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

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

DATE 02/94
DR 94/02

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

Abstract: *This work presents a new code for solving the multicommodity network flow problem with a linear objective function considering additional linear side constraints linking arcs of the same or different commodities. The code implements a specialization of the simplex method through primal partitioning of the basis, with use of a reduced working basis. 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. 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 random problems obtained from various network generators and real problems arising from the field of long-term hydro-thermal scheduling of electricity generation, with up to 150,000 variables and 45,000 constraints.*

Keywords: Multicommodity Network Flows, Network Simplex Methods, Primal Partitioning, Side Constraints, Network Generators, Hydro-Thermal Scheduling, Electricity Generation.

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 [14], no report can be found of its computational performance with large multicommodity problems of different characteristics. This work aims to fill this void by describing the computational results obtained with an efficient implementation of primal partitioning to solve multicommodity network flow problems of many types and sizes.

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

$$\min_{X_1,X_2,\ldots,X_K} \quad \sum_{k=1}^{K} C_k^t X_k \tag{1}$$

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

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

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

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 C_k the cost vector of the flows for each commodity. $A \in \mathbb{R}^{m \times n}$ (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 MP problem has been extended to include linear side constraints defined by:

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

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 MPC 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 [14]. 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. The best codes are two to five times faster than a general-purpose linear programming code [1]. 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 [13] for a description of a code using such methodology).

The code here presented mainly follows the underlying ideas in the primal partitioning method (it will be referred to as the PPRL code in the rest of the document). Some aspects, especially those related to the management of a working matrix, have been upgraded with respect to the original formulation of the problem described in [14], substantially improving the performance of the algorithm. Moreover, the code can deal with additional side constraints like (5) (PPRL solves the MPC problem instead of the MP one). In fact the mutual capacity constraints (4) are nothing but side constraints with special structure. From this point of view, codes for solving network flow problems with side constraints could be used to solve the MP problem. Descriptions of two such codes can be found in [11,15]. Therefore, it can be said that PPRL combines the features of the codes which solve the multicommodity network flow problem and of those which solve the network flow problem with side constraints. Moreover, the fact of using a primal partitioning of the basis facilitates the extension of the code to consider nonlinear objective functions. A first attempt was already made in a previous work by the authors [5], where Murtagh and Saunder's strategy of considering superbasic variables was applied [16].

The rest of the document will be subdivided into five main sections. Firstly, the primal partitioning methodology will be revised. Secondly the specific implementation developed in the PPRL code will be described. The next section will be devoted to the updating process of the working matrix, which is instrumental in the performance of the algorithm. Once the main features of the code have been reported, the two types of test problems that have been employed will be presented. These problem types come either from random network generators or from real problems from short and long-term hydro-thermal scheduling of electricity generation. The last main section will show the computational results obtained, with each test problem previously detailed.

2. 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 [14] for a comprehensive description.

2.1. 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 kth 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:

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

each B_k being a nonsingular matrix associated with the kth spanning tree. L_1 can be represented at every iteration by K spanning trees following the methodology described in [3,9].

• 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).

2.2. Motivation for using a working matrix.

During the optimization process systems $Bx = b$ and $x^t B = b^t$ 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 [14] 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.

2.2.1. 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:

$$L_1 x_1 + R_1 x_2 = b_1 \tag{8}$$

$$L_2 x_1 + R_2 x_2 = b_2 \tag{9}$$

$$L_3 x_1 + R_3 x_2 + x_3 = b_3 \tag{10}$$

Isolating x_1 from (8) we obtain $x_1 = L_1^{-1} b_1 - L_1^{-1} R_1 x_2$. Substituting this expression into (9) yields $L_2 L_1^{-1} b_1 - L_2 L_1^{-1} R_1 x_2 + R_2 x_2 = b_2$. Now x_2 is directly obtained from this equation:

$$x_2 = (R_2 - L_2 L_1^{-1} R_1)^{-1} (b_2 - L_2 L_1^{-1} x_1) \tag{11}$$

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.

2.2.2. 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 [9]. 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).

2.3. 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 $\quad dim(Q_{mc}) = |\mathcal{A}_{mc}| \times |\mathcal{A}| \quad$ and $\quad dim(Q_{sc}) = |\mathcal{A}_{sc}| \times |\mathcal{A}|$.

The expression for computing Q involves the calculation of $L_1^{-1} R_1$. Since L_1 is a block diagonal matrix where the kth block is a minimum spanning tree for the kth commodity, and R_1 expresses for each complementary arc of the kth commodity its connection to the kth minimum spanning tree, then solving $L_1^{-1} R_1$ is equivalent to having the paths (denoted by $P_j, j = 1 \div |\mathcal{A}|$) 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 jth column of Q, $j = 1 \div |\mathcal{A}|$.
- mc_i the mutual capacity constraint of the ith 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 ith 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}:

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

It is clear from this procedure that the information regarding the mutual capacity constraints is not stored, and is implicitly assumed in the construction of the Q_{mc} submatrix. Moreover, the current implementation of the PPRL code does not require the Boolean function $B(a, n)$ when computing Q_{sc} (which is done in a more efficient way without using $B(a, n)$), but it has been introduced here to simplify the concept. However, the abovementioned function $B(a, n)$ is needed in other parts of the algorithm, requiring us to store the information about the side constraints in sparse form in columns (that is, for each arc we have the side constraints where it appears) and sorted according to the side constraint number. Thus the Boolean function $B(a, n)$ can be reduced to a binary search (far more efficient than an exhaustive search). A full description of the computation of Q can be found in [4].

3. Implementation of primal partitioning.

The implementation of the primal partitioning method developed in the PPRL 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 phase 1 is closer to phase 2 than to phase 0, since 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 MPC 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 MPC2) 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 PPRL cannot deal with lower bounds other than zero in the variables.

3.1. 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 [9] 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.

3.2. 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 $\hat{X}_k, k = 1 \ldots K$ 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:

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

The problem solved in phase 1 is:

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

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

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

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

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

Where both matrices $\mathbb{1}^e \in \mathbb{R}^{n \times n}$ 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 MPC2 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 PPRL 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.

3.3. Phase 2.

Once a feasible point has been obtained in phases 0 and 1, phase 2 will reach the optimizer without leaving the feasible region. The problem to be minimized now is the original MCP2 problem defined in (21). Basically, the only difference between phases 1 and 2 is the different objective function to be minimized, but in both cases the primal partitioning technique is applied.

The fact of considering a partition of the basis as presented in (7) permits an efficient solution of systems $Bx = b$ and $x^t B = b^t$ (as was shown in former sections). This means updating this partition at each iteration whenever a pivotal operation is performed. 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 kth tree.* A complementary arc of the kth commodity, e.g. the jth complementary arc, having the leaving arc in its path P_j, must be found to replace the leaving arc in the kth tree. This complementary arc will always exist (otherwise the basis would become singular). The row and column of Q associated with the entering slack and the jth 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 kth tree.* A complementary arc of the kth commodity, e.g. the jth complementary arc, having the leaving arc in its path P_j, is sought. If this arc is found, it will replace the leaving arc in the kth 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 kth 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}.

4. Updating the working matrix.

The way in which the working matrix is dealt with is instrumental in ensuring the efficiency of the algorithm, since it is the only matrix to be factorized. 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 PPRL 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 [10] 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 [7].

An initial description of how to update this matrix was made by Kennington and Helgason in [14]. 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 PPRL 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 [4].

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 [4]. 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

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

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 PPRL directly stores the inverses of E and F, which continue to be products of eta and permutation matrices.

It has been shown that working with extended matrices avoids the problems of a variable dimension updating. Clearly a good choice of i — the number of iterations between two successive recomputations of Q — is important. This value should be neither too small (Q would be recomputed too often) nor too big (the dimension of the extended matrix would be too high and the round-off could increase). The default value for i of code PPRL is to recompute Q each 50 iterations (this value can be modified by the user). Another important point is to maintain the identity matrix at the bottom right part of Q_j. It is this that allows updating Q_{j+1} from Q_j at every step. It can be viewed as an invariant condition that must be preserved at each iteration. Of course another invariant condition could have been chosen (e.g., a diagonal matrix of 2, i.e., $2 \times \mathbb{1}$). Nevertheless, the choice of the identity matrix is justified by its simplicity.

5. Test problems employed.

Two types of problems have been employed to test the performance of the code. The first type was obtained from various network generators, while the second one arises from the field of long and short-term hydro-thermal coordination of electricity generation. The following subsections will describe the main features of such problems, and the particular instances employed in testing the code.

5.1. Network generator problems.

Four network generators (Rmfgen, Grid-on-torus, Gridgen and Gridgraph) have been employed, taken from the suite of generators distributed for the First DIMACS International Algorithm Implementation Challenge [6]. They are freely distributed and have been obtained via anonymous ftp from *dimacs.rutgers.edu* at directory */pub/netflow*. Some features are common to all four network generators:

- The output network is written in DIMACS format, producing the following information:
  - The number of nodes and arcs of the network.
  - The input and output flows at production and demand nodes.
  - For each arc, the source and target nodes, the maximum and minimum flow capacities and the linear cost.
- Although code PPRL can deal with side constraints, the generators produce networks without them. Thus all test problems obtained through generators have no side constraints.
- These network generators do not consider the case of multicommodity flows. Thus the output network obtained with the generators must be converted to a multicommodity network problem. This has been done through the following algorithm (where m is the number of nodes, n the number of arcs, K the desired number of commodities, $dem_{i,1}, i = 1 \div m$ the demand vector of the single commodity network, $cap_{i,1}, i = 1 \div n$ the upper capacity vector of the single commodity network and $cost_{i,1}, i = 1 \div n$ the cost vector of the single commodity network)

```
for_each k = 1 to K
        p_k := random(1/2 , 1)
end_for_each
for_each i = 1 to m
        r := dem_{i,1}
        for_each  k = 1 to K
                dem_{i,k} := p_k · r
        end_for_each
end_for_each
for_each i = 1 to m
        r := cap_{i,1}
        for_each k = 1 to K
                cap_{i,k}:= p_k · r
        end_for_each
end_for_each
for_each i = 1 to n
        r := cost_{i,1}
        for_each k = 1 to K
                cost_{i,k} := random(0 , r)
        end_for_each
end_for_each
```

where "random(a,b)" is a function which returns a random value $x, a \leq x \leq b$ (derived from [19]). It must be noticed that the arc capacities and node demands in the multicommodity network are obtained as a fraction p_k of the original capacities and demands in the single network, where p_k will always remain between $\frac{1}{2}$ and 1. The reason for not permitting values of p_k of less than $\frac{1}{2}$ is to avoid obtaining multicommodity networks where either the demands could be so small that no active mutual capacity constraints would ever appear in the optimal solution, or the upper capacities would be so close to 0 than no feasible solution could be obtained.

TABLE I. RMFGEN: MEDIUM-SIZED NETWORK—FEW COMMODITIES.

test	a	b	K	c_1	c_2	*cost*	*dem*	nodes	arcs	rows A	columns$A$
$P1_1$	16	8	1	0	10000	1000	1000	2048	9472	2048	9472
$P1_2$	16	8	4	0	10000	1000	1000	2048	9472	17664	47360
$P1_3$	16	8	8	0	10000	1000	1000	2048	9472	25856	85248
$P1_4$	16	8	16	0	10000	1000	1000	2048	9472	42240	161024

TABLE II. RMFGEN: SMALL-SIZED NETWORK—MANY COMMODITIES.

test	a	b	K	c_1	c_2	*cost*	*dem*	nodes	arcs	rows A	columns$A$
$P2_1$	4	8	50	0	4000	1000	1000	128	496	6896	25296
$P2_2$	4	8	100	0	4000	1000	1000	128	496	13296	50096
$P2_3$	4	8	150	0	6000	1000	1000	128	496	19696	74896
$P2_4$	4	8	200	0	8000	1000	1000	128	496	26096	99696

Eight particular instances have been created with each generator. These eight instances can be classified into two groups of four instances. The first group is made up of problems with few commodities and medium-sized networks, whereas the second group is composed of small networks with many commodities. The following subsections will present the main features of each generator and the description of the test instances taken into account. For all the cases created the seed used (to generate random values) was the integer 3141592.

5.1.1. The Rmfgen generator.

The Rmfgen generator, developed by Goldfarb and Grigoriadis [8], from the input parameters a, b, c_1 and c_2 produces a network with b pieces of frames of size $a \times a$ (so the number of vertices of the whole network is $a \times a \times b$). In each frame all the vertices are connected with their neighbours (forth and back). In addition the vertices of a frame are connected one to one with the vertices of the next frame using a random permutation of those vertices. The source is the lower left vertex of the first frame and the sink is the upper right vertex of the bth frame. The capacities are randomly chosen integers from the range of (c_1, c_2) in the case of interconnecting edges, and $c_2 \times a \times a$ for the in-frame edges.

In this work the original Rmfgen generator has been extended with two more parameters *cost* and *dem* used to produce a cost for each arc (randomly chosen from the range $(0, cost)$) and a demand/supply at the sink/source nodes (chosen from the range $(0, dem)$).

Tables I and II show the input parameters and dimension of the eight test problems created considering a medium-sized network and few commodities in the first case, and a small network with many commodities in the second. The first column, "test", is the name given to the test instance. Columns named a, b, c_1, c_2, *cost* and *dem* refer to the input parameters of the generator. Column K denotes the number of commodities considered in the test. 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.

5.1.2. The Grid-on-torus generator.

The Grid-on-torus generator was developed by A.V. Goldberg (1991, Dept. of Computer Science, Stanford University). It produces a capacitated transportation problem laid out on a grid-on-torus. It requires five integer parameters, which are:

- M: the number of nodes.
- N: the number of arcs.
- MAXCAP: the maximum capacity of arcs (the minimum capacity is always 0).
- MAXCOST: the maximum cost of arcs.
- SEED: for random number generators.

Table III shows the instances generated considering a medium-sized network and few commodities. In all cases the parameters of the Grid-on-torus generator were: M=1500, N=9000, MAXCAP=10000, MAXCOST=1000, SEED=3141592. The demand/supply obtained was afterwards divided by 10 to

TABLE III. GRID-ON-TORUS: MEDIUM-SIZED NETWORK—FEW COMMODITIES.

test	K	nodes	arcs	rows A	columns$A$
$P3_1$	1	1500	9000	1500	9000
$P3_2$	4	1500	9000	15000	45000
$P3_3$	8	1500	9000	21000	81000
$P3_4$	16	1500	9000	33000	153000

TABLE IV. GRID-ON-TORUS: SMALL-SIZED NETWORK—MANY COMMODITIES.

test	K	nodes	arcs	rows A	columns$A$
$P4_1$	50	100	600	5600	30600
$P4_2$	100	100	600	10600	60600
$P4_3$	150	100	600	15600	90600
$P4_4$	200	100	600	20600	120600

TABLE V. GRIDGRAPH: MEDIUM-SIZED NETWORK—FEW COMMODITIES.

test	K	supply	nodes	arcs	rows A	columns$A$
$P5_1$	1	15000	2502	5000	2502	5000
$P5_2$	4	15000	2502	5000	15008	25000
$P5_3$	8	3000	2502	5000	25016	45000
$P5_4$	16	3000	2502	5000	45032	85000

obtain feasible multicommodity flows. The meaning of the columns is the same as for those described when presenting the Rmfgen generator.

In the case of many commodities and a small network, the input parameters of the Grid-on-torus generator were: M=100, N=600, MAXCAP=10000, MAXCOST=1000, SEED=3141592. As in the previous case, the demand/supply was divided (now by 100) to allow feasible multicommodity flows. The dimensions of the tests obtained are shown in Table IV.

5.1.3. The Gridgraph generator.

The Gridgraph generator was developed by M.G.C. Resende (1991, AT&T Bell Laboratories). It requires five parameters (h, w, MAXCAP, MAXCOST and SEED), and from these it produces a graph made up of $h \times w + 2$ nodes: a source s, a sink t and a rectangular grid of h times w nodes (thus, a max-flow min-cost problem). Arcs go:

- From s to h nodes of the first grid column.
- From h nodes of the last grid column to t.
- From (i, j) node to $(i + 1, j)$ and $(j + 1, i)$ nodes except for the last row which goes from (h, j) to $(h, j + 1)$ and the last column which goes from (i, w) to t.

The capacities for grid arcs are uniformly distributed between 0 and MAXCAP. The arcs connecting the source node s or the sink one t are uncapacitated. The costs for grid arcs are uniformly distributed between 0 and MAXCOST. The arcs linking s and t with the grid have zero cost. The supply/demand at nodes s and t is the maximum flow that can be transported. This value must be reduced in the case of considering a multicommodity network to preserve feasibility.

Table V shows the test instances obtained when considering few commodities and medium-sized networks. The input parameters to Gridgraph were: h=50, w=50, MAXCAP=10000, MAXCOST=1000, SEED=3141592. The column "supply" is the supply of flow injected into the network, instead of the maximum flow given by Gridgraph. The remaining columns have the same meaning that in previous tables.

TABLE VI. GRIDGRAPH: MEDIUM-SIZED NETWORK—FEW COMMODITIES.

test	K	supply	nodes	arcs	rows A	columns$A$
$P6_1$	50	400	227	450	11800	22950
$P6_2$	100	400	227	450	23150	45450
$P6_3$	150	125	227	450	34500	67950
$P6_4$	200	125	227	450	45850	90450

For the case of the small network with many commodities the input parameters used were: h=15, w=15, MAXCAP=10000, MAXCOST=1000, SEED=3141592. The test instance dimensions obtained are shown in Table VI:

5.1.4. The Gridgen generator.

The Gridgen generator was delevoped by Y. Lee and J. Orlin. This network generator creates a grid-like network plus a super node. In addition to the arcs connecting the nodes in the grid, there is an arc from each supply node to the super node and from the super node to each demand node to guarantee feasibility. These arcs have very high costs and very large capacities.

The idea of this network generator is as follows. First, a grid of $n1 \times n2$ nodes is generated. The nodes are numbered from 1 to $n1 \times n2$, and the super node is numbered as $n1 \times n2 + 1$. Then arcs between adjacent nodes are generated. For these arcs, the user may choose between generating two-way arcs or one-way arcs. If two-way arcs are to be generated, two arcs, one in each direction, will be generated between each adjacent node pair. Otherwise, only one arc will be generated. If this is the case, the arcs will be generated in alternative directions. Then the arcs between the super node and the source/sink nodes are added as mentioned above. If the required number of arcs is still not reached, additional arcs will be added by uniformly picking random node pairs. There is no checking to prevent multiple arcs between any pair of nodes. However, there will be no self-arcs (arcs that point back to their own tail node) in the network. The source and sink nodes are selected uniformly in the network, and the imbalances of each source/sink node are also assigned by uniform distribution.

The input parameters of the generator are:

- Two-way arcs: 1 if links in both directions must be generated, and 0 otherwise.
- Random number seed: it must be a positive integer.
- Number of nodes: the number of nodes generated might be slightly different from specified to make the network a grid.
- Grid width.
- Number of sources and number of sinks.
- Average degree: to count the arcs to and from the super node.
- Total flow.
- Distribution of arc costs: 1 for uniform, and 2 for exponential. If 1 is chosen, then two more parameters must be supplied: the lower bound and upper bound of the arc costs. If the exponential distribution is chosen, then the lambda value of the distribution is required.
- Distribution of arc capacities: 1 for uniform, and 2 for exponential. If 1 is chosen, then two more parameters must be supplied: the lower bound and upper bound of the arc capacities. If the exponential distribution is chosen, then the lambda value of the distribution is required.

Tables VII and VIII show the instances generated for both kinds of problems: medium-sized network and few commodities, and small network and many commodities. The input parameters for the first type were: two-way arcs=0, seed=3141592, #nodes=1000, grid width=75, #sources=50, #sinks=30, average degree=8, total flow=50000, arc cost distribution=1, minimum cost=0, maximum cost=1000, arc capacity distribution=1, minimum capacity=0, maximum capacity=10000. The parameters for the second kind of problems were: two-way arcs=0, seed=3141592, #nodes=100, grid width=20, #sources=10, #sinks=10, average degree=6, total flow=200, arc cost distribution=1,

TABLE VII. GRIDGEN: MEDIUM-SIZED NETWORK—FEW COMMODITIES.

test	K	nodes	arcs	rows A	columns$A$
$P7_1$	1	976	7808	976	7808
$P7_2$	4	976	7808	11712	39040
$P7_3$	8	976	7808	15616	70272
$P7_4$	16	976	7808	23424	132736

TABLE VIII. GRIDGEN: SMALL-SIZED NETWORK—MANY COMMODITIES.

test	K	nodes	arcs	rows A	columns$A$
$P8_1$	50	101	606	5656	30906
$P8_2$	100	101	606	10706	61206
$P8_3$	150	101	606	15756	91506
$P8_4$	200	101	606	20806	121806

minimum cost=0, maximum cost=1000, arc capacity distribution=1, minimum capacity=0, maximum capacity=20000.

5.2. 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 [18] and [12] (where a comprehensive explanation of the models can be found).

5.2.1. 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.

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 Kth 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. This is a nonlinear multicommodity network flow problem and a simplified linearized objective function has been minimized here.

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).

The simplified linear objective function considered penalizes — with high positive costs — the spillage flows (s_{ji}^k) and incentivates — with negative costs — the discharges (d_{ji}^k), which lead to hydrogeneration, weighting each commodity with their availability in terms of $p_1, \ldots, p_{K-1}$. Stored

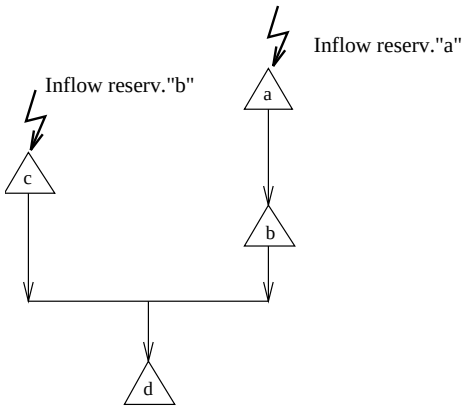

Fig. 1 Example of a four-reservoir system.

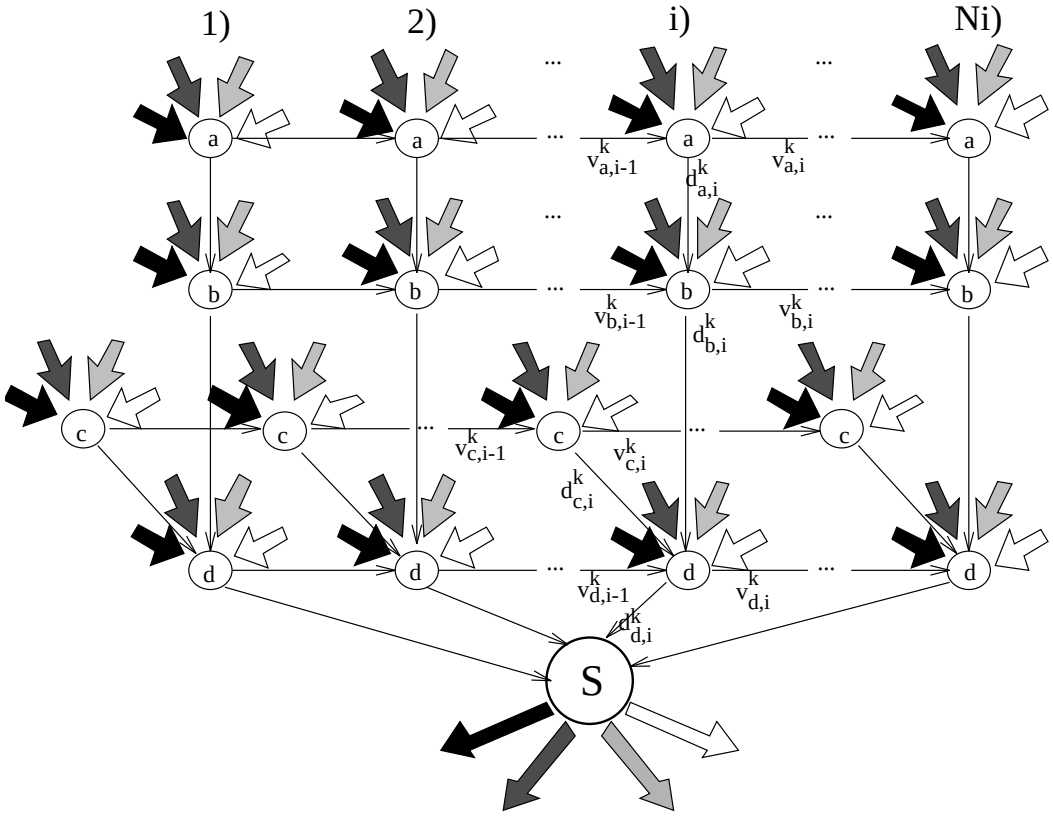

Fig. 2 Replicated multicommodity water network.

volumes (v_{ji}^k) do not take part in the linear objective function, the initial and final volumes being data of each problem. Side constraints to account for generation or irrigation limitations can be imposed.

5.2.2. Short-term hydro-thermal scheduling problems.

The solution to the short-term hydrothermal 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.

The network model usually employed for short-term hydrogeneration optimization has been extended to include thermal units in a new and undecoupled way [12], 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

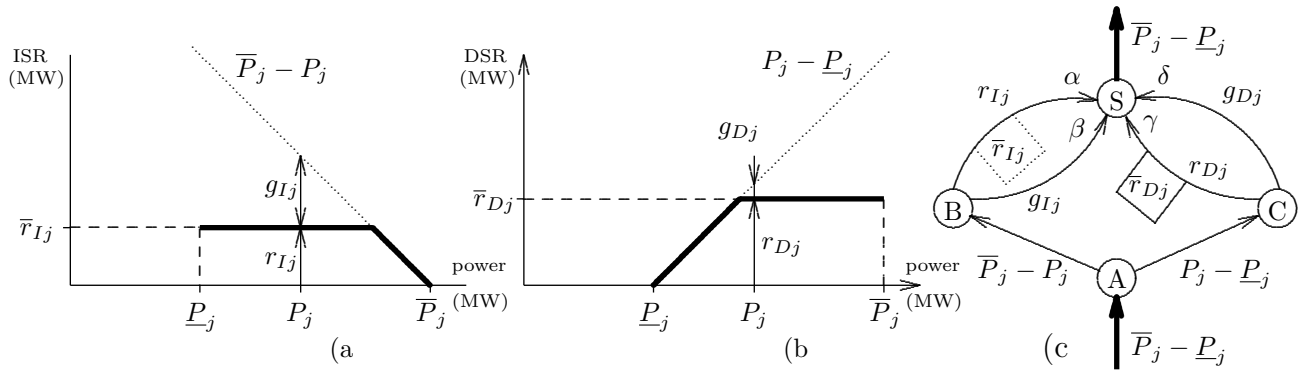

Fig. 3 a) Incremental Spinning Reserve (ISR) function of the j^{th} thermal unit.
b) Decremental Spinning Reserve (DSR) function of the j^{th} thermal unit.
c) Network representation of thermal generation variables.

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. 3(c. 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. 3(a, 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 linear cost function of generation.

Table IX shows the characteristics for the instances generated of the long-term (tests $P9_i$) and short-term (tests $P10_i$) hydro-thermal models. The column "#s.c." gives the number of side

TABLE IX. HYDRO-THERMAL SCHEDULING PROBLEMS.

test	K	#s.c.	nodes	arcs	rows A	columns$A$
$P9_1$	4	2	37	117	267	587
$P9_2$	4	12	37	153	313	777
$P9_3$	4	18	25	98	216	508
$P9_4$	4	3	99	315	714	1578
$P9_5$	4	3	685	2141	4884	10708
$P10_1$	1	36	444	181	217	480
$P10_2$	1	504	6216	2521	3025	6720

constraints considered in the problem. The remaining columns have the same meaning as in previous tables.

6. Computational results.

This section will present the results obtained with the PPRL code, with the test problems described formerly. The PPRL code has been compared with the general-purpose package Minos 5.3 [17]. 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 X–XIV show the results obtained for the groups of test problems presented in former tables, with both the PPRL code and Minos 5.3. For PPRL the information disclosed includes columns:

- "Ph.0": number of iterations at phase 0.
- "Ph.1": number of iterations at phase 1.
- "Ph.2": number of iterations at phase 2.
- "Obj.Value": optimum objective function value.
- "$|\mathcal{A}|$": number of active mutual capacity and side constraints at the optimizer (i.e., the dimension of the working matrix).
- "cpu sec.": cpu seconds spent by the execution.

For Minos 5.3 only the information in columns "Ph.1", "Ph.2", "Obj.Value" and "cpu sec." is given (it must be noticed that in the Minos 5.3 package — and others that implement the simplex method — phase 1 finds a feasible point — task performed by phases 0 and 1 in PPRL —, whereas phase 2 reaches the optimizer).

It must be pointed out that network generator tests with few commodities have been executed with the option of finding the optimum spanning tree at phase 0, whereas for the cases with many commodities and for long and short-term hydro-thermal scheduling problems a feasible spanning tree only was obtained (this is clearly reflected in the number of iterations at phase 0 of each type of case). Thus for cases with only one commodity, phase 0 alone is executed (in these cases the comparison is made between the Minos 5.3 package and the single network flow code that implements phase 0, instead of the primal partitioning algorithm).

Some tests were not performed with Minos 5.3, given that because its size the execution time would be too expensive. This is denoted in tables with the message "Not executed". The message "Error during execution" can also be found in some table cells for some tests attempted with Minos 5.3. It denotes that the program stopped before arriving at the optimum (in some cases with a controlled error and in others with a failure). The message "Too many constraints" appears sometimes because the number of constraints is stored as a two-byte signed integer, thus the maximum number of constraints possible is $2^{15} - 1 = 32767$, and test with more constraints cannot be performed by Minos 5.3.

From tables X–XIV some interesting conclusions can be drawn. These are shown in Table XV (only for instance tests whose execution was completed for both codes), where each column means:

- "cpu sec./iter.": average cpu time per iteration in seconds, for Minos and for PPRL. Only the iterations at phases 1 and 2 for PPRL are considered (and only its associated cpu time).

TABLE X. RESULTS FOR RMFGEN TESTS.

	PPRL						Minos 5.3			
test	Ph.0	Ph.1	Ph.2	Obj.Value	$\|\mathcal{A}\|$	cpu sec.	Ph.1	Ph.2	Obj.Value	cpu sec.
$P1_1$	3152	0	0	370685.0	0	13.1	1	4350	370685.0	412.6
$P1_2$	12046	7	654	2037823.4	2	69.2	5	20179	2037823.4	3545.1
$P1_3$	24050	293	4942	4453121.9	4	568.3	9	49411	4453123.8	15903.0
$P1_4$	49284	6774	25102	9805550.3	6	5896.9	Too many constraints			
$P2_1$	2500	3972	9554	11728727.3	24	332.1	3514	22669	11728728.2	2385.4
$P2_2$	5000	10462	43123	26902869.7	52	2552.4	19634	131938	26902871.4	29993.9
$P2_3$	7500	14660	76875	39449268.2	56	7490.9	Error during execution			
$P2_4$	10000	19911	107055	53828985.7	57	14560.8	Not executed			

TABLE XI. RESULTS FOR GRID-ON-TORUS TESTS.

	PPRL						Minos 5.3			
test	Ph.0	Ph.1	Ph.2	Obj.Value	$\|\mathcal{A}\|$	cpu sec.	Ph.1	Ph.2	Obj.Value	cpu sec.
$P3_1$	3727	0	0	1881.3	0	12.5	21	5670	1881.3	455.1
$P3_2$	13233	531	3848	75797.87	25	242.1	704	42601	75798.0	6584.3
$P3_3$	28259	3298	20054	989167.2	66	2267.5	Error during execution			
$P3_4$	55379	9698	107176	5234488.2	183	23051.2	Not executed			
$P4_1$	2500	982	5008	5004639.0	36	151.2	3003	14301	5004640.0	1096.2
$P4_2$	5000	3779	16817	12485241.0	102	1175.2	41918	76372	12485241.0	19271.4
$P4_3$	7500	10313	52715	21967864.7	166	5603.7	128630	231337	21967864.7	90131.8
$P4_4$	10000	26862	121659	35830254.4	237	18294.1	Not executed			

TABLE XII. RESULTS FOR GRIDGRAPH TESTS.

	PPRL						Minos 5.3			
test	Ph.0	Ph.1	Ph.2	Obj.Value	$\|\mathcal{A}\|$	cpu sec.	Ph.1	Ph.2	Obj.Value	cpu sec.
$P5_1$	3318	0	0	94657071.1	0	6.8	869	3000	94657071.1	256.4
$P5_2$	13076	2462	10147	355442380.7	244	716.7	11820	48781	355442380.7	10528.4
$P5_3$	21088	2387	8607	128385992.5	108	1042.0	Error during execution			
$P5_4$	43173	11447	45162	252898663.4	221	11077.3	Not executed			
$P6_1$	2500	2587	9501	27675286.0	23	557.4	3989	15623	27675286.04	3002.6
$P6_2$	5000	9793	36611	64840298.33	60	3829.8	Error during execution			
$P6_3$	7500	6734	30441	26928530.2	27	4583.2	Too many constraints			
$P6_4$	10000	11852	52914	37788372.1	38	11010.3	Not executed			

• "iter. ratio": ratio of the average cpu time per iteration between Minos and PPRL $\left(\frac{\text{Minos cpu time per iter.}}{\text{PPRL cpu time per iter.}}\right)$, that is, how many times faster PPRL is with respect to Minos in performing one single phase 1/phase 2 iteration.

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

Looking at Table XV, it can be observed that the phase 0 code (in tests $P1_1$, $P3_1$, $P5_1$ and $P7_1$) is about 30 times faster than Minos 5.3. On the other hand, PPRL is two to six times faster than Minos in performing a single simplex iteration (phases 1 and 2). This is due to the fact of managing the working matrix instead of the whole basis (notice that this is so even in tests $P9_i$ and $P10_i$ with side constraints — not considered in the original formulation of the primal partitioning method). However, comparing the total execution time, code PPRL is much faster than Minos 5.3. The reason is the

TABLE XIII. RESULTS FOR GRIDGEN TESTS.

	PPRL						Minos 5.3			
test	Ph.0	Ph.1	Ph.2	Obj.Value	$\|\mathcal{A}\|$	cpu sec.	Ph.1	Ph.2	Obj.Value	cpu sec.
$P7_1$	3584	0	0	5532381.8	0	10.4	256	3657	5532382.0	275.6
$P7_2$	14577	689	5776	22883042.3	41	255.7	7524	69308	22883042.2	9349.4
$P7_3$	28317	3038	35791	61345575.9	111	2779.8	53617	730324	61345575.1	173952.8
$P7_4$	57502	11317	251673	164680474.5	263	38061.0	Not executed			
$P8_1$	2550	44	6531	1409470.3	1	199.1	1617	9779	1409470.3	922.2
$P8_2$	5177	273	13269	2940217.3	3	773.0	5336	31486	2940217.2	6079.0
$P8_3$	7850	501	19460	4603847.52	5	1608.4	8813	41614	4603847.51	12860.7
$P8_4$	10350	989	28290	6440385.5	9	3486.9	Error during execution			

TABLE XIV. RESULTS FOR HYDRO-THERMAL SCHEDULING TESTS.

	PPRL						Minos 5.3			
test	Ph.0	Ph.1	Ph.2	Obj.Value	$\|\mathcal{A}\|$	cpu sec.	Ph.1	Ph.2	Obj.Value	cpu sec.
$P9_1$	174	84	106	-1282540.0	27	0.5	238	85	-1282540.0	2.4
$P9_2$	156	89	60	-571293.8	38	0.5	183	149	-571293.8	3.0
$P9_3$	200	268	303	-400778.3	38	1.0	222	208	-400778.3	2.3
$P9_4$	543	344	1564	-995120.3	111	7.0	570	829	-995120.3	12.9
$P9_5$	5168	4257	10134	71104987.7	730	327.0	12758	7426	71104987.8	995.4
$P10_1$	250	88	89	3248.1	12	0.6	134	192	3250.7	3.8
$P10_2$	3850	2069	1382	40701.8	168	69.8	3051	5777	40739.2	727.7

TABLE XV. TIME COMPARISON BETWEEN MINOS AND PPRL.

	cpu sec./iter.		iter.	time
test	PPRL	Minos	ratio	ratio
$P1_1$	—	0.094	—	31.5
$P1_2$	0.028	0.175	6.2	51.2
$P1_3$	0.089	0.321	3.6	28.0
$P2_1$	0.023	0.091	3.9	7. 2
$P2_2$	0.047	0.197	4.1	11.8
$P3_1$	—	0.079	—	36.4
$P3_2$	0.045	0.152	3.3	27.2
$P4_1$	0.023	0.063	2.7	7.3
$P4_2$	0.056	0.162	2.8	16.4
$P4_3$	0.088	0.250	2.8	16.1
$P5_1$	—	0.066	—	37.7
$P5_2$	0.054	0.173	3.2	14.7
$P6_1$	0.045	0.153	3.4	5.4

	cpu sec./iter.		iter.	time
test	PPRL	Minos	ratio	ratio
$P7_1$	—	0.070	—	26.5
$P7_2$	0.033	0.121	3.6	36.6
$P7_3$	0.069	0.221	3.2	62.6
$P8_1$	0.029	0.080	2.7	4.6
$P8_2$	0.055	0.165	3.0	7.9
$P8_3$	0.079	0.255	3.2	8.0
$P9_1$	0.0026	0.0074	2.8	4.8
$P9_2$	0.0033	0.0090	2.7	6.0
$P9_3$	0.0017	0.0053	3.1	2.3
$P9_4$	0.0036	0.0092	2.5	1.8
$P9_5$	0.022	0.049	2.2	3.0
$P10_1$	0.0033	0.0117	3.5	6.3
$P10_2$	0.0188	0.0824	4.3	10.4

number of iterations required by each program: in general, code PPRL requires far fewer iterations than Minos, due to the special procedure of finding a feasible point in phases 0 and 1 (only in tests $P9_3$ and $P9_4$ did the number of iterations degrade the performance of the program). Thus, combining the primal partitioning technique (with the special management of the working matrix presented) and the phase 0-phase 1-phase 2 procedure seems to be a good choice to solve linear multicommodity problems with side constraints.

7. Conclusions.

A new linear multicommodity network flow code has been presented, implementing the primal partitioning method. Special features included in it are: managing the working matrix and dividing the optimization process into three phases. Moreover, the code is capable of solving problems with side constraints. The code has shown to be very efficient in solving some test problems obtained from network generators and long-term hydrothermal scheduling models, with sizes ranging from 500–150,000 variables and 200–45,000 constraints. This algorithm can be extended to consider nonlinear objective functions, and some work has already been done in this direction.

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 & H. Lall. 1980 *Computational comparison among three multicommodity network flow algorithms.* Operations Research, v. 28, pp. 995–1000

[3] 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.

[4] 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).

[5] 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.

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

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

[8] Goldfarb, D and M. D. Grigoriadis. 1988. *A Computational Comparison of the Dinic and Network Simplex Methods for Maximum Flow.* Annals of Operations Research, v. 13, pp. 83–128.

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

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

[11] 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).

[12] Heredia, F.J. and N. Nabona. 1993. *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.

[13] 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.

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

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

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

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

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

[19] Schrage, L. 1979. *A More Portable FORTRAN Random Number Generator.* ACM Transactions on Mathematical Software, June.