting for synchronisation is relatively large, resulting in poor performances: all the above results could perhaps be summarised by saying that the larger and more difficult an instance is, the better its relative efficiency is likely to be, possibly indicating that the PMMCFB is well-suited for large-scale, difficult (MMCF)s with many commodities - i.e. the class of problems where a parallel approach appears to be really instrumental.

II.3.5.3 Results for the asymmetric version (Cray T3D + Cray C90)

As remarked at the begin of this paragraph, it is rather difficult even to define the speedup of the asymmetric PMMCFB: at least two speed-up definitions are possible, as the "serial execution time" could be that of MMCFB on the C90 *or* on a single PE of the T3D - and both these choices disregard the fact that the two architectures behave differently on different parts of the code. Some technical difficulties further complicated the matter: one relevant problem was that, in the available environment, the C90 was *not* run in a dedicated mode, so that the Master was repeatedly descheduled during the execution of the algorithm, increasing the idle time of the (dedicated) T3D; even worse, the C90-T3D communications using PVM pass through a separate C90 process (the PVM *daemon*), so that receiving one message possibly involved waiting for *two* processes being scheduled for execution. Although the latter problem could be solved by abolishing PVM and directly using the sockets, eliminating the extra idle time appeared to be hopeless: in fact, the wall-clock time of the asymmetric PMMCFB actually *increased* of up to an order of magnitude, and was fluctuating with the workload of the C90 (that was, however, always very high). The only way to get rid of the problem is to approximate the total execution time by the sum of the CPU time of the C90 and the wall-clock time of the T3D, excluding from both the time spent in the C90-T3D communications: such an estimate is very close to the time that would be obtained with a dedicated C90, since it only

disregards the (very low) cost of the actual data transfer. Actually, this is true only if the C90 does *not* solve (MCF)s itself (so that it is immediately available when the T3D finishes), i.e. when the two machines are *never* working at the same time: this may definitely be a bad choice for a small p, but with 16 or 64 processors it is unlikely to diminish the maximum number of (MCF)s that have to be solved by a PE of the T3D, and hence to improve the performances. Note that, when the C90 solves ($\Pi_{\beta t}$) and a single PE of the T3D solves all the (MCF)s, such an estimate could be taken as another reasonable "serial execution time", giving a third way for defining relative speedup and efficiency: since the T3D is almost always faster than the C90 in solving the (MCF)s, this can be interpreted as the time of the code run on an "abstract" sequential machine combining the advantages of both, since all the communication times have been disregarded. Finally, if *efficiencies* had to be compared, the question of how one CPU of C90 has to contribute to the *number* of processors (p) in the definition should be answered: although this further degree of freedom could be avoided by using of *speedups* only, all these problems call for the simpler solution of avoiding any efficiency measure but the plain execution times.

Hence, the results reported in Table 28 are the times of the symmetric and asymmetric versions of PMMCFB: in both cases, T_p has to be intended as the time obtained with p PEs of the T3D, and the relative gap Δt_p between the corresponding values is used as the only "goodness" indicator (remind, however, that due to the above mentioned synchronisation problems such gap is only *potential*).

		Symmetric		Asymmetric				
	C90	T_{16}	T_{64}	T_1	T_{16}	T_{64}	Δt_{16}	Δt_{64}
128-128-A	21.41	1.72	0.66	20.81	1.72	0.66	0%	0%
128-128-B	30.46	2.42	0.94	30.22	2.38	0.96	1%	-1%
128-128-C	174.99	14.35	5.91	171.80	13.90	5.82	3%	1%
128-128-D	405.03	30.97	12.32	384.71	29.45	10.77	5%	13%
256-128-A	177.12	13.76	5.35	175.87	13.92	5.28	-1%	1%
256-128-B	279.83	20.94	7.17	277.20	20.33	7.29	3%	-2%
256-128-C	522.13	42.77	16.98	530.81	42.32	15.28	1%	10%
256-128-D	1610.55	129.47	54.94	1639.63	117.80	44.32	9%	19%

Table 28: time comparison for the C90, the symmetric and asymmetric version

Although clearly still preliminary, these results show that the asymmetric version can be significantly faster, saving up to 20% of total time, especially on the larger and more difficult groups; note that, due to the limited CPU time available, the asymmetric version was tested only on the medium-size 128-128 and 256-128 classes, although larger benefits might have been obtained on larger instances.

II.3.5.4 Conclusions

The aim of this work was to test whether exploitation of the natural parallelism inherent in a Cost Decomposition algorithm for (MMCF), i.e. the parallel solution of the k (MCF)s at each step, would lead to an efficient parallel code even by starting with a complex-coordinator code not originally designed for a parallel implementation: the main advantage of such an approach, if carried along in conjunction with suitable software development techniques, is to produce a highly-portable parallel code starting from a sequential one whose effectiveness has already been proved. The computational experience obtained with the Bundle-based Cost Decomposition algorithm MMCFB seems to show

that the resulting parallel code can be highly efficient even with a large number of processors: perhaps, the most interesting finding is that the classes of problems on which it attains its better results are exactly those where the sequential code is known to outperform several other approaches, i.e. "difficult" problems with relatively "many" commodities w.r.t. the size of the graph. The results of the "symmetric" version are already quite satisfactory, with e.g. the running time of the very large instances cut down from 6 hours to 7 minutes, but it has also been possible to further improve on them by exploiting (apparently for the first time) a common characteristic of nowadays supercomputing facilities, i.e. the side-to-side availability of vector and massively parallel supercomputers: a proper combination of algorithmic design, software development techniques and parallel architectures seem to have a great potential for the solution of very large-scale Multicommodity Flow Problems.

Acknowledgements

For the material presented in this Chapter, I gratefully acknowledge the invaluable contribution of Paola Cappanera, who wrote, debugged and ran most of the parallel code as a part of her Master thesis [Ca96]: also, in the early phase of the project we used a temporary account of the Fourth Summer School on Parallel and Vector Supercomputing, attended on a volounteer basis by Paola in the Summer of 1995. The largest part of this work has been supported by grants provided by CINECA (Casalecchio di Reno, Bologna, Italy): I also wish to acknowledge Giovanni Erbacci and Francesca Durì for their help and suggestions during Paola's stay at CINECA and after.

II.4 Other Dual-Ascent Approaches

The Bundle approach of the two previous Chapters is only one possible Dual-Ascent method based on the theory developed in Part I: barring the (LP)-based versions, that will be (hopefully) tested in a near future, several other possibilities can be considered. In this Chapter, preliminary theoretical and computational results about three alternative approaches are described: although none of them have yet reached the status of a mature technology, they offer interesting directions for future research. In particular, §II.4.1 and §II.4.2 are devoted to the development of ε-Steepest Ascent approaches to the Cost Decomposition and Potential Decomposition of (MMCF) respectively, and in the first one preliminary computational results are presented: then, in §II.4.3 some ideas are presented about a more "combinatorial" approach to Potential Decomposition, that might be extended to other forms of network-structured problems such as the Min-Cost Flow problem on Hypergraphs.

II.4.1 An ε-Steepest Ascent Cost Decomposition approach

One of the basic hypotheses in the development of Bundle methods is that the $\varphi()$ is available only as a "black box", i.e. that the whole subdifferential $\partial\varphi()$ is not known: this is not actually true in our case, since Theorem I.1.2.1 implies that

$$\partial\varphi() = \left\{ \Sigma_{h \in \mathcal{K}} \boldsymbol{x}^h - \boldsymbol{u} : \boldsymbol{x}^h \in X^h(\boldsymbol{y}) \ \forall\ h \in \mathcal{K} \right\}$$

where $X^h(\boldsymbol{y})$ is the set of all optimal solutions of the h-th (MCF) subproblem

$$(\mathrm{RMF}_{\boldsymbol{y}}^h) \qquad \min \left\{ (\boldsymbol{c}^h + \boldsymbol{y})\boldsymbol{x}^h : \boldsymbol{E}\boldsymbol{x}^h = \boldsymbol{b}^h,\ \mathbf{0} \le \boldsymbol{x}^h \le \boldsymbol{u}^h \right\}$$

and it has a nice description in terms of the original problem. In fact, let $\pi^h = \pi^h(\boldsymbol{y})$ be *any* vector of *optimal potentials* for $(\mathrm{RMF}_{\boldsymbol{y}}^h)$ and

$$\bar{c}_{ij}^h = \bar{c}_{ij}^h(\pi^h) = c_{ij}^h - \pi_i^h + \pi_j^h$$

be the corresponding *reduced costs*: some general duality results (Cf. Sec. II.1.4.2) show that $\bar{\boldsymbol{x}}^h \in X^h(\boldsymbol{y})$ if and only if it is feasible, i.e. $\boldsymbol{E}\bar{\boldsymbol{x}}^h = \boldsymbol{b}^h$ and $\mathbf{0} \le \bar{\boldsymbol{x}}^h \le \boldsymbol{u}^h$, and it satisfies the *Complementary Slackness Conditions* (*CSC*)

$$\bar{c}_{ij}^h < 0 \quad ((i,j) \text{ is } \textit{active}) \quad \Rightarrow \quad x_{ij}^h = u_{ij}^h$$

$$\bar{c}_{ij}^h > 0 \quad ((i,j) \text{ is } \textit{inactive}) \quad \Rightarrow \quad x_{ij}^h = 0.$$

Hence, $X^h(\boldsymbol{y})$ is the set of all the flows in which the active and inactive arcs are *blocked* at their

upper and lower bounds respectively, while the flow on *balanced* arcs (s.t. $\bar{c}_{ij}^h = 0$) is *free* to have any value in $[0, u_{ij}^h]$, provided that the flow conservation equations are respected: then, the *steepest ascent direction* finding problem

$$(\mathrm{SAD}_{\boldsymbol{y}}) \qquad \operatorname{argmin}_{\boldsymbol{g}} \left\{ \left\| \boldsymbol{g} \right\|_2^2 : \boldsymbol{g} \in \partial\varphi(\boldsymbol{y}) \right\} =$$
$$\min_{\boldsymbol{x}} \left\{ \left\| \sum_{h \in \mathcal{K}} \boldsymbol{x}^h - \boldsymbol{u} \right\|_2^2 : \boldsymbol{x}^h \in X^h(\boldsymbol{y}) \; \forall \, h \in \mathcal{K} \right\}$$

is a Quadratic single-commodity Min-Cost Flow with separable feasible region but nonseparable objective function, that can be used in a *steepest ascent* approach for maximizing $\varphi()$. Actually, $(\mathrm{SAD}_{\boldsymbol{y}})$ is not enough, since the *constrained maximization* of $\varphi()$ is required: however, with the usual device of introducing the (concave) *characteristic function* of the nonnegative orthant

$$\varphi^+(\boldsymbol{y}) = \begin{cases} 0 & \text{if } \boldsymbol{y} \geq \boldsymbol{0} \\ -\infty & \text{otherwise} \end{cases}$$

the Lagrangean Dual of (MMCF) can be restated as the unconstrained maximization problem

$$(\mathrm{DMMCF}) \qquad \max\{ \varphi(\boldsymbol{y}) + \varphi^+(\boldsymbol{y}) : \boldsymbol{y} \in \mathcal{R}^m \}$$

and the issue becomes that of the characterisation of $\partial[\varphi + \varphi^+](\boldsymbol{y})$; this is the approach that leads to the general definition of *conditional subdifferential* [LPS96], but a direct derivation is preferable here. For any fixed $\boldsymbol{y} \geq \boldsymbol{0}$, $\partial\varphi^+(\boldsymbol{y})$ is the Cartesian product $V(y_1) \times V(y_2) \times \ldots \times V(y_m)$, where

$$V(y) = \begin{cases} \{ 0 \} & \text{if } y > 0 \\ \mathcal{R}_+ & \text{otherwise} \end{cases}$$

so that, since $\partial[\varphi + \varphi^+](\boldsymbol{y}) = \partial\varphi(\boldsymbol{y}) + \partial\varphi^+(\boldsymbol{y})$, the *constrained* steepest ascent direction finding problem is

$$(\mathrm{SAD}_{\boldsymbol{y}}') \qquad \min_{\boldsymbol{x},\boldsymbol{v}} \left\{ \left\| \sum_{h \in \mathcal{K}} \boldsymbol{x}^h - \boldsymbol{u} + \boldsymbol{v} \right\|_2^2 : \boldsymbol{v} \in \partial\varphi^+(\boldsymbol{y}) \, , \boldsymbol{x}^h \in X^h(\boldsymbol{y}) \; \forall \, h \in \mathcal{K} \right\}.$$

This can be put in a more natural form by defining the projection operator "modulo y"

$$[\, w \,]_+^y = \begin{cases} 0 & \text{if } w < 0 \text{ and } y = 0 \\ w & \text{otherwise} \end{cases}$$

and its obvious componentwise extension to vectors of $\mathcal{R}^m$: then,

$$(\mathrm{SAD}_{\boldsymbol{y}}') \qquad \min_{\boldsymbol{x}} \left\{ \left\| [\, \sum_{h \in \mathcal{K}} \boldsymbol{x}^h - \boldsymbol{u} \,]_+^{\boldsymbol{y}} \right\|_2^2 : \boldsymbol{x}^h \in X^h(\boldsymbol{y}) \; \forall \, h \in \mathcal{K} \right\}$$

since any negative component of $\sum_{h \in \mathcal{K}} \boldsymbol{x}^h - \boldsymbol{u}$ corresponding to a zero component of $\boldsymbol{y}$ can be turned to zero by suitably choosing the vector $\boldsymbol{v} \in \partial\varphi^+(\boldsymbol{y})$, while positive components are not modified; hence, $(\mathrm{SAD}_{\boldsymbol{y}}')$ requires finding the least-norm *projected* subgradient w.r.t. the *active* constraints. In this form, the strong similarities with the Augmented Lagrangean problem $(\mathrm{AL}_{\boldsymbol{y},t})$ (Cf. Sec. II.1.4.2) are apparent, as well as the differences:

- the costs of (MMCF) have disappeared, followed by the penalty parameter t, because the sets $X^h(\boldsymbol{y})$ contain all equivalent solutions for $(\mathrm{RMF}_{\boldsymbol{y}}^h)$;
- hence, the costs now induce "hard" constraints on the feasible region;
- the projection operator now takes into account the current point $\boldsymbol{y}$.

Intuitively, (SAD_y') should be "easier" than ($AL_{y,t}$), since it has a very nicely conditioned quadratic objective function and a smaller feasible region: in fact, if the optimal solution of (RMF_y^h) is *unique*, the $\boldsymbol{x}^h$ variables are fixed and disappear from the problem, and this is true for *all* $h \in \mathcal{K}$ (i.e. $\varphi()$ is differentiable) with probability 1 in a random point $\boldsymbol{y} \in \mathcal{R}^m$ (unfortunately, this is very unlikely to be true in the optimal point $\boldsymbol{y}^*$).

A "pure" steepest-ascent approach is prone to the typical problems of first-order methods, i.e. zig-zagging and slow convergence: a possible way to avoid them is to use *steepest* ε-*ascent* directions instead, i.e. work with the ε-*subdifferential* (Cf. §I.1.1) of $[\varphi + \varphi^+](\boldsymbol{y})$. It is well-known (see e.g. [Be91]) that every $\bar{\boldsymbol{x}}^h \in X_\varepsilon^h(\boldsymbol{y})$, i.e feasible ($\boldsymbol{E}\bar{\boldsymbol{x}}^h = \boldsymbol{b}^h$ and $\boldsymbol{0} \le \bar{\boldsymbol{x}}^h \le \boldsymbol{u}^h$) and satisfying the ε-*Complementary Slackness Conditions* (ε-*CSC*)

$$\bar{c}_{ij}^h < \varepsilon \quad ((i,j) \text{ is } \varepsilon\text{-}active) \;\Rightarrow\; x_{ij}^h = u_{ij}^h$$

$$\bar{c}_{ij} > \varepsilon \quad ((i,j) \text{ is } \varepsilon\text{-}inactive) \;\Rightarrow\; x_{ij}^h = 0$$

is ($n\varepsilon$)-*optimal* for (RMF_y^h), so that

$$\partial_{kn\varepsilon}\varphi() = \left\{ \boldsymbol{w} = \Sigma_{h \in \mathcal{K}} \boldsymbol{x}^h - \boldsymbol{u} : \boldsymbol{x}^h \in X_\varepsilon^h(\boldsymbol{y}) \;\forall\, h \in \mathcal{K} \right\};$$

the ($m\delta$)-*subdifferential* of $\varphi^+()$ is the Cartesian product $V_\delta(y_1) \times V_\delta(y_2) \times \ldots \times V_\delta(y_m)$ (see Figure 13) for

$$V_\delta(y) = \left[\, 0 \;,\; \frac{\delta}{y} \,\right] \qquad \text{with the customary assumption that } \delta/0 = +\infty \text{ for any } \delta \ge 0.$$

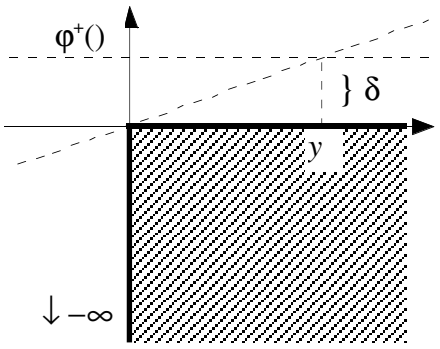

Figure 13: two "extreme" δ-subgradients of $\varphi^+()$ in y

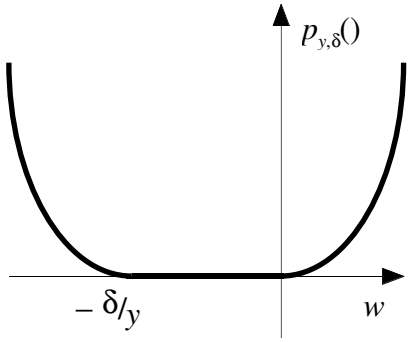

Figure 14: an interpretation of ($\varepsilon\delta$-SAD_y')

Hence, the ($kn\varepsilon + m\delta$)-*steepest* ascent direction finding problem is

($\varepsilon\delta$-SAD_y') $\quad \min_{\boldsymbol{x}} \left\{ \left\| \left[\Sigma_{h \in \mathcal{K}} \boldsymbol{x}^h - \boldsymbol{u} \right]_+^{y,\delta} \right\|_2^2 : \boldsymbol{x}^h \in X_\varepsilon^h(\boldsymbol{y}) \;\forall\, h \in \mathcal{K} \right\}$

having a larger feasible region (the *free* arcs are now the ε-*balanced* ones having $\left| \bar{c}_{ij}^h \right| < \varepsilon$) and a different "projection" operator

$$[w]_+^{y,\delta} = \begin{cases} \min\{ w + \delta/y \,,\, 0 \} & \text{if } w < 0 \\ w & \text{otherwise} \end{cases};$$

the function $p_{y,\delta}(w) = ([w]_+^{y,\delta})^2$ depicted in Figure 14 can be thought as a different form of nonquadratic smooth penalty function, and different ε^h and δ_{ij} could be used for each different commodity and mutual capacity constraint. This leads to the definition of a scaling approach, where the problem is solved for a pair (ε, δ) until the termination criterion $z(\varepsilon\delta\text{-}SAD_y') = 0$ signals that at

least one of the two has to be decreased: a nice feature of such a scheme is that, upon termination of a "scaling phase", an ($kn\varepsilon + m\delta$)-*optimal* solution of (MMCF) is available, since the optimal solution $\bar{\boldsymbol{x}}$ of ($\varepsilon\delta$-SAD$'_{\boldsymbol{y}}$) must clearly satisfy

$$-\delta/y_{ij} \leq \Sigma_{h \in \mathcal{K}} \bar{x}^h_{ij} \leq u_{ij} \qquad \forall\ (i, j) \in \mathcal{A}$$

and hence it is *feasible*.

($\varepsilon\delta$-SAD$'_{\beta,\boldsymbol{y}}$) can be solved with a Simplicial Decomposition [VHo77] approach, where vertices of the sets $X^h_\varepsilon(\ \boldsymbol{y}\ )$ are efficiently generated by exploiting the (MCF) structure as in the Bundle and Linear-Quadratic Penalty function methods [PZ92] [PZ94]: the Master problem of the Simplicial Decomposition is

($\varepsilon\delta$-SAD$'_{\beta,\boldsymbol{y}}$) $\quad \min_{\theta,\mu} \left\{ \tfrac{1}{2} \left\| \boldsymbol{G}_\beta \theta + \mu \right\|^2_2 : \theta \in \Theta\ ,\ \delta \boldsymbol{Y}^{-1} \boldsymbol{e} \geq \mu \geq \boldsymbol{0} \right\}$

where $\boldsymbol{Y}$ is the diagonal matrix having $\boldsymbol{y}$ as the diagonal elements, and $\boldsymbol{G}_\beta$ is the matrix of the subgradients $\boldsymbol{g}_i = \Sigma_{h \in \mathcal{K}} \boldsymbol{x}^h_i - \boldsymbol{u}$ corresponding to the vertices $\boldsymbol{x}^h_i \in X^h_\varepsilon(\ \boldsymbol{y}\ )$ found so far, and kept in the "local Bundle" β. For the special case $\delta = 0$, the upper bound on μ_i is zero if $y_i > 0$ and $+\infty$ otherwise, so that the problem only has the $\mu_\chi \geq \boldsymbol{0}$ variables, where $\chi = \chi(\ \boldsymbol{y}\ ) = \{\ i : y_i = 0\ \}$: hence, it can be (efficiently) solved with the BTT algorithm developed in Chapter I.3. In fact, since (Cf. §I.3.5) its Quadratic Dual is

($\varepsilon\delta$-DSAD$'_{\beta,\boldsymbol{y}}$) $\quad \min_{\boldsymbol{d},v} \left\{ v + \tfrac{1}{2} \left\| \boldsymbol{d} \right\|^2_2 : v\boldsymbol{e} \geq \boldsymbol{d}\boldsymbol{G}_\beta\ ,\ \boldsymbol{0} \leq \boldsymbol{d}_\chi \right\}$

a specialized version of BTT has been developed for dealing with just nonnegativity (instead of general "box") constraints, where some of the operations are performed more efficiently since the RHS is known to be zero. At each step, the optimal solution ($\bar{\theta}$, $\bar{\mu}_\chi$) of ($\varepsilon\delta$-SAD$'_{\beta,\boldsymbol{y}}$) determines a tentative optimal point $\bar{\boldsymbol{x}}^h = \Sigma_{i \in \beta} \boldsymbol{x}^h_i$, and the dual optimal solution $\bar{\boldsymbol{d}} = \boldsymbol{G}_\beta \bar{\theta} + \bar{\mu}_\chi$ is the *gradient* $\Sigma_{h \in \mathcal{K}} \bar{\boldsymbol{x}}^h - \boldsymbol{u}$ of the objective function at $\bar{\boldsymbol{x}}$ projected onto the $\boldsymbol{0} \leq \boldsymbol{d}_\chi$ constraints: hence, the *linearized subproblem* that needs to be solved is

(ε-LP$'_{\bar{\boldsymbol{d}},\boldsymbol{y}}$) $\quad \min_{\boldsymbol{x}} \left\{ \bar{\boldsymbol{d}} (\Sigma_{h \in \mathcal{K}} \boldsymbol{x}^h - \boldsymbol{u}\ ) : \boldsymbol{x}^h \in X^h_\varepsilon(\ \boldsymbol{y}\ )\ \forall\ h \in \mathcal{K} \right\}$

which can be restated as

(ε-LP$'_{\bar{\boldsymbol{d}},\boldsymbol{y}}$) $\quad \min_{\boldsymbol{w}} \left\{ \bar{\boldsymbol{d}}\boldsymbol{w} : \boldsymbol{w} \in \partial_{kn\varepsilon} \varphi(\ \boldsymbol{y}\ ) \right\}$

i.e. the problem of determining the *kn*ε-*directional derivative* of $\varphi()$ in $\boldsymbol{y}$ along $\bar{\boldsymbol{d}}$ (Cf. Chapter I.1); in other words, z(ε-LP$'_{\bar{\boldsymbol{d}},\boldsymbol{y}}$) is the *exact slope* of $\varphi()$ along the (feasible) direction $\bar{\boldsymbol{d}}$, clearly giving an excellent stopping criterion for the early termination of the Simplicial Decomposition.

From the practical viewpoint, the typical implementation of a Relaxation Min-Cost Flow solver can be easily modified in order to "block" the flow on the arcs that are active/inactive in the optimal solution: for instance, in the `RelaxIV` class three vectors `X[]`, `Cap[]` and `b[]` contain the current flows x_{ij}, residual capacities $u_{ij} - x_{ij}$ and node imbalances $\Sigma_{j:\,(i,j) \in \mathcal{A}} x_{ij} - \Sigma_{j:\,(j,i) \in \mathcal{A}} x_{ji} - b_i$ respectively, so that an arc (i, j) will have `X[` *ij* `]` $= u_{ij}$ and `Cap[` *ij* `]` $= 0$ if active, or `X[` *ij* `]` $= 0$ and `Cap[` *ij* `]` $= u_{ij}$ if inactive. Hence, to block an active/inactive arc it is sufficient to set `X[` *ij* `]` $= 0$ or `Cap[` *ij* `]` $= 0$ respectively *without* changing the node balance vector `b[]`: since the code reoptimizes from the current solution, it will keep the arcs to the current value until they are "unblocked" by resetting the

correct values of the `X[]` and `Cap[]` vectors.

Despite these promising features, the computational results obtained so far are rather disappointing: in Table 29, the "pure" Steepest Ascent approach (*SA*), its ε-variant (ε-*SA*) and MMCFB are compared for the solution of a subset of the Canad problems (Cf. Sec. II.2.3.1 and II.2.4.1), aggregated in groups of similar sizes.

				SA	ε- SA					MMCFB	
k	*n*	*m*	*size*	Time	Time	φ# SD	Time SD	φ# LS	Time LS	Time	φ#
10	50	469	4685	13.28	13.58	110	8.85	70	1.58	1.08	22
10	50	625	6250	3600*	1580.00	10078	1160.40	976	47.30	36.00	37
40	20	252	10064	20.64	15.20	84	7.66	46	1.64	0.92	17
10	100	1600	16000	3600*	239.00	700	169.00	293	20.50	18.00	69
10	100	2500	25000	24.00	25.50	26	15.50	25	3.90	2.55	15
200	20	267	53360	18.68	18.20	31	6.38	12	2.14	1.32	7
100	35	545	54450	39.15	40.00	46	11.68	22	4.21	2.10	10
100	50	625	62500	412.00	277.00	316	190.30	65	2.00	18.00	33
100	100	1600	160000	136.50	88.00	10	74.50	9	3.75	5.25	6

Table 29: comparative results of the [ε-]Steepest Ascent and Bundle approach

The "pure" Steepest Ascent version has serious convergence problems on the more difficult instances (on the entries marked with "*" it was stopped after having run for one hour without producing the result), that are solved by the introduction of the ε-scaling mechanism: nevertheless, the ε-SA algorithm is always one order of magnitude slower than the Bundle approach, being up to a factor of 50 worse on the difficult instances. The results show that the solution of (ε-SAD$'_y$) is very costly, in terms of the number of (ε-LP$'_{\bar{d},y}$) subproblems solved, if compared with both the Line Search phase and the *overall* MMCFB: moreover, the surprising negative result is that the solution times of (ε-LP$'_{\bar{d},y}$) are from two to three times *larger* than those of the Lagrangean subproblems (RMF^h_y), despite that on average the 95% of the arcs is blocked. Hence, although substantially negative, these results suggest a number of possible implementative and algorithmic improvements:

- a different implementation should be used for solving (ε-LP$'_{\bar{d},y}$), that be capable of exploiting the large number of blocked arcs: for instance, the optimal solution of (RMF^h_y) is primal but not dual feasible for the *h*-th component of (ε-LP$'_{\bar{d},y}$), so that a primal (MCF) solver might be more efficient than the (dual) Relaxation method, at least for the first call;
- on the difficult instances, the direction is not of a good quality, resulting in a too large number of iterations: this may be in part due to the "myopic" view of the constraints, i.e. the $\delta = 0$ choice, so that one sensible (although not theoretically "clean") alternative could be to solve the slightly different (Dual) Master problem

  $$\bar{\boldsymbol{d}} = \text{argmin}_{d,v} \left\{ v + 1/2 \left\| \boldsymbol{d} \right\|_2^2 : v\boldsymbol{e} \geq \boldsymbol{d}\boldsymbol{G}_\beta \, , -\boldsymbol{y} \leq \boldsymbol{d} \right\}$$

  that would ensure $\boldsymbol{y} + \bar{\boldsymbol{d}}$ to be feasible, in the spirit of the Bundle approach;
- analogously, it may be possible to replace the Line Search with a "one step" approach where the steplength be derived by the available information on $\boldsymbol{d}$: this is benefical in the Bundle case;
- the ε-scaling mechanism can be improved, for instance by gradually decreasing ε without waiting for an ε-optimal solution to be obtained: this is analogous to the gradual increase of the penalty

parameter t in the Augmented Lagrangean approach, that is known to be crucial for the performances;

- finally, it is possible to replace the Simplicial Decomposition method with an entirely different, more efficient algorithm: for instance, the Block-Jacobi method used in [ZaA95] within an Augmented Lagrangean approach could be used in this setting as well.

To conclude, ε-Steepest Ascent approaches to the maximization of the Lagrangean function could be an interesting alternative, due to their capability of greatly reducing the size of the Quadratic Problem to be solved (e.g. with respect to Augmented Lagrangean approaches) and to produce an approximate optimal (primal) solution at the end of each ε-scaling phase: however, further research is needed for obtaining good computational results in practice.

II.4.2 An ε-Steepest Ascent Potential Decomposition approach

Although the most natural, the Multicommodity structure is not the only source of decomposition in (MMCF): a seldom noticed alternative is the *Potential Decomposition*, i.e. the Lagrangean relaxation of the flow conservation constraints with the vector of potentials $\Pi = [\, \pi^1 \,..\, \pi^k \,]$

$$(\mathrm{PDMF}_\Pi) \quad \left\{ \begin{array}{l} \min \sum_{h \in \mathcal{K}} (\boldsymbol{c}^h - \pi^h \boldsymbol{E}) \boldsymbol{x}^h \\ \boldsymbol{0} \le \boldsymbol{x}^h \le \boldsymbol{u}^h \quad \forall\ h \in \mathcal{K} \\ \sum_{h \in \mathcal{K}} \boldsymbol{x}^h \le \boldsymbol{u} \end{array} \right\} + \sum_{h \in \mathcal{K}} \pi^h \boldsymbol{b}^h,$$

that decomposes into m independent *continuous Knapsack* problems

$$(\mathrm{K}_\Pi^{ij}) \qquad \psi_{ij}(\Pi) = \min\Big\{ \sum_{h \in \mathcal{K}} (c_{ij}^h - \pi_i^h + \pi_j^h) x_{ij}^h : 0 \le x_{ij}^h \le u_{ij}^h \ \forall\ h \in \mathcal{K}, \sum_{h \in \mathcal{K}} x_{ij}^h \le u_{ij} \Big\},$$

and the corresponding Lagrangean Dual

$$(\mathrm{PDMCF}) \quad \max_\Pi \Big\{ \psi(\Pi) = \sum_{(i,j) \in \mathcal{A}} \psi_{ij}(\Pi) + \sum_{h \in \mathcal{K}} \pi^h \boldsymbol{b}^h : \Pi \in \mathcal{R}^{n \times k} \Big\}.$$

This approach differs from the Relaxation and Row Action methods (Cf. Sec. II.1.4.5) in that (PDMF_Π) decomposes in only m independent "complex" subproblems, while in the Relaxation method the mutual capacity constraints are relaxed as well, so that there are km "simple" independent subproblems, and in the Row Action method the bounds $[0, u_{ij}^h]$ are also relaxed at the expense of introducing $2km$ more Lagrangean variables.

Requiring the (unconstrained) maximization of a NonDifferentiable function, (PDMCF) can be attacked with the same approaches devised for (DMMCF), such as a Bundle method: this has not been done in the past since $\psi(\Pi)$ can be expected to be "more difficult" to maximize than $\varphi(\boldsymbol{y})$. In fact, if k is large then Π is much longer than $\boldsymbol{y}$ (kn instead of m variables); furthermore, being Lagrangean variables of *equality* constraints all the Πs can be expected to be "active" in the optimum (almost, since $\boldsymbol{E}$ is well-known to be $c(G)$-rank deficitary, where $c(G)$ is the number of distinct maximal connected components of G). However, it is also likely that $\psi()$ can be computed (much) faster than $\varphi()$; furthermore, the "difficulty" of (PDMCF) can be expected to be less dependent from the "tightness" of the mutual capacity constraints, so that it could be competitive on instances that are

"hard" for (DMMCF); finally, there are cases where kn is not "too larger" than m. Actually, the PDS problems (Cf. Sec. II.2.3.3) have all the above characteristics: they have large m and small k, so that $\varphi(\boldsymbol{y})$ requires the solution of (few) large (MCF)s while $\psi(\Pi)$ requires the solution of (many) tiny Knapsacks; they are known (Cf. Sec. II.2.4.3) to be difficult; they are "sparse" ($m / n \approx 3$) and have "active" mutual capacity constraints on about the 30% of the arcs, so that the $kn \approx 3m$ and the "active" Πs can be expected to be "only" about 10 times the "active" $\boldsymbol{y}$s.

Hence, a Bundle approach to (PDMCF) might be sensible, but in this Paragraph the development of a ε-Steepest Ascent (Cf. §II.4) will be sketched instead: with the natural definition of *reduced cost* $\bar{c}^h_{ij} = c^h_{ij} - \pi^h_i + \pi^h_j$, and dropping the superscript (i, j) for ease of notation, the subproblems are

$$(K_\Pi) \qquad \psi(\Pi) = \min\left\{ \sum_{h \in \mathcal{K}} \bar{c}^h x^h : 0 \le x^h \le u^h \ \forall\, h \in \mathcal{K}, \sum_{h \in \mathcal{K}} x^h \le u \right\}$$

(where no confusion should arise with the "global" $\psi()$) and their feasible region will be denoted by Φ_{ij}; it will also be sometimes convenient to consider the multiflow $\boldsymbol{x} = [\, \boldsymbol{x}^1 \,..\, \boldsymbol{x}^k \,]$ as partitioned into m vectors $\boldsymbol{x}_{ij} = [\, \boldsymbol{x}^1_{ij} \,..\, \boldsymbol{x}^k_{ij} \,]$. (K_Π) is a "simple" continuous knapsack problem (all the items have "size"1), and its optimal solutions set can be easily characterised: if $\sigma = \sigma_{ij} = \sigma_{ij}(\Pi) : \mathcal{K} \to \mathcal{K}$ is (one of) the permutation(s) that orders the reduced costs in nonincreasing sense, i.e.

$$\bar{c}^{\sigma(1)} \le \bar{c}^{\sigma(2)} \le \ldots \le \bar{c}^{\sigma(k)},$$

then an optimal solution of (K_Π) can be - roughly speaking - obtained by "picking entirely" the objects $\sigma(1), \sigma(2)$ ... until either an object with $\bar{c} \ge 0$ or the total capacity u is reached, eventually choosing only a fraction of the last object (the one with the higher cost). More formally, a flow vector $\boldsymbol{x}$ is a *pseudoflow* if $\boldsymbol{x}_{ij} \in \Phi_{ij}\ \forall\ (i, j) \in \mathcal{A}$, and for any pseudoflow $\boldsymbol{x}$ let the *aggregated (pseudo)flow* be $\bar{\boldsymbol{x}} = \sum_{h \in \mathcal{K}} \boldsymbol{x}^h$ $(\le \boldsymbol{u})$: given a pair $(\Pi, \boldsymbol{x})$, the vector of *critical reduced costs* η is

$$\eta_{ij} = \begin{cases} 0 & \text{if } \bar{x}_{ij} < u_{ij} \\ \max_{h \in \mathcal{K}}\{\, \bar{c}^h_{ij} : x^h_{ij} > 0 \,\} & \text{if } \bar{x}_{ij} = u_{ij} \end{cases}$$

and the arc (i, j) is *saturated* if $\bar{x}_{ij} = u_{ij}$, and *nonsaturated* otherwise; note that any nonsaturated arc has $\eta_{ij} = 0$, but the converse is not true, i.e. η_{ij} can be zero even if (i, j) is saturated. Then, for a given pair $(\Pi, \boldsymbol{x})$, the *commodity h* over the arc (i, j) is said

active if $\bar{c}^h_{ij} < \eta_{ij}$

balanced if $\bar{c}^h_{ij} = \eta_{ij}$

inactive if $\bar{c}^h_{ij} > \eta_{ij}$

and let A^{ij}_Π (respectively B^{ij}_Π, I^{ij}_Π) be the set of active (resp. balanced, inactive) commodities on (i, j): $(\Pi, \boldsymbol{x})$ satisfies the Complementary Slackness Conditions if and only if for each (i, j)

$$(CSC_{ij}) \qquad \begin{cases} h \in A^{ij}_\Pi & \Rightarrow\ x^{ij}_\Pi = u^{ij}_\Pi \\ h \in I^{ij}_\Pi & \Rightarrow\ x^{ij}_\Pi = 0 \end{cases}$$

i.e. $\boldsymbol{x}_{ij}$ is optimal for the Lagrangean subproblem (K^{ij}_Π).

Lemma II.4.2.1: *for any fixed* Π, $\boldsymbol{x}_{ij}$ *satisfies* (CSC_{ij}) *if and only if* $\boldsymbol{x}_{ij}$ *is optimal for* (K_Π^{ij}).

Proof: A flow (possibly not a pseudoflow) $\boldsymbol{x}$ is optimal for the Lagrangean relaxation of (K_Π)

($RK_{\Pi,y}$) $\quad \min_{\boldsymbol{x}} \{ \sum_{h \in \mathcal{K}} (\bar{c}^h + y) x^h : 0 \le x^h \le u^h \; \forall \; h \in \mathcal{K} \} - yu$

(for notational simplicity, the index ij is disregarded) if and only if

$$x^h \in \begin{cases} \{ \; 0 \; \} & \text{if } \bar{c}^h > -y \\ \{ \; u^h \; \} & \text{if } \bar{c}^h < -y \\ [\; 0, \; u^h \;] & \text{if } \bar{c}^h = -y \end{cases} ;$$

since $y \ge 0$, all the x^h with positive reduced costs will be zero, while possibly not all the x^h with negative reduced costs will be nonzero. y^* is an optimal solution of the corresponding Lagrangean Dual if and only if there exists such an $\boldsymbol{x}$ (w.r.t. y^*) that is also a pseudoflow; for any permutation σ as defined above, it is clear that $\bar{c}^{\sigma(h)} < -y \Rightarrow \bar{c}^{\sigma(p)} < -y \; \forall \; p \le h$, and therefore the *critical* index $\bar{h}$

$$\bar{h} = \max \{ \; h \in \mathcal{K} : \sum_{p \le h} u^p < u \; , \; \bar{c}^{\sigma(h)} < \bar{c}^{\sigma(h+1)} \; , \; \bar{c}^{\sigma(h)} < 0 \; \}$$

(or, more precisely, $\sigma(\bar{h})$) indicates the last "object" that *must* be entirely into the knapsack in *any* optimal solution of (K_Π). In fact, the object $p = \sigma(\bar{h}) + 1$

- either cannot be put entirely into the knapsack without exceeding the capacity u;
- or has $\bar{c}^p \ge 0$, hence it must not (necessarily, if $\bar{c}^p = 0$) be put into the knapsack;
- or, finally, can be put entirely into the knapsack without exceeding the capacity u, but it has exactly the same *strictly negative* reduced cost of other objects, not all of which can be put entirely into the knapsack without exceeding u: in this case, p has exactly the same "right" of the others to be put into the knapsack, so it *can* be chosen but it *must not be necessarily* chosen.

Then, it is easy to see that $y^* = \max \{ -\bar{c}^p , 0 \} = -\eta_{ij}$, and the proof is over. ♦

Hence, (Π, $\boldsymbol{x}$) satisfies (CSC) and $\boldsymbol{x}$ satisfies the flow conservation constraints if and only if Π and $\boldsymbol{x}$ are optimal solutions of (PDMCF) and (MMCF) respectively: also, the optimal solutions set $X(\Pi)$ of (K_Π) is the set of all pseudoflows $\boldsymbol{x}$ such that

$$\sum_{h \in B_\Pi} x^h \le u - \sum_{h \in A_\Pi}$$
$$x^h = u^h \qquad \forall \; h \in A_\Pi$$
$$x^h = 0 \qquad \forall \; h \in I_\Pi$$

so that the Steepest Ascent direction finding problem

(SAD_Π) $\quad \min_{\boldsymbol{x}} \{ \sum_{h \in \mathcal{K}} \sum_{i \in \mathcal{N}} (\sum_{j:\,(i,j) \in \mathcal{A}} x_{ij}^h - \sum_{j:\,(j,i) \in \mathcal{A}} x_{ji}^h - b_i^h)^2 : \boldsymbol{x}_{ij} \in X_{ij}(\boldsymbol{y}) \; \forall \; (i,j) \in \mathcal{A} \}$

is again a (QP) with separable feasible region and nonseparable objective function. (SAD_Π) has a more "complicated" Hessian than ($SAD_{\boldsymbol{y}}'$) (Cf. §II.4.1), but the feasible region is much "simpler": hence, the subproblems of a block-coordinate descent approach to the former would be simpler than the Quadratic (MCF)s needed for the latter, yelding for instance a finer-grain parallel block-Jacobi method than in [ZaA95]. As in §II.4.1, the extension to an ε-steepest ascent approach is quite straightforward, essentially requiring only the definition of ε-*Complementary Slackness Conditions*

characterising the set of ε-optimal solutions $X_{ij}^{\varepsilon}(\Pi)$: rather than further elaborating on the details, however, we prefer to move to a related but different idea, that will be presented in the next Paragraph.

II.4.3 Towards a combinatorial approach

The above Potential Decomposition approach suggests to extend to the Linear (MMCF) the Primal-Dual methods for the single-commodity (MCF) (Cf. Sec. II.1.4.6): some steps in this direction - but, unfortunately, not yet a complete algorithm - will be presented in this Paragraph. A more detailed description of the algorithm for the single-commodity case is in order: Primal-Dual methods attack the "Potential Decomposition" (into *m* univariate subproblems) of (MCF)

(PD) $\quad \max_{\pi} \{ \psi(\pi) : \pi \in \mathcal{R}^n \}$

where

(PR$_\pi$) $\quad \psi(\pi) = \Sigma_{(i,j) \in \mathcal{A}} \min_x \{ (c_{ij} - \pi_i + \pi_j) x_{ij} : 0 \le x_{ij} \le u_{ij} \} - \Sigma_{i \in \mathcal{N}} \pi_i b_i$

but, in the linear case, the coordinate ascent method is not sufficient due to the nondifferentiability of $\psi()$ or, equivalently, due to the fact that $X(\pi)$ - the set of optimal solutions of (PR$_\pi$) - is not necessarily a singleton. As in §II.4.1, however, $X(\pi)$ (and therefore $\partial\psi(\pi)$) can be described in terms of Complementary Slackness Conditions, i.e. of the partition $\mathcal{A} = A_\pi \cup I_\pi \cup B_\pi$ of the arcs among active, inactive and balanced: it is easy to see that $X(\pi)$ is the set of all pseudoflows where active and inactive arcs are *fixed* to their (upper and lower) bounds, while balanced arcs are *free*. Therefore, Primal-Dual methods maintain a pair $(\pi, \boldsymbol{x})$, where $\boldsymbol{x} \in X(\pi)$ is changed in the *primal phase* to reduce the total infeasibility $\| \boldsymbol{Ex} - \boldsymbol{b} \|_1$, i.e. the ($L_1$-)norm of the corresponding subgradient: at the end of the primal phase, either an $\boldsymbol{x}' \in X(\pi)$ is found such that $\boldsymbol{Ex}' = \boldsymbol{b}$, or an ascent direction is identified that is used in the *dual phase* to obtain better potentials π'. From the "dual" viewpoint, the distinguishing feature of Primal-Dual methods is the use of *elementary* ascent directions only, corresponding to the characteristic vector $\boldsymbol{e}_{\mathcal{S}}$ of some cutset $\mathcal{S} \subset \mathcal{N}$: the following nice characterisation of the *directional derivative* $\psi'(\pi, \boldsymbol{e}_{\mathcal{S}})$ with respect to an elementary direction is a key point in their analysis. The *imbalance* of a cutset $\mathcal{S}$ w.r.t. the pseudoflow $\boldsymbol{x}$ is

$$\Delta_{\mathcal{S}}(\boldsymbol{x}) = \Sigma_{i \in \mathcal{S}} \Delta_i(\boldsymbol{x})$$

where $\quad \Delta_i(\boldsymbol{x}) = \sum_{j:(j,i) \in \mathcal{A}} x_{ji} - \sum_{i:(i,j) \in \mathcal{A}} x_{ij} - b_i$

and the sets of *forward* and *backward* arcs in the induced cut are

$$\mathcal{S}^+ = \{ (i,j) : i \in \mathcal{S}, j \notin \mathcal{S} \} \qquad \text{and} \qquad \mathcal{S}^- = \{ (i,j) : i \notin \mathcal{S}, j \in \mathcal{S} \}$$

(with a little abuse of notation, the cut $\mathcal{S}^+ \cup \mathcal{S}^-$ will also be denoted by $\mathcal{S}$): then, with the "capacity" of $\mathcal{S}$ w.r.t. $\boldsymbol{x}$ defined as

$$\beta_{\mathcal{S}}(\boldsymbol{x}) = \left(\sum_{(i,j) \in \mathcal{S}^+ \cap B} u_{ij} - x_{ij} \right) + \left(\sum_{(i,j) \in \mathcal{S}^- \cap B} x_{ij} \right);$$

it is possible to show that

$$\psi'(\pi, \boldsymbol{e}_{\mathcal{S}}) = \Delta_{\mathcal{S}}(\boldsymbol{x}) - \beta_{\mathcal{S}}(\boldsymbol{x}).$$

Clearly, $\Delta_S(\boldsymbol{x})$ units of flow *must* be "pushed away" from $\mathcal{S}$ to make all the nodes balanced, and $\beta_S(\boldsymbol{x})$ measures how much flow *can* be "pushed away" from $\mathcal{S}$ by only using balanced arcs (Figure 15): if the imbalance of a cut exceeds its capacity, no feasible $\boldsymbol{x}$ (in which all the nodes, and hence all those in $\mathcal{S}$, are balanced) can exist in $X(\pi)$, and the potentials π_S must be increased to create some new balanced arcs that allow more flow to be "extracted" from $\mathcal{S}$. Relaxation methods try to use "small" cutsets $\mathcal{S}$: some node i with $\Delta_i(\boldsymbol{x}) > 0$ is used as a tentative $\mathcal{S}$, and a coordinate ascent step is performed if the above relation holds. Otherwise, the *reduced graph* $G_\pi(\mathcal{N}, B_\pi)$ (containing the balanced arcs only) is visited starting from i in search of a node j with $\Delta_j(\boldsymbol{x}) < 0$, where (i, j) is used "forward" if it is not full ($x_{ij} < u_{ij}$) and "backward" if it is not empty ($x_{ij} > 0$): any balanced node h reached during the process is added to $\mathcal{S}$ (Figure 16), each time checking for strict positivity of $\psi'()$ and performing a dual step in case. If a suitable j is found, the imbalance of both i and j is strictly decreased by sending flow on the (balanced) path joining them: otherwise, the visit terminates because no "forward nonfull" or "backward nonempty" arc exists for the current $\mathcal{S}$, but then $\beta_S(\boldsymbol{x}) = 0$ while $\Delta_S(\boldsymbol{x}) > 0$ and $\mathcal{S}$ provides an ascent direction. Hence, the search for an (elementary) ascent direction is performed via efficient combinatorial operations on G_π, that are in fact a *labelling* (or *augmenting path*) approach to the Max-Flow problem aimed at finding the pseudoflow $\boldsymbol{x} \in X(\pi)$ that minimizes the total imbalance $\Delta_{\mathcal{N}}(\boldsymbol{x})$: the Steepest Ascent method just solves the Max-Flow exactly, obtaining a Min-Cut with $\beta_S(\boldsymbol{x}) = 0$ and therefore a "normalized" L_1-Steepest Ascent direction (despite the "negative" result in §I.4.1).

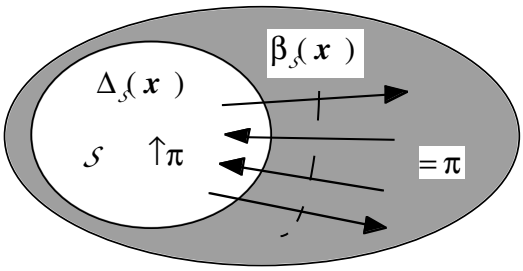

Figure 15: the "dual" phase of Relaxation methods

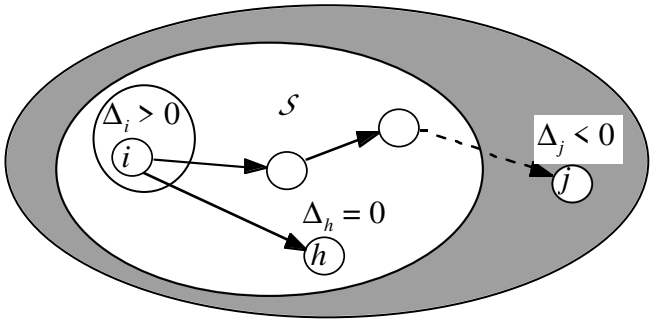

Figure 16: the "primal" phase of Relaxation methods

Unfortunately, generalization of the primal phase to (MMCF) is problematic, since Multicommodity Max-Flow problems lack of efficient combinatorial approaches corresponding to the labelling algorithm: conversely, the characterisation of $\psi'(\pi, \cdot)$ can be quite easily extended. In fact, for a pair $(\Pi, \boldsymbol{x})$ where $\boldsymbol{x}$ is a pseudoflow and the (CSC) are satisfied - i.e. $\boldsymbol{x}_{ij}$ is optimal for (K_Π^{ij}) - the extended definitions of active, balanced and inactive commodities on (i, j) have already been given in §II.4.2, so that it is only necessary to define what an *elementary* direction is in the (MMCF) case; several alternative definitions are - at least conceptually - possible, perhaps the most general one being

Definition II.4.3.1: *given a set of k cuts* $\Sigma = \{ \mathcal{S}^1, .., \mathcal{S}^k \}$ $(\mathcal{S}^h \subseteq \mathcal{N} \ \forall h \in \mathcal{K})$ *the direction* $\boldsymbol{d}_\Sigma = [\boldsymbol{e}_{\mathcal{S}^1}, .., \boldsymbol{e}_{\mathcal{S}^k}]$ *in the space of the dual multipliers* Π *is called an elementary direction; furthermore,* $\boldsymbol{d}_\Sigma$ *is called*

- *horizontal if* $\Sigma = \{ \varnothing, .., \mathcal{S}^h, .., \varnothing \}$ $(\mathcal{S}^h \subset \mathcal{N})$
- *vertical if* $\mathcal{S}^1 = \mathcal{S}^2 = .. = \mathcal{S}^k$

Horizontal and vertical directions (the names come from imagining each commodity living on a

separate copy of G, and the copies in a vertical stack) will be succinctly indicated by $\boldsymbol{d}_{\mathcal{S},h}$ and $\boldsymbol{d}_{\mathcal{S}}$ respectively: a characterisation of $\psi'()$ will be given for the horizontal directions case. By the definition of $\psi()$, its directional derivative in Π w.r.t. $\boldsymbol{d}_{\mathcal{S},h}$

$$\psi'(\Pi, \boldsymbol{d}_{\mathcal{S},h}) = \Sigma_{i \in \mathcal{S}} b_i^h + \Sigma_{(i,j) \in \mathcal{A}} \psi'_{ij}(\Pi, \boldsymbol{d}_{\mathcal{S},h})$$

is completely described by the m separate directional derivatives of the simpler functions $\psi_{ij}()$: since each $\psi_{ij}()$ is a polyhedral function, its restriction over a line is piecewise linear, so that for a "sufficiently small" $\alpha > 0$ (to be derived precisely later)

$$\delta_{ij} = \psi'_{ij}(\Pi, \boldsymbol{d}_{\mathcal{S},h}) = \frac{\psi_{ij}(\Pi + \alpha \boldsymbol{d}_{\mathcal{S},h}) - \psi_{ij}(\Pi)}{\alpha}.$$

Moving along the horizontal elementary direction $\boldsymbol{d}_{\mathcal{S},h}$, only the dual multipliers π^h ($\Rightarrow$ the reduced costs $\bar{\boldsymbol{c}}^h$) of commodity h are modified, so that the usual notions of *forward* and *backward* arcs in $\mathcal{S}$ can be used - the matter is clearly more complicated in the general case: hence, one has

$$\bar{c}^h_{ij}(\pi^h + \alpha \boldsymbol{e}_{\mathcal{S}}) = \bar{c}^h_{ij}(\pi^h) + \begin{cases} 0 & \text{if } (i,j) \notin \mathcal{S} \\ -\alpha & \text{if } (i,j) \in \mathcal{S}^+ \\ +\alpha & \text{if } (i,j) \in \mathcal{S}^- \end{cases}$$

i.e. $(i,j) \notin \mathcal{S} \Rightarrow \delta_{ij} = 0$. Moreover, since h is fixed the set A_Π (respectively I_Π, B_Π) of active (resp. inactive, balanced) arcs can be defined as

$$A_\Pi = \{ (i,j) : h \in A^{ij}_\Pi \text{ (resp. } I^{ij}_\Pi, B^{ij}_\Pi) \}$$

so that, recalling the definition of critical reduced cost η_{ij} from §II.4.2, for all

$$\alpha < \bar{\alpha} = \min_{(i,j) \in \mathcal{S}} \{ |\bar{c}^h_{ij} - \eta_{ij}| : ((i,j) \in \mathcal{S}^+ \cap I_\Pi) \text{ or } ((i,j) \in \mathcal{S}^- \cap A_\Pi) \}$$

no active or inactive arc becomes balanced: hence, for each $\alpha < \bar{\alpha}$, $\psi_{ij}(\Pi + \alpha \boldsymbol{d}_{\mathcal{S},h}) - \psi_{ij}(\Pi)$ is a linear function of α. If (i,j) is either active or inactive, $\boldsymbol{x}_{ij}$ remains optimal for (K^{ij}_Π), therefore

$$\psi_{ij}(\Pi + \alpha \boldsymbol{d}_{\mathcal{S},h}) - \psi_{ij}(\Pi) = \bar{c}^h_{ij}(\pi^h + \alpha \boldsymbol{e}_{\mathcal{S}}) - \bar{c}^h_{ij}(\pi^h) = \pm \alpha x^h_{ij}$$

so that $$\delta_{ij} = \begin{cases} 0 & \text{if } (i,j) \in I \\ -u^h_{ij} & \text{if } (i,j) \in \mathcal{S}^+ \cap A \\ +u_{ij}{}^h & \text{if } (i,j) \in \mathcal{S}^- \cap A \end{cases} ;$$

on the contraty, any $(i,j) \in B_\Pi$ becomes either active or inactive for any $\alpha > 0$, and the optimal solution of (K^{ij}_Π) (possibly) changes. In this case, the *free forward space*

$$\Delta_{ij} = u_{ij} - \sum_{p \in A^{ij}_\Pi} x^p_{ij} = u_{ij} - \sum_{p \in A^{ij}_\Pi} u^p_{ij}$$

and the *free backward space*

$$\nabla^h_{ij} = \begin{cases} x^h_{ij} & \text{if } \eta_{ij} = 0 \\ \sum_{p \in B^{ij}_\Pi / \{h\}} u^p_{ij} - x^p_{ij} & \text{if } \eta_{ij} < 0 \end{cases}$$

of (i,j) w.r.t. $(\Pi, \boldsymbol{x})$ and h play a role: the rationale of these definitions is clarified by the proofs of the following two Lemmas where, for notiational simplicity, the reference to (i,j) is dropped.

Lemma II.4.3.2: *if (i, j)* $\in \mathcal{S}^+ \cap B_\Pi$, *then* $\delta_{ij} = -\min\{\ \Delta_{ij}\ ,\ u_{ij}^h\ \}$.

Proof: The total flow and the contribution to the value of $\psi()$ of the balanced commodities are

$$y = \sum_{p \in B_\Pi} x^p \qquad \text{and} \qquad z = \sum_{p \in B_\Pi} \bar{c}^p x^p = \eta y$$

since $\bar{c}^p = \eta$ for all $p \in B_\Pi$: it is easy to see that $z = \eta\Delta$, as if $\eta < 0$ then $\bar{x} = u$ and $\Delta = y$, otherwise $\eta = 0$ and then $z = 0 = \eta\Delta$. Now, let $\Pi(\alpha) = \Pi + \alpha \boldsymbol{d}_{\mathcal{S},h}$ and consider any optimal solution of $(K_{\Pi(\alpha)})$: since $\bar{c}^h$ has decreased, flow from the (former) balanced commodities is "transferred" to the h-th until either $x^h = u^h$ or the maximum quantity of flow that can be transferred, i.e. $\Delta - x^h$, has been assigned to the h-th commodity; the flows of active and inactive commodities over (i, j) don't change, hence their contribution to δ is 0. Therefore, if $v = \min\{\ \Delta\ ,\ u^h\ \}$ is the new value of x^h, the total contribution of the commodities in (the former) B_Π to the (new) value of $\psi(\ \Pi(\ \alpha\ )\ )$ is $\eta(\ \Delta - v\ ) + (\ \eta - \alpha\ )v = \eta\Delta - \alpha v$, hence $\delta = -v$. ♦

Lemma II.4.3.3: *if (i, j)* $\in \mathcal{S}^- \cap B_\Pi$, *then* $\delta_{ij} = \max\{\ x_{ij}^h - \nabla_{ij}^h\ ,\ 0\ \}$.

Proof: if $\eta = 0$ ($\Rightarrow \nabla^h = x^h$ by definition $\Rightarrow \delta = 0$), x^h can be set to 0 leaving the rest of the optimal solution unchanged, so that the contribution of the h-th commodity to z is $\eta x^h = 0x^h = (\ \eta - \alpha\ )0 = 0$. If $\eta < 0$, then in the optimal solution of $(K_{\Pi(\alpha)})$ x^h will decrease as long as the other "nonfull" balanced commodities can "absorb" it: hence, ∇^h represents the maximum quantity of flow that can be "absorbed away" from the h-th commodity, and two different cases can occur

- either $\nabla^h \geq x^h$ ($x^h - \nabla^h \leq 0$), i.e. the h-th commodity can be "absorbed" by other (balanced) commodities, with the same reduced cost η: in this case, $\psi()$ doesn't change, therefore $\delta = 0$;
- or $\nabla^h < x^h$, i.e. *not all* the h-th commodity can be "absorbed": then, $v = x^h - \nabla^h > 0$ is the h-th commodity "left", and $\psi(\ \Pi(\ \alpha\ )\ ) = \eta(\ \Delta - v\ ) + (\ \eta + \alpha\ )v = \eta\Delta + \alpha v$, therefore $\delta = v$.

Hence, the definition of ∇^h only serves to unify the two cases $\eta = 0$ and $\eta < 0$. ♦

As a consequence of the above Lemmas, one has

$$\psi'(\ \Pi\ ,\ \boldsymbol{d}_{\mathcal{S},h}\ ) = b_{\mathcal{S}}^h - \sum_{(i,j) \in \mathcal{S}^+ \cap A} u_{ij}^h + \sum_{(i,j) \in \mathcal{S}^- \cap A} u_{ij}^h$$
$$- \sum_{(i,j) \in \mathcal{S}^+ \cap B} \min\{\ \Delta_{ij}\ ,\ u_{ij}^h\ \} + \sum_{(i,j) \in \mathcal{S}^- \cap B} \max\{\ x_{ij}^h - \nabla_{ij}^h\ ,\ 0\ \};$$

however, this can be turned in a more significative form by defining

$$\Delta_{\mathcal{S}}^h(\ \boldsymbol{x}\ ) = b_{\mathcal{S}}^h - \sum_{(i,j) \in \mathcal{S}^+} x_{ij}^h + \sum_{(i,j) \in \mathcal{S}^-} x_{ij}^h$$

(the total imbalance of $\mathcal{S}$ w.r.t. $\boldsymbol{x}^h$) and studying the quantity $\beta_{\mathcal{S}}^h(\ \boldsymbol{x}\ ) = \Delta_{\mathcal{S}}^h(\ \boldsymbol{x}\ ) - \psi'(\ \Pi\ ,\ \boldsymbol{d}_{\mathcal{S},h}\ )$.

Theorem II.4.3.4:

$$\beta_{\mathcal{S}}^h(\ \boldsymbol{x}\ ) = \sum_{(i,j) \in \mathcal{S}^+ \cap B} (\ \min\{\ \Delta_{ij}\ ,\ u_{ij}^h\ \} - x_{ij}^h\ ) + \sum_{(i,j) \in \mathcal{S}^- \cap B} \min\{\ \nabla_{ij}^h\ ,\ x_{ij}^h\ \}$$

Proof: For enumeration of the terms in the expression for $\Delta_{\mathcal{S}}^h$ and the directional derivative:

- $b_{\mathcal{S}}^h$ is present in both, hence it elides;

– an *inactive* arc (i, j) gives no contribution to $\psi'()$, and contribution $x_{ij}^h = 0$ to Δ_S^h;
– an *active* arc (i, j) gives the same contribution $\pm u_{ij}^h$ (depending if it is a backward or forward arc) to $\psi'()$ and to Δ_S^h;
– a *balanced forward* arc (i, j) gives contribution $-\min\{ \Delta_{ij} , u_{ij}^h \}$ to $\psi'()$ and $-x_{ij}^h$ to Δ_S^h, hence $\min\{ \Delta_{ij} , u_{ij}^h \} - x_{ij}^h$ to β_S^h;
– a *balanced backward* arc (i, j) gives contribution $\max\{ x_{ij}^h - \nabla_{ij}^h , 0 \}$ to $\psi'()$ and x_{ij}^h to Δ_S^h, hence $x_{ij}^h - \max\{ x_{ij}^h - \nabla_{ij}^h , 0 \} = x_{ij}^h + \min\{ \nabla_{ij}^h - x_{ij}^h , 0 \} = \min\{ \nabla_{ij}^h , x_{ij}^h \}$ to β_S^h. ♦

Theorem II.4.3.4 justifies the definitions of *nonempty* and *nonfull* balanced arcs (w.r.t. $(\Pi, \boldsymbol{x})$ and h) as the $(i, j) \in B_\Pi$ such that

$$\min\{ \nabla_{ij}^h , x_{ij}^h \} > 0 \qquad \text{and} \qquad \min\{ \Delta_{ij} , u_{ij}^h \} > x_{ij}^h$$

respectively: since $\Delta_{ij} \geq x_{ij}^h$, all the terms in the expression for β_S^h are nonnegative, hence $\beta_S^h \geq 0$. As in the single-commodity case, β_S^h is the flow that can be "pushed away" from S, so that the result

$$\Delta_S^h > 0 \;\Rightarrow\; \psi'(\Pi , \boldsymbol{d}_{S,h}) > 0 \Leftrightarrow \text{the total imbalance of } S \text{ is strictly larger than the flow that can be "pushed away"}$$

is readily obtained; an immediate corollary is that if all the balanced arcs in the cut are *forward full* or *backward empty*, then $\boldsymbol{d}_{S,h}$ is a direction of ascent.

Analogous results can be derived for vertical elementary directions and for general elementary directions, but they require a clumsier notation and a set of proper definitions: a simpler proof can be derived by considering the more general class of Min-Cost Flow problems on Hypergraphs. In a (directed) *hypergraph* $G(\mathcal{N} , \mathcal{A})$ [GLN93], *hyperarcs* can have multiple heads and tails: more formally, $\mathcal{A} \subseteq 2^{\mathcal{N}} \times 2^{\mathcal{N}}$, i.e. each $a \in \mathcal{A}$ is an (ordered) pair $(H(a), T(a))$ of (possibly empty) subsets of the nodes, $H(a)$ being the *head* nodes and $T(a)$ being the *tail* nodes.
One important class of hypergraphs is that of *B-hypergraphs*, where each hyperarc can have multiple tails but at most one tail: the B-Hypergraph Min-Cost Flow problem

(BHMCF) $\quad \min_{\boldsymbol{x}} \{ \boldsymbol{cx} : \boldsymbol{Ex} = \boldsymbol{b} , \boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{u} \}$

where $\boldsymbol{E}$ is the node-hyperarcs incidence matrix having $| H(a) | + | T(a) |$ nonzeroes in the column $\boldsymbol{E}^a$ relative to hyperarc a, is general enough to properly contain (MMCF) as a subclass, since any instance of (MMCF) can be transformed into one of (BHMCF) on a graph with $kn + m$ nodes, km hyperarcs with two tails and one head and m "tailless" hyperarcs, as depicted in Figure 17. Nevertheless, a nice characterisation can be given for the directional derivative (w.r.t. an elementary direction $\boldsymbol{d}_S$) of the Lagrangean function

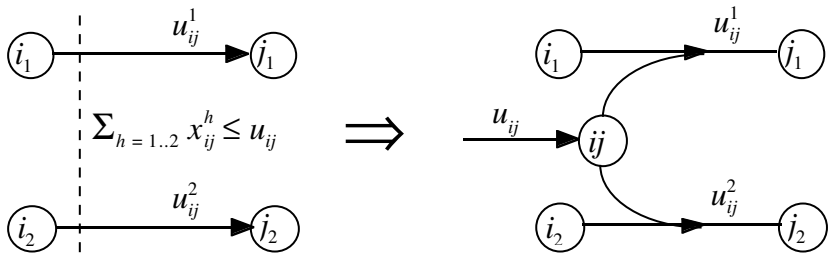

Figure 17: the reduction of (MMCF) to (BHMCF)

$$\psi(\pi) = \Sigma_{a \in \mathcal{A}} \min\{ \bar{c}_a(\pi) x_a : 0 \leq x_a \leq u_a \} + \pi \boldsymbol{b} ,$$

where the reduced costs are

$$\bar{c}_a(\pi) = c_a - \Sigma_{i \in T(a)} \pi_i + \Sigma_{j \in H(a)} \pi_j ,$$

by only properly generalizing the notion of *forward* and *backward* (hyper)arc. If the *directionality* of an hyperarc a with respect to a cutset $\mathcal{S}$ is

$$K(\mathcal{S}, a) = \left| T(a) \cap \mathcal{S} \right| - \left| H(a) \cap \mathcal{S} \right| = - \boldsymbol{e}_{\mathcal{S}} \boldsymbol{E}^a$$

a is forward if $K(\mathcal{S}, a) > 0$ and backward if $K(\mathcal{S}, a) < 0$ (Figure 18): the rationale of the definition is that a forward hyperarc has more tails than heads in $\mathcal{S}$, so that increasing x_a decreases the total imbalance $\Delta_{\mathcal{S}}(\boldsymbol{x})$ of $\mathcal{S}$, while the converse is true for a backward one.

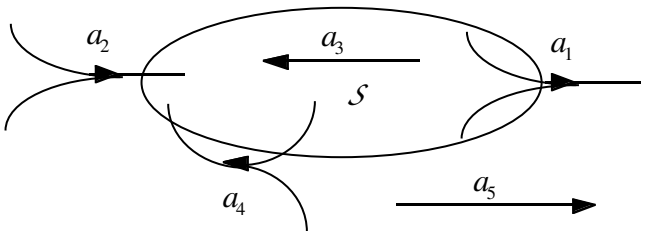

Figure 18: directionality - $K(\mathcal{S}, a_1) = 2$, $K(\mathcal{S}, a_1) = -1$, $K(\mathcal{S}, a_3) = K(\mathcal{S}, a_4) = K(\mathcal{S}, a_5) = 0$

Then, it can be easily shown that

$$\psi'(\pi, \boldsymbol{e}_{\mathcal{S}}) = \Delta_{\mathcal{S}}(\boldsymbol{x}) - \beta_{\mathcal{S}}(\boldsymbol{x})$$

where $\Delta_{\mathcal{S}}(\boldsymbol{x}) = \Sigma_{i \in \mathcal{S}} \left(\Sigma_{a : i \in H(a)} x_a - \Sigma_{a : i \in T(a)} x_a - b_i \right)$,

$$\beta_{\mathcal{S}}(\boldsymbol{x}) = \left(\sum_{a \in B : K(\mathcal{S}; a) > 0} K(\mathcal{S}, a)(u_a - x_a) \right) + \left(\sum_{a \in B : K(\mathcal{S}; a) < 0} K(\mathcal{S}, a) x_a \right)$$

and the set B of balanced hyperarcs is defined in the obvious way: note that hyperarcs having the same numbers of heads and tails in $\mathcal{S}$ (i.e. $K(\mathcal{S}, a) = 0$) do not contribute to $\beta_{\mathcal{S}}(\boldsymbol{x})$. Hence, $\psi'()$ is again the difference between the total imbalance of $\mathcal{S}$ and the flow that can be "pushed away" on the balanced hyperarcs: this can be easily extended to the case of "gains" in the hyperarcs, i.e. of not-all-unitary coefficients in the node-hyperarcs incidence matrix $\boldsymbol{E}$, and can be directly used to derive the results about general elementary directions in (MMCF).

However, no "pure" Relaxation method for (MMCF) or (HBMCF) has been obtained yet, because the primal phase - that finds either some sort of elementary direction or a feasible flow - seems to be much more complex than in the (MCF) case: in fact, the Max-Flow versions of the problems are not easily solvable, and it is not even clear if and how they would grant an ascent direction. Developing a "combinatorial" primal phase would be an interesting result, and could lead to efficient implementations: however, an alternative approach could be to provide *heuristics* for efficiently finding ascent directions. In fact, the convergence of a Primal-Dual method can be easily ensured by "immerging" it inside the Steepest Ascent approach of §II.4.2: the Steepest Ascent (QP) subproblem (SAD_{Π}) provides an "exact" primal phase to be used as "last resort", while the heuristics should provide low-cost ascent directions. Actually, the heuristics might also be capable of solving most instances, as indicated by some encouraging - although preliminary - computational results that have been independently obtained [DBR96] for Hypergraph (MCF) problems. The heuristics might exploit the special structure of the problem at hand, for instance searching for horizontal or vertical elementary directions for (MMCF), that are not likely to be identified when looking at the more general (BHMCF). Altogether, the lines of approach described in this Chapter seem to indicate that "hybrid" algorithms integrating smooth, nondifferentiable and combinatorial technique may in a future provide efficient means of solving (MMCF) or even more general classes of problems.

II.5 Other Network-Structured Problems

In the previous Chapters, various Dual Ascent approaches for the Linear and Continuous (MMCF) have been discussed: since the problem is "easy", the viewpoint was that of obtaining exact solutions, although algorithms that rapidly provide "good" solutions can also be of practical interest. Here, the Dual Ascent techniques are rather applied to quickly obtaining Lower Bounds to the value of the integer solution for two $\mathcal{NP}$-hard network-structured problems, the Reserve Problem arising in telecommunications in §II.5.1 and the general Fixed Charge (MMCF) in §II.5.2.

II.5.1 The Reserve Problem

The Reserve Problem can be clearly illustrated by referring to the real-world application that originated the data set used in this Paragraph[Ω]: a (small) directed graph $G(\mathcal{N}, \mathcal{A})$, with n nodes and m arcs, represents a *telecommunication network*, such as the backbone of a telephone company in a large city where each arc represents an optic fibre cable, capable of thousands of lines. The network is "up and functional", i.e. k Origin/Destination pairs are given with the relative (estimated) traffic demands and the *fixed* (multi)routings that, for each (O_k, D_k) pair, are used to satisfy the demand: however, some of the arcs are subject to "catastrophic" *failures* (landslides, street works ...), and the problem is that of optimally allocating *reserve capacities* on the arcs to face such failures. An usual assumption is that the probability of two simultaneous failures is negligible, even though the model can be extended to the case of multiple arc and *node* failures; also, the network is assumed to be 2-*connected*, so that the topology is not an issue. The difficulty of the problem lies in the fact that the reserve capacity needed for facing a given failure can (and should) be exploited in the case of any other failure: solving for each failure separately and merging all the obtained solutions together can deliver suboptimal solutions, as shown in Figure 19 for the case of uniform demands (the crossed arcs are those prone to failure). Formally, a set Γ of possible failures is given, and for each $a \in \Gamma$ the set $K(a)$ of the commodities that are affected by the failure is given: in the current setting, failures are arcs (in all the instances $\Gamma = \mathcal{A}$) and $K(a)$ is the set of O/D pairs that are routed through arc a. Hence, the vector $\boldsymbol{b}_k^a$ of the demands generated by the failure $a \in \Gamma$ for commodity k is known: the structure of $\boldsymbol{b}_k^a$ depends on the *re-routing strategy*, i.e. if the re-routing of commodity k is done "locally" (from i to j if $a = (i, j)$) or "globally" (from O_k to D_k). An Integer Linear Programming formulation of the

Ω This line of research started from a visit at the Université Libre de Bruxelles, where Martine Labbé and Christelle Wynats were working on a real-world application concerning a local telephone company.

Reserve Problem is

$$(\mathrm{RP}) \qquad \begin{cases} \min \boldsymbol{c}\boldsymbol{z} & \\ \boldsymbol{E}^a \boldsymbol{x}_k^a = \boldsymbol{b}_k^a & a \in \Gamma,\ k \in \mathrm{K}(a) \\ \Sigma_{k \in \mathrm{K}(a)} \boldsymbol{x}_k^a \le \boldsymbol{z} & a \in \Gamma,\ k \in \mathrm{K}(a) \\ \boldsymbol{x}_k^a,\ \boldsymbol{z} \in \mathcal{N}^m & \end{cases}$$

where $\boldsymbol{E}^a$ is node-arc incidence matrix of $G^a(\mathcal{N}, \mathcal{A} / \{ a \})$: it has $k(m-1)\left|\Gamma\right|$ ($\approx km^2$, assuming from now on $\left|\Gamma\right| = m$) integer variables, but only the *design variables* $\boldsymbol{z}$ have costs, although routing costs could also be taken into account. As the Integer (MMCF), (RP) is a large (ILP) with block-structured constraint matrix and network structure within each block: however, (RP) has m^2 *complicating* constraints $\Sigma_{k \in \mathrm{K}(a)} \boldsymbol{x}_k^a \le \boldsymbol{z}$, and only the commodities belonging to the same K(a) (re-routings generated by the same failure) compete for the capacity of the arcs. Hence, (RP) has another possible level of decomposition, in that the problem could be (heuristically) approached by solving for each commodity separately: as shown in Figure 20, however, this can be suboptimal again.

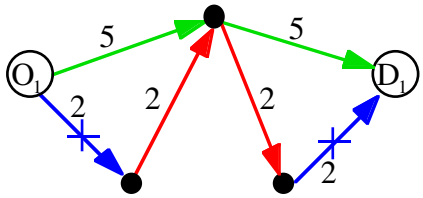

Figure 19: solving for each k and merging is not optimal

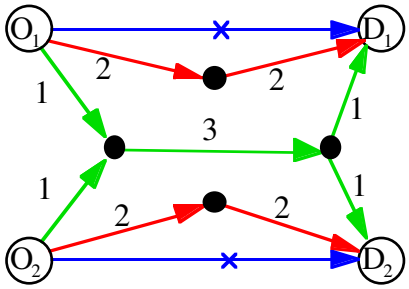

Figure 20: solving for each a and merging is not optimal

As for any (difficult) (ILP), rapidly obtaining tight lower bounds on z(RP) is crucial for a Branch and Bound approach, but it can also be useful for "validating" heuristics: furthermore, Lagrangean-based bounds can serve as a base for Lagrangean-based heuristics, where the dual multipliers and/or the unfeasible solutions of the Lagrangean subproblems are used for constructing (hopefully) good feasible solutions. The Lagrangean relaxation w.r.t. the "linking" constraints

$$(\mathrm{LRP}_{\boldsymbol{y}}) \qquad \varphi(\boldsymbol{y}) = \varphi_0(\boldsymbol{y}) + \Sigma_{a \in \Gamma} \Sigma_{k \in \mathrm{K}(a)} \varphi_k^a(\boldsymbol{y}^a)$$

decomposes into km independent (SPT)s

$$(\mathrm{LRP}'_{\boldsymbol{y},a,k}) \qquad \varphi_k^a(\boldsymbol{y}^a) = \min\{ \boldsymbol{y}^a \boldsymbol{x}_k^a : \boldsymbol{E}^a \boldsymbol{x}_k^a = \boldsymbol{b}_k^a,\ \boldsymbol{x}_k^a \in \mathcal{N}^m \}$$

plus the simple problem solvable by inspection

$$(\mathrm{LRP}''_{\boldsymbol{y}}) \qquad \varphi_0(\boldsymbol{y}) = \min\{ (\boldsymbol{c} - \Sigma_{a \in \Gamma} \boldsymbol{y}^a)\boldsymbol{z} : \boldsymbol{z} \in \mathcal{N}^m \} = \begin{cases} 0 & \text{if } \boldsymbol{c} \ge \Sigma_{a \in \Gamma} \boldsymbol{y}^a \\ -\infty & \text{otherwise} \end{cases};$$

note that the m^2 dual multipliers are partitioned into m vectors $\boldsymbol{y}^a \in \mathcal{R}^m$, one for each failure $a \in \Gamma$. The corresponding Lagrangean Dual is

$$(\mathrm{DRP}) \qquad \max_{\boldsymbol{y}} \{ \varphi(\boldsymbol{y}) : \boldsymbol{y} \in Y \} \qquad \text{where} \qquad Y = \{ \boldsymbol{y} \ge 0 : \Sigma_{a \in \Gamma} \boldsymbol{y}^a = \boldsymbol{c} \}$$

and therefore $\varphi_0(\boldsymbol{y}) = 0$ for each $\boldsymbol{y} \in Y$: the "=" sign in the definition can be justified by writing the

Dual of (RP) and noticing that the explicit constraint $\Sigma_{a \in \Gamma} \boldsymbol{y}^a \le \boldsymbol{c}$ must be satisfied to equality by *all* the optimal solutions. Altough (LRP'_y) could simply be disregarded, it is possible to exploit it for enhancing the "flexibility" of the method: since *any* $\boldsymbol{y} \in Y$ satisfies $\Sigma_{a \in \Gamma} \boldsymbol{y}^a = \boldsymbol{c}$, *any* $\boldsymbol{z} \ge \boldsymbol{0}$ is an optimal solution of (LRP'_y), so that *any* vector of the form

$$\boldsymbol{g}_{\bar{\boldsymbol{x}}}(\boldsymbol{z})^a = [\, \Sigma_{k \in K(a)} \bar{\boldsymbol{x}}^a_k - \boldsymbol{z} \,]$$

(where $\bar{\boldsymbol{x}}^a_k$ is an optimal solution of ($LRP'_{y,a,k}$)) belongs to $\partial\varphi(\boldsymbol{y})$. Since it is possible to choose $\boldsymbol{z} \ge \boldsymbol{0}$ at will, it may be useful to make a "wise" choice: a possibility is clearly

$$\boldsymbol{z} = \operatorname{argmin} \left\{ \Sigma_{a \in \Gamma} \left\| \Sigma_{k \in K(a)} \bar{\boldsymbol{x}}^a_k - \boldsymbol{z} \right\|^2 : \boldsymbol{z} \ge \boldsymbol{0} \right\}$$

that (since $\bar{\boldsymbol{x}}$ is fixed) is the Steepest Ascent direction finding problem restricted to a (small) subset of $\partial\varphi(\boldsymbol{y})$. Fortunately, a closed-form solution of the problem exists, since by letting $\tilde{\boldsymbol{x}}^a = \Sigma_{k \in K(a)} \bar{\boldsymbol{x}}^a_k$ it can be rewritten as

$$\Sigma_{(i,j) \in \mathcal{A}} \min_z \left\{ \Sigma_{a \in \Gamma} (\tilde{x}^a_{ij} - z_{ij})^2 : z_{ij} \ge 0 \right\}$$

so that $\bar{z}_{ij} = (\Sigma_{a \in \Gamma} \tilde{x}^a_{ij})^2 / m$ nullifies the derivative $2(m z_{ij} - \Sigma_{a \in \Gamma} \tilde{x}^a_{ij})$ of the objective function (note that $\bar{z}_{ij} \ge 0$ since all the $\tilde{x}^a_{ij}$ are): this $\bar{\boldsymbol{z}}$ is especially appealing for a "pure" subgradient method, where $\boldsymbol{g}_{\bar{\boldsymbol{x}}}(\bar{\boldsymbol{z}})$ is used as the direction, because $\boldsymbol{y}(\alpha) = \boldsymbol{y} + \alpha \boldsymbol{g}_{\bar{\boldsymbol{x}}}(\bar{\boldsymbol{z}})$ is feasible w.r.t. the $\Sigma_{a \in \Gamma} \boldsymbol{y}^a = \boldsymbol{c}$ constraints for all $\alpha \ge 0$. This can also be extended for taking into account the nonnegativity constraints, because finding the minimum-norm *projected* subgradient w.r.t. the *active* constraints amounts at solving the m independent problems

$$\min \left\{ \Sigma_{a \in \Gamma} ([\, \tilde{x}^a_{ij} - z_{ij} \,]_+^{y^a_{ij}})^2 : z_{ij} \ge 0 \right\}$$

(where the projection operator "modulo y" $[\cdot]_+^y$ has been defined in §II.4.1) for each $(i, j) \in \mathcal{A}$: this can be done with the efficient algorithm described in Figure 21, the arc index being dropped from the notation.

```
B_i = Γ; B' = { a ∈ Γ : y^a = 0 }; i = 1;
do
    z_i = ( Σ_{a∈Γ} x̃^a ) / |B_i|; x_i = min{ x̃^a : a ∈ B_i ∩ B' };      /* x_i = +∞ if B_i ∩ B' = ∅ */
    if( z_i ≤ x_i )
        then  z* = z_i; STOP;
        else  B_{i+1} = B_i / { a ∈ B' : x_i = x̃^a }; i = i + 1;
while( ~STOP )
```

Figure 21: finding the minimum norm projected subgradient

The algorithm terminates in at most $|B'|$ iterations, and its straightforward implementation is $O(\max\{m, |B'| \lg(|B'|)\})$ (but a linear time variant exists): correctness follows from the fact that, despite $\{z_i\}$ being a strictly decreasing sequence, $z_{i+1} \ge x_i > x_{i-1}$ for each i, since

$$z_i = \frac{(|B_i| - p_i) z_{i+1} + p_i x_i}{|B_i|} > x_i$$

(where $p_i = |\{ a \in B_i \cap B' : x_i = \tilde{x}^a \}|$) i.e. even deleting *all* the p_i objects from B_i cannot make z_{i+1} to become smaller than x_i. Hence, upon termination $\tilde{x}^a \le z^*$ for each $a \in B_i / B'$ and $\tilde{x}^a > z^*$ for

$a \in B_i \cap B'$, so that z^* is exactly the average of the entries that are *not* turned to zero by the projection, and reasoning as above it is easy to conclude that it is optimal.

However, this was not used in the Bundle-based approach to (DRP) - hereafter named BRP - because we were more interested in comparing it with the existing Subgradient-based implementation [LSW95]. The task was challenging, because the ($LRP'_{y,a,k}$) subproblems are very small, so that calculation of $\varphi(\boldsymbol{y})$ is quite fast: this clearly gives an advantage to the Subgradient algorithm, that has an extremely fast "coordination" phase. Conversely, (DRP) has "many" dual variables and the dual constraint set Y is "complicated", requiring both m^2 nonnegativity and m "general" linear constraints, so that the solution time required by ($\Pi'_{\beta t}$) is relatively high. However, a specialized implementation of the basic "interface" between the Bundle solver and the BTT algorithm helped in keeping the cost bearable: typically, calculation of $z_h = \Sigma_{i \in \beta} [\, \boldsymbol{g}_i \,]^h\theta_i + \Sigma_{j \in \gamma} [\, \boldsymbol{h}_j \,]^h\mu_j$ at each step (Cf. §I.3.3 and §I.3.5) is the most expensive operation, so that exploitation of the structure of the constraint $\boldsymbol{h}_j$ and the *row-wise* (opposed to the "natural" column-wise) implementation of the matrix $\boldsymbol{G}_\beta$ can reduce the cost. In this case, the great flexibility provided by the design of the `BMinQuad` class had really been crucial: the importance of other "standard" characteristics of BRP, such as the Lagrangean Variables Generation (Cf. §II.2.1) and the "Long Term" *t*-strategies (Cf. §I.5.3) will be discussed later on.

The Subgradient algorithm - hereafter named SGRP - is a good practical implementation: in fact, it was used in the *production* code for solving the real-world problems [LSW95]. SGRP uses the last subgradient only as direction, the "standard" bound-based steplength [HWC74] and projection of the trial point [LPS95] on the feasible region Y by means of a specialized code: although the experiences from the "tuning phase" of the code have not been reported, it is likely that the choice of not using the previous direction [CFM75] [Cr76] is due to the projection step on the "complicated" feasible region, that "messes up" the direction somewhat.

Both the codes were tested on 9 real-world problems, 6 small ones and 3 larger ones, described in Table 30: there, $kt = \Sigma_{a \in \Gamma} | K(a) |$ is the number of (SPT)s to be solved at each iteration and v is the number of Lagrangean multipliers $\boldsymbol{y}$ of (DRP); the computational results are presented in Table 31, Table 32 and Table 33. The codes use very different stopping rules, since BRP stops when a relative precision of ε is attained while SGRP stops if no improvement is observed in a certain number q of consecutive iterations (the same parameter is also used in the steplenght updating): hence, the three tables report the performances for "comparable" choices of ε and q. BRP is far better than SGRP, obtaining comparable or better bounds in a very small fraction of the iterations: despite the considerable time spent in the the ($\Pi'_{\beta t}$) solution, BRP is over an order of magnitude faster than SGRP, proving that a "complex" coordinator can dramatically improve the overall performances of the method. A clear representation of the different behaviour of the two algorithms is given in Chart 24, where the value of $\varphi()$ obtained by BRP and SGRP is plotted as a function of the iterations count for the largest instance: the first 10 iterations, where most of the overall improvement is obtained, have been disregarded to underline the long-term behaviour. However, the real advantage of BRP over SGRP is *not* just to obtain much better bounds in the same number of iterations, but rather to correctly predict the obtained accuracy, and therefore to be able to stop shortly after than the required accuracy has been achieved: furthermore, increasing the q parameter for SGRP is not guaranteed to increase the obtained precision, while the Bundle code has a reliable stopping mechanism.

	n	*m*	*kt*	*v*
network.20	20	30	138	783
network.21	20	30	164	812
network.22	20	30	107	783
network.22	20	34	133	825
network.24	20	29	143	644
network.25	20	31	113	810
datab1.txt	45	63	615	3844
datab2.txt	45	63	606	3844
datab3.txt	45	63	829	3844

Table 30: characteristics of the test problems

BRP - $\varepsilon = 10^{-2}$			SGRP - $q = 3$		
time	*iter*	*bound*	*time*	*iter*	*bound*
5.64	60	6001	99	1661	6005
4.74	44	7612	102	1292	7524
5.36	60	5221	79	1723	5239
11.26	114	5778	45	587	5734
5.22	85	4453	46	807	4442
4.01	71	8015	73	1434	7974
151.98	171	12361	1396	2390	12293
163.34	139	11349	2305	4113	11321
232.76	233	17626	3249	4179	17579

Table 31: BRP with $\varepsilon = 10^{-2}$ and SGRP with $q = 3$

BRP - $\varepsilon = 10^{-3}$			SGRP - $q = 9$		
time	*iter*	*bound*	*time*	*iter*	*bound*
6.35	89	6046	207	3507	6018
11.59	149	7572	319	4101	7548
12.99	173	5253	181	3999	5246
25.98	241	5824	157	2052	5777
21.83	308	4502	127	2254	4467
7.24	127	8106	294	4609	8056
429.29	502	12517	4020	6973	12397
221.09	230	11424	1858	3354	11328
641.53	539	17936	8133	10737	17706

Table 32: BRP with $\varepsilon = 10^{-3}$ and SGRP with $q = 9$

BRP - $\varepsilon = 10^{-3}$			SGRP - $q = 15$		
time	*iter*	*bound*	*time*	*iter*	*bound*
6.35	89	6046	309	5251	6022
11.59	149	7572	252	3213	7547
12.99	173	5253	174	3817	5247
25.98	241	5824	288	2979	5778
21.83	308	4502	177	3195	4477
7.24	127	8106	241	4573	8058
429.29	502	12517	3332	5798	12393
221.09	230	11424	2276	4087	11316
641.53	539	17936	5417	7192	17677

Table 33: BRP with $\varepsilon = 10^{-3}$ and SGRP with $q = 15$

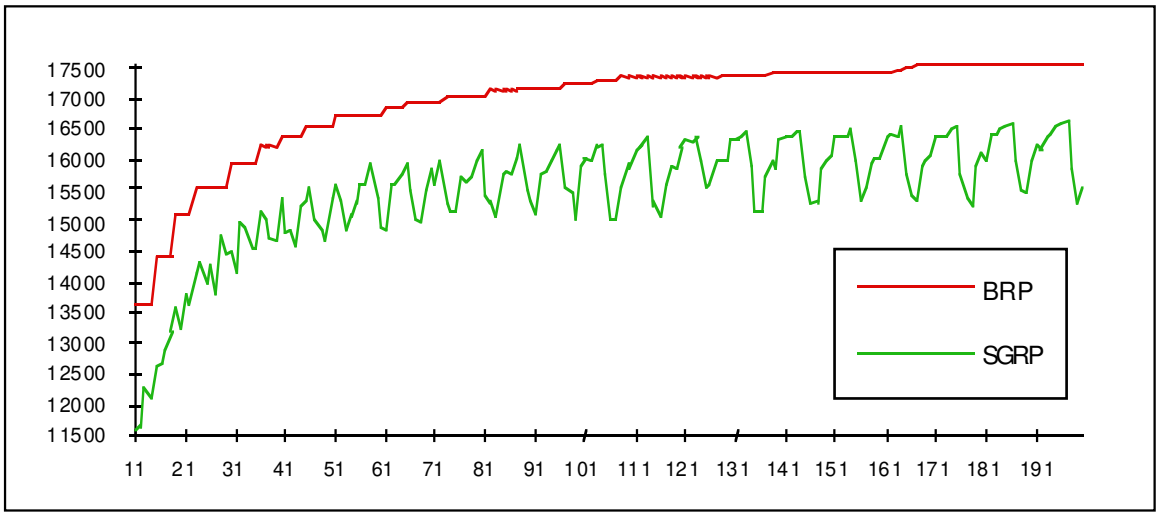

Chart 24: evolution of the lower bounds for BRP and SGRP as a function of the iterations count for datab3.txt

These results, however, would have surely been much worse without the careful implementation of some technical details: the clearest example is the need for a specialized (QP) solver, that is revealed in Chart 25 where the fraction of the time spent in the (LRP_y') (*SPT*) and in the ($\Pi_{\beta t}'$) solution (*QP*) is depicted for all the instances, showing how - despite the efficient BTT algorithm and the specialized implementation of the "core" operations - the coordination step always requires about 90% of time.

Another important detail is the Lagrangean Variables Generation strategy, whose effects are reported in Table 34 and the corresponding Chart 26 and Chart 27: as for (MMCF) (Cf. §II.2.1), the Monotone LVG (*M*) where variables are only added and never deleted is considerably better than no LVG at all (*N*), but in this case the Full LVG (*F*) (where variables are both added and deleted) is even more efficient, cutting down on the total time of a factor of four; the difference with (MMCF) means that - probably due to the more "complicated" constraint region - many of variables that are "active" in the initial iterations are not so in the optimal solution.

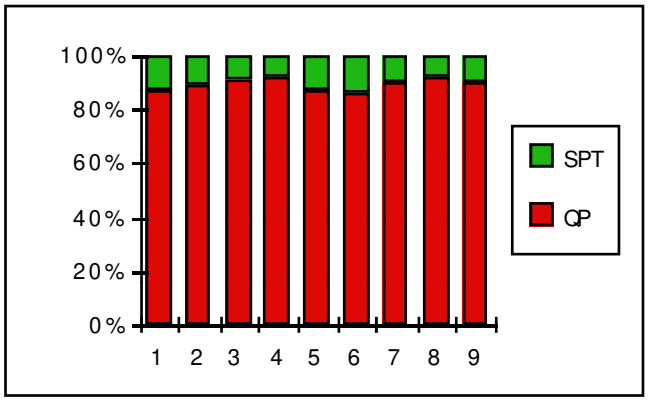

Chart 25: time spent in the (LRP_y') and in the ($\Pi_{\beta t}'$) solution

F		M		N	
Tot	*SPT*	*Tot*	*SPT*	*Tot*	*SPT*
6.44	0.96	15.17	1.32	20.96	1.30
27.55	3.26	46.99	2.63	58.38	2.88
27.89	2.88	28.31	1.45	33.13	1.52
70.21	6.27	104.38	4.30	116.20	4.35
57.03	8.76	57.53	5.56	60.97	5.26
10.73	1.74	14.04	1.28	26.34	1.83
427.58	46.45	915.54	31.07	1317.25	34.70
218.77	19.58	639.50	20.05	934.49	19.98
641.59	67.76	1532.74	61.01	1869.29	59.52

Table 34: effect of the different LVG strategies

N		S		H	
iter	*time*	*iter*	*time*	*iter*	*time*
1000	38	294	22	471	28
1000	83	457	36	443	25
1000	34	311	21	595	26
1000	90	761	77	796	51
1000	30	434	24	950	47
1000	23	270	15	480	16
2000	1230	520	412	660	428
861	615	257	314	355	254
2000	774	443	352	788	681

Table 35: effect of the different *t*-strategies

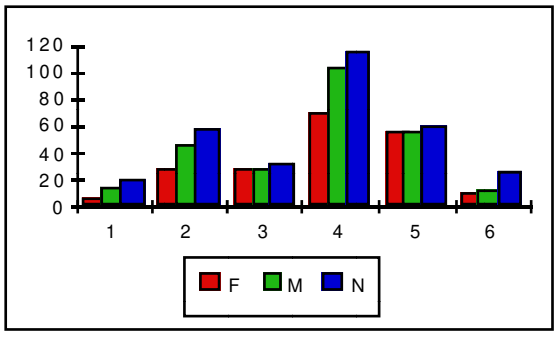

Chart 26: the LVG strategies, small instances

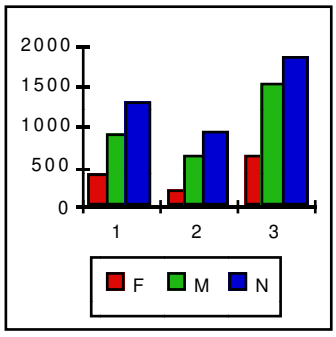

Chart 27: the LVG strategies, large instances

Finally, the effect of the Long-Term *t*-strategies is reported in Table 35 and the corresponding Chart 28 and Chart 29: on all but one problem, BRP using the Heuristic *t*-strategies alone (*N*) was not able to converge before the maximum number of iterations (1000 for the small problems, 2000 for the large ones), while the Hard (*H*) and Soft (*S*) Long-Term *t*-strategies both ensured much faster termination. The m_2 parameter (Cf. §II.2.1) has been set to the "normal" value of .9, allowing *t* to decrease all the way long the execution of the algorithm: an interesting picture of the effect of the *t*-strategies has already been given in Chart 7 and Chart 8 of §I.5.3, to which the reader is referred to for a more detailed discussion.

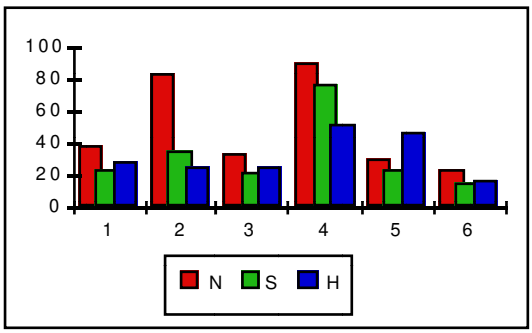

Chart 28: the *t*-strategies, small instances

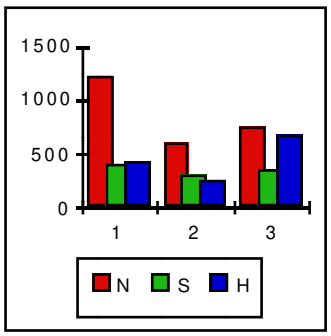

Chart 29: the *t*-strategies, large instances

In conclusion, the Bundle-type approach for the solution of (DRP) has shown to be superior to the "standard" Subgradient approach, especially in term of iterations count: due to the high cost of the ($\Pi_{\beta t}'$) solution, however, a number of implementation details in the specialized (QP) solver, the Lagrangean Variables Generation strategy and the *t*-strategies have to be carefully chosen in order to obtain significantly better running times. These results show that properly implemented Bundle methods can be a valuable mean for the approximate solution of Lagrangean Duals of large-scale structured integer programs.

II.5.2 The Fixed Charge Multicommodity Flow problem

The Fixed Charge Multicommodity (Min-Cost) Flow problem has several applications in transportation models[@]: (FCMF) is a generalization of (MMCF), belonging to the broad class of Network Design problems, in which the arcs must be constructed, paying a fixed cost, before the flow of any commodity can be routed along them. A standard (MIP) formulation is

$$
\text{(FCMF)} \quad \begin{cases}
\min \sum_{h \in \mathcal{K}} \sum_{(i,j) \in \mathcal{A}} c_{ij}^h x_{ij}^h + \sum_{(i,j) \in \mathcal{A}} f_{ij} z_{ij} & \\
\sum_{j:(i,j) \in \mathcal{A}} x_{ij}^h - \sum_{j:(j,i) \in \mathcal{A}} x_{ji}^h = b_i^h & \forall\ i \in \mathcal{N},\ \forall\ h \in \mathcal{K} \\
0 \le x_{ij}^h \le u_{ij}^h z_{ij} & \forall\ (i,j) \in \mathcal{A},\ \forall\ h \in \mathcal{K} \\
\sum_{h \in \mathcal{K}} x_{ij}^h \le u_{ij} z_{ij} & \forall\ (i,j) \in \mathcal{A} \\
z \in \{0,1\}^m,\ \boldsymbol{x}^h \in \mathcal{R}^m & \forall\ h \in \mathcal{K}
\end{cases}
$$

even though other constraints may be imposed on the *m design variables* z: here the flows are allowed to take fractional values, even though the Integer (FCMF) is sometimes also studied. For simplicity, only the version generalizing (MMCF) will be dealt with in this paragraph, but analogous results can be obtained e.g. for the Generalized Fixed Charge Multicommodity Flow: in fact, the Reserve Problem of §II.5.1 is a special case of the Fixed Charge Nonsimultaneous (MMCF). Another formulation can be obtained by replacing the $0 \le x_{ij}^h \le u_{ij}^h z_{ij}$ constraints with simple bounds $0 \le x_{ij}^h \le u_{ij}^h$: although equivalent as a MIP, this formulation - that will be denoted by (FCMF´) -

[@] This line of research have been carried along in collaboration with Bernard Gendron and Teodor Crainic of the Centre de Recherche sur les Transports of the Université de Montréal.

has a potentially much weaker continuous relaxation. In fact, since the design costs f_{ij} can be assumed to be strictly positive ($f_{ij} \leq 0 \Rightarrow z_{ij} = 1$ in some optimal solution), in any optimal solution of (FCMF´) z_{ij} must be equal to $(u_{ij} - \Sigma_{h \in \mathcal{K}} x^h_{ij}) / u_{ij}$: hence, (FCMF´) is equivalent to

$$\text{(FCMF˝)} \quad \min\{ \Sigma_{h \in \mathcal{K}} \hat{c}^h \boldsymbol{x}^h : \boldsymbol{E}\boldsymbol{x}^h = \boldsymbol{b}^h \;\; \boldsymbol{0} \leq \boldsymbol{x}^h \leq \boldsymbol{u}^h \; \forall \; h \in \mathcal{K}, \Sigma_{h \in \mathcal{K}} \boldsymbol{x}^h \leq \boldsymbol{u} \}$$

$$\text{where} \qquad \hat{c}^h_{ij} = c^h_{ij} + (f_{ij} / u_{ij})$$

that is the (MMCF) obtained by constructing all the arcs but "spreading" the construction cost on the flow cost. By "playing" with these alternative formulations, a number of different Lagrangean relaxations can be proposed for (FCMF) [GC95]: the two more natural ones are

$$\text{(RFC}_y\text{)} \qquad \Sigma_{h \in \mathcal{K}} \min\{ (\boldsymbol{c}^h + \boldsymbol{y} + \boldsymbol{y}^h) \boldsymbol{x}^h : \boldsymbol{E}\boldsymbol{x}^h = \boldsymbol{b}^h , \boldsymbol{0} \leq \boldsymbol{x}^h \leq \boldsymbol{u}^h \}$$

$$+ \; \Sigma_{(i,j) \in \mathcal{A}} \min\{ (f_{ij} - y_{ij} u_{ij} - \Sigma_{h \in \mathcal{K}} y^h_{ij} u^h_{ij}) z_{ij} : z_{ij} \in \{ 0 , 1 \} \}$$

$$\text{(RFC}_y\text{´)} \qquad \Sigma_{h \in \mathcal{K}} \min\{ (\boldsymbol{c}^h + \boldsymbol{y}) \boldsymbol{x}^h : \boldsymbol{E}\boldsymbol{x}^h = \boldsymbol{b}^h , \boldsymbol{0} \leq \boldsymbol{x}^h \leq \boldsymbol{u}^h \}$$

$$+ \; \Sigma_{(i,j) \in \mathcal{A}} \min\{ (f_{ij} - y_{ij} u_{ij}) z_{ij} : z_{ij} \in \{ 0 , 1 \} \}$$

corresponding to (FCMF) and (FCMF´) respectively where all the constraints linking the flow and design variables have been dualized; the Lagrangean Dual (DFCMF´) corresponding to (RFC$_y$´) has only m dual multipliers $\boldsymbol{y}$, while (DFCMF) has $m(k + 1)$ multipliers $[\boldsymbol{y} , \boldsymbol{y}^1 , .. , \boldsymbol{y}^k]$.
Another possibility is

$$\text{(PDMF}_\Pi\text{)} \quad \Sigma_{(i,j) \in \mathcal{A}} \left\{ \begin{array}{l} \min \; \Sigma_{h \in \mathcal{K}} \bar{c}^h_{ij}(\pi^h) x^h_{ij} + f_{ij} z_{ij} \\ 0 \leq x^h_{ij} \leq u^h_{ij} \qquad \forall \; h \in \mathcal{K} \\ \Sigma_{h \in \mathcal{K}} x^h_{ij} \leq u_{ij} z_{ij} , \; z_{ij} \in \{ 0 , 1 \} \end{array} \right\} + \Sigma_{h \in \mathcal{K}} \pi^h \boldsymbol{b}^h$$

that decomposes into m independent Knapsack problems: in fact, the contribution of an arc is 0 if $z_{ij} = 0$ and f_{ij} plus the optimal solution of the ij-th Knapsack if $z_{ij} = 1$, so that it is enough to choose the smaller between the two. Note that the first two relaxations give the same bound for the Continuous and the Integer versions of (FCMF), as the (MCF) subproblems have the Integrality Property, while this is not true for (PDFCMF) using the Knapsacks relaxation (PDMF$_\Pi$): hence, better bounds can in principle be obtained by the latter, at the cost of solving m $\mathcal{NP}$-hard Integer Knapsack subproblems (for which relatively efficient techniques exist) at each step.

Having the MMCFB code described in §II.2.1, implementing a code for the solution of both (DFCMF) and (DFCMF´) was really straightforward: it was only necessary to develop a specialized version of the `Graph` class, the derived `FCGraph` class, for handling the easy subproblems concerning the design variables $\boldsymbol{z}$. Hence, all the observations about the implementation issues of MMCFB are valid here as well and will not be repeated: as the unique remark, the LVG strategy is extremely important for problems with many variables like (DFCMF) and (RP) (Cf. §II.5.1), in fact the results reported in §I.3.7 about the advantages of an Active Set strategy for a Bundle solver have been obtained in the present context.

However, it became rapidly clear that the Bundle method applied to the solution of (DFCMF)

(hereafter named BFC) was not competitive with the Subgradient method of [GC95] (SGFC): this is shown in Table 36 for a small subset of large instances, where *v* is the number of Lagrangean multipliers.

As it happened for (RP), the time spent in the calculation of φ() (*Fi*) is only a small percentage of the total computing time (*Tot*): in this case, however, BFC was not able to converge before the 500 iterations limit that was assigned to both the methods (*Iter*), thus requiring a much larger running time than SGFC because of the cost of the "coordination" phase. Therefore, it was necessary to construct a Subgradient-based (DFCMF) solver for performing more accurate comparisons, and eventually providing the more efficient algorithm to the procedures that require the solution of the Lagrangean Dual [CG95]: this was easy, however, thanks to the internal structure of the `MMCFBundle` class (Cf. §II.2.1), that is derived from the *abstract* general-purpose `Bundle` class. The `Bundle` class implements all the NDO-specific operations, such as the β-strategy, the *t*-strategy and the interface with the `[B]MinQuad` class, leaving the computation of φ() and the subgradient to the *pure virtual* method `FiAndGi()` to be instantiated in the derived classes, such as `MMCFBundle`: hence, it was only necessary to construct an alternative `SubGrad` class with the same structure, and derive `MMCFBundle` from it instead (Figure 22).

		BFC		SGFC	
v	*Iter*	*Tot*	*Fi*	*Tot*	*Fi*
40400	501	2066	261	249	220
40400	501	1041	245	243	215
40400	501	1232	258	244	215
40400	501	927	235	241	213
63125	501	2410	457	399	354
63125	501	2538	432	379	334

Table 36: comparison of Bundle and Subgradient for solving (DFCMF) on some large instances

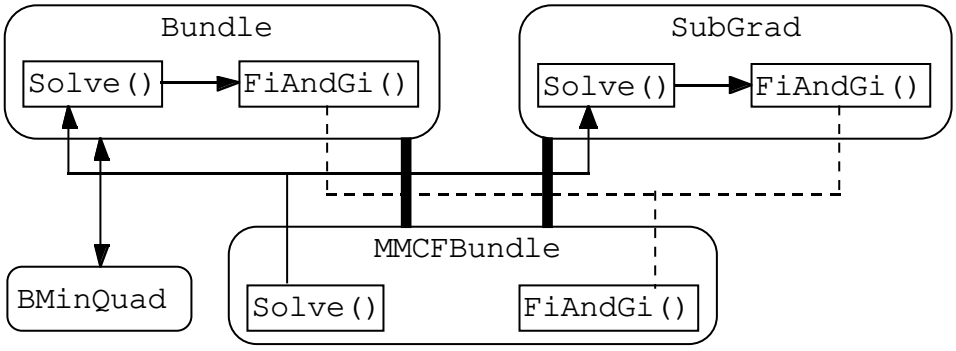

Figure 22: the internal structure of `MMCFBundle` and its relations with `Bundle` and `SubGrad`

Which NDO solver is used is decided by a compile-time switch, but it is also possible to derive `MMCFBundle` from *both*, so that "mixed" strategies could be used: one possible way could be to leave the "few" Lagrangean multipliers $\boldsymbol{y}$ under the control of the Bundle algorithm, while the "many" [$\boldsymbol{y}^1$ .. $\boldsymbol{y}^k$] are handled by the "quick and dirty" Subgradient.

The implementation of the `SubGrad` class allows the choice among the several different strategies that have been proposed over the years [HWC74] [CFM75] [Cr76] [LPS95]:

– the formula for calculating the direction at the *i*-th step is

$$\boldsymbol{d}_i = \boldsymbol{g}_i + \tau_i \boldsymbol{d}_{i-1}$$

and the "smoothing" parameter τ_i can be either fixed [Cr76] (if $\tau_i = 0$ the previous direction is simply disregarded [HWC74]) or set according to (one of) the heuristic rule(s)

$$\tau_i = \begin{cases} 0 & \text{if } \boldsymbol{g}_i \boldsymbol{d}_{i-1} \geq m_3 \| \boldsymbol{d}_{i-1} \|^2 \\ \dfrac{-m_2(\boldsymbol{g}_i \boldsymbol{d}_{i-1})}{\| \boldsymbol{d}_{i-1} \|^2} \quad or \quad \dfrac{\| \boldsymbol{g}_i \|}{\| \boldsymbol{d}_{i-1} \|} & \text{otherwise} \end{cases}$$

where m_2 and m_3 are fixed parameters in (0, 1] [CFM75]: for the current data sets, $\tau_i = .9$ was found to be the better choice;

- once $\boldsymbol{d}_i$ has been chosen, the generic formula for the step to be taken along it is

$$t_i = \lambda \frac{\min\{ \bar{t}\bar{\varphi}_i , \bar{\varphi}^* \} - \varphi(\boldsymbol{y}_i)}{\| \boldsymbol{g}_i \|^2}$$

where $\bar{\varphi}_i$ is the best value of $\varphi()$ found up to iteration i, $\bar{\varphi}^*$ (if available, which is *not* true in this case) is an upper bound to the optimum value of $\varphi()$, $\bar{t}$ is some fixed number > 1 and λ is a scalar that is initialised with some $\lambda_0 > 0$ and then multiplied for some $\underline{t} < 0$ once every p iterations (but never decreases under a fixed $\underline{\lambda}$); such a formula offers a large flexibility, i.e. many parameters that can be "hand tuned" for enhancing the performances, and for the current data sets the best setting ever found is $p = 25$, $\bar{t} = 2$, $\underline{t} = .5$, $\lambda_0 = 2$ and $\underline{\lambda} = 10^{-3}$;

- the algorithm is stopped if $\| \boldsymbol{g}_i \|^2 \leq \varepsilon\bar{\varphi}_i$, after q consecutive "nonincreasing" steps (where $\varphi(\boldsymbol{y}_i) - \varphi(\boldsymbol{y}_{i-1}) \leq m_1 \varphi(\boldsymbol{y}_{i-1})$) or after r steps if nothing else succeeds: for most of the problems, no setting of ε, m_1 and q could avoid the code to just perform all the $r = 500$ steps;

- the $\boldsymbol{y} \geq \boldsymbol{0}$ constraints are faced by means of projection, and the code offers the possibility of either projecting $\boldsymbol{g}_i$ *before* that the above formulas are applied [LPS95], or projecting the new current point $\boldsymbol{y}_{i+1} = \boldsymbol{y}_i + t_i \boldsymbol{d}_i$ where $\boldsymbol{d}_i$ has been obtained with the non-projected gradient (this could be needed anyway, if $\tau_i \neq 0$): for the current data sets, the second choice has proved to be largely superior with any choice of the direction/step formulae.

Although very easy to implement, Subgradient methods usually require a careful choice among the above update formulae and the relative (many) algorithmic parameters: this allows for some "human expertise" to be exploited, but requires a possibly long tuning phase.

As a first step, (FCMF) and (FCMF´) have been solved with Cplex in order to obtain z(DFCMF) and z(DFCMF´) for the 96 (FCMF) described in Sec. II.2.3.1 (see also Table 8 in Sec. II.2.4.1): the resulting (LP)s are very large, and the solution times - shown in Table 37 (*Tot*) in the Cplex and Cplex´ columns - were enormous, up to 35 hours on the largest instances; furthermore, the largest (FCMF) instances in all the three groups could not be solved because the workstation went trashing. Interestingly, the network presolver of Cplex is uneffective on these problems, and the IP algorithm is neatly superior to both the Simplex variants: unfortunately, it requires much more memory, making it impossible to solve all but the smaller instances, but this could indicate that specialized IP methods could be more competitive for (FCMF) than they are for (MMCF).
The bounds given by (FCMF´) are much worse than those given by (FCMF) on groups A and B, as shown in column *Gap* where the relative gap between the two is reported: for the instances where (FCMF) was not solved, the best bound obtained with all the other methods was used for comparison. On the contrary, the difference is minimal for group C: this is due to the fact that the random generator is not very good at determining "tight" individual capacities u_{ij}^h for nonbipartite problems with multiple sources and sinks for each commodity, while such capacities are obvious for

bipartite or single-source graphs; in fact, the lower bounds provided by (DFCMF) on z(FCMF) seem [CG95] to be consistently weaker for group C than for both groups A and B. These results clearly show the importance of developing alternative approaches: the most obvious ones are a Bundle or a Subgradient algorithm applied to the solution of (DFCMF) or (DFCMF´), whose results are again reported in Table 37.

	Cplex	Cplex´		SGFC		SGFC´		BFC´	
	Tot	*Tot*	*Gap*	*Tot*	*Gap*	*Tot*	*Gap*	*Tot*	*Gap*
A	107	19	8.19%	22	.07%	17	.11%	23	.01%
	335	59	9.97%	35	.11%	27	.11%	34	.01%
	2252	485	7.46%	112	.08%	84	.11%	88	.01%
	9093	1004	8.58%	175	.13%	130	.12%	142	.02%
	32275	2895	24.16%	244	.80%	143	.11%	123	.02%
	107193	9149	11.70%	379	.61%	247	.12%	219	.03%
	* *	36972	48.62%	1235	1.79%	622	.15%	616	.27%
	* *	128397	19.01%	1896	.67%	1098	.13%	1033	.20%
B	23	2	13.07%	17	.01%	8	.11%	4	.00%
	8	2	15.46%	21	.00%	10	.11%	9	.00%
	15282	805	40.93%	101	1.41%	42	.12%	40	.00%
	4386	504	34.84%	107	.83%	44	.12%	49	.00%
	11001	1187	33.80%	124	1.13%	53	.12%	47	.00%
	2183	249	34.51%	136	.82%	55	.11%	55	.01%
	* *	10923	33.23%	450	.92%	177	.12%	180	.02%
	* *	16754	42.72%	573	1.45%	231	.13%	227	.03%
C	87	23	.10%	37	.06%	30	.12%	34	.02%
	133	32	.22%	45	.07%	36	.12%	40	.04%
	4864	900	.11%	194	.06%	152	.12%	152	.01%
	4678	1082	.02%	223	.10%	176	.12%	176	.02%
	9117	1450	.20%	234	.08%	179	.12%	177	.02%
	7516	1575	.05%	279	.21%	217	.13%	216	.04%
	* *	16898	.00%	910	.01%	708	.12%	677	.02%
	* *	29303	.00%	1137	.06%	898	.12%	850	.02%

Table 37: comparison of methods for the solution of (FCMF), (FCMF´), (DFCMF) and (DFCMF´)

With an iteration limit set to 500, none of the Lagrangean-based codes was able to solve the problems to optimality (but BFC´ on the first two sets of group B), but they were always able to quickly approximate z(DFCMF) and z(DFCMF´) to a reasonable accuracy: BFC´ obtained a much smaller *relative Gap* than SGFC´ (w.r.t. to z(DFCMF´)) with the same number of iterations and in almost the same running time (*Tot*). However, a much better use of the LP solver was possible, because the pre-processor of Cplex is apparently unable to substitute for the z variables in (FCMF´): in fact, by solving (FCMF˝) instead, enormous savings could be obtained - up to a factor of 150 - as shown in the Cplex˝ column of Table 38. Still, the largest instances of (FCMF˝) require more than 3 hours each to be solved, hence the MMCFB code of Chapter II.2 was also used for the purpose: the table shows that applying the Bundle method to (FCMF˝) (BFC˝) is much better than solving (DFCMF´) with the same tool. This may seem counterintuitive, since the two Lagrangean subproblems have a very close structure: however, while in the first the z variables are "automatically" set to their correct value when the optimal $\boldsymbol{x}$ is found, in the second both the optimal $\boldsymbol{x}$ *and* z have to be obtained as a convex combination of those generated as solutions of the subproblems. Hence, a Subgradient-based variant

of the code (SGFC˝) was also arranged (with the same technique of substituting the `SubGrad` class to the `Bundle` class) and (FCMF˝) was solved by means of both: while neither the Bundle nor the Subgradient are able to solve (DFCMF´) in less than 500 iterations, this is done - and sometimes in very few iterations - for (FCMF˝). Here, the Bundle is far superior to the Subgradient, because it always stops in (up to 30 times) less iterations due to its much more effective stopping rules; furthermore, tuning the parameters such as the Subgradient "exactly" solves (FCMF˝) and stops quickly enough is not easy, since the only parameter that can be used to this purpose is the number of consecutive "bad" steps ($q = 5$).

	Cplex´´	SGFC´´		BFC´´	
	Tot	*Iter*	*Tot*	*Iter*	*Tot*
A	10.21	153	6.32	153	3.68
	20.11	109	7.05	109	4.21
	122.55	227	28.05	227	23.06
	183.68	107	24.82	107	14.92
	152.87	23	9.92	23	2.04
	2131.09	50	33.38	50	9.86
	1322.76	9	15.47	9	5.11
	867.95	19	43.51	19	11.20
B	3.57	5	1.90	5	0.05
	4.09	7	2.64	7	0.07
	80.94	7	5.40	7	0.25
	83.07	11	7.00	11	0.35
	119.09	6	9.26	6	0.28
	136.12	7	14.20	7	0.33
	2541.58	16	34.18	16	1.86
	689.90	6	30.46	6	1.21
C	9.75	11	1.25	11	0.14
	9.01	7	1.13	7	0.13
	420.29	14	5.90	14	0.89
	705.48	22	8.37	22	1.26
	409.32	8	3.65	8	0.72
	959.42	19	7.60	19	1.39
	12389.32	9	13.96	9	3.17
	11123.80	17	25.01	17	6.22

Table 38: comparison of methods for solving (FCMF˝)

Hence, applying a Bundle algorithm to (FCMF˝) is a very effective method for rapidly obtaining lower bounds on instances, such as those of group C, where the individual capacities u_{ij}^h are "loose": however, on other instances these bounds are very weak, even if compared with z(FCMF). Hence, we also experimented wether the solution of (PDFCMF), by a Bundle-based code (KB) or a Subgradient-based code (KS) was a feasible alternative: with the `Bundle`, `SubGrad` and `Graph` classes at hand, the implementation was relatively easy, essentially requiring only an efficient sorting routine (a specialized *QuickSort* without recoursive calls) for solving the Continuous Knapsack subproblems. Conversely, making the Subgradient to converge was not at all easy, as already reported in [CG95]: it was finally possible to make (almost) all the instances to converge only by "warm-starting" the algorithm, i.e. using the optimal potentials of the k separate ($RMF_{\mathbf{0}}^h$) rather than $\Pi = [\ \mathbf{0}\ ..\ \mathbf{0}\ ]$ as a starting point. This was neither necessary nor useful for the Bundle, demonstrating its inherent superior stability: from the results reported in Table 39 it is also easy to see that, in the same number of iterations (500 or 1000), KB attains much better bounds than KS. However, the running time of Bundle is up to 3 times that of the Subgradient, due extra cost of the ($\Pi_{\beta t}$) solution: examining the results, it is possible to note that

- the ratio between the running times decreases as the size increases, being 1.5 - 2 on the largest instances;
- increasing the number of iterations from 500 to 1000 only marginally reduces the gap of KS on most instances, while it always at least halves it for KB: for most instances, especially of group A, KB reaches in 500 iterations a much better bound than that obtained by KS in 1000 iterations (and comparable running time);
- for a small minority of instances, KS diverges and it is not capable of improving the (weak) bound

given by the initialization: these are three instances in group A, corresponding to the three sets with much larger Gap than the others (all the figures are averaged within a set of four instances, the other three ones in the set having about .1% gap).

	KS - 500			KB - 500			KS - 1000		KB - 1000	
	Tot	*Fi*	*Gap*	*Tot*	*Fi*	*Gap*	*Tot*	*Gap*	*Tot*	*Gap*
	3.05	2.86	.11%	9.72	3.18	.02%	6.04	.10%	29.20	.01%
	4.78	4.58	.12%	13.47	4.79	.01%	9.49	.11%	53.03	.00%
	9.52	9.14	5.76%	23.96	12.17	.03%	16.88	5.75%	63.76	.01%
A	18.83	18.27	.12%	32.71	18.14	.02%	37.38	.11%	85.61	.01%
	33.56	31.64	7.22%	108.00	42.26	.03%	66.35	7.21%	322.49	.01%
	65.33	62.74	.14%	138.01	63.74	.05%	130.19	.13%	382.90	.01%
	127.94	123.42	7.48%	322.53	164.02	.03%	252.94	7.46%	863.20	.00%
	242.54	236.79	.13%	420.87	249.32	.04%	431.82	.12%	1083.76	.00%
	7.51	7.29	.05%	6.36	4.20	.00%	14.97	.05%	6.34	.00%
	9.30	9.09	.06%	4.41	3.44	.00%	18.52	.06%	4.41	.00%
	50.83	49.59	.24%	94.49	50.45	.17%	101.03	.19%	242.10	.07%
B	50.22	49.11	.18%	76.62	49.45	.16%	100.10	.16%	182.03	.07%
	58.39	57.18	.21%	107.04	62.73	.15%	105.20	.18%	264.86	.06%
	64.64	63.61	.13%	86.96	63.98	.09%	129.02	.10%	194.59	.03%
	265.83	260.08	.14%	433.36	269.26	.12%	529.48	.09%	1063.74	.00%
	345.67	339.72	.18%	506.23	341.27	.14%	683.74	.12%	1226.36	.00%
	7.64	7.39	.56%	19.10	7.75	.34%	14.78	.45%	55.59	.10%
	9.42	9.14	.53%	21.44	9.54	.36%	18.72	.43%	59.84	.16%
	50.39	49.12	1.00%	103.62	50.23	.73%	100.34	.80%	292.71	.27%
C	50.35	49.14	1.07%	90.53	49.90	.89%	100.25	.87%	251.98	.31%
	62.82	61.35	.92%	113.39	62.70	.69%	125.04	.73%	293.75	.32%
	64.44	63.22	1.11%	104.51	64.40	.90%	128.46	.90%	272.15	.35%
	255.62	249.10	1.59%	448.33	258.07	1.41%	509.46	1.27%	1155.95	.64%
	337.64	330.91	1.53%	537.79	336.82	1.32%	673.17	1.24%	1350.13	.56%

Table 39: comparison of methods for the solution of (PDFMCF)

There are noticeable differences between the groups, since group A is “easier” than group B which is in turn “easier” than group C: for instance, 500 iterations of KB are always capable of approximating z(FCMF) within .05% on group A, a much better result than SGFC, while the gap can be as large as 1.4% on group C (where, conversely, BFC″ is extremely efficient).

Summing up, Lagrangean-based techniques offer several different ways for obtaining lower bounds to z(FCMF): according to the characteristics of the instances at hand, and on the desired quality of the bound, the best methods seem to be a Bundle approach to (FCMF″) or a Bundle/Subgradient approach to (PDFMCF). These results are preliminary, since many other issues still have to be addressed: one is the influence of the different algorithms on the Lagrangean-based heuristics that will be constructed upon them, and some initial results seem to show that the better multipliers produced by a Bundle method have a surprisingly positive effect. Furthermore, a more accurate tuning of the parameters or improvements in the Bundle technology - such as the “disaggregated” and/or (LP)-based Bundle algorithms described in Part I - could provide better results.

This work has clearly shown that the potential weakness of Bundle methods - the need for $(\Delta_{\beta t})$ solution - can actually be an issue for problems with many Lagrangean variables, but is has also

confirmed that Bundle methods are neatly superior to Subgradient methods in terms of speed of convergence; possibly more important, the ability of evaluating the obtained precision is crucial to avoid performing many unnecessary steps, and it can lead to relevant performances improvements especially on "easy" instances. Several directions are open for future research:

- implementation and test of Lagrangean-based heuristics, that provide (hopefully) good solutions of (FCMF) exploiting the information provided by the Lagrangean multipliers, the solutions of the Lagrangean subproblems or the "aggregate" solution, in case of a Bundle: the corresponding upper bounds on z(FCMF) could also be useful for improving the behaviour of the NDO algorithms;
- implementation of the "mixed" approach sketched above, where the Bundle handles the $\boldsymbol{y}$ variables alone: the resulting algorithm can be viewed as an approach to the Lagrangean Dual of (FCMC) w.r.t. the $0 \le x_{ij}^h \le u_{ij}^h z_{ij}$ constraints only, that have a Lagrangean subproblem of the (FCMF´) type which is efficiently solved by a Bundle approach;
- parallelization of some of the approaches on loosely coupled, low-cost parallel architectures such as clusters of workstations;
- development of alternative relaxations and approaches, that be even more suited for solution by means of Lagrangean techniques.