COMPUTATIONAL TESTS OF A NONLINEAR MULTICOMMODITY NETWORK FLOW CODE WITH LINEAR SIDE CONSTRAINTS THROUGH PRIMAL PARTITIONING

Jordi Castro & Narcís Nabona
Statistics & Operations Research Dept.
UPC

DATE 9/94
DR 94/05

COMPUTATIONAL TESTS OF A NONLINEAR MULTICOMMODITY NETWORK FLOW CODE WITH LINEAR SIDE CONSTRAINTS THROUGH PRIMAL PARTITIONING.

Abstract: *This work is an extension of a previous one [8] where a specialized code for solving the multicommodity network flow problem with a linear objective function and linear side constraints was presented. The new code here described is able to minimize a nonlinear objective function, which generally requires longer executions than linear cases. The code implements a specialization for the multicommodity network problem of Murtagh & Saunders' strategy of dividing the set of variables into basic, nonbasic and superbasic. All the computations with the basis matrix are reduced to simpler operations with a reduced working matrix, following a primal partitioning technique. An ad hoc update of this working matrix has been developed, which improves the performance of the method considerably. The optimization process followed consists of three phases (instead of the usual two in the simplex method). The first merely solves a single network flow problem for each commodity, the second phase attempts to obtain a feasible point and the last reaches the optimizer of the nonlinear function. This methodology has proved to be very powerful, especially in cases where the number of active linking multicommodity constraints is small. Several tests are reported, using artificial problems and real problems arising from the fields of long and short-term hydro-thermal scheduling of electricity generation, electrical network expansion planning and traffic assignment, with up to ≈17,000 variables and ≈5,000 constraints.*

Keywords: Multicommodity Network Flows, Network Simplex Methods, Nonlinear Optimization, Primal Partitioning, Side Constraints, Hydro-Thermal Scheduling, Electricity Generation, Traffic Equilibrium, Network Expansion.

1. Introduction.

Although primal partitioning has been reported for quite some time to be an appropriate technique for solving the multicommodity linear network flow problem and its algorithm has been described in detail [17], no work has been done adapting this methodology to the nonlinear objective function case. This work aims to fill this void by describing the computational results obtained with an efficient implementation considering different nonlinear objective functions. This is an extension of a previous work by the authors [8] where a code for the linear case and its computational results for a set of tests were presented.

The nonlinear multicommodity network flow problem (which will be referred to as the NMP problem) can be cast as:

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

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

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

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

where $X_k \in \mathbb{R}^n$, (n: number of arcs) is the flow array for each commodity k ($k = 1 \div K$), K being the number of commodities of the problem, and h being a $\mathbb{R}^{K \times n} \rightarrow \mathbb{R}^1$ real valued function. $A \in \mathbb{R}^{m \times n}$ (m: number of nodes) is the arc–node incidence matrix. Constraints (3) are simple bounds on the flows, $\overline{X}_k \in \mathbb{R}^n, k = 1 \div K$, being the upper bounds. Equation (4) represents the mutual capacity constraints, where $T \in \mathbb{R}^n$.

In this work the original NMP problem has been extended to include linear side constraints defined by:

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

where L_k: $L_k \in \mathbb{R}^{p \times n}$, $k = 1 \div K$, and $L, U \in \mathbb{R}^p$ (p: number of side constraints). These side constraints can link arcs of the same or different commodities. Therefore, the final formulation of the problem considered can be stated as follows:

$$\min_{X_1, X_2, \ldots, X_K} (1) \text{ ; subj. to } (2\text{–}5) \tag{6}$$

and will be referred to as the NMPC problem.

The code here presented (which will be referred to as the PPRN code [6] in the rest of the document) can be viewed as a general purpose code for solving the NMPC problem. The NMPC problem was first approached using the price-directive decomposition [23] but this procedure does not seem to be as computationally efficient as primal partitioning [7]. If the set of side constraints (5) is empty, code PPRN will only solve a nonlinear multicommodity network flow problem (NMP). If the number of commodities is equal to one, it will work as a specialized nonlinear network flow code with side constraints, as described in [14,17,18]. Even in the latter case the PPRN code can be more efficient than a plain network flow code with side constraints, due to the fact of considering a variable-dimension working matrix instead of a fixed one (as will be shown in later sections). This can improve the performance of the algorithm when the number of active side constraints at the optimum is small with respect to the whole set of side constraints. When the objective function (1) is linear the PPRN code implements a specialized simplex algorithm based on a primal partitioning of the basis, as described in [8].

The rest of the document will be subdivided into six main sections. Firstly, a brief revision of the literature on this subject for the linear case will be made. Secondly, the primal partitioning methodology will be revised. The next section will present the specific implementation of the code developed. The updating process of the working matrix, which is instrumental in the performance of the algorithm, will be described in the following section. Once the main features of the code have been reported, the tests problems that have been employed will be presented. These problems are either artificial problems or real problems arising mainly from long and short-term hydro-thermal scheduling of electricity generation, electrical network expansion planning and traffic assignment. The last main section will show the computational results obtained, with each test problem previously detailed. Comparisons of computational performance with a general-purpose nonlinear optimization code are included.

2. Linear Multicommodity Network Flows.

If the objecive function (1) is linear we will refer to the nonlinear multicommodity problem (NMP) merely as the MP problem. To solve the MP problem by exploiting the network structure, various techniques have been described in the literature. Some of them deal with the mutual capacity constraints (4) in an exact fashion whereas others replace them by a Lagrangian relaxation in the objective function. The price-directive decomposition, resource-directive decomposition and primal partitioning methods belong to the first class. A complete description of these three methodologies can be found in [17]. The Lagrangian relaxation technique does not guarantee finding the optimal flows, but it can achieve good approximations simply by solving decoupled single network flow problems obtained by relaxing constraints (4). This methodology is briefly described in [1].

A first attempt, comparing the above multicommodity network techniques (except the Lagrangian relaxation one), was already made in [2]. In that work primal partitioning and price-directive decomposition seemed to be the best methods.

Several codes have been developed to solve the MP problem, although up to now none have become standard. Recently, interior point methods have provided another approach to solve the MP problem. These methods appear to be really efficient when the size of the network is very great (see [16] for a description of a code using such methodology).

In [8] the authors already presented a code for solving the MP problem considering additional side constraints. That code and the PPRN code here described mainly follow the underlying ideas in the primal partitioning method. Some aspects, especially those related to the management of a working matrix are the same in both codes, and have been upgraded with respect to the original formulation of the problem described in [17], substantially improving the performance of the algorithm. The primal partitoning technique and the special management of the working matrix are revised in later sections.

3. The primal partitioning method.

In this section a brief description of the primal partitioning method will be presented, paying special attention to the changes brought about by considering the additional side constraints (5). See [17] for a comprehensive description.

3.1. Structure of the problem.

Given that constraints (2), (4) and (5) in (6) are linear, it is possible to consider the problem constraint matrix A. Then each variable j_k (that is, each flow j of the k-th commodity) has an associated column A^{j_k} in A, with the following no null components:

$$(A^{j_k})^t = \left(\underbrace{\overset{s}{\downarrow}\ldots 1 \ldots \overset{t}{\downarrow} -1\ldots}_{\text{Network}} \;\Big|\; \underbrace{\overset{j}{\downarrow}\ldots 1\ldots}_{\text{Mutual Capacity}} \;\Big|\; \underbrace{a_{j_1}\ldots a_{j_p}}_{\text{Side Constraints}} \right)^t$$

where s and t identify the source and target nodes of arc j_k. It can be noticed that each variable appears in three clearly different types of constraint: network, mutual capacity and side constraints. Network constraints are always active (they are equality constraints), whereas mutual capacity and side constraints may not be (as they are inequality constraints).

Every basis in the primal partitioning method can be decomposed as follows:

$$B = \begin{array}{|c|c|c|} \hline L_1 & R_1 & 0 \\ \hline L_2 & R_2 & 0 \\ \hline L_3 & R_3 & \mathbb{1} \\ \hline \end{array} \tag{7}$$

L_1, R_2 and $\mathbb{1}$ being square matrices where:

- L_1 refers to the network constraints and arcs of the K spanning trees. The topology of this matrix is:

$$L_1 = \begin{array}{|c|c|c|c|} \hline B_1 & & & \\ \hline & \boxed{B_2} & & \\ \hline & & \ddots & \\ \hline & & & \boxed{B_K} \\ \hline \end{array}$$

each B_k being a nonsingular matrix associated with the k-th spanning tree. L_1 can be represented at every iteration by K spanning trees following the methodology described in [4,12].

- R_1 refers to the network constraints and complementary arcs of the K commodities. Complementary arcs do not belong to any spanning tree and are required to preserve the nonsingularity of the basis.
- L_2 refers to the active mutual capacity and side constraints, for the arcs of the spanning trees.
- R_2 refers to the active mutual capacity and side constraints, for the complementary arcs.
- L_3 refers to the inactive mutual capacity and side constraints, for the arcs of the spanning trees.
- R_3 refers to the inactive mutual capacity and side constraints, for the complementary arcs.
- $\mathbb{1}$, an identity matrix, refers to the slacks of the inactive mutual capacity and side constraints. (It should be noticed that constraints whose slacks are in matrix $\mathbb{1}$ are treated as inactive constraints, even though the slack values are zero).

3.2. Motivation for using a working matrix.

During the optimization process systems $Bx = b$ and $x^tB = b^t$ must be solved at each iteration, x and b being the variable and independent term vectors respectively. A description of the solution technique can be found in [17] and is briefly outlined here. Considering for x and b a partition such as the one employed above for the basis B, the next subsections show how to efficiently obtain the solution of both systems.

3.2.1. Computing $Bx = b$.

This system can be written as:

$$\begin{array}{|c|c|c|}\hline L_1 & R_1 & 0 \\ \hline L_2 & R_2 & 0 \\ \hline L_3 & R_3 & \mathbb{1} \\ \hline \end{array} \quad \begin{array}{|c|}\hline x_1 \\ \hline x_2 \\ \hline x_3 \\ \hline \end{array} = \begin{array}{|c|}\hline b_1 \\ \hline b_2 \\ \hline b_3 \\ \hline \end{array}$$

By block multiplication we obtain:

$$L_1x_1 + R_1x_2 = b_1 \tag{8}$$
$$L_2x_1 + R_2x_2 = b_2 \tag{9}$$
$$L_3x_1 + R_3x_2 + x_3 = b_3 \tag{10}$$

Isolating x_1 from (8) we obtain $x_1 = L_1^{-1}b_1 - L_1^{-1}R_1x_2$. Substituting this expression into (9) yields $L_2L_1^{-1}b_1 - L_2L_1^{-1}R_1x_2 + R_2x_2 = b_2$. Now x_2 is directly obtained from this equation:

$$x_2 = (R_2 - L_2L_1^{-1}R_1)^{-1}(b_2 - L_2L_1^{-1}x_1) \tag{11}$$

Once x_2 is known, x_1 is computed directly:

$$x_1 = L_1^{-1}b_1 - L_1^{-1}R_1x_2 \tag{12}$$

And finally, with x_1 and x_2, x_3 is computed as:

$$x_3 = b_3 - L_3x_1 - R_3x_2 \tag{13}$$

Thus by solving (11), (12) and (13) consecutively we obtain the solution of the original system.

3.2.2. Computing $x^tB = b^t$.

This system can be written as:

$$\begin{array}{|c|c|c|}\hline x_1 & x_2 & x_3 \\ \hline \end{array} \quad \begin{array}{|c|c|c|}\hline L_1 & R_1 & 0 \\ \hline L_2 & R_2 & 0 \\ \hline L_3 & R_3 & \mathbb{1} \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline b_1 & b_2 & b_3 \\ \hline \end{array}$$

By block multiplication we obtain:

$$x_1L_1 + x_2L_2 + x_3L_3 = b_1 \tag{14}$$
$$x_1R_1 + x_2R_2 + x_3R_3 = b_2 \tag{15}$$
$$x_3 = b_3 \tag{16}$$

The x_3 value is found directly from (16). Isolating x_1 from (14) we obtain:

$$x_1 = (b_1 - x_3L_3 - x_2L_2)L_1^{-1} \tag{17}$$

Using (17) and (16) at (15) we get:

$$\begin{aligned} (b_1 - b_3L_3 - x_2L_2)L_1^{-1}R_1 + x_2R_2 + b_3R_3 &= b_2 \\ (b_1 - b_3L_3)L_1^{-1}R_1 + x_2(R_2 - L_2L_1^{-1}R_1) &= b_2 - b_3R_3 \\ \big((b_2 - b_3R_3) - (b_1 - b_3L_3)L_1^{-1}R_1\big)(R_2 - L_2L_1^{-1}R_1)^{-1} &= x_2 \end{aligned} \tag{18}$$

Thus by solving (16), (18) and (17) we obtain the solution of the original system.

As shown in the two previous subsections, to solve systems $Bx = b$ and $x^tB = b^t$ it suffices to invert submatrix L_1 and a matrix whose expression is $R_2 - L_2L_1^{-1}R_1$. This last matrix will be referred to as the working matrix, and denoted by Q. It must be noticed that the fact of inverting L_1 does not involve too much work, given that L_1 is a block diagonal matrix, where each block represents a spanning tree. This kind of system can be solved by simply exploiting the tree structure

of the matrix and highly efficient procedures have been developed [12]. Therefore, the problem of solving both systems of equations is reduced to factorizing the working matrix Q instead of basis B, and having a procedure to update this factorization at each iteration. Since the dimension of the working matrix is small compared with the whole dimension of basis B, it can be expected that the computation time of an algorithm using this primal partitioning will likewise be small compared with a general-purpose linear optimization package. On the other hand, the dimension of basis B is fixed during the optimization process, whereas the dimension of Q is variable, given that it depends on the number of active mutual capacity and side constraints. That implies that the updating process of the Q factorization must be able to deal with variable size dimensions, increasing the difficulty of the algorithm (as will be shown in later sections).

3.3. Computing the working matrix Q.

Some new concepts must first be defined in order to use them in an efficient procedure for computing Q:

- $\mathcal{A}_{sc}$: set of active side constraints at current iteration.
- $\mathcal{A}_{mc}$: set of active mutual capacity constraints at current iteration.
- $\mathcal{A}$: set of active constraints (mutual capacity and side constraints), that is, $\mathcal{A} = \mathcal{A}_{sc} \cup \mathcal{A}_{mc}$.
- $|\mathcal{C}|$: number of elements of set $\mathcal{C}$.
- $dim(M)$: dimension of matrix M.

Given that R_2 and L_2 are associated with the active mutual capacity and side constraints, they can be subdivided into two submatrices as follows:

$$R_2 = \begin{bmatrix} R_{2_{mc}} \\ R_{2_{sc}} \end{bmatrix} \qquad L_2 = \begin{bmatrix} L_{2_{mc}} \\ L_{2_{sc}} \end{bmatrix}$$

where $R_{2_{mc}}$ and $L_{2_{mc}}$ refer to constraints belonging to $\mathcal{A}_{mc}$, and $R_{2_{sc}}$ and $L_{2_{sc}}$ refers to constraints of the set $\mathcal{A}_{sc}$. Since $Q = R_2 - L_2 L_1^{-1} R_1$, it can also be considered as subdivided into two submatrices thus:

$$Q = \begin{bmatrix} Q_{mc} \\ Q_{sc} \end{bmatrix} = \begin{bmatrix} R_{2_{mc}} \\ R_{2_{sc}} \end{bmatrix} - \begin{bmatrix} L_{2_{mc}} \\ L_{2_{sc}} \end{bmatrix} L_1^{-1} R_1$$

whose dimensions are $dim(Q_{mc}) = |\mathcal{A}_{mc}| \times |\mathcal{A}|$ and $dim(Q_{sc}) = |\mathcal{A}_{sc}| \times |\mathcal{A}|$.

The expression for computing Q involves the calculation of $L_1^{-1} R_1$. Since L_1 is a block diagonal matrix where the k-th block is a minimum spanning tree for the k-th commodity, and R_1 expresses for each complementary arc of the k-th commodity its connection to the k-th minimum spanning tree, then solving $L_1^{-1} R_1$ is equivalent to having the paths (denoted by $P_j, j = 1 \div |\mathcal{A}|$) 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 source node of the complementary arc j; otherwise, it has reverse orientation.

If we denote by:

- a_j the arc associated with the j-th column of Q, $j = 1 \div |\mathcal{A}|$.
- mc_i the mutual capacity constraint of the i-th row of $Q, i = 1 \div |\mathcal{A}_{mc}|$ (this capacity constraint refers to the saturated arc mc_i).
- sc_i the side constraint of the i-th row of $Q, i = |\mathcal{A}_{mc}| + 1 \div |\mathcal{A}|$.
- $B(a, n)$ a logical function which becomes $true$ 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}:

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

Submatrix Q_{sc}:

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

A full and more detailed description of the computation of Q can be found in [5].

4. Implementation of primal partitioning.

The implementation of the primal partitioning method developed in the PPRN code follows three stages, called phases 0, 1 and 2, instead of the two classical phases of the simplex method. Phases 0 and 1 attempt to obtain a feasible starting point, whereas phase 2 achieves the optimizer. However, although phases 0 and 1 work sequentially to find a feasible point, it can be said that primal partitioning is only applied in phases 1 and 2. The following subsections will clarify these ideas by describing each phase.

For computational purposes the inequality constraints (4) and (5) in the original NMPC problem are replaced by equality constraints by adding slacks, obtaining:

$$\sum_{k=1}^{K} X_k + s = T \quad ; \quad \underline{0} \leq s \tag{19}$$

$$\sum_{k=1}^{K} L_k X_k + t = U \quad ; \quad \underline{0} \leq t \leq U - L \tag{20}$$

where $s \in \mathbb{R}^n$ and $t \in \mathbb{R}^p$. In this formulation equations (19) and (20) replace the original equations (4) and (5). Then the formulation of the problem considered by the algorithm (which will be referred to as NMPC2) is:

$$\min_{X_1, X_2, \ldots, X_K} (1) \text{ ; subj. to (2–3), (19–20)} \tag{21}$$

It can be noticed that the current version of PPRN cannot deal with lower bounds other than zero in the variables.

4.1. Phase 0.

In phase 0 the algorithm considers only the network constraints and bounds on the variables of the problem, without any constraint linking the flows of different commodities. It attempts to obtain for each commodity $k, k = 1 \div K$, a feasible starting point for the linear network problem:

$$\begin{aligned} \min_{X_k} \quad & C_k^t X_k \\ \text{subj. to} \quad & A X_k = R_k \\ & \underline{0} \leq X_k \leq \overline{X}_k \end{aligned} \tag{22}$$

This problem is solved by applying a specialization of the simplex algorithm for networks. The implementation developed mainly follows the ideas described in [12] with regard to the pivotal operations when managing the spanning trees. It is important to note that phase 0 has nothing to do with primal partitioning, as it only solves single network problems.

The code developed can either merely obtain a feasible point for (22) or reach its optimum solution. (The default option is to obtain a feasible point). When at the optimal solution of problem (21) the number of active mutual capacity and side constraints $|\mathcal{A}|$ is small, that implies that this point is not far from the point obtained by joining the solutions of the K single network problems (22). Then it seems to be a good choice to obtain the optimum value when solving (22) for each commodity,

because at the end of phase 0 the current point will be near the optimum solution desired. Otherwise, simply obtaining a feasible point for (22) will suffice.

It has been observed in phase 0 that keeping up and checking a degenerate pivot list to avoid degenerate steps is less efficient than allowing degenerate pivots. Thus the default option in the code developed is to permit degenerate steps.

4.2. Phase 1.

The K points obtained in phase 0 will not satisfy in general the mutual capacity and side constraints, thus giving rise to a pseudofeasible 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 pseudofeasible 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:

$$\min_{X_1, X_2, \dots, X_K, s, t, e, f} \quad \sum_{i \in s^-} e_i + \sum_{i \in t^-} f_i + \sum_{i \in t^+} f_i \tag{23}$$

$$\text{subj. to} \quad (2\text{–}3)$$

$$\sum_{k=1}^{K} X_k + s + \mathbb{1}^e e = T \tag{24}$$

$$\sum_{k=1}^{K} L_k X_k + t + \mathbb{1}^f f = U \tag{25}$$

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

Where both matrices $\mathbb{1}^e \in \mathbb{R}^{n \times n}$ and $\mathbb{1}^f \in \mathbb{R}^{p \times p}$ in (24) and (25) are diagonal and defined as follows:

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

It can be noticed that the objective function (23) at phase 1 is nothing but the sum of infeasibilities of the mutual capacity and side constraints. Therefore, the NMPC2 problem defined in (21) will be feasible if, at phase 1, the value of (23) at the optimizer is 0.

Dividing the process of finding a feasible starting point for problem MCP2 into two stages (phases 0 and 1) has proved to be very efficient in number of iterations with respect to methods that starting from any given point consider the sum of infeasibilities for all constraints. In the PPRN code, at the beginning of phase 1 it is known that no infeasibilities should be considered for the network constraints, because in phase 0 K spanning trees have been obtained. Besides, the obtention of the K

spanning trees does not involve much computation time, given that the available methods for solving linear network problems are very efficient.

4.3. Phase 2.

Once a feasible point has been obtained, phase 2 attempts to achieve the optimizer of the nonlinear objective function. The primal partitioning method, as presented in [17], was intended for linear objective functions. However, when optimizing nonlinear functions, primal partitioning can be applied together with Murtagh and Saunders' strategy —described in [19]— of dividing the set of variables into basic, superbasic and nonbasic variables:

$$A = \left[B|S|N\right] \tag{26}$$

A being the matrix of constraints (2), (4), and (5). The efficiency in managing the working matrix Q with respect to the whole basis B is preserved in the nonlinear case. Furthermore, the structure of network, mutual capacity and side constraints, can be exploited, improving the computation time with respect to general methods of optimization where these constraints are treated in a general way.

Consider that at iteration i we have (subindex $_i$ is omitted in almost all cases to simplify the notation):

•$x_i, h(x_i)$: 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. B is represented by just K spanning trees and an LU decomposition of the working matrix Q.

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

•Z: a representation matrix of the null subspace of the constraint matrix A defined in (26). The expression of Z is:

$$Z = \begin{bmatrix} -B^{-1}S \\ \mathbb{1} \\ 0 \end{bmatrix} \tag{27}$$

It can easily be observed that $AZ = 0$.

•g_z, ϵ_{g_z}: the current reduced gradient $g_z = Z^t g(x_i)$, and a tolerance to estimate 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 one can take advantage of the particular structure of the 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 $Z^t H_i Z P_S = -g_z$, where $H_i = \nabla^2 h(x_i)$. (*)
- ii) Solve $BP_B = -SP_S$. (*)

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

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

STEP (6): Update reduced gradient g_z.
- i) Solve $\pi^t B = g_B^t$. (*)
- ii) Perform $g_z = g_S - S^t\pi$. (*)

iii) 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 involves 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 comments should be made about the finer points of this algorithm:

4.3.1. Computing the descent direction.

The current implementation of the program makes it possible to solve the system $Z^tH_iZP_S = -g_z$ in step (3) i) by two methods: through a truncated-Newton algorithm, or using a quasi-Newton approximation of the projected Hessian Z^tH_iZ. In neither case is an analytical expression for the Hessian of the objective function (1). Thus the PPRN code only needs the evaluation of the objective function and its gradient. The following subsections will briefly show both methodologies.

4.3.1.1. Truncated-Newton algorithm.

The truncated-Newton algorithm employed for solving $Z^tH_iZP_S = -g_z$ follows mainly the description of [9]. This is based on the conjugate gradient method for solving linear systems of equations $Mx = b$, M being symmetric and positive definite. In this case the matrix system M is directly the projected HessianZ^tH_iZ.

The steps for solving $Z^tH_iZP_S = -g_z$ are:

Step 0) Set $j = 0$, $p_0 = 0$, $r_0 = g_z$, $d_0 = -r_0$, $\delta_0 = d_0^t d_0$, $\eta = \theta^2(g_z^t g_z)$.

Step 1) $q_j = Z^tH_iZd_j$
if $d_j^t q_j \leq \sqrt{\varepsilon_M}\ \delta_j$ then exit: $P_S = \begin{cases} -g_z & \text{if j=0} \\ p_j & \text{otherwise} \end{cases}$
else $\alpha_j = r_j^t r_j / d_j^t q_j$

Step 2) $p_{j+1} = p_j + \alpha_j d_j$
$r_{j+1} = r_j + \alpha_j q_j$
if $r_{j+1}^t r_{j+1} \leq \eta$ then exit: $P_S = p_{j+1}$

Step 3) $\beta_j = r_{j+1}^t r_{j+1} / r_j^t r_j$
$d_{j+1} = -r_{j+1} + \beta_j d_j$
$\delta_{j+1} = r_{j+1}^t r_{j+1} + \beta^2 \delta_j$
$j = j + 1$
go to step 1

The three critical points of this algorithm are the computation of q_j and the controlling of the two exit conditions. The next three points will briefly describe how PPRN proceeds with these tasks.

i) Given that the current Hessian H_i is unknown, it is necessary to approximate it in some way. The steps for computing $q_j = Z^tH_iZd_j$ are:

- $U = Zd_j$
- $V = \big(g(x_i + \gamma U) - g(x_i)\big)/\gamma$, $g(x)$ being the gradient of the objective function.
- $q_j = Z^tV$

The second step comes from expanding $g(x_i + \gamma U)$ in Taylor series:

$$g(x_i + \gamma U) \approx g(x_i) + \gamma \nabla g(x_i) U$$
$$\nabla g(x_i) U \approx \big(g(x_i + \gamma U) - g(x_i)\big)/\gamma = V$$

Thus, since $\nabla g(x_i)U = H_iU = H_iZd_j$, V can be considered a good approximation to H_iZd_j. The γ difference interval used by the code is $\gamma = \sqrt{\varepsilon_M}/\sqrt{\delta_j}$, where ε_M is the machine precision, and $\delta_j = d_j^t d_j$ is updated at each iteration in step 3. In fact, the correct value that should be used is $\gamma = \sqrt{\varepsilon_M}/\sqrt{U^tU}$ where $U = Zd_j$. The effect of choosing such a γ value would be equivalent to performing finite differences using the unit vector $U/||U||_2$ (where $||.||_p$ denotes the p-norm of a vector) and a difference interval equal to $\sqrt{\varepsilon_M}$, which is a more desirable situation. However, that would involve computing the 2-norm of U at each iteration, whereas the value $\sqrt{\delta_j} = ||d_j||_2$ has been already computed. This value is chosen, then, to avoid extra computations, although in some cases (where $||U||_2 >> ||d_j||_2$) the difference approximation could become meaningless.

ii) The first exit condition in step 1 attempts to control whether or not the hessian H_i is positive definite. If the value $d_j^t Z^t H_i Z d_j$ was negative we could conclude that $Z^t H_i Z$ is not positive definite. However, given that we only have an approximation of $Z^t H_i Z d_j$, we have to compare $d_j^t Z^t H_i Z d_j$ with a value greater than 0. The PPRN code uses the value $\sqrt{\varepsilon_M}\ \delta_j$, where $\delta_j = d_j^t d_j$. Using this value, the comparison test can be seen as:

$$\begin{aligned} d_j^t Z^t H_i Z d_j &\leq \sqrt{\varepsilon_M}\ \delta_j \\ d_j^t Z^t H_i Z d_j &\leq \sqrt{\varepsilon_M}\ ||d_j||_2^2 \\ \frac{d_j^t}{||d_j||_2} Z^t H_i Z \frac{d_j}{||d_j||_2} &\leq \sqrt{\varepsilon_M} \end{aligned}$$

which means than unit vectors are being considered, decreasing the possible round-off error.

iii) The second exit condition in step 2 will be active when a sufficient reduction in the vector r_j has been achieved. Initially $r_0 = g_z$, and (assuming full precision) at each iteration $||r_j||_2 \leq ||r_{j-1}||_2,\ j = 1 \div s$, s being the number of superbasic variables (which is the dimension of the system). After s iterations the r_s vector will have a value of 0, which means that p_s is the solution of the system. However, in order to avoid too many computations, the process will be stopped when a sufficient decrease in the 2-norm of r_j has been reached. In fact, the exit condition should be written as:

$$\begin{aligned} r_{j+1}^t r_{j+1} &\leq \eta \\ r_{j+1}^t r_{j+1} &\leq \theta^2 (g_z^t g_z) \\ \frac{r_{j+1}^t r_{j+1}}{g_z^t g_z} &\leq \theta^2 \\ \frac{||r_{j+1}||_2}{||g_z||_2} &\leq \theta = \max\{\varepsilon_1, \min\{\varepsilon_2, ||g_z||_2\}\} \end{aligned}$$

where ε_1 is the maximum required precision (in the PPRN code $\varepsilon_1 = \sqrt{\varepsilon_M}$), ε_2 is the required precision when the current iterate x_i is far from the optimum of the current subspace ($\varepsilon_2 \in [0, 1]$ and can be chosen by the user), and $||g_z||_2$ is used as the required precision when we are near the optimum of the current subspace.

A clear disadvantage of this methodology is that at each iteration an evaluation of the objective function has to be made for computing q_j through finite differences, which considerably increases the cost of the algorithm. On the other hand, no data structure need be stored to maintain $Z^t H_i Z$ at each iteration (as quasi-Newton methods do), and only four vectors of dimension s are needed for p_j, r_j, d_j and q_j. If the number of superbasics is high a great memory saving can be made, which can speed up considerably the performance of the program.

4.3.1.2. Quasi-Newton method.

The second way of solving $Z^t H_i Z P_S = -g_z$ consists of approximating the $s \times s$ matrix $Z^t H_i Z$ by a factorization $R^t R$ following the description in [19]. This is the default option in the PPRN code. Thus, the superbasic direction P_S in step (3) i) is found directly by solving $R^t R P_S = -g_z$ by forward and backward substitution. This factorization must be updated whenever a superbasic variable enters the set S (step (2) iii)), a superbasic variable leaves the set S (step (7) i)), there is a basis change (step

(7) ii)) or there is a movement from x_i to x_{i+1} and a quasi-Newton update of R must be performed (step (6)). Let us describe briefly how each operation is performed.

i) A superbasic variable enters the set S.

In this case the current implementation of the PPRN code merely adds $\mathbb{1}_{s+1}$ (the $(s+1)$-th unit vector) to R as a new column. The new dimension of matrix R will be $(s+1)\times(s+1)$. This implies that when solving $R^tRP_S = -g_z$ at the next iteration $P_{S_{s+1}} = -g_{z_{s+1}}$ (the gradient direction will be used for the $(s+1)$-th component of P_S). However, in successive iterations new second-order information will be added to this component through the quasi-Newton update.

ii) A superbasic variable leaves the set S.

In this case a column of R (i.e., the q-th column) must be removed from R. The resulting upper-Hessenberg matrix must be restored to its triangular form. This is performed by pre-multiplying R with suitable Givens matrices $G_{l,l+1},\ l = q \div s-1$.

iii) A basis change is performed.

Consider that the p-th basic variable is replaced by the q-th superbasic variable. Thus, the new null space matrix $\bar{Z}$ can be written as $\bar{Z} = Z(\mathbb{1} + \mathbb{1}_q u^t)$, where $\mathbb{1}_q$ is the q-th unit vector, and $u = -1/y_q(y + \mathbb{1}_q)$, y being $y = B_p^{-1}S$ (B_p^{-1} is the p-th row of the inverse of the basis, S is the superbasic matrix and y_q the q-th component of y). Thus, since $R^tR \approx Z^tH_iZ$, the next factorization $\bar{R}$ can be obtained as follows:

$$\begin{aligned}
\bar{R}^t\bar{R} &\approx \bar{Z}^tH_i\bar{Z} \\
&\approx (\mathbb{1} + u\mathbb{1}_q^t)Z^tH_iZ(\mathbb{1} + \mathbb{1}_q u^t) \\
&\approx (\mathbb{1} + u\mathbb{1}_q^t)R^tR(\mathbb{1} + \mathbb{1}_q u^t) \\
&\approx (R^t + uR_q^t)(R + R_q u^t) \\
&\Downarrow \\
\bar{R} &= (R + R_q u^t)
\end{aligned}$$

R_q being the q-th column of R. A more detailed description of this process can be found in [19].

iv) Quasi-Newton update.

When there is a movement from x_i to x_{i+1} the approximation R^tR of Z^tH_iZ must be modified to take into account the new reduced Hessian $Z^tH_{i+1}Z$. The update used by the PPRN code is the well-known BFGS formula

$$\bar{R}^t\bar{R} = R^tR + \frac{1}{\alpha^* y^tP_S}yy^t + \frac{1}{g_z^tP_S}g_zg_z^t \tag{28}$$

where α^* is the linesearch value obtained in step (5) i), g_z is the reduced gradient $Z^tg(x_i)$ at point x_i, P_S is the superbasic direction found in step (3) i), and $y = g(x_{i+1}) - g(x_i)$. Appendix 1 shows how the factorization update $\bar{R}$ is performed in the PPRN code.

The main advantage of the quasi-Newton method with respect to the truncated-Newton algorithm is that no extra objective function evaluations are required. For objective functions with high evaluation computation cost this can mean a significant time saving. On the other hand, the R matrix must be stored in dense form, and should the number of superbasics be very great its management would be prohibitive. The default option of the PPRN code is to work with the quasi-Newton method until a certain number of superbasics is reached (this threshold value can be modified by the user), whereupon a switch is made to the truncated-Newton method.

4.3.2. Optimality test in the current subspace.

At each iteration it must be tested whether the optimum point of the current subspace has already been reached (step (1) i) of the algorithm) . However, the test performed is much more exhaustive than simply ascertaining whether $||g_z|| \geq \epsilon_{g_z}$. Actually, the code discerns between two situations: when we have still not reached the optimum active constraint set (thus being far from the optimizer) and when we are already in the optimum active constraint set and merely a final, more accurate subspace minimization is required. The variable *cs* tells us which is the *c*urrent *s*ituation, and it can take the values "far" or "near" depending on whether we are in the first or the second case. The code

uses six logical variables (T_i) to decide whether or not the optimum point has been achieved in the current subspace. Each T_i is defined as follows.

$T_1 := \big(\alpha^* ||P_S||_1 \leq (\varepsilon_x^{cs} + \sqrt{\varepsilon_M})(1 + ||x_s||_1)\big)$
$T_2 := \big(|\Delta h| \leq (\varepsilon_f^{cs} + \varepsilon_M)(1 + |h|)\big)$
$T_3 := \big(||g_z||_\infty \leq T_{g_z}\big)$
$T_4 := \big(||g_z||_\infty \leq \max\{T_{g_z}/10, \varepsilon_g \epsilon(||\pi||_1)\}\big)$
$T_5 := \big((T_5 \text{ is active}) \; and \; (cs = \text{“far”}) \; and \; (nsame \leq MAXsame)\big)$
$T_6 := \big(ncurrent \leq MAXcurrent\big)$

The first test, T_1, controls whether the 1-norm of the current movement in the superbasic variables $\alpha^* ||P_S||_1$ is significant with respect to the 1-norm of the superbasic components of the current iterate $||x_s||_1$, using for such comparison the machine precision ε_M and the value ε_x^{cs} which depends on the variable cs (if cs= “near” this value will be much smaller than when cs= “far”, requiring a smaller movement to satisfy the test).

The second test, T_2, will be true when the variation in the objective function $|\Delta h|$ is not significant with respect to the absolute value of h at the current iterate ($|h|$). The value ε_f^{cs} used in the comparison depends also on the variable cs, and, as in the previous case, $\varepsilon_f^{\text{“near”}} << \varepsilon_f^{\text{“far”}}$.

At test T_3 the tolerance T_{g_z} has been previously computed as $T_{g_z} = \eta_{g_z}^{cs} ||g_z^0||_\infty$, where g_z^0 was the reduced gradient vector at the first point of the current subspace, and $\eta_{g_z}^{cs} \in [0, 1]$ is a value that can be chosen by the user. Thus, this test attempts to control when a sufficient reduction in the reduced gradient has been made since the minimization in the current subspace started. When cs =“near” it is desirable to require a greater reduction in the projected gradient, so, as in previous cases, $\eta_{g_z}^{\text{“near”}} << \eta_{g_z}^{\text{“far”}}$.

The next test, T_4, will be true when the reduced gradient is so small that the current point can be considered to be the optimum one of the current subspace. In this case the value ε_g does not depend on how far we are from the optimum active constraints set (variable cs), and $\epsilon(||\pi||_1)\}$ is a function that depends on the π vector computed in step (6) i) of the phase 2 algorithm. In the current implementation of the PPRN code $\epsilon(||\pi||_1)$ has been defined as:

$$\epsilon(||\pi||_1) = \max\{1, \frac{||\pi||_1}{\sqrt{mK + n + p}}\} \tag{29}$$

where m was the number of nodes, K the number of commodities, n the number of arcs and p the number of side constraints. The coefficient $\sqrt{mK + n + p}$ is an attempt at scaling the optimality tolerances depending on problem size.

The last two tests have been included for highly non-smooth functions where the four previous tests could mean very slow convergence. The first of these two tests (T_5) is inactive by default (the user can decide to activate it if he/she so desires) and can only be applied when we are far from the optimum. T_5 will become true if the first three tests, T_1, T_2 and T_3, gave the same result during $MAXsame$ consecutive iterations in the current subspace (where the value $MAXsame$ can be chosen by the user). The second one (T_6) controls whether the number of iterations in the current subspace $ncurrent$ is greater than a maximum allowable value $MAXcurrent$.

Once the six logical tests have been made, the code will consider that the optimum in the current subspace has been reached if the logical variable T is true, where T is defined as:

$$T := (T_1 \text{ and } T_2 \text{ and } T_3) \text{ or } T_4 \text{ or } T_5 \text{ or } T_6 \tag{30}$$

Thus, in step (1) i), the condition that is really verified as the criterion for going to step (3) is *“if T is true”* instead of *“If $||g_z|| \geq \epsilon_{g_z}$ ”*.

4.3.3. Choosing a nonbasic variable to become superbasic.

In step (2) ii) the process of choosing a nonbasic variable to enter the superbasic set was reduced to *“choose a suitable λ_q and the associated column N_q”*. In fact, the PPRN code implements a more elaborated algorithm for this point, which is most crucial since a bad choice or poor tolerances may mean slow convergence when finding the optimum active constraint set.

The following algorithm is the very sequence of steps in which point (2) ii) is expanded in the PPRN code. The algorithm uses a tolerance T_{λ_q} for choosing a good multiplier λ_q. At the beginning of phase 2 this tolerance is initialized with an arbitrary high value $T^0_{\lambda_q}$. The variable cs for knowing how far we are from the optimal active constraints set is also consulted and updated in this algorithm. The function $\epsilon(||\pi||_1)$ was defined in (29), and the tolerance T_{g_z} —to detect a sufficient reduction in the norm of the reduced gradient $||g_z||_\infty$— was already introduced in the previous subsection. The value ϵ_{opt}, chosen by the user, is the optimality precision required at the optimum point (by default $\epsilon_{opt} = 10^{-6}$).

0) At the beginning of phase 2 set $T_{\lambda_q} = T^0_{\lambda_q}$ and $cs =$ "far".
1) $T_{\lambda_q} = \max\{T_{\lambda_q}, 1.12 \cdot ||g_z||_\infty\}$.
2) Find the first λ_q such that $|\lambda_q| \geq T_{\lambda_q}$ or, if there is none, the greatest $|\lambda_q|$.
3) If $(|\lambda_q| \geq T_{\lambda_q})$ go to 7).
4) No multiplier satisfies the current tolerance T_{λ_q}.
   If $(|\lambda_q| \geq \epsilon_{opt}\epsilon(||\pi||_1))$ then
   i) $T_{\lambda_q} = \max\{|\lambda_q|/10, \epsilon_{opt}\epsilon(||\pi||_1)\}$.
   ii) Go to 7).
5) No multiplier is greater than the optimality tolerance $\epsilon_{opt}\epsilon(||\pi||_1)$. We are near the optimum.
   If $((cs =$ "near") or $(||g_z||_\infty \leq \epsilon_{opt}\epsilon(||\pi||_1)))$ then go to 8).
6) No multiplier is greater than the optimality tolerance and the point does not satisfy the optimality conditions. We adjust the tolerances for a more accurate optimization —probably the last— in the current subspace.
   i) $T_{g_z} = \dfrac{\min\{T_{g_z}, ||g_z||_\infty\}}{10}$.
   ii) If $(T_{g_z} < \epsilon_{opt}\epsilon(||\pi||_1)$ then
      - cs= "near".
      - $T_{g_z} = \epsilon_{opt}\epsilon(||\pi||_1)$.
   iii) Continue with step (3) of the phase 2 algorithm.
7) A suitable λ_q has been found:
   i) Update $||g_z||_\infty = \max\{|\lambda_q|, ||g_z||_\infty\}$.
   ii) Update $T_{g_z} = \eta^{cs}_{g_z}||g_z||_\infty$.
   iii) If $(cs =$ "near") then cs= "far".
   iv) Continue with step (2) iii) of the phase 2 algorithm.
8) The current point satisfies the optimality conditions. STOP: OPTIMAL SOLUTION FOUND.

The main idea of this algorithm is to use initially an arbitrary high tolerance T_{λ_q} and to reduce it when no multiplier can be found greater than this tolerance. This value is reduced until it reaches the optimality tolerance $\epsilon_{opt}\epsilon(||\pi||_1)$. When that happens it can be considered that the optimum constraints set has been found, and the tolerance T_{g_z} is adjusted for a final, more accurate optimization in the current optimum active constraints set. It must be noted that at step 1 we always choose the greater value between $1.12 \cdot ||g_z||_\infty$ and T_{λ_q} for finding a good λ_q, following the recommendation in [19]. This is done because the chosen λ_q will be added as a component of the new reduced gradient, which means that this value will be significant with respect to the rest of components of the current g_z.

4.3.4. Quasi-active superbasic variables.

When solving $Z^t H Z P_S = -g_z$, PPRN makes it possible to consider a special set of superbasic variables that will be referred to as "quasi-active superbasic variables". The quasi-active set $\mathcal{I}^+$ is defined as:

$$\mathcal{I}^+ = \{i : |x_{s_i}| < \varepsilon_q \text{ and } g_{z_i} < 0\} \cup \{i : |x_{s_i} - \overline{x_{s_i}}| < \varepsilon_q \text{ and } g_{z_i} > 0\} \tag{31}$$

x_s being the superbasic variables, $\overline{x_s}$ their upper bounds and g_z the reduced gradient. It will be considered that the remaining superbasic variables belong to the set $\mathcal{I}^-$. When the option for considering this kind of superbasics is active the code will solve $Z^tHZP_S = -g_z$ by performing

$$\begin{array}{|c|c|} \hline Z^tHZ_{\mathcal{I}^-} & 0 \\ \hline 0 & \mathbb{1}_{\mathcal{I}^+} \\ \hline \end{array} \quad \begin{array}{|c|} \hline P_{S_{\mathcal{I}^-}} \\ \hline P_{S_{\mathcal{I}^+}} \\ \hline \end{array} = \begin{array}{|c|} \hline -g_{z_{\mathcal{I}^-}} \\ \hline -g_{z_{\mathcal{I}^+}} \\ \hline \end{array} \tag{32}$$

where $M_{\mathcal{I}^*}$ is the vector (matrix) made of the i-th components (columns and rows) of M such that $i \in \mathcal{I}^*$. Thus the solution of (32) is directly

$$Z^tHZ_{\mathcal{I}^-}\ P_{S_{\mathcal{I}^-}} = -g_{z_{\mathcal{I}^-}} \tag{33}$$

$$P_{S_{\mathcal{I}^+}} = -g_{z_{\mathcal{I}^+}} \tag{34}$$

By definition of (31) it is clear that using the descent direction $P_{S_{\mathcal{I}^+}}$ computed at (34) means than the quasi-active superbasic variables will acquire a value outside their bound when updating the new point in step (5) ii). The motivation for using such a P_S is, then, to avoid possible descent directions found through $Z^tHZP_S = -g_z$ that could provide a zero value for α_{max} at step (4) i) of the phase 2 algorithm (that is, degenerate steps). The fact of avoiding these degenerate steps removes some cycling problems in the process of leaving and entering superbasic variables. The default option is not working with quasi-active variables. However, when a degenerate step is found due to a quasi-active variable, the option is automatically activated —only at next iteration— in order to avoid the initiation of cycling.

4.3.5. 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 the K spannings trees and the working matrix Q. During the pivotal operations the dimension of matrix Q can be modified, since $dim(Q) = |\mathcal{A}|$ (where $|\mathcal{A}|$ is the number of active mutual capacity and side constraints). 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 on the type of variable entering and leaving the basis, the following six cases can be observed (denoting by "E:–" the case of an entering variable and by "L:–" the case of a leaving variable):

• *E: slack–L: slack.* The row of Q associated with the entering slack is removed and replaced 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. $Dim(Q)$ must be updated as $dim(Q) - 1$.

• *E: slack–L: arc of k-th tree.* A complementary arc of the k-th commodity, e.g. the j-th complementary arc, having the leaving arc in its path P_j, must be found to replace the leaving arc in the k-th tree. This complementary arc will always exist (otherwise the basis would become singular). The row and column of Q associated with the entering slack and the j-th complementary arc are removed. $Dim(Q)$ must be updated as $dim(Q) - 1$.

• *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 a complementary arc — is also added to the working matrix. $Dim(Q)$ must be updated as $dim(Q) + 1$.

• *E: arc–L: complementary arc.* The column of Q associated with the leaving complementary arc is removed, and replaced by a column corresponding to the entering arc, which will become a complementary arc. $Dim(Q)$ is not modified.

• *E: arc–L: arc of k-th tree.* A complementary arc of the k-th commodity, e.g. the j-th complementary arc, having the leaving arc in its path P_j, is sought. If this arc is found, it will replace the leaving arc in the k-th tree, and the entering arc will become a complementary arc. If no complementary arc is found, then the entering arc will replace the leaving arc in the k-th tree. One of the two possibilities described will always happen, otherwise the basis would become nonsingular. $Dim(Q)$ is not modified.

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

5. Updating the working matrix.

The way in which the working matrix is handled is instrumental in ensuring the efficiency of the algorithm, since it is the only matrix to be factorized (together with matrix R that has been presented in the former section). Several tests have shown that the sparsity of Q is, in general, high (Q has less than 10% non zero elements). The current implementation of code PPRN performs a sparse LU decomposition of Q with partial pivoting allowing a choice between two ways of pre-reordering the matrix: applying either the P3 algorithm developed by Hellerman and Rarick [13] or a pre-reorder which attempts to put all the spikes at the end of the matrix. The latter pre-reorder is taken as default, since very good results have been obtained with it. Details of this subject can be found in [11].

An initial description of how to update this matrix was made by Kennington and Helgason in [17]. Two importants remarks should be made on the approach described there:

• It only considers the updating of the Q matrix with mutual capacity constraints. As mentioned above, the updating of Q in code PPRN has been 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 means that updating Q^{-1} is a costly operation if it is stored as a sparse matrix, since it is necessary to add or remove columns in a sparse structure. On the other hand, it seems inappropiate to store Q^{-1} as a dense matrix, given its high sparsity. This led one of the authors to develop an ad hoc and very efficient update of Q, instead of its inverse [5].

It is beyond the scope of this document to describe all the formulae required in the updating process, as they were developed in a previous work [5]. Nevertheless, a brief outline will be given here.

Let us consider that at iteration p the working matrix Q_p is recomputed (not merely updated), with dimension $dim(Q_p) = n_p$, and that it will not be newly recomputed until after i iterations (that is, until iteration $p+i$), where its dimension will be $dim(Q_{p+i}) = n_{p+i}$. Since the dimension of Q can only increase at most by a row and column at each iteration, it follows that $n_j \leq n_p + i,\ \forall\ j\ \ p \leq j \leq p+i$, $p+i$ being the maximum dimension of Q_j between iterations p and $p+i$. Thus the proposed procedure would be to work with an *extended matrix* $\overline{Q}_j$ at iterations $j,\ p \leq j \leq p+i$, where $\overline{Q}_j$ is defined as

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

Dimensions n_j and l_j of matrices Q_j and identity $\mathbb{1}$ satisfy $n_j + l_j = n_p + i$, i.e., the extended matrix $\overline{Q}_j$ has at every step the maximum dimension that Q_j can achieve between iterations p and $p+i$.

Thus the structure that will be updated will be that of the extended matrices $\overline{Q}_j$, even though the systems to be solved are systems $Q_j x_j = b_j$ and $x_j^t Q_j = b_j^t$. In fact these systems can be directly computed from $\overline{Q}_j$, using $\overline{x}_j$ and $\overline{b}_j$, which are extensions of x_j and b_j such that

$$\overline{x}_j = \begin{array}{c} n_j \\ \\ l_j \end{array} \begin{pmatrix} x_j \\ — \\ \alpha_j \end{pmatrix} \qquad \overline{b}_j = \begin{array}{c} n_j \\ \\ l_j \end{array} \begin{pmatrix} b_j \\ — \\ 0 \end{pmatrix}$$

Then

$$\overline{Q}_j \overline{x}_j = \overline{b}_j \Longleftrightarrow \begin{pmatrix} Q_j & 0 \\ 0 & \mathbb{1} \end{pmatrix} \begin{pmatrix} x_j \\ \alpha_j \end{pmatrix} = \begin{pmatrix} b_j \\ 0 \end{pmatrix} \Longleftrightarrow \begin{cases} \alpha_j = 0 \\ \boxed{Q_j x_j = b_j} \end{cases}$$

the marked expression being the desired result. Analagously $x_j^t Q_j = b_j^t$ can be solved in the same way.

The increase (decrease) in the number of rows/columns in Q_j can now be treated through direct pre and post-multiplications by eta and permutation matrices, implying that n_j will become

$n_j + 1$ ($n_j - 1$) and the identity submatrix in the bottom right part of $\overline{Q}_j$ will lose (gain) a unit in dimension. Therefore, it is clear than $\overline{Q}_{j+1}$ can be updated from $\overline{Q}_j$ through $\overline{Q}_{j+1} = E_j \overline{Q}_j F_j$, where E_j and F_j are made up of eta and permutation matrices. Recursively it is possible to write $\overline{Q}_{j+1} = E_j E_{j-1} \overline{Q}_{j-1} F_{j-1} F_j$, and so on, until reaching iteration p where the working matrix was recomputed. Thus it can be written in a general form $\forall\ j, \quad p \leq j < p + i$, $\overline{Q}_j = E \overline{Q}_p F$, where $E = \prod_{l=1}^{j-p} E_{j-l}$ and $F = \prod_{l=1}^{j-p} E_{p+l-1}$. So the solution to system $\overline{Q}_j \overline{x}_j = E \overline{Q}_p F \overline{x}_j = \overline{b}_j$ can be computed as follows:

$$\begin{aligned} \overline{Q}_p F \overline{x}_j &= E^{-1} \overline{b}_j \\ \overline{Q}_p z_j &= E^{-1} \overline{b}_j \ , \quad \text{where } z_j = F \overline{x}_j \\ z_j &= \overline{Q}_p^{-1} E^{-1} \overline{b}_j \\ \text{And finally} \quad \overline{x}_j &= F^{-1} z_j \end{aligned}$$

Since Q_p has been factorized when recomputed, to solve the required systems E and F must simply be inverted at each iteration. Nevertheless, the inverses of E and F are directly computed, since they are nothing but products of eta and permutation matrices. In fact, code PPRN directly stores the inverses of E and F, which continue to be products of eta and permutation matrices.

6. Test problems employed.

Two types of problems have been employed to test the performance of the code. The first type are expensive (in computation time) artificial problems, while the second arises from the fields of long and short-term hydro-thermal coordination of electricity generation, electrical network expansion planning and traffic assignment. The following subsections will describe the main features of such problems, and the particular instances employed in testing the code.

6.1. Artificial problems.

Three different artificial objective functions have been tested. However, the networks used are real and have been obtained from long-term hydro-thermal scheduling problems that will be described later. The reason for choosing such networks instead of others obtained from different network generators —e.g., those distributed for the First DIMACS International Algorithm Implementation Challenge [10]— is that, for problems of the same size, the hydro-electrical networks seem to be much more difficult to solve than the randomly obtained ones (this can be observed in a previous work by the authors for linear objective functions [8]).

The first two artificial objective functions are simple convex functions defined by:

$$h^{(1)}(X_1, X_2, \ldots, X_K) = \sum_{k=1}^{K} \sum_{i=1}^{n} x_{ki}^2 \tag{35}$$

$$h^{(2)}(X_1, X_2, \ldots, X_K) = \sum_{k=1}^{K} \sum_{i=1}^{n} x_{ki}^4 \tag{36}$$

x_{ki} being the flow of the i-th arc and k-th commodity. The third objective function is derived from that described in [25], and is defined as:

$$h^{(3)}(X_1, X_2, \ldots, X_K) = \sum_{k=1}^{K} F_k(X_k) \tag{37}$$

where

$$F_k(X_k) = \frac{1}{c_1} \sum_{i=1}^{n} x_{ki}^2 + \frac{1}{c_2} \left(\sum_{i=1}^{n-1} \sqrt{1 + x_{ki}^2 + (x_{ki} - x_{k,i+1})^2} + \frac{1}{c_3} \Big(10 + \sum_{i=1}^{n} (-1)^i x_{ki} \Big)^4 \right) \tag{38}$$

TABLE I. LONG-TERM HYDRO-THERMAL SCHEDULING PROBLEMS.

test	K	#s.c.	nodes	arcs	rows A	columns$A$
$P_1^{(j)}$	4	12	37	153	313	777
$P_2^{(j)}$	4	3	99	315	714	1578
$P_3^{(j)}$	4	3	685	2141	4884	10708

$c_1, c_2, c_3 \in \mathbb{R}$ (in the executions performed $c_1 = 1000$, $c_2 = 1000$ and $c_3 = 1200$). Despite their simplicity, these three objective functions have solutions with a high number of superbasic variables, which increases the execution time considerably.

Table I shows the dimensions of the long-term hydro-thermal scheduling networks used with these artificial objective functions. The first column, "test", is the name given to the test problem, where the subindex $_i$ identifies the network used and the supraindex $^{(j)}, j = 1 \div 3$ refers to the artificial objective function used ((35), (36) or (37) respectively). Column K denotes the number of commodities considered in the test (usually four commodities are used for these long-term hydro-thermal networks). Column "#s.c." shows the number of side constraints considered in the problem. Columns "nodes" and "arcs" give the number of nodes and arcs of the single-commodity network. Finally, columns "rows A" and "columns A" give the dimensions of the constraint matrix of the multicommodity network problem to be solved.

6.2. Hydro-thermal scheduling problems.

The second type of problems has been obtained as instances of long and short-term hydro-thermal scheduling of electricity generation according to the models proposed in [21] and [15] (where a comprehensive explanation of the models can be found).

6.2.1. Long-term hydro-thermal scheduling problems.

The solution to the long-term hydrogeneration optimization problem in a power utility with thermal and hydro power stations indicates how to distribute throughout a long period of time the hydroelectric generation in each reservoir of the reservoir system in order that the expected cost of thermal generation over the period under consideration be a minimum. In long-term hydrogeneration optimization the availability of a thermal plant, the demand of electricity and the water inflows in the reservoirs are not deterministic but only known as probability distributions. The probabilistic demand and thermal plant availability can be modelled through functions of probabilistic production cost versus hydrogeneration. The problem left is that of minimizing the sum of the expected generation cost of each interval, taking into account that the water inflows at each interval are to be regarded as stochastic.

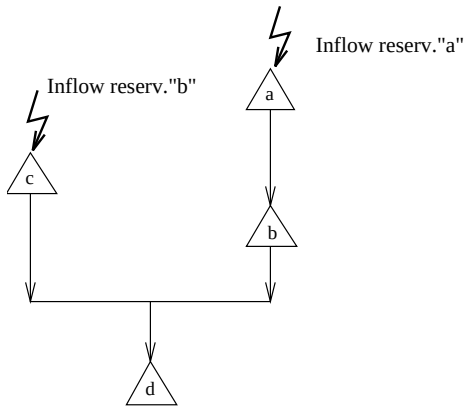

Fig. 1 Example of a four-reservoir system.

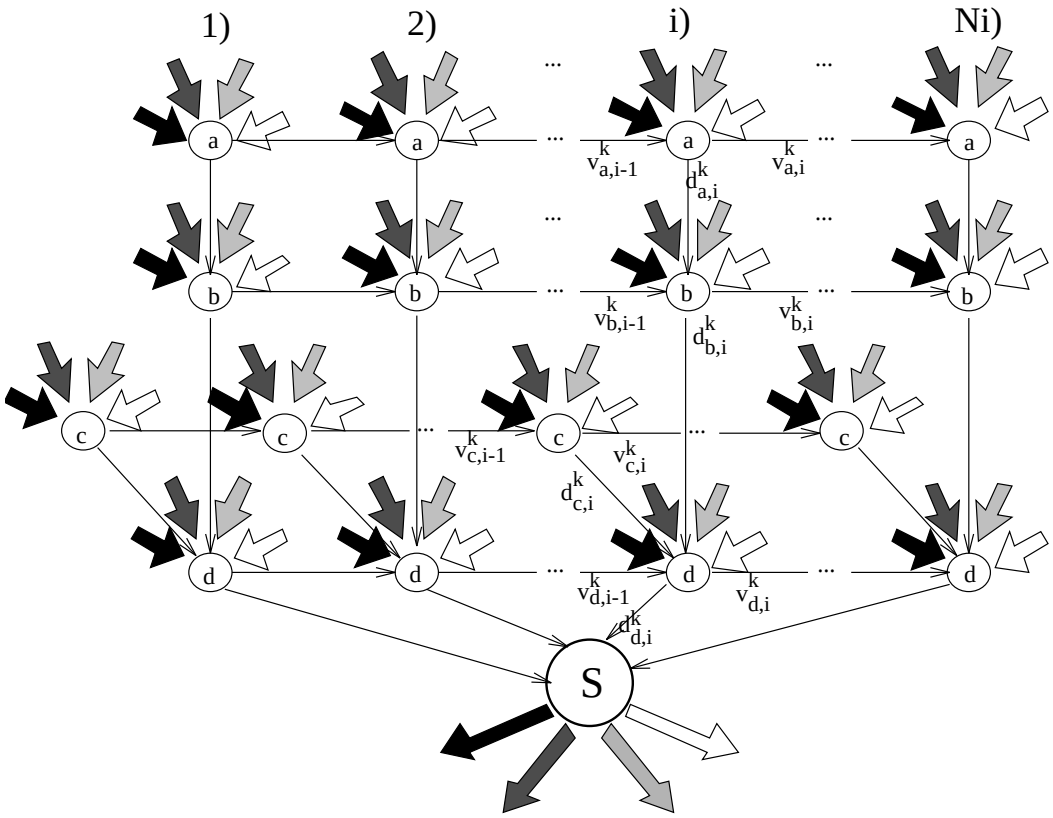

Fig. 2 Replicated multicommodity water network.

It is important to optimize taking into account the whole set of probable water inflows. A model that considers several types of water according to their probability of occurrence must be employed. This can be made approximating the probability density function of water inflow, in each reservoir over each time interval considered, by a block probability density function with $K-1$ rectangular blocks of probability areas $p_1, \ldots, p_{K-1}$, $\sum_{k=1}^{K-1} p_k = 1$. The commodities of the problem are thus amounts of water corresponding to different probabilities. The first commodity is the deterministic water (from the origin of the probability density function to the axis position where the first rectangular block starts), the second commodity is the water under the p_1 block, and so up to the K-th commodity under the block of area p_{K-1}. An expression of the expected cost of generation at each interval in terms of amounts of the different types of water dedicated to storage, generation, pumping or spillage at each reservoir has been derived, and the objective function to be minimized is the sum of the expected costs of all intervals, which can be considered highly nonlinear (more details can be found in [21]).

The characteristics of the tests problems used for this model were already described in table I. We shall refer to the concrete problems with this objective function as $\mathrm{P}_i^{(4)}$ (since the supraindex $^{(j)}, j = 1 \div 3$, are related to the artificial objective functions).

Given a reservoir system such that of Fig. 1, a replicated hydro network of it, as shown in Fig. 2, is employed to represent the water balance constraints over the intervals. Total reservoir volumes and maximum discharges are imposed as mutual capacity constraints to multicommodity flows on volume arcs (the horizontal ones) and discharge arcs (the vertical or slanted ones).

6.2.2. Short-term hydro-thermal scheduling problems.

The solution to the short-term hydrot-hermal coordination indicates how to distribute the hydroelectric generation (cost-free) in each reservoir of the reservoir system and how to allocate generation to thermal units committed to operate over a short period of time so that the fuel expenditure during the period is minimized. Load and spinning reserve constraints tie up hydro and thermal generation.

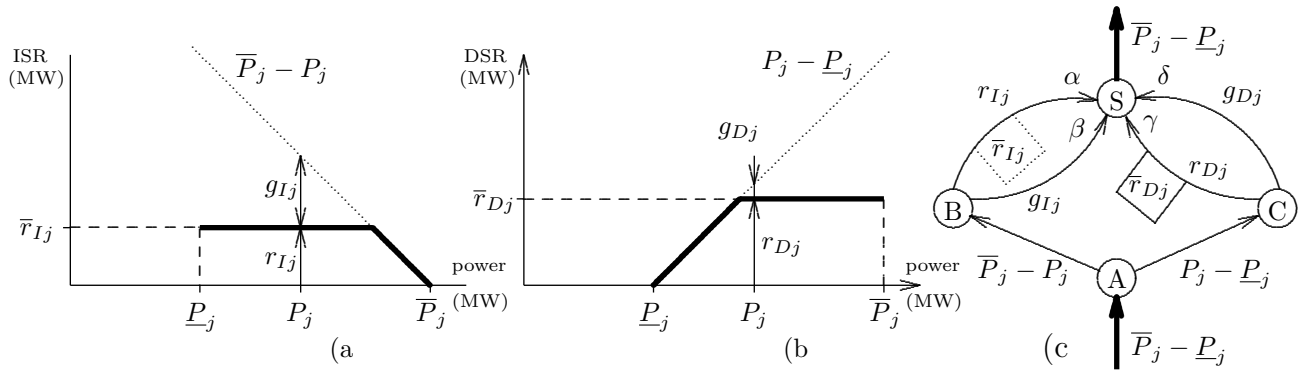

Fig. 3 a) Incremental Spinning Reserve (ISR) function of the j^{th} thermal unit.
b) Decremental Spinning Reserve (DSR) function of the j^{th} thermal unit.
c) Network representation of thermal generation variables.

The network model usually employed for short-term hydrogeneration optimization has been extended to include thermal units in a new and undecoupled way [15], imposing single load and spinning reserve constraints on both hydro and thermal generation and minimizing directly thermal production costs without decoupling the problem into hydro and thermal subproblems. When constraints are added so that hydrogeneration plus thermal generation match the load and satisfy a spinning reserve requirement at each interval, pure network flow algorithms are no longer applicable; however, if these constraints are linearized, efficient specialised algorithms for optimizing network flows with linear side constraints can be employed. Hydrogeneration has thus been linearized in terms of the network variables (initial and final volumes and discharges at each reservoir) in order that all side constraints are linear.

The basis of the thermal generation model used is the following. Let P_j be the power output of the j^{th} thermal unit and let $\overline{P}_j$ and $\underline{P}_j$ be its upper and lower operating limits: $\underline{P}_j \leq P_j \leq \overline{P}_j$. The incremental spinning reserve (ISR) r_{Ij} of unit "j" is the amount of power by which the current generation P_j can be increased within a given time lapse. $\overline{r}_{Ij}$ is the maximum possible ISR. Similarly, the decremental spinning reserve (DSR) r_{Dj} is the amount of power by which one can decrease the current power P_j. Its maximum value will be $\overline{r}_{Dj}$. The ISR r_{Ij} and the DSR r_{Dj} of the j^{th} unit satisfy: $r_{Ij} = min\{\overline{r}_{Ij}, \overline{P}_j - P_j\}$ and $r_{Dj} = min\{\overline{r}_{Dj}, P_j - \underline{P}_j\}$, which is represented by the thick lines of Fig. 3a) and 3b)

At power P_j we have an ISR r_{Ij} and a DSR r_{Dj}, and there is a power gap $g_{Ij} \geq 0$ from the ISR r_{Ij} to $\overline{P}_j - P_j$ so that $r_{Ij} + g_{Ij} = \overline{P}_j - P_j$ and also a power gap $g_{Dj} \geq 0$ between the DSR r_{Dj} and $P_j - \underline{P}_j$ thus $r_{Dj} + g_{Dj} = P_j - \underline{P}_j$. The generation of a thermal unit, its ISR and DSR, the associated power gaps, and its operating limits lend themselves well to being modeled through network flows as shown in Fig. 3c). An upper limit of $\overline{r}_{Ij}$ on arc α must be imposed to prevent the reserve from getting over its limit. To assure that flows on arcs α and β are like the variables in Fig. 3a), a small positive weighing cost on the flow of arc β must be placed while arc α has zero cost. Similarly arc γ will have zero cost while arc δ will have a small positive cost $w_{\beta\delta}$ like arc β in order to divert as much flow as possible from arc β and δ to arcs α and γ respectively. The flow $P_j - \underline{P}_j$ from node A to node C is associated to the generation cost to be minimized

The model just described for one generator can be extended to all committed thermal units at a given interval "i". A single network will represent the generation, ISR, DSR and power gaps of all committed units. The networks of each single unit can share the sink node S. The network described would correspond to the thermal generation and spinning reserve for a single interval "i", and will be referred to as therm.net "i". One such network, connected to a single sink node S, must be considered for each interval. A load constraint and an ISR and a DSR requirement, for each interval, are linear side constraints.

The resulting network model is a single commodity one and the primal partitioning algorithm described has been employed to find its optimum when minimizing a nonlinear cost function of generation. A complete description of this nonlinear objective function can be found in [15].

TABLE II. SHORT-TERM HYDRO-THERMAL SCHEDULING PROBLEMS.

test	K	#s.c.	nodes	arcs	rows A	columns$A$
$P_1^{(5)}$	1	528	1345	4416	1873	4944
$P_2^{(5)}$	1	840	1975	6048	2815	6888
$P_3^{(5)}$	1	840	2479	8064	3319	8904
$P_4^{(5)}$	1	1848	4741	15600	2289	17448

Table II shows the characteristics of the test problems employed with the short-therm objective function. The meaning of the columns is the same that was previously detailed. In all cases a single-commodity network problem is obtained.

6.3. Electrical network expansion planning problems.

The network expansion problem addressed to is that of finding the lower cost expansion of an existing transmission network in order to account for load demand growth and increasing power generation availability. The investment cost of new transmission and distribution lines and power transformers from some power source points to load points and the operating cost expressed as estimated cost of active power losses on the existing plus new network should be minimized. Problem data are: a set of generating points with their geographical coordinates, generating capacity and voltage level, a set of load points with their geographical coordinates, load value and voltage level, a set of existing lines and transformers uniting some generation and load points and a set of functions of change of investment cost of lines and transformers at different voltage levels with power rating. Finally a price to evaluate power losses, an estimation of the usage rate (in percentage of time over a year) of the transmission and distribution equipment, the interest rate to be considered and the investment pay-off time length.

It will be assumed that, although all loads have an specified voltage level (usually the lowest), power can be delivered to them at any voltage level equal or higher than the load voltage. This implies that at all load points all voltage levels envisaged equal or higher than the load voltage should be considered. The user may also decide which voltage levels to consider in generation sources. Thus at each load and generation point all transformers from any voltage to all lower levels considered will be taken into account.

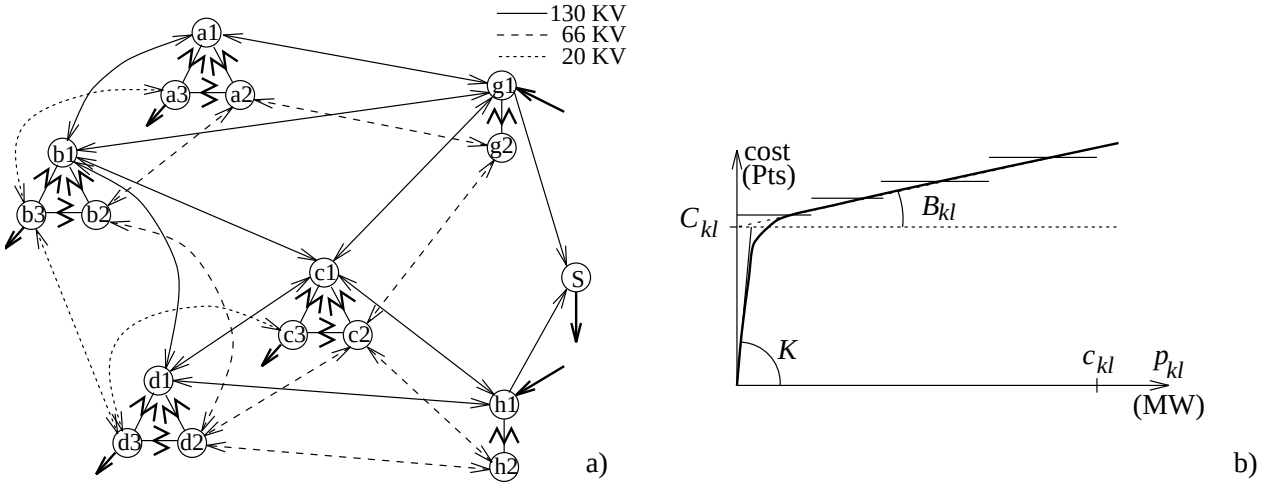

Fig. 4. a) Example of superabundant network from generating sources g and h to load nodes a, b, c and d at three voltage levels
b) Real cost function of transmission line or transformer for several structural options, in terms of power to be carried and continuous approximation to real cost function.

TABLE III. ELECTRICAL NETWORK EXPANSION PLANNING PROBLEMS.

test	#volt	K	#s.c.	nodes	arcs	rows A	columns$A$
$P_1^{(6)}$	3	20	4	73	508	1972	10672
$P_2^{(6)}$	4	20	4	92	508	2352	10672
$P_3^{(6)}$	4	20	4	92	560	2404	11764

In what regards to transmission lines, if any load or generation point at a given voltage level would be connected to the rest of the load or generation points at the same voltage, we would have an enormous number of variables to consider. In order to have a reasonable number of variables rules have been defined to decide which lines to take into account at each voltage level. The set of lines and transformers considered will be referred to as superabundant network (see Fig 4a) and it should include the optimum network subset. An optimization process will try to find out this optimum network. Taking a network with the same nodes as the superabundant network and taking as arcs its lines and transformers the variables to be optimized will be the flows on the arcs. The arcs with zero flow at the solution are not worth building up whereas those with flow bigger than zero are to be constructed. The flow on the arcs of the networks will be active power and each arc between two nodes is duplicated with one arc in each direction to account for nonnegative flows representing power flow in either direction

The objective function considered refers to the installation cost of equipment and a continuous function as that in Fig. 4b) has been considered for each line or transformer. This function has a linear part plus an step type cost modeled by one minus a decreasing exponential times the basic investment cost. Transmission losses as a quadratic function of power flow are also part of the objective function.

Some constraints will have to be imposed on the variables (power flows) in order that the solution obtained resembles the electrical flows on the lines and transformers to be installed. This is possible by considering for the constraints the parameters of the the d.c. aproximation. The flow balance equations in the network ensure the compliance of Kirchhoff's current law.

Security constraints are usually considered either limiting the bus voltage magnitudes and bus swing angles, or protecting the network against single line contingencies. In this model special security constraints have been included: that at least two lines at the same voltage level must carry the load to a given load point and that there should be disjoint paths from each load point to at least two different generation points. Should these security constraints be not included, a radial type network expansion would be the optimal solution.

These security constraints can not be imposed with the formulation described so far but can be easily expressed through the following multicommodity model. The power corresponding to each load will be considered a different commodity, so there will be as many commodities flowing on the network as load nodes. Instead of having a nonlinear single commodity network flow problem with side constraints we will have to solve a nonlinear multicommodity network flow problem with side constraints. According to the classical multicommodity formulation there will be an specific arc capacity for each commodity and a mutual capacity constraint for each arc, which will coincide with the arc capacity considered all along.

Imposing limits to the single commodity arc capacities and to the single commodity generation point injections it is possible to satisfy that each load is delivered through more than one line and that it is supplied from more than one generation point. To get that a given load is delivered through lines at the same voltage level a multicommodity product penalty term can be added to the objective function. A comprehensive description of this model can be found in [22].

Table III shows the characteristics of the problem instances employed with this test function, all of them corresponding to the same real case but modifying the number of voltages and dimensions of the superabundant network generated by the heuristic. The column "#volt" gives the number of voltages at which the network expansion will be considered. The meaning of the remaining columns is the same as that in former tables.

6.4. Traffic assignment problems.

The static user equilibrium traffic assignment problem with inelastic demand and separable link cost functions was first formulated as an optimization problem by Beckmann [3]. Given I origins and

TABLE IV. TRAFFIC ASSIGNMENT PROBLEM.

test	K	#s.c.	nodes	arcs	rows A	columns$A$
$P_1^{(7)}$	16	0	182	309	3221	5253

J destinations, the static user equilibrium problem consists of finding a flow distribution amongst paths p_{ij} joining origin $i \in I$ with destination $j \in J$ such that all paths with nonzero flow assigned to them have equal cost u_{ij}^* and flows without flow assigned to them result in a cost $(u_{ij})_p \geq u_{ij}^*$. This problem can be formulated as a multicommodity problem, considering that each commodity k_{ij} is the traffic flow leaving from origin $i \in I$ and arriving at destination $j \in J$. However, this can mean having a great number of commodities ($K = I \times J$, K being the number of commodities). Another possibility is grouping the traffic flows by origin or destination nodes, thus having $K = I$ or $K = J$ depending on the choice, with a great reduction in the number of commodities and variables. In the executions made the grouping was according to origin nodes.

The objective function to minimize is the sum for each arc a of the network of a cost function defined in terms of the arc traffic volume v_a. This function can be stated as:

$$h^{(7)}(V_1, V_2, \ldots, V_K) = \sum_{\forall a} \int_0^{v_a} C_a(v)dv \tag{39}$$

where $C_a(v)$ is the instantaneous value cost function for a given volume, and $v_a = \sum_{\forall k} V_{k_a}$. By solving the integral function, (39) can be written as:

$$h^{(7)}(V_1, V_2, \ldots, V_K) = \sum_{\forall a} c_{1_a} + c_{2_a} l_a \left(1 + \alpha_a \Big(\frac{v_a + c_{3_a}}{c_{4_a}}\Big)^{\beta_a}\right) \tag{40}$$

l_a being the length of the line, and c_{i_a}, α_a and β_a some line-dependent coefficients. The expression (40) has been used as the assignment problem objective function.

It must be said that the approach here considered for solving this problem is slightly different from those usually employed and described in the literature on this field [24]. Along with some heuristic methodologies, the Frank-Wolfe and Simplicial Decomposition algorithms have been those most extensively used. The main advantage of these algorithms is that they make it possible to deal with large networks, as problem structure is greatly exploited. However, they have slow convergence and finding accurate solutions becomes prohibitive. The formulation of the problem and the algorithm employed in this work overcomes this difficulty, even though they cannot deal with very large networks (since each arc means having K volumes, one for each commodity, thus obtaining a final problem with too many variables).

Only one problem test has been used with this objective function. This problem corresponds to a subnetwork of the city of Barcelona (thus being a real case network). The dimensions of this problem are shown in table IV.

7. Computational results.

This section will present the results obtained using the PPRN code with the test problems described formerly. The PPRN code has been compared with the general-purpose package MINOS 5.3 [20], since no other specialized code for solving the NMPC problem is known. The default values have been used for all the adjustable parameters in both codes. All runs were carried out on a Sun Sparc 10/41 (one CPU), having a risc-based architecture, with 40MHz clock, ≈100Mips and ≈20Mflops CPU, and 32Mbytes of main memory.

Tables V–XI show the results obtained for each group of test problems presented in the previous section, with both the PPRN code and MINOS 5.3. For PPRN, information disclosed includes rows:

- "Ph.0": number of iterations at phase 0.
- "Ph.1": number of iterations at phase 1.
- "Ph.2": number of iterations at phase 2.
- "$h(x^*)$": optimum objective function value.

TABLE V. RESULTS FOR THE ARTIFICIAL OBJECTIVE FUNCTION $h^{(1)}(x)$.

	$P_1^{(1)}$		$P_2^{(1)}$		$P_3^{(1)}$	
	PPRN	MINOS	PPRN	MINOS	PPRN	MINOS
Ph.0	107	—	460	—	4083	—
Ph.1	63	163	226	617	2896	16598
Ph.2	654	687	1459	1753	9458	12401
$h(x^*)$	285515.68	285515.68	867915.88	867915.87	210894646.82	210894647.06
$\|\mathcal{A}\|$	34	—	87	—	810	—
s	272	272	544	544	2723	2727
CPU.sec	10.4	17.0	87.2	139.7	3086.7	23579.6
ϵ^*_{opt}	$7.7{\cdot}10^{-7}$	$9.9{\cdot}10^{-8}$	$9.2{\cdot}10^{-7}$	$1.0{\cdot}10^{-7}$	$5.3{\cdot}10^{-7}$	$1.6{\cdot}10^{-7}$
$\#h(x)$	1200	1271	3437	2798	59198	18541
T.It 1	1.10	3.41	2.31	11.40	21.31	24.28
T.It 2	15.02	23.93	58.69	86.70	315.47	1868.88
T.It $2-h$	14.72	23.63	57.91	86.10	295.21	1864.04

TABLE VI. RESULTS FOR THE ARTIFICIAL OBJECTIVE FUNCTION $h^{(2)}(x)$.

	$P_1^{(2)}$		$P_2^{(2)}$		$P_3^{(2)}$	
	PPRN	MINOS	PPRN	MINOS	PPRN	MINOS
Ph.0	107	—	460	—	4083	—
Ph.1	63	163	226	689	2896	16598
Ph.2	845	1160	2127	2297	15463	17019
$h(x^*)$	$3.9718{\cdot}10^{10}$	$3.9718{\cdot}10^{10}$	$6.0465{\cdot}10^{10}$	$6.0465{\cdot}10^{10}$	$1.9431{\cdot}10^{14}$	$1.9431{\cdot}10^{14}$
$\|\mathcal{A}\|$	34	—	95	—	788	—
s	211	233	449	449	1238	1900
CPU.sec	10.4	22.5	83.1	118.4	5823.3	16524.7
ϵ^*_{opt}	$4.8{\cdot}10^{-7}$	$2.3{\cdot}10^{-7}$	$1.8{\cdot}10^{-7}$	$2.5{\cdot}10^{-8}$	$5.0{\cdot}10^{-7}$	$7.7{\cdot}10^{-8}$
$\#h(x)$	1780	2299	4010	4320	152579	24174
T.It 1	1.10	3.48	2.31	10.39	20.65	73.56
T.It 2	11.18	18.90	37.46	48.42	367.96	899.21
T.It $2-h$	10.55	18.31	36.43	47.40	330.70	893.84

• "$|\mathcal{A}|$": number of active mutual capacity and side constraints at the optimizer (i.e., the dimension of the working matrix).
• "s": number of superbasic variables at the optimum point.
• "CPU sec.": CPU seconds spent by the execution.
• "ϵ^*_{opt}": optimality precision of the solution point, where $\epsilon^*_{opt} = ||g^*_z||_\infty/\epsilon(||\pi^*||_1)$. In all cases this value should be less than 10^{-6}, which is the default value.
• "$\#h(x)$": number of objective function evaluations.
• "T.It 1": time per phase 1 iteration (in milliseconds).
• "T.It 2": time per phase 2 iteration (in milliseconds).
• "T.It $2-h$": time per phase 2 iteration excluding the time spent evaluating the objective function (in milliseconds).
For MINOS 5.3, the same information is given excluding rows "Ph.0" and "$|\mathcal{A}|$". The values "T.It 1", "T.It 2" and "T.It $2-h$" were computed exactly in the PPRN executions, whereas in the MINOS runs they had to be approximated (with the result that some noise could be present in the results exposed).

TABLE VII. RESULTS FOR THE ARTIFICIAL OBJECTIVE FUNCTION $h^{(3)}(x)$.

	$P_1^{(3)}$		$P_2^{(3)}$		$P_3^{(3)}$	
	PPRN	MINOS	PPRN	MINOS	PPRN	MINOS
Ph.0	175	—	460	—	4083	—
Ph.1	74	256	226	689	2896	16598
Ph.2	1640	1638	3539	3772	18289	28314
$h(x^*)$	971.69	971.69	891.69	891.69	212682.61	212682.61
$\|\mathcal{A}\|$	27	—	81	—	713	—
s	266	266	546	546	1927	1929
CPU.sec	30.9	34.4	236.0	292.3	6831.2	19398.0
ϵ^*_{opt}	$1.5{\cdot}10^{-7}$	$2.8{\cdot}10^{-9}$	$4.8{\cdot}10^{-7}$	$4.1{\cdot}10^{-8}$	$6.1{\cdot}10^{-7}$	$7.4{\cdot}10^{-8}$
$\#h(x)$	5851	3666	14337	8212	139047	47686
T.It 1	1.08	6.78	2.16	6.37	17.92	73.45
T.It 2	17.75	19.94	65.37	76.32	367.16	642.04
T.It 2$-h$	14.32	17.79	54.99	70.74	235.49	612.87

TABLE VIII. RESULTS FOR THE LONG-TERM HYDRO-THERMAL SCHEDULING PROBLEM.

	$P_1^{(4)}$		$P_2^{(4)}$		$P_3^{(4)}$	
	PPRN	MINOS	PPRN	MINOS	PPRN	MINOS
Ph.0	175	—	460	—	4083	—
Ph.1	74	261	226	574	2813	16069
Ph.2	395	202	4457	5092	14186	16569
$h(x^*)$	$-3.7747{\cdot}10^{12}$	$-3.7747{\cdot}10^{12}$	$1.0792{\cdot}10^{8}$	$1.2228{\cdot}10^{8}$	$-7.9171{\cdot}10^{9}$‡	$-6.7860{\cdot}10^{9}$‡
$\|\mathcal{A}\|$	51	—	99	—	642	—
s	2	2	260	69	112	118
CPU.sec	3.2	3.9	183.1	213.9	2504.1	4284.9
ϵ^*_{opt}	$4.7{\cdot}10^{-8}$	$8.0{\cdot}10^{-8}$	$3.1{\cdot}10^{-3}$†	$1.7{\cdot}10^{-2}$ †	$7.5{\cdot}10^{-8}$	$3.8{\cdot}10^{-10}$
$\#h(x)$	432	211	12379	11699	21068	20559
T.It 1	0.94	4.94	2.57	9.12	17.85	72.09
T.It 2	6.60	12.91	44.12	40.97	169.98	188.68
T.It 2$-h$	2.60	9.09	8.77	11.79	46.10	85.15

† The required optimality tolerance $\epsilon_{opt} = 10^{-6}$ could not be achieved.

‡ Different local minima were reached.

Some comments should be made about the results presented. In some problems ($P_3^{(4)}$, $P_1^{(6)}$, $P_2^{(6)}$ and $P_3^{(6)}$) PPRN and MINOS obtained different solutions. This is due to the high nonconvexity of the objective functions for the long-term hydro-thermal and network expansion models, which permits many local optimum points. Thus, to compare the behavior of both codes for these problems, attention must be payed to the CPU time per iteration (for each phase), instead of the total CPU time spent in the run.

In all runs the optimality tolerance required was $\epsilon_{opt} = 10^{-6}$. Only for test problem P_2^4 could this tolerance not be achieved for both codes, due to the nonlinearities in the long-term hydro-thermal objective function. In this case, the PPRN code reduced ϵ^*_{opt} more than the MINOS package, thus reaching a best optimum point (this is why both codes have a different objective function value, rather than considering different local minima).

As stated previously, PPRN and MINOS were executed with the default options. This meant that the PPRN code computed $Z^t H Z P_S = -g_z$ using the quasi-Newton methodology while $s \leq 500$

TABLE IX. RESULTS FOR THE SHORT-TERM HYDRO-THERMAL SCHEDULING PROBLEM.

	$P_1^{(5)}$		$P_2^{(5)}$		$P_3^{(5)}$		$P_4^{(5)}$	
	PPRN	MINOS	PPRN	MINOS	PPRN	MINOS	PPRN	MINOS
Ph.0	3419	—	5209	—	7044	—	11650	—
Ph.1	1177	2653	915	2905	1216	2965	3341	12257
Ph.2	1692	2522	1608	2299	1850	2455	6768	7507
$h(x^*)$	0.4009	0.4009	0.8715	0.8715	0.3844	0.3844	1.0920	1.0917 †
$\|\mathcal{A}\|$	470	—	530	—	510	—	1525	—
s	66	66	335	338	288	296	230	231
CPU.sec	110.2	226.5	156.7	357.8	217.3	485.5	2026.0	3316.9
ϵ^*_{opt}	$7.4{\cdot}10^{-13}$	$1.9{\cdot}10^{-11}$	$4.1{\cdot}10^{-7}$	$5.8{\cdot}10^{-8}$	$8.4{\cdot}10^{-11}$	$1.0{\cdot}10^{-17}$	$4.5{\cdot}10^{-7}$	$1.2{\cdot}10^{-8}$
$\#h(x)$	2036	6219	2583	6359	2714	6897	9821	19092
T.It 1	19.47	28.79	27.85	38.52	31.58	58.46	120.87	128.62
T.It 2	45.89	59.52	71.22	106.94	91.05	127.14	230.58	231.83
T.It $2{-}h$	41.19	49.91	63.07	92.92	79.85	105.69	208.56	193.25

† In this execution the "feasibility tolerance" parameter of the MINOS package was increased considerably to permit obtaining a feasible solution. The different value $h(x^*)$ for MINOS and PPRN can be due to this fact.

TABLE X. RESULTS FOR THE ELECTRICAL NETWORK EXPANSION PLANNING PROBLEMS.

	$P_1^{(6)}$		$P_2^{(6)}$		$P_3^{(6)}$	
	PPRN	MINOS	PPRN	MINOS	PPRN	MINOS
Ph.0	303	—	274	—	312	—
Ph.1	11	147	0	86	0	84
Ph.2	936	1331	985	829	860	1217
$h(x^*)$	1072.7775 †	1198.2819 †	1107.6211 †	1104.9917 †	996.0407 †	929.9943 †
$\|\mathcal{A}\|$	1	—	1	—	1	—
s	1	1	0	0	1	1
CPU.sec	15.5	107.7	12.0	80.1	14.0	115.0
ϵ^*_{opt}	$1.1{\cdot}10^{-9}$	$4.9{\cdot}10^{-9}$	0.0	0.0	0.0	0.0
$\#h(x)$	106	1028	29	839	45	1231
T.It 1	4.54	53.58	—	64.14	—	60.06
T.It 2	9.56	74.99	8.29	89.96	10.88	90.34
T.It $2{-}h$	8.97	70.99	8.02	83.22	10.44	81.80

† Different local minima were reached.

(s being the number of superbasic variables) and changed to the truncated-Newton algorithm when $s > 500$. On the other hand, the MINOS package always performs a quasi-Newton update. This fact is instrumental in the performance of the test problems $P_3^{(1)}$, $P_3^{(2)}$ and $P_3^{(3)}$, where the number of superbasic variables at the optimum is very great. In these three cases the time spent by PPRN is much less than that required by MINOS, even though the PPRN code performs many more objective function evaluations (#h(x)) —since it is using the truncated-Newton algorithm. Thus, it can be concluded that the different behavior of both codes in these examples is mainly due to the different algorithm used for computing the superbasic descent direction P_S when the number of superbasic variables is very high, and that, clearly, the truncated-Newton algorithm seems to be much more efficient than the quasi-Newton update in such cases.

From tables V–XI some interesting conclusions can be drawn. These are shown in Table XII, where each column means:

TABLE XI. RESULTS FOR THE TRAFFIC ASSIGNMENT PROBLEM.

	$P_1^{(7)}$	
	PPRN	MINOS
Ph.0	869	—
Ph.1	78	660
Ph.2	460	526
$h(x^*)$	288.9697	288.9697
$\|\mathcal{A}\|$	1	—
s	9	9
CPU.sec	11.6	69.6
ϵ^*_{opt}	$1.3{\cdot}10^{-10}$	$2.7{\cdot}10^{-8}$
$\#h(x)$	258	657
T.It 1	6.66	50.88
T.It 2	19.60	68.47
T.It $2{-}h$	16.93	62.52

TABLE XII. TIME COMPARISON BETWEEN MINOS AND PPRN.

test	T.It 1 ratio	T.It 2 ratio	T.It $2{-}h$ ratio	time ratio	test	T.It 1 ratio	T.It 2 ratio	T.It $2{-}h$ ratio	time ratio
$P_1^{(1)}$	3.10	1.59	1.60	1.63	$P_2^{(4)}$	3.54	0.92	1.34	1.16
$P_2^{(1)}$	4.93	1.47	1.48	1.60	$P_3^{(4)}$	4.03	1.11	1.84	1.71
$P_3^{(1)}$	1.13	5.92	6.31	7.63	$P_1^{(5)}$	1.47	1.29	1.21	2.05
$P_1^{(2)}$	3.16	1.69	1.73	2.16	$P_2^{(5)}$	1.38	1.50	1.47	2.28
$P_2^{(2)}$	4.49	1.29	1.30	1.42	$P_3^{(5)}$	1.85	1.39	1.32	2.23
$P_3^{(2)}$	3.56	2.44	2.70	2.83	$P_4^{(5)}$	1.06	1.00	0.92	1.63
$P_1^{(3)}$	6.27	1.12	1.24	1.11	$P_1^{(6)}$	11.80	7.84	7.91	6.90
$P_2^{(3)}$	2.94	1.22	1.28	1.23	$P_2^{(6)}$	—	10.85	10.37	6.67
$P_3^{(3)}$	4.09	1.74	2.60	2.83	$P_3^{(6)}$	—	8.30	7.83	8.21
$P_1^{(4)}$	5.25	1.95	3.49	1.21	$P_1^{(7)}$	7.63	3.49	3.69	6.00

• "T.It 1 ratio": ratio of the CPU time per phase 1 iteration between MINOS and PPRN $\left(\frac{\text{T.It 1 (MINOS)}}{\text{T.It 1 (PPRN)}}\right)$, that is, how many times faster PPRN is with respect to MINOS in performing one single phase 1 iteration.

• "T.It 2 ratio": ratio of the CPU time per phase 2 iteration between MINOS and PPRN $\left(\frac{\text{T.It 2 (MINOS)}}{\text{T.It 2 (PPRN)}}\right)$, that is, how many times faster PPRN is with respect to MINOS in performing one single phase 2 iteration.

• "T.It $2{-}h$ ratio": ratio of the CPU time per phase 2 iteration (excluding the time spent making objective function evaluations) between MINOS and PPRN $\left(\frac{\text{T.It } 2{-}h \text{ (MINOS)}}{\text{T.It } 2{-}h \text{ (PPRN)}}\right)$, that is, how many times faster PPRN is with respect to MINOS in performing one single phase 2 iteration without considering the objective function evaluations.

• "time ratio": ratio of the total CPU time between MINOS and PPRN $\left(\frac{\text{CPU. sec (MINOS)}}{\text{CPU. sec (PPRN)}}\right)$, that is, how many times faster the PPRN code is with respect to MINOS 5.3.

Looking at column "time ratio" of table XII it can be observed that PPRN is faster than MINOS in all the executions. Besides, in some tests (e.g., $P_3^{(1)}$) the "time ratio" value is higher than the other ratios; the reason is that PPRN performs in such cases much fewer iterations than MINOS. Another interesting fact is that, in most cases, PPRN is much faster in phase 1 than in phase 2 iterations with respect to MINOS, since it can take advantage of optimizing a linear function only.

8. Conclusions.

A new nonlinear multicommodity network flow code has been presented, implementing the primal partitioning method. Special features included in it are: managing the working matrix, dividing the optimization process into three phases and using Murtagh & Saunders' strategy of considering basic, superbasic and nonbasic variables. Moreover, the code is capable of solving problems with side constraints, optimizing a linear objective function through an ad hoc simplex algorithm, and working as a specialized single-commodity network flow code with side constraints. The code has shown to be very efficient in solving some test problems obtained from artificial functions and real models (long and short-term hydro-thermal scheduling of electricity generation, electrical network expansion planning and inelastic traffic assignment), with sizes ranging from 800 to 17,000 variables and 300 to 5,000 constraints.

Appendix 1. Factorization update of $\bar{R}$.

The BFGS formula used in the code was defined previously in equation (28). Next proposition will show how the new factorization matrix $\bar{R}$ can be obtained from R adding only one rank-one matrix.

Proposition.
Given the BFGS update formula

$$\bar{R}^t\bar{R} = R^tR + \frac{1}{\alpha^* y^t P_S} yy^t + \frac{1}{g_z^t P_S} g_z g_z^t$$

and defining $\delta_1 = \dfrac{1}{\sqrt{-\,g_z^t P_S}}$, $\delta_2 = \dfrac{1}{\sqrt{\alpha^* y^t P_S}}$, v such that $R^t v = g_z$, $\bar{v} = -\delta_1 v$, and $p = \delta_2 y + \delta_1 g_z$
($\delta_1, \delta_2 \in \mathbb{R}$; $v, \bar{v}, p \in \mathbb{R}^s$, s being the dimension of R),
then $\bar{R}$ can be computed as $\bar{R} = R + \bar{v}p^t$

Proof.
It will be shown that defining $\bar{R} = R + \bar{v}p^t$, then $\bar{R}^t\bar{R} = R^tR + \frac{1}{\alpha^* y^t P_S} yy^t + \frac{1}{g_z^t P_S} g_z g_z^t$, thus $\bar{R}$ being the correct factorization update matrix. Indeed,

$$\begin{aligned}
\bar{R}^t\bar{R} &= (R^t + p\bar{v}^t)(R + \bar{v}p^t) \\
&= R^tR + R^t\bar{v}p^t + p\bar{v}^tR + p(\bar{v}^t\bar{v})p^t \\
&\qquad \text{(using that } \bar{v} = -\delta_1 v\text{)} \\
&= R^tR + (-\delta_1)R^tvp^t + (-\delta_1)pv^tR + \delta_1^2(v^tv)pp^t \\
&\qquad \text{(using that } R^tv = g_z\text{)} \\
&= R^tR + (-\delta_1)g_zp^t + (-\delta_1)pg_z^t + \delta_1^2(v^tv)pp^t \\
&\qquad \text{(using that } p = \delta_2 y + \delta_1 g_z\text{)} \\
&= R^tR - \delta_1\delta_2 g_z y^t - \delta_1^2 g_z g_z^t - \delta_1\delta_2 y g_z^t - \delta_1^2 g_z g_z^t + \delta_1^2(v^tv)pp^t \\
&= R^tR - \delta_1\delta_2(g_z y^t + y g_z^t) - 2\delta_1^2 g_z g_z^t + \delta_1^2(v^tv)pp^t
\end{aligned} \tag{41}$$

Since $p = \delta_2 y + \delta_1 g_z$ then pp^t can be directly expressed as:

$$\begin{aligned}
pp^t &= (\delta_2 y + \delta_1 g_z)(\delta_2 y^t + \delta_1 g_z^t) \\
&= \delta_2^2 yy^t + \delta_2\delta_1 y g_z^t + \delta_1\delta_2 g_z y^t + \delta_1^2 g_z g_z^t
\end{aligned} \tag{42}$$

Now it will be shown that the coefficient $\delta_1^2(v^tv)$ is equal to 1. The descent direction P_S is computed as $R^tRP_S = -g_z$. Given that $R^tv = g_z$ it follows that $v = -RP_S$. Thus $v^tv = P_S^tR^tRP_S = -P_S^tg_z$. Since $\delta_1 = \frac{1}{\sqrt{-g_z^tP_S}}$ we directly have

$$\delta_1^2(v^tv) = \frac{1}{-g_z^tP_S}(-P_S^tg_z) = 1 \tag{43}$$

Substituting (42) and (43) in (41) we have:

$$\begin{aligned}\bar{R}^t\bar{R} &= R^tR - \delta_1\delta_2(g_zy^t + yg_z^t) - 2\delta_1^2 g_z g_z^t + \delta_2^2 yy^t + \delta_2\delta_1 yg_z^t + \delta_1\delta_2 g_z y^t + \delta_1^2 g_z g_z^t \\ &= R^tR - \delta_1^2 g_z g_z^t + \delta_2^2 yy^t \\ &= R^tR + \frac{1}{g_z^t P_S} g_z g_z^t + \frac{1}{\alpha^* y^t P_S} yy^t\end{aligned}$$

which was the desired result.

■

REFERENCES

[1] Ahuja, R.K., T.L. Magnanti, and J.B. Orlin. 1993. *Network Flows.* Prentice Hall, Englewood Cliffs, New Jersey.

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

[3] Beckmann M., C.B. McGuire, and C.B. Winsten. 1956. *Studies in the Economics of Transportation.* Yale University Press, New Haven CT.

[4] Bradley, G.H., G.G. Brown, and G.W. Graves. 1977. *Design and implementation of large scale primal transshipment algorithms.* Management Science, Vol.24, N.1, pp 1–34.

[5] Castro, J. 1993. *Efficient computing and updating of the working matrix of the multicommodity network flow problem with side constraints through primal partitioning.* DR 93/03. Statistics and Operations Research Dept., Universitat Politècnica de Catalunya, Barcelona. (written in Catalan).

[6] Castro, J. 1994. *PPRN 1.0, User's Guide.* DR 94/06. Statistics and Operations Research Dept., Universitat Politècnica de Catalunya, Barcelona.

[7] Castro, J. and N. Nabona. 1994. *Nonlinear multicommodity network flows through primal partitioning and comparison with alternative methods.* System Modelling and Optimization. Proceedings of the 16th IFIP Conference. J. Henry and J.-P. Yvon editors. pp. 875-884. Springer-Verlag.

[8] Castro, J. and N. Nabona. 1994. *Computational tests of a linear multicommodity network flow code with linear side constraints through primal partitioning.* DR 94/02. Statistics and Operations Research Dept., Universitat Politècnica de Catalunya, Barcelona.

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

[10] DIMACS. 1991. *The first DIMACS international algorithm implemyentation challenge: The bench-mark experiments.* Technical report, DIMACS, New Brunswick, NJ.

[11] Duff, I.S., A.M. Erisman, and J.K. Reid. 1986. *Direct Methods for Sparse Matrices.* Oxford University Press, New York.

[12] Grigoriadis, M.D. 1986. *An Efficient Implementation of the Network Simplex Method.* Mathematical Programming Study, v. 26, pp. 83–111.

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

[14] Heredia, F.J. and N. Nabona. 1990. *The FXCB Program for Linear Network Flows with Side Constraints.* RR 90/06. Computer Sciences Faculty of Barcelona, Universitat Politècnica de Catalunya, Barcelona. (written in Catalan).

[15] Heredia, F.J. and N. Nabona. 1994. *Optimum Short-Term Hydro-thermal Scheduling with Spinning Reserve through Network Flows.* Submitted to IEEE for consideration for publication in IEEE Trans. on Power Systems.

[16] Kamath, A.P, N.K. Karmarkar, and K.G. Ramakrishnan. 1993. *Computational and Complexity Results for an Interior Point Algorithm on Multicommodity Flow Problems.* Presented at Netflow93, San Miniato, Italy, October 3–7 1993.

[17] Kennington, J.L. and R.V. Helgason. 1980. *Algorithms for network programming.* John Wiley & Sons, New York.

[18] Kennington, J.L. and A. Whisman. 1988. *NETSIDE User's Guide.* TR86-OR-01 (revised April 1988), Southern Methodist University, Dallas, Tex., 75275, USA.

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

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

[21] Nabona, N. 1993. *Multicommodity network flow model for long-term hydrogeneration optimization.* IEEE Trans. on Power Systems, v. 8, num. 2, pp. 395–404.

[22] Nabona, N., J. Castro, J.A. González, and F.J. Heredia. 1994. *Optimum Transmission and Distribution Network Expansion at Several Voltage Levels through Multicommodity Network Flows.* Submitted to IEEE for consideration for publication in IEEE Trans. on Power Systems.

[23] Nabona, N. and J.M. Verdejo. 1992. *Numerical implementationof nonlinear multicommodity network flows with linear side constraints through price-directive decomposition*, in P. Kall (Ed.) System Modelling and Optimization, Proccedings of the 15th IFIP Conference, Zurich. Springer–Verlag. pp. 311-320.

[24] Sheffi, Y. 1985. *Urban Transportation Networks. Equilibrium Analysis with Mathematical Methods.* Prentice Hall, Englewood Cliffs, New Jersey.

[25] Toint, Ph. L. and D. Tuyttens. 1990. *On large scale nonlinear network optimization.* Mathematical Programming, Series B, v. 48(1), pp. 125–159.