NONLINEAR MULTICOMMODITY NETWORK FLOWS THROUGH PRIMAL PARTITIONING AND COMPARISON WITH ALTERNATIVE METHODS

Jordi Castro & Narcís Nabona
Statistics and Operations Research Dept., Universitat Politècnica de Catalunya
c. Pau Gargallo 5, 08071 Barcelona
e-mail: jcastrop@eio.upc.es

Abstract : *This paper presents a specialized code for solving the linear and nonlinear multicommodity network flow problem with linear side constraints using primal partitioning techniques. The computational performance of the program developed is reported and compared with that of another specialized code for the same problem —based on price-directive decomposition— and also compared to the performance of a general purpose nonlinear optimization package. Test problems of many sizes corresponding to real cases of nonlinear multicommodity network flows have been employed.*

Keywords: Linear and Nonlinear Network Flows, Multicommodity Network Flows, Network Simplex Methods, Nonlinear Optimization, Side Constraints.

1. Introduction.

Primal partitioning is described and used in [5] to solve the linear multicommodity network flow problem(LMP). An specialization of the simplex method using primal partitioning has been developed for linear problems. Furthermore, this methodology has been extended to minimize a continuous nonlinear objective function (1) and to include linear side constraints (5). The formulation studied considers a mutual capacity constraint at each arc of the network and a certain number of side constraints. The nonlinear multicommodity problem (NMP) can be formulated as:

$$\min_{X_1,X_2,\ldots,X_K} \quad h(X_1, X_2, \ldots, X_K) \tag{1}$$

$$\text{subj. to} \quad AX_k = R_k \qquad k = 1, \ldots, K \tag{2}$$

$$\underline{0} \leq X_k \leq \overline{X}_k \qquad k = 1, \ldots, K \tag{3}$$

$$\sum_{k=1}^{K} X_k \leq T \tag{4}$$

$$L \leq \sum_{k=1}^{K} L_k X_k \leq U \tag{5}$$

where $X^k \in \mathbb{R}^m$, (m: number of arcs) is the vector of flows of commodity k ($k = 1, \ldots, K$), and h being a $\mathbb{R}^{K \times m} \rightarrow \mathbb{R}^1$ real valued function. $A \in \mathbb{R}^{n \times m}$ (n: number of nodes) is a network matrix. Constraints (3) are simple bounds with $\overline{X}^k \in \mathbb{R}^m, k = 1, \ldots, K$ being upper limits and constraints (4) are the mutual capacity constraints with

$T \in \mathbb{R}^m$ and (5) are the linear side-constraints defined by matrices L^k: $L^k \in \mathbb{R}^{p \times m}, k = 1, \ldots, K$ with elements of any type, and $L, U \in \mathbb{R}^p$ (p: number of side constraints).

Every basis using the primal partitioning method can be written as:

$$B = \begin{array}{|c|c|c|} \hline L_1 & R_1 & 0 \\ \hline L_2 & R_2 & 0 \\ \hline L_3 & R_3 & \mathbb{1} \\ \hline \end{array}$$

being L_1, R_2 and $\mathbb{1}$ square matrices, and where: L_1 refers to the network constraints and arcs of the K spanning trees. The topology of this matrix is:

$$L_1 = \begin{pmatrix} B_1 & & & \\ & B_2 & & \\ & & \ddots & \\ & & & B_K \end{pmatrix}$$

being each B_k a nonsingular matrix associated with the kth spanning tree. L_1 can be represented at every iteration by K spanning trees following the methodology described in [2]. R_1 refers to the network constraints and complementary arcs of the K commodities. Complementary arcs are required to preserve the nonsingularity of the basis. L_2 refers to saturated mutual capacity and side constraints, for the arcs of the spanning trees. R_2 refers to saturated mutual capacity and side constraints, for the complementary arcs. L_3 refers to unsaturated mutual capacity and side constraints, for the arcs of the spanning trees. R_3 refers to unsaturated mutual capacity and side constraints, for the complementary arcs. $\mathbb{1}$ refers to the slacks of the unsaturated mutual capacity and side constraints. (It has to be noticed that constraints whose slacks are in matrix $\mathbb{1}$ are treated as unsaturated constraints, even though the values of slacks are zero).

2. Influence of the side constraints.

To perform the required operations with the basis B (compute the Lagrange multipliers $\pi^t B = c^t$, determine the descent direction of basic variables $Bp_B = b$) it is only necessary to store and use a nonsingular working matrix, that we will denote by Q. The expression of this matrix is $Q = R_2 - L_2 L_1^{-1} R_1$. This result can be obtained directly solving the previous systems —considering the partition of the basis exposed before— and doing some algebraic manipulations (a detailed explanation can be found in [5]). The fact of inverting Q instead of the whole basis B improves substantially the efficiency of the method.

If we denote the set of saturated mutual capacity constraints by S_{mc}, the set of saturated side constraints by S_{sc} and the number of elements of a set S by $|S|$ then the dimension of the matrix Q ($dim(Q)$) can be expressed as $dim(Q) = |S_{mc}| + |S_{sc}|$. Since Q has full rank it follows than the number of complementary arcs in the basis must be equal to $|S_{mc}| + |S_{sc}|$. We can consider this matrix divided into two submatrices

$Q = \begin{bmatrix} Q_{mc} \\ Q_{sc} \end{bmatrix}$ where Q_{mc} is the submatrix whose rows refer to constraints $\in S_{mc}$, and Q_{sc} is the submatrix whose rows are associated with constraints $\in S_{sc}$.

The expression for computing Q involves the calculation of $L_1^{-1}R_1$. Since L_1 is a block diagonal matrix where the kth block is a minimum spanning tree for the kth commodity, and R_1 expresses for each complementary arc of the kth commodity its connection to the kth minimum spanning tree, then solving $L_1^{-1}R_1$ is equivalent to having the paths (denoted by $P_j, j = 1 \dots dim(Q)$) of complementary arcs in their associated spanning trees. Given an arc $a \in P_j$, we will say than a has normal orientation if it points to the origin node of the complementary arc j; otherwise, it has reverse orientation.

If we denote by:

- a_j the arc associated with the jth column of Q, $j = 1 \dots dim(Q)$.
- mc_i the mutual capacity constraint of the ith row of $Q, i = 1 \dots |S_{mc}|$ (this capacity constraint refers to the arc mc_i).
- sc_i the side constraint of the ith row of $Q, i = |S_{mc}| + 1 \dots dim(Q)$.
- $B(a, n)$ a logical function than returns $true$ if the arc a appears in the side constraint n, and $false$ otherwise.
- $c_{a,n}$ the coefficient of the arc a in the side constraint n.

Then we can compute directly the matrix Q as follows:

Submatrix Q_{mc}:

$$\underset{\substack{i=1\dots|S_{mc}| \\ j=1\dots dim(Q)}}{Q_{ij}} = \begin{cases} +1, & \text{if } a_j = mc_i \\ +1, & \text{if } mc_i \in P_j \text{ with normal orientation} \\ -1, & \text{if } mc_i \in P_j \text{ with reverse orientation} \\ 0, & \text{otherwise} \end{cases}$$

Submatrix Q_{sc}:

$$\underset{\substack{i=|S_{mc}|+1\dots dim(Q) \\ j=1\dots dim(Q)}}{Q_{ij}} = \begin{cases} & \text{Following next 4 steps:} \\ 1) & \text{Set } Q_{ij} = 0 \\ 2) & \text{if } B(a_j, sc_i) \text{ then } Q_{ij} = c_{a_j,sc_i} \\ & \text{for each } a \in P_j, \text{ do next 2 steps} \\ 3) & \text{if } B(a, sc_i) \text{ and } a \text{ has normal orientation then} \\ & \qquad Q_{ij} = Q_{ij} + c_{a,sc_i} \\ 4) & \text{if } B(a, sc_i) \text{ and } a \text{ has reverse orientation then} \\ & \qquad Q_{ij} = Q_{ij} - c_{a,sc_i} \end{cases}$$

It is clear from this procedure than the information of the mutual capacity constraints is not stored and is implicitly assumed in the construction of the Q_{mc} submatrix. However when computing submatrix Q_{sc} a function such as the logical function $B(a, sc_i)$ employed before, is required. In this implementation, information about the side constraints is stored in sparse form by columns (that is, for each arc we have the side constraints where it appears), and sorted by the number of side constraint. Thus the boolean function $B(a, sc_i)$ is reduced to a binary search.

3. The Algorithm

The algorithm presented has three different phases called phase 0, 1 and 2. In phases 0 and 1 the algorithm finds a feasible starting point, while phase 2 tries to achieve the optimizer without leaving the feasible region. Phases 0 and 1 are common to the linear and nonlinear problem. We will describe each phase in next three subsections. When referring to phase 2, only that of the nonlinear problem will be adressed.

For computational purposes the inequality constraints (4) and (5) in the original NMP problem are substituted by equality constraints by adding slacks variables.

$$\sum_{k=1}^{K} X_k + s = T \quad ; \quad \underline{0} \leq s \tag{6}$$

$$\sum_{k=1}^{K} L_k X_k + t = U \quad ; \quad \underline{0} \leq t \leq U - L \tag{7}$$

where $s \in \mathbb{R}^m$ and $t \in \mathbb{R}^p$. In this formulation equations (6) and (7) substitute original equations (4) and (5). Then the formulation of the problem considered by the algorithm (that will be referred to as NMP2) is the minimization of (1) subject to (2), (3), (6) and (7)

3.1. Phase 0.

In phase 0 the algorithm considers only the network constraints and bounds of the varibles of the problem. Without any constraint linking the flows of differents commodities, it solves for each commodity $k, k = 1 \dots K$ the linear network problem

$$\begin{aligned} \min_{X_k} \quad & C_k^t X_k \\ \text{subj. to} \quad & A X_k = R_k \\ & \underline{0} \leq X_k \leq \overline{X}_k \end{aligned}$$

where the vector costs C_k can be introduced by the user. This can be useful to guide the algorithm towards good initial feasible points, if information about the model is known before the execution of the program. Once phase 0 has finished, the algorithm has a minimum spanning tree for each commodity.

3.2. Phase 1.

The K points obtained in phase 0 will not satisfy in general the mutual capacity and side constraints, thus having a pseudo-feasible point. That implies than some slack variables s for the mutual capacity constraints or t for the side constraints will be out of bounds. Let $\hat{X}_k, k = 1 \dots K$ be the pseudo feasible point obtained, then, the following index sets are defined:

- $s^- = \{i : (\sum_{k=1}^{K} \hat{X}_k)_i > T_i \Leftrightarrow s_i < 0\}$.

- $t^- = \{i : (\sum_{k=1}^{K} L_k \hat{X}_k)_i > U_i \Leftrightarrow t_i < 0\}$.
- $t^+ = \{i : (\sum_{k=1}^{K} L_k \hat{X}_k)_i < L_i \Leftrightarrow t_i > (U-L)_i\}$

Introducing new artificial variables e and f, and fixing initial values for s and t such that:

- $(\sum_{k=1}^{K} \hat{X}_k)_i + s_i - e_i = T_i \;\; ; \;\; s_i = 0 \;\; ; \quad \forall i \in s^-$
- $(\sum_{k=1}^{K} L_k \hat{X}_k)_i + t_i - f_i = U_i \;\; ; \;\; t_i = 0 \;\; ; \quad \forall i \in t^-$
- $(\sum_{k=1}^{K} L_k \hat{X}_k)_i + t_i + f_i = U_i \;\; ; \;\; t_i = (U-L)_i \;\; ; \quad \forall i \in t^+$

The problem solved in phase 1 is:

$$\min_{X_1, X_2, \ldots, X_K, s, t, e, f} \quad \sum_{i \in s^-} e_i + \sum_{i \in t^-} f_i + \sum_{i \in t^+} f_i \tag{8}$$

$$\text{subj. to} \quad (1) \quad \text{and} \quad (2)$$

$$\sum_{k=1}^{K} X_k + s + \mathbb{1}^e e = T \tag{9}$$

$$\sum_{k=1}^{K} L_k X_k + t + \mathbb{1}^f f = U \tag{10}$$

$$\underline{0} \leq t \leq U - L \;\; ; \;\; \underline{0} \leq s \;\; ; \;\; \underline{0} \leq e \;\; ; \;\; \underline{0} \leq f$$

Where both matrices $\mathbb{1}^e \in \mathbb{R}^{m \times m}$ and $\mathbb{1}^f \in \mathbb{R}^{p \times p}$ in (9) and (10) are diagonal and defined as follows:

$$(\mathbb{1}^e)_{ii} = \begin{cases} -1 & \text{if } i \in s^- \\ 0 & \text{otherwise} \end{cases} \qquad (\mathbb{1}^f)_{ii} = \begin{cases} -1 & \text{if } i \in t^- \\ +1 & \text{if } i \in t^+ \\ 0 & \text{otherwise} \end{cases}$$

Problem NMP2 will be feasible if, at phase 1, a point where the value of (8) is 0, is achieved.

3.3. Phase 2

Once a feasible point has been obtained, phase 2 tries to achieve the optimizer of the nonlinear function (1). The primal partitioning method, as presented in [5], was thought for linear objective functions. However, when optimizing nonlinear functions, primal partitioning can be applied together with the Murtagh and Saunders' strategy —described in [6]— of dividing the set of variables in Basic, Superbasic and Nonbasic variables: $\hat{A} = [B|S|N]$, being $\hat{A}$ the matrix of constraints (2), (3), (6) and (7). The

efficiency in managing the working matrix Q with respect the whole basis B is preserved in the nonlinear case. Also the structure of either network, mutual capacity and side constraints can be exploited improving the computation time with respect general methods of optimization where this constraints are treated in a general way.

Consider that at iteration k we have (the subindex $_k$ is avoided in almost all cases to simplify the notation):

•$x_k, h(x_k)$: the current feasible point and the value of the objective function at this point.

•B, S, N: the sets of basic, superbasic and nonbasic variables. Having B is means having just K spanning trees and a LU decomposition of the working matrix Q.

•$g(x_k)$: where $g(x_k) = \nabla h(x_k)$ divided into $g(x_k) = \left[g_B|g_S|g_N\right]$ for basic, superbasic and nonbasic variables.

•Z: a way to perform the operations $Z'x$ and Zy, being Z a matrix of the null subspace of the current set of active constraints.

•g_z, ϵ_{g_z}: the current reduced gradient $g_z = Zg_S$, and a tolerance to stimate when its norm is considered sufficiently small.

•π: a vector satisfying $\pi^t B = g_B^t$.

Then the algorithm of phase 2 can be expressed as the following succession of steps (steps where profit can be taken of the particular form of constraints are marked with $(^*)$):

STEP (1): Optimality test in the current subspace.
i) If $||g_z|| \geq \epsilon_{g_z}$ go to step (3).

STEP (2): Price nonbasic variables.
i) Compute lagrange multipliers $\lambda = g_N - N^t\pi$. $(^*)$
ii) Choose a suitable λ_q and the associated column N_q. If no multiplier can be chosen go to step (8)
iii) Update data structures: remove N_q from N and add it to S; add λ_q as a new component of g_z.

STEP (3): Find descent direction $P^t = \left[P_B|P_S|0\right]^t$ for basic and superbasic variables.
i) Solve $Z^t H_k Z P_S = g_z$, where $H_k = \nabla^2 h(x_k)$. $(^*)$
ii) Solve $BP_B = -SP_S$. $(^*)$

STEP (4): Ratio test.
i) Find $\alpha_{max} \geq 0$ such that $x_k + \alpha_{max} P$ is feasible.
ii) if $\alpha_{max} = 0$ go to step (7).

STEP (5): Linesearch.
i) Find α^* such that $h(x_k + \alpha^* P) = \min_{0 \leq \alpha \leq \alpha_{max}} h(x_k + \alpha P)$
ii) Update new point $x_{k+1} = x_k + \alpha^* P$, and compute $h(x_{k+1})$ and g_{k+1}.

STEP (6): Update reduced gradient g_z.
i) Solve $\pi^t B = g_B^t$. $(^*)$
ii) Perform $g_z = g_S - S^t\pi$. $(^*)$
ii) if $\alpha < \alpha_{max}$ go to step (1).

STEP (7): A basic or a superbasic variable becomes nonbasic (it reaches its lower or upper bound).
i) If a superbasic variable S_p hits its bound then:

- Remove the component of g_z associated with the column S_p from S.
- Remove S_p from S and add it to N.

ii) If a basic variable B_p hits its bound then:

- Find a superbasic variable S_q to replace B_p in B preserving the nonsingularity of the basis. (*)
- Remove B_p from B and add it to N. Remove S_q from S and add it to B (pivot operation). This implies updating the working matrix Q since a change in the basis B has been made.
- Update π.
- Perform $g_z = g_S - S^t\pi$. (*)

iii) Go to step (1)

STEP (8): Optimal solution found.

Some remarks have to be made about fine points of this algorithm:

3.3.1 Computing the descent direction.

The current implementation of the program solves the system $Z^t H_k Z P_S = g_z$ by using a truncated-newton algorithm, following the description in [3]. This is based on the conjugate gradient method for solving linear systems of equations $Ax = b$ being A symmetric and positive definite. One of the next tasks to do would be testing the behaviour of the program with a quasi-newton update of $Z^t H_k Z$ as described in [6].

3.3.2 Linesearch.

Routine *GETPTC* of the Minos.5.3 package [7] has been employed to find the optimum step α^*. The good performance of this routine against others than have been tested justify the choice.

3.3.3 Pivot operation.

When a basic variable hits its bound in step (7) *ii*), a column of the basis B is removed and replaced by a column of the superbasic set S. The new basis (denoted by B_n) could be expressed as $B_n = B\eta$ being η a convenient eta-matrix. However the algorithm does not work with the whole basis B. For our purposes it is necessary to reflect how this change in the basis affects to the K spannings trees and the working matrix Q. Given that the $dim(Q)$ is equal to the number of saturated mutual capacity and side constraints it is clear than $dim(Q)$ can increase or decrease at each iteration (In this context "saturated constraint" means "constraint whose associated slack is not in the basis B". Of course, in the original formulation of the primal partitioning when an slack was not in the basis it was a nonbasic variable at value 0. In the nonlinear extension with side constraints, when an slack is not basic it can be superbasic or nonbasic at its upper bound, having a non-zero value. Then it is not correct talking about "saturated constraints", but this expression has to be understadood as "constraints whose slacks are not basic", as a reminiscence of the linear problem). Considering that the variables of the problem can be arcs or slacks (and the arcs of the basis B can be subdivided into arcs of the K spanning trees or complementary arcs), then, depending of the type of the variable entering and leaving the basis, the following 6 cases must be observed with

the related operations (denoting by "E:–" the type of entering variable and by "L:–" the type of leaving variable):

• *E: slack–L: slack.* The row of Q associated with the entering slack is removed and substituted by a new row for the leaving slack. $Dim(Q)$ is not modified.
• *E: slack–L: complementary arc.* The row and column of Q associated with the entering slack and leaving complementary arc respectively are removed. Update $dim(Q) = dim(Q) - 1$.
• *E: slack–L: arc of kth tree.* A complementary arc of the kth commodity, e.g. the jth complementary arc, having the leaving arc in its path P_j, must be found to replace the leaving arc in the kth tree. This complementary arc will always exist (otherwise the basis would become singular). The row an column of Q associated with the entering slack and the jth complementary arc are removed. Update $dim(Q) = dim(Q) - 1$.
• *E: arc–L: slack.* A new row associated with the leaving slack is added to Q. To maintain the nonsingularity of Q a new column for the entering arc —which will become complementary arc— is also added to the working matrix. Update $dim(Q) = dim(Q) + 1$.
• *E: arc–L: complementary arc.* The column of Q associated with the leaving complementary arc is removed, and substituted for a column for the entering arc, which will become complementary arc. $Dim(Q)$ is not modified.
• *E: arc–L: arc of kth tree.* A complementary arc of the kth commodity, e.g. the jth complementary arc, having the leaving arc in its path P_j, is searched for. If this arc is found, it will replace the leaving arc in the kth tree, and the entering arc will become complementary arc. If no complementary arc is found, then the entering arc will substitute the leaving arc in the kth tree. One of the two last cases must be always possible to preserve the nonsigularity of the basis. $Dim(Q)$ is not modified.

It has not been explicited, but it must be noticed that, when rows of matrix $Q = \begin{bmatrix} Q_{mc} \\ Q_{sc} \end{bmatrix}$ are removed of added, depending on the kind of the associated slack (if it is an slack of mutual capacity or side constraints) the operations will afect submatrix Q_{mc} or/and Q_{sc}.

4. Updating matrix Q.

The efficiency of the method is directly related to the efficiency of the routines than manage matrix Q. Obviously the first step is having a way of updating matrix Q instead of recalculate it at each iteration. It is not the purpose of this work to prove all the formulae required to obtain the expressions for updating Q_{k+1} from Q_k (where the subindex k refers to the current iteration). An approach of how to obtain it can be found in [5]. Two important remarks have to be made on this approach:
• It only considers the updating of the Q matrix with mutual capacity constraints. The updating used by the algorithm here presented is extended to include side constraints.
• It considers an updating of Q^{-1} instead of Q. The difficulty of the variable dimension of Q at each iteration makes that updating Q^{-1} is a costly operation if it is stored as a sparse matrix. On the other hand, it seems not appropiate to store Q^{-1} as a dense matrix (tests made with problems of many sizes showed that the number of nonzero

elements of Q^{-1} was always less than 10%). This led us to work with an update of Q, and not with its inverse.

Consider that at iteration k we recalculate the working matrix Q_k, which has dimension $dim(Q_k) = n_k$, and that it will be recalculated after i iterations (that is, at iteration $k+i$), where it will have dimension n_{k+i}. Once the working matrix is recalculated, a LU decomposition of the same is performed. The LU routine developed tries to exploit the sparse structure of Q by using either the $P3$ Hellerman-Rarick algorithm [4] or an adhoc variant of the same. Given that at each iteration the dimension of the working matrix can, at most, increase only in one column and row, then it follows than $n_j \leq n_k + i,\ k \leq j \leq k+i$. The method proposed consists in working with an *extended matrix* $\overline{Q}_j$ at each iteration $j,\ k \leq j \leq k+i$, where $\overline{Q}_j$ is defined as:

$$\overline{Q}_j = \begin{array}{c} \\ n_j \\ l_j \end{array} \begin{array}{c} \begin{array}{cc} n_j & l_j \end{array} \\ \begin{pmatrix} Q_j & 0 \\ 0 & \mathbb{1} \end{pmatrix} \end{array}$$

Where the dimensions n_j and l_j of matrices Q_j and identity $\mathbb{1}$ satisfy $n_j + l_j = n_k + i$. That is, the extended matrix has the maximum dimension that can achieve the working matrix at iteration $k+i$, in the worst case of increasing one column and row at each iteration j. This form of the extendend matrix must be preserved as an invariant condition at each iteration when the working matrix is updated (pre and post multiplying by eta an permutation matrices). This permets us working with the original LU decomposition when the working basis was recalculated, even when its dimension is modified. It must be noticed than solving the systems $Q_j x = b$ and $x^t Q_j = b^t$ is equivalent to solving $\overline{Q}_j x = b$ and $x^t \overline{Q}_j = b^t$ (only must be increased the dimension of the independent term b adding l_j zeros and, once solved the system, considering only the first n_j components of the solution vector x).

5. Computational results.

Problems of Long-Term Hydro-Thermal Coordination of Electricity Generation, modeled as nonlinear multicommodity network flow problems with side constraints have been used as test (a complete description of the objective function can be shown in [8] and constraints). For these problems 4 commodities are normally used although a higher number of commodities could have also been envisaged. There are two important features relative to the nonlinear multicommodity flow problem derived from the hydro-thermal model:

- $\overline{X}^k = T$, $k = 1, \ldots, K$, which means that, for each arc of the network, each single commodity can saturate the mutual capacity limit and, as a consequence, there are as many potentially active mutual capacity constraints as there are arcs
- the nonlinearity of the objective function is high.

Four models of different sizes have been used. Each model has been tried with three objective functions: a linear function, a convex nonlinear function and a highly nonlinear (and possibly nonconvex) function (this last is the nonlinear function of the Electricity Generation problem). The description of the models (that will be referred to as M1, M2, M3 and M4) is displayed in table 1, where for each model the number of

nodes, arcs, commodities and side constraints is given, together with the total number of variables and constraints.

Table 1. Definition of the models.

	# nodes	# arcs	# comm.	# SC	# var.	# cons.
M1	37	117	4	2	468	267
M2	37	153	4	12	612	313
M3	99	315	4	3	1260	714
M4	685	2141	4	3	8564	4884

Results obtained for each objective function with the algorithm proposed have been compared with those obtained with the Minos.5.3 general purpose optimization package. The executions have been made on a SUN Sparc-10/41 machine. Table 2 shows the CPU time in seconds, number of iterations at phase 1, number of iterations at phase 2, optimal value of the objective function ($h(x^*)$), dimension of matrix Q at the optimizer (dim(Q)), number of superbasics variables (# SB VAR.) and number of objective function evaluations (# O.F.EVAL) for Primal Partitioning algorithm (referred to as PP) and Minos.5.3 package (referred to as MI) for each objective function.

As it can be appreciated, for the linear objective function the gain of CPU time of the PP algorithm with respect to the general optimization package is considerable. For the convex nonlinear function this effect is still preserved. However in the highly nonlinear function of the Electricity Generation problem, Minos is faster than the PP algorithm. It must be noticed than the number of evaluations of the objective function is higher in the PP algorithm than in Minos. This could explain the low performance of PP algorithm for the highly nonlinear function, given that it is a costly function to compute. The fact of finding descent directions by a truncated-newton algorithm could explain why PP algorithm requires more function evaluations than Minos (which uses a quasi-newton method). The preparation of a revised version of the PP algorithm employing a quasi-newton update routine is in project.

Another important point is the behaviour of primal partitioning as compared to another specialized multicommodity network algorithms: price directive decomposition. The computational performance of multicommodity network flows through primal partitioning and through price directive decomposition (referred to as PPD) has been compared, for linear test problems, by Ali & al. [1]. This comparison indicates that PPD performs much better than PP. The experience of the authors does not agree with that of Ali & al. It appears that for nonlinear problems PP is better than the other method. In general PPD has a very efficient phase 1 but as the number of active vertices [5,9] increases in phase 2 the iterations become less effective and, after a given number of iterations —or after an analogous number of function evaluations— PP has reduced more the objective function than PPD Reference [9] describes the principles and performance of a nonlinear multicommodity network flow code implementing PPD. Although some improvements have been recently introduced by the authors in this code, its convergence patern remains of the same type. This code follows the same stages of phase 0, phase 1 and phase 2, as the PP algorithm of this work (the code of phase 0 is the same in both programs). Figure 1 shows the behaviour of each code for the model

Table 2. Results using the different objective functions.

	sec.	iter. Phase 1	iter. Phase 2	$h(x^*)$	dim(Q)	# SB VAR.	# O.F. EVAL
Linear function							
M1 (PP)	0.4	69	26	1057375.00	28		
M1 (MI)	2.5	238	43	1057375.07			
M2 (PP)	0.4	59	18	2.27×10^{-12}	20		
M2 (MI)	2.6	190	30	2.36×10^{-11}			
M3 (PP)	2.1	219	276	4477156.58	82		
M3 (MI)	8.8	569	265	4477156.60			
M4 (PP)	83.0	2081	1711	249354122.	616		
M4 (MI)	745.0	12712	3009	249354124.			
Convex nonlinear function							
M1 (PP)	10.5	69	1203	5.42797×10^{9}	36	249	4511
M1 (MI)	20.0	256	887	5.42796×10^{9}		249	1754
M2 (PP)	12.9	59	1491	3.97185×10^{9}	34	238	5323
M2 (MI)	23.0	163	1165	3.97183×10^{9}		231	2315
M3 (PP)	89.8	219	4540	6.04754×10^{9}	96	453	16798
M3 (MI)	105.9	617	1991	6.04651×10^{9}		449	3862
M4 (PP)	4249	2081	35306	1.94319×10^{14}	771	1292	120176
M4 (MI)	14308	17008	16498	1.94319×10^{14}		2077	23766
Highly nonlinear function (possibly non convex)							
M1 (PP)	3.5	69	364	-2.5164×10^{11}	52	2	490
M1 (MI)	3.4	261	202	-2.5164×10^{11}		2	211
M2 (PP)	8.1	59	630	-7.9577×10^{10}	47	2	941
M2 (MI)	5.8	103	354	-7.9578×10^{10}		2	365

M1 and the highly nonlinear function for phase 1 (unfeasibilities) and phase 2 (objective function $h(x)$). Time in figure 1 corresponds to system time and not CPU time (flat parts in phase 2 are due to system tasks performed periodically).

REFERENCES

[1] Ali, A., R.V. Helgason, J.L. Kennington & H. Lall. 1980. *Computational comparison among three multicommodity network flow algorithms.* Operations Research, v. 28, pp. 995-1000

[2] Bradley, G.H.; G.G. Brown & G.W. Graves. 1977. *Design and implementation of large scale primal transshipment algorithms.* Management Science, Vol.24, N.1, pages 1-34.

[3] Dembo, R.S. and T. Steihaug. 1983. *Truncated-Newton algorithms for large-scale unconstrained optimization.* Mathematical Programming, v.26, pp. 190-212.

[4] Hellerman, E and D. Rarick. 1971. *Reinversion with the preassigned pivot procedure.* Mathematical Programming, v. 1, pp. 195-216.

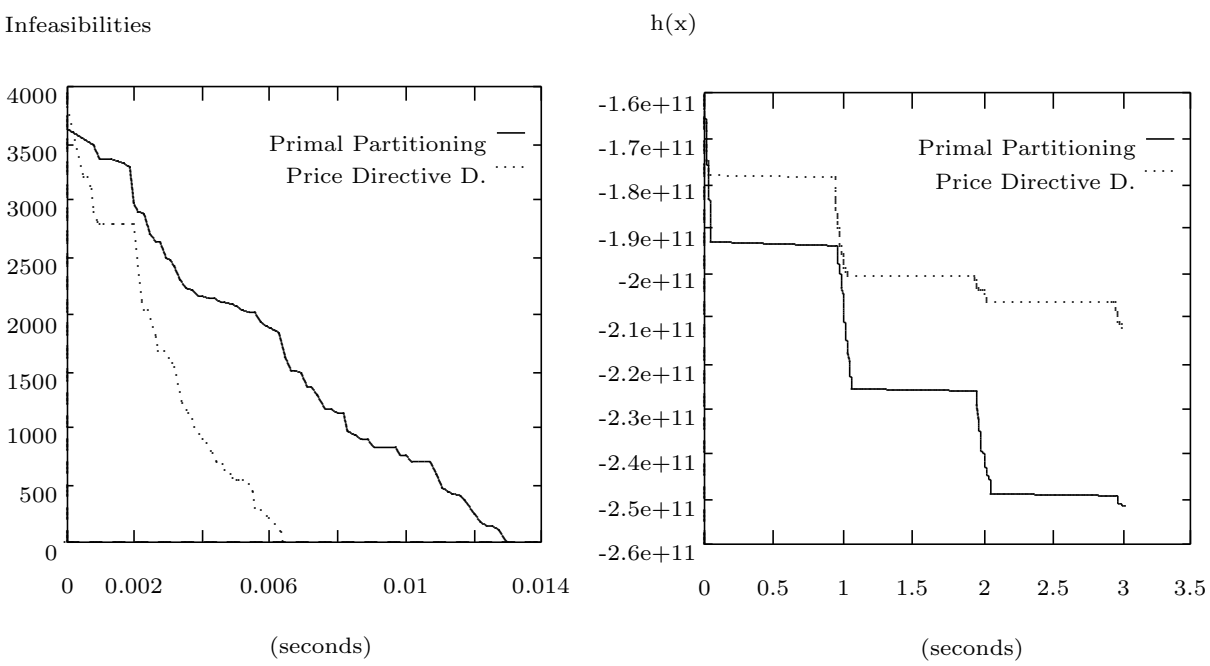

Figure 1. Unfeasibilites and objective function versus time

[5] Kennington, J.L. and R.V. Helgason. 1980. *Algorithms for network programming.* John Wiley & sons.

[6] Murtagh, B.A. and M.A. Saunders. 1978. *Large-scale linearly constrained optimization.* Mathematical Programming, v. 14, pp. 41-72

[7] Murtagh, B.A. and M.A. Saunders. 1983. *MINOS 5.0. User's guide.* Dpt. of Operations Research, Stanford University, CA 9430, USA.

[8] Nabona, N. 1992. *Multicommodity network flow model for long–term hydro– generation optimization.* Paper 92 WM 137–O PWRS presented at the IEEE 1992 PES Winter Meeting, New York

[9] Nabona, N. and J.M. Verdejo. 1991. *Numerical implementation of nonlinear multicommodity network flows with linear side constraints through price–directive decomposition.* Proccedings (to appear) of the 15th IFIP Conference on System Modelling and Optimization, Zurich. Springer–Verlag