ELSEVIER

European Journal of Operational Research 92 (1996) 37–53

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Theory and Methodology

An implementation of linear and nonlinear multicommodity network flows

J. Castro *, N. Nabona

Statistics & Operations Research Dept., Universitat Politècnica de Catalunya, Pau Gargallo 5, 08071 Barcelona, Spain

Received December 1994; revised May 1995

Abstract

This work presents a new code for solving the multicommodity network flow problem with a linear or nonlinear objective function considering additional linear side constraints that link arcs of the same or different commodities. For the multicommodity network flow problem through primal partitioning the code implements a specialization of Murtagh and Saunders' strategy of dividing the set of variables into basic, nonbasic and superbasic. Several tests are reported, using random problems obtained from different network generators and real problems arising from the fields of long and short-term hydrothermal scheduling of electricity generation and traffic assignment, with sizes of up to 150000 variables and 45000 constraints. The performance of the code developed is compared to that of alternative methodologies for solving the same problems: a general purpose linear and nonlinear constrained optimization code, a specialised linear multicommodity network flow code and a primal-dual interior point code.

Keywords: Linear programming; Multicommodity Network flows; Network programming; Nonlinear programming; Primal partitioning

1. Introduction

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 [20], but no report can be found either of its comparative computational performance with large scale multicommodity problems or its adaptation to the nonlinear objective function case. This work aims at filling this void by describing the algorithmic principles followed in an extension of the classical techniques and the comparative computational results obtained with an efficient implementation.

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

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

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

$$0 \leqslant X_k \leqslant \overline{X}_k, \quad k = 1, \ldots, K, \tag{3}$$

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

where $X_k \in \mathbb{R}^n$ (n is the number of arcs) is the flow array for each commodity k ($k = 1, \ldots, K$), K being the number of commodities of the problem, and h being a $\mathbb{R}^{K \times n} \to \mathbb{R}^1$ real valued linear or nonlinear function. $A \in \mathbb{R}^{m \times n}$ (m is the number of nodes) is the

* Corresponding author.

SSDI 0377-2217(95)00137-9

arc–node incidence matrix. Constraints (3) are simple bounds on the flows, $\overline{X}_k \in \mathbb{R}^n$, $k = 1, \ldots, K$, being the upper bounds. Eq. (4) represents the mutual capacity constraints, where $T \in \mathbb{R}^n$.

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

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

where L_k: $L_k \in \mathbb{R}^{p \times n}$, $k = 1, \ldots, K$, and $L, U \in \mathbb{R}^p$ (p is the number of side constraints). These side constraints can link arcs of the same or different commodities. In fact, constraints (4) are a special case of constraints (5), but in this work they have been specifically treated, instead of being considered as plain side constraints. This is made in order to exploit the inherent structure of the mutual capacity constraints. Therefore, the final formulation of the problem considered can be stated as follows:

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

and will be referred to as the MCNFC problem.

The code here presented (which will be referred to as the PPRN code [7] in the rest of the document) can be viewed as a general purpose code for solving the MCNFC problem. The MCNFC problem was first approached for nonlinear objective functions using the price-directive decomposition [25] but this procedure does not seem to be as computationally efficient as primal partitioning [8]. If the set of side constraints (5) is empty, code PPRN will only solve a multicommodity network flow problem (MCNF). If the number of commodities is equal to one, it will work as a specialized linear and nonlinear network flow code with side constraints, as described in Refs. [20,21]. 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.

2. Linear multicommodity network flows

If the objective function (1) is linear we will refer to the multicommodity problem (MCNF) merely as the LMCNF problem. To solve the LMCNF 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 [20] belong to the first class, and a first attempt at comparing these techniques can be found in Ref. [3]. 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) [1].

Interior point methods are an alternative approach to solve the LMCNF problem. These methods appear to be really efficient when the size of the network is very large [11,18].

3. The primal partitioning method

A brief description of the primal partitioning method [20] will be presented, paying special attention to the changes brought about by considering the additional side constraints (5).

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 $\hat{A}$. Then each variable j_k (that is, each flow j of the kth commodity) has an associated column $\hat{A}^{j_k}$ in $\hat{A}$, with the following nonzero components:

$$(\hat{A}^{j_k})^{\mathrm{T}} = \left(\underbrace{\overset{s}{\downarrow}\ \overset{t}{\downarrow}\ \ldots 1 \ldots -1 \ldots}_{\text{Network}} \;\middle|\; \underbrace{\overset{j}{\downarrow}\ \ldots 1 \ldots}_{\text{Mutual capacity}} \;\middle|\; \underbrace{a_{j_1} \ldots a_{j_p}}_{\text{Side constraints}} \right)^{\mathrm{T}}$$

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. The structure of the network and mutual capacity constraints is independent of the commodity, but the structure of side constraints may be different for each commodity. 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:

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

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

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

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

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 Refs. [5,15].

- R_1 refers to the network constraints and complementary arcs of the K commodities. Complementary arcs do not belong to any spanning tree and they are just additional arcs exchanged against the active constraints (4) or (5).
- 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.
- $\mathbf{1}$, 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 $\mathbf{1}$ 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 $u^{\mathrm{T}}B = c^{\mathrm{T}}$ must be solved at each iteration, x, u and b, c being the variable and independent term vectors, respectively. A description of the solution technique can be found in Ref. [20] and is briefly outlined here. Considering for x, u and b, c partitions $x_1, x_2, x_3, u_1, u_2, u_3$ and $b_1, b_2, b_3, c_1, c_2, c_3$ as the one employed above for the basis B we have

- For $Bx = b$

$$x_2 = (R_2 - L_2L_1^{-1}R_1)^{-1}(b_2 - L_2L_1^{-1}x_1)\,, \tag{8}$$

$$x_1 = L_1^{-1}b_1 - L_1^{-1}R_1x_2\,, \tag{9}$$

$$x_3 = b_3 - L_3x_1 - R_3x_2\,. \tag{10}$$

Thus by solving (8), (9) and (10) consecutively we obtain the solution of the original system.

- For $u^{\mathrm{T}}B = c^{\mathrm{T}}$

$$u_3 = c_3\,, \tag{11}$$

$$u_2 = \big((c_2 - c_3R_3) - (c_1 - c_3L_3)L_1^{-1}R_1\big) \times (R_2 - L_2L_1^{-1}R_1)^{-1}\,, \tag{12}$$

$$u_1 = (c_1 - u_3L_3 - u_2L_2)L_1^{-1}\,. \tag{13}$$

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

Thus it is enough to invert the 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. There is no need to invert L_1, given that 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 [5]. 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 [15]. 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 factorization of Q must be able to deal with variable size, 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,

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

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 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, $j = 1, \ldots, |\mathcal{A}|$) of complementary arcs in their associated spanning trees. Given an arc $a \in P_j$, we will say that 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, $j = 1, \ldots, |\mathcal{A}|$;
- mc_i the mutual capacity constraint of the ith row of Q, $i = 1, \ldots, |\mathcal{A}_{mc}|$ (this capacity constraint refers to the saturated arc mc_i);
- sc_i the side constraint of the ith row of Q, $i = |\mathcal{A}_{mc}| + 1, \ldots, |\mathcal{A}|$;
- $B(a, n)$ 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}:

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

- Submatrix Q_{sc} with entries Q_{ij}, $i = |\mathcal{A}_{mc}| + 1, \ldots, \dim(Q)$, $j = 1, \ldots, \dim(Q)$, computed following the following steps:

(1) Set $Q_{ij} = 0$,
(2) If $B(a_j, sc_i)$ then $Q_{ij} = c_{a_j, sc_i}$.
(3) For each $a \in P_j$, perform next 2 steps.
(4) If $B(a, sc_i)$ and a has normal orientation then $Q_{ij} = Q_{ij} + c_{a,sc_i}$.
(5) If $B(a, sc_i)$ and a has reverse orientation then $Q_{ij} = Q_{ij} - c_{a,sc_i}$.

A full and more detailed description of the computation of Q can be found in Ref. [6].

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 MCNFC problem

are replaced by equality constraints by adding slacks, obtaining:

$$\sum_{k=1}^{K} X_k + s = T, \quad 0 \leqslant s, \tag{14}$$

$$\sum_{k=1}^{K} L_k X_k + t = U, \quad 0 \leqslant t \leqslant U - L, \tag{15}$$

where $s \in \mathbb{R}^n$ and $t \in \mathbb{R}^p$. In this formulation Eqs. (14) and (15) replace the original Eqs. (4) and (5). Then the formulation of the problem considered by the algorithm (which will be referred to as MCNFC2) is

$$\min_{X_1, X_2, \ldots, X_K} \quad (1)$$
$$\text{subject to} \quad (2), (3), (14), (15). \tag{16}$$

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, $k = 1, \ldots, K$, a feasible starting point for the linear network problem:

$$\min_{X_k} \quad C_k^{\mathrm{T}} X_k$$
$$\text{subject to} \quad A X_k = R_k,$$
$$0 \leqslant X_k \leqslant \overline{X}_k, \tag{17}$$

where C_k refers to the cost vector for commodity k if (1) is linear, or an arbitrary cost vector provided by the user (which can be the gradient at some initial point x_0) if the problem to be solved is nonlinear.

This problem is solved by applying a specialization of the simplex algorithm for networks. The implementation developed mainly follows the ideas described in Ref. [15] 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 (17) or reach its optimum solution. (The default option is to obtain a feasible point).

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 $\hat{X}_k, k = 1 \ldots K$ be the pseudofeasible point obtained; then the following index sets are defined:

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

$$t^- = \left\{ i : \left(\sum_{k=1}^{K} L_k \hat{X}_k \right)_i > U_i \Leftrightarrow t_i < 0 \right\},$$

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

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

$$\left(\sum_{k=1}^{K} \hat{X}_k \right)_i + s_i - e_i = T_i; \quad s_i = 0; \ \forall i \in s^-,$$

$$\left(\sum_{k=1}^{K} L_k \hat{X}_k \right)_i + t_i - f_i = U_i; \quad t_i = 0; \ \forall i \in t^-,$$

$$\left(\sum_{k=1}^{K} L_k \hat{X}_k \right)_i + t_i + f_i = U_i; \quad t_i = (U - L)_i; \quad \forall i \in t^+.$$

The problem to be solved in phase 1 is

$$\min_{X_1, X_2, \ldots, X_K, s, t, e, f} \quad \sum_{i \in s^-} e_i + \sum_{i \in t^-} f_i + \sum_{i \in t^+} f_i \tag{18}$$
$$\text{subject to} \quad (2), (3),$$
$$\sum_{k=1}^{K} X_k + s + \mathbf{1}^e e = T, \tag{19}$$
$$\sum_{k=1}^{K} L_k X_k + t + \mathbf{1}^f f = U, \tag{20}$$

$$0 \leqslant t \leqslant U - L, \quad 0 \leqslant s,$$
$$0 \leqslant e, \quad 0 \leqslant f,$$

where both matrices $\mathbf{1}^e \in \mathbb{R}^{n\times n}$ and $\mathbf{1}^f \in \mathbb{R}^{p\times p}$ in (19) and (20) are diagonal and defined as follows:

$$(\mathbf{1}^e)_{ii} = \begin{cases} -1, & \text{if } i \in s^-, \\ 0, & \text{otherwise;} \end{cases}$$

$$(\mathbf{1}^f)_{ii} = \begin{cases} -1, & \text{if } i \in t^-, \\ +1, & \text{if } i \in t^+, \\ 0, & \text{otherwise.} \end{cases}$$

It can be noticed that the objective function (18) at phase 1 is nothing but the sum of the infeasibilities of the mutual capacity and side constraints. Therefore, the MCNFC2 problem defined in (16) will be feasible if, at phase 1, the value of (18) at the optimizer is 0.

Dividing the process of finding a feasible starting point for problem MCNFC2 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.

4.3. Phase 2

Once a feasible point has been obtained, phase 2 attempts to achieve the optimizer of the objective function. The primal partitioning method, as presented in Ref. [20], was intended for linear objective functions. In this case PPRN works as a specialized simplex algorithm based on primal partitioning, where the main implementation detail is based on the pivot operations described later.

However, when optimizing nonlinear functions, primal partitioning can be applied together with Murtagh and Saunders' strategy – described in Ref. [22] – of dividing the set of variables into basic, superbasic and nonbasic variables:

$$\hat{A} = [B \mid S \mid N],$$

$\hat{A}$ 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.

Suppose that at iteration i we have (the subscript 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 $g(x_i) = [g_B \mid g_S \mid g_N]$ for basic, superbasic and nonbasic variables.
- Z: a representation matrix of the null subspace of the constraint matrix A defined in (4.3). The expression for Z is

$$Z = \begin{bmatrix} -B^{-1}S \\ \mathbf{1} \\ 0 \end{bmatrix}.$$

It can easily be observed that $\hat{A}Z = 0$.

- g_z, ϵ_{g_z}: the current reduced gradient $g_z = Z^{\mathrm{T}} g(x_i)$, and a tolerance to estimate when its norm is small enough.
- π: a vector satisfying $\pi^{\mathrm{T}} B = g_B^{\mathrm{T}}$.

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 $\|g_z\| \geqslant \epsilon_{g_z}$ go to Step 3.

Step 2. Price nonbasic variables.
(i) Compute Lagrange multipliers $\lambda = g_N - N^{\mathrm{T}}\pi$. (*)
(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 $P^{\mathrm{T}} = [P_B \mid P_S \mid 0]^{\mathrm{T}}$ for basic and superbasic variables.
(i) Solve $Z^{\mathrm{T}} H_i Z P_S = -g_z$, where $H_i = \nabla^2 h(x_i)$. (*)
(ii) Solve $BP_B = -SP_S$. (*)

Step 4. Ratio test.
(i) Find $\alpha_{\max} \geqslant 0$ such that $x_i + \alpha_{\max} P$ is feasible.
(ii) If $\alpha_{\max} = 0$ go to Step 7.

Step 5. Line search.
(i) Find α^* such that

$$h(x_i + \alpha^* P) = \min_{0 \leqslant \alpha \leqslant \alpha_{\max}} h(x_i + \alpha P)$$

(ii) Update the 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:

(a) Computing the descent direction. The current implementation of the program makes it possible to solve the system $Z^T H_i Z P_S = -g_z$ in Step 3 (i) by two methods: through a truncated-Newton algorithm [12], or using a quasi-Newton approximation of the projected Hessian $Z^T H_i Z$ [22]. In neither case is an analytical expression for the Hessian of the objective function (1) required. Thus the PPRN code only needs the evaluation of the objective function and its gradient.

The algorithmic details on this implementation can be found in Ref. [10].

(b) 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 $||g_z|| \geqslant \epsilon_{g_z}$. 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 current situation, 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.

$$T_1 := \left(\alpha^* ||P_S||_1 \leqslant (\varepsilon_x^{cs} + \sqrt{\varepsilon_M})(1 + ||x_s||_1)\right),$$

$$T_2 := \left(|\Delta h| \leqslant (\varepsilon_f^{cs} + \varepsilon_M)(1 + |h|)\right),$$

$$T_3 := \left(||g_z||_\infty \leqslant T_{g_z}\right),$$

$$T_4 := \left(||g_z||_\infty \leqslant \max\{T_{g_z}/10, \varepsilon_g \epsilon(||\pi||_1)\}\right),$$

$$T_5 := \left((T_5 \text{ is active}) \text{ and } (cs = \text{"far"}) \text{ and } (nsame \leqslant MAXsame)\right),$$

$$T_6 := \left(ncurrent \leqslant MAXcurrent\right).$$

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, $\varepsilon_f^{\text{"near"}} \ll \varepsilon_f^{\text{"far"}}$.

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, $\eta_{g_z}^{\text{"near"}} \ll \eta_{g_z}^{\text{"far"}}$.

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

$$\epsilon(||\pi||_1) = \max\left\{1, \frac{||\pi||_1}{\sqrt{mK+n+p}}\right\}, \qquad (21)$$

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 nonsmooth 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

$$T := (T_1 \text{ and } T_2 \text{ and } T_3) \text{ or } T_4 \text{ or } T_5 \text{ or } T_6. \qquad (22)$$

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* $||g_z|| \geqslant \epsilon_{g_z}$.

(c) 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 (21), 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 $\epsilon_{\text{opt}} = 10^{-6}$).

(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 $|\lambda_q| \geqslant T_{\lambda_q}$ or, if there is none, the greatest $|\lambda_q|$.
(3) If $(|\lambda_q| \geqslant T_{\lambda_q})$ go to (7).
(4) No multiplier satisfies the current tolerance T_{λ_q}. If $(|\lambda_q| \geqslant \epsilon_{\text{opt}}\epsilon(||\pi||_1))$ then
  (i) $T_{\lambda_q} = \max\{|\lambda_q|/10, \epsilon_{\text{opt}}\epsilon(||\pi||_1)\}$.
  (ii) Go to (7).
(5) No multiplier is greater than the optimality tolerance $\epsilon_{\text{opt}}\epsilon(||\pi||_1)$. We are near the optimum. If $((cs = \text{"near"})$ or $(||g_z||_\infty \leqslant \epsilon_{\text{opt}}\epsilon(||\pi||_1)))$ 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 $(T_{g_z} < \epsilon_{\text{opt}}\epsilon(||\pi||_1)$ then
    – *cs*= "near".
    – $T_{g_z} = \epsilon_{\text{opt}}\epsilon(||\pi||_1)$.
  (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 $(cs = \text{"near"})$ then *cs*= "far".
  (iv) Continue with Step 2(iii) of the phase 2 al-

gorithm.

(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 Ref. [22]. 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.

(d) Pivot operation. In case of a nonlinear objective function (1), 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. If the objective function is linear a column of the basis is replaced by a nonbasic column. In both cases 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 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 the 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 the 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% nonzero 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 [16] 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.

An initial description of how to update this matrix was made by Kennington and Helgason in [20]. Two importants remarks should be made on the approach described there:

(i) 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.

(ii) 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 inappropriate 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 [6].

It is beyond the scope of this paper to describe all the formulae required in the updating process, as they were developed in a previous work [6]. 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 $n_j \leqslant n_p + i$, $\forall\, j \;\; p \leqslant j \leqslant p+i$, $p+i$ 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, $p \leqslant j \leqslant p+i$, where $\overline{Q}_j$ is defined as

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

Dimensions n_j and l_j of matrices Q_j and identity $\mathbf{1}$ 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 $x_j^{\mathrm{T}} Q_j = b_j^{\mathrm{T}}$. 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

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

Then

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

$$\Longleftrightarrow \begin{cases} \alpha_j = 0 \\ \boxed{Q_j x_j = b_j} \end{cases}$$

the marked expression being the desired result. Analogously $x_j^{\mathrm{T}} Q_j = b_j^{\mathrm{T}}$ 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 $\forall\, j, \;\; p \leqslant j < p+i$, $\overline{Q}_j = E \overline{Q}_p F$, 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:

$$\overline{Q}_p F \overline{x}_j = E^{-1} \overline{b}_j,$$

$$\overline{Q}_p z_j = E^{-1} \overline{b}_j, \quad \text{where } z_j = F \overline{x}_j,$$

$$z_j = \overline{Q}_p^{-1} E^{-1} \overline{b}_j$$

and finally

$\overline{x}_j = F^{-1} z_j$.

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. Computational results

This section will present the results obtained on a set of linear and nonlinear test problems, comparing PPRN with other available codes. PPRN is mainly written in Fortran-77, with the routines for the dynamic memory assignment coded in ANSI-C. All runs were carried out on a Sun Sparc 10/41 (one CPU), having a risc-based architecture, with a 40 MHz clock, about 100 Mips and about 20 Mflops CPU, and 64 Mbytes of main memory (32 real and 32 mapped in disk).

6.1. Tests of linear problems

Two types of linear problems have been employed to test the performance of the code. The first type was obtained from five network generators, while the second one arises from the field of long and short-term hydrothermal coordination of electricity generation.

The first four network generators employed (Rmfgen [14], Grid-on-torus, Gridgen and Gridgraph) were taken from the suite of generators distributed for the First DIMACS International Algorithm Implementation Challenge [13]. They are freely distributed and have been obtained via anonymous ftp from *dimacs.rutgers.edu* at directory */pub/netflow*. These network generators do not consider the case of multicommodity flows, and the output network had to be converted to a multicommodity one. The conversion algorithm is described in Ref. [9]. Besides, although code PPRN can deal with side constraints, the generators produce networks without them. Thus all test problems obtained through generators have no side constraints. Eight particular instances have been created with each of these four generators. 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 input parameters for each generator are fully detailed in Ref. [9]. Each test problem will be denoted as $L_i^{(j)}$, $i = 1, \ldots, 8$, $j = 1, \ldots, 4$, j denoting the generator employed (1 for Rmfgen, 2 for Grid-on-torus, 3 for Gridgen and 4 for Gridgraph).

The fifth generator used is Mnetgen [2], a multicommodity generator without side constraints. Six tests have been performed with this generator; they will be denoted as $L_i^{(5)}$, $i = 1, \ldots, 6$.

The second type of problems was obtained as instances of long and short-term hydrothermal scheduling of electricity generation according to the models proposed in Refs. [24,17] (where a comprehensive explanation of the models can be found). The original nonlinear objective function has been linearized so that it can be solved as a linear problem. The long-term model produces multicommodity tests (that will be denoted as $L_i^{(6)}$, $i = 1, \ldots, 5$), whereas the short-term model gives rise to single-commodity problems with side constraints (that will be denoted as $L_i^{(7)}$, $i = 1, 2$).

Table 1 shows the characteristics of the linear problems. The first column, "test", is the name given to the test instance. Column K denotes the number of commodities considered in the test. The column "#s.c." gives 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. Columns "rows $\hat{A}$" and "columns $\hat{A}$" give the dimensions of the constraint matrix of the standard form of the multicommodity network problem to be solved (that is, with inequality constraints converted into equalities by adding slacks). It can be observed that "rows $\hat{A}$"= K·"nodes"+"arcs"+"#s.c.", and "columns $\hat{A}$"= K·"arcs"+"arcs"+"#s.c." (these last two terms correspond to the slack variables associated to constraints (4) and (5)). The last column shows the optimal objective function value.

PPRN has been compared with the general-purpose package MINOS 5.3 [23], the MCNF85 code [19] (a specialized code for linear multicommodity problems – it does not support side constraints) and the LoQo package [27] (a state of the art primal-dual interior point implementation). The default values were

Table 1
Linear test problems

Test	K	#s.c.	Nodes	Arcs	Rows $\hat{A}$	Columns $\hat{A}$	Optimal value
$L_1^{(1)}$	1	0	2048	9472	2048	9472	375675.1
$L_2^{(1)}$	4	0	2048	9472	17664	47360	2027285.0
$L_3^{(1)}$	8	0	2048	9472	25856	85248	4506263.3
$L_4^{(1)}$	16	0	2048	9472	42240	161024	9870432.7
$L_5^{(1)}$	50	0	128	496	6896	25296	11839382.1
$L_6^{(1)}$	100	0	128	496	13296	50096	27150952.5
$L_7^{(1)}$	150	0	128	496	19696	74896	39835825.1
$L_8^{(1)}$	200	0	128	496	26096	99696	54343948.3
$L_1^{(2)}$	1	0	1500	9000	1500	9000	36896.8
$L_2^{(2)}$	4	0	1500	9000	15000	45000	187962.0
$L_3^{(2)}$	8	0	1500	9000	21000	81000	1197048.7
$L_4^{(2)}$	16	0	1500	9000	33000	153000	5876840.3
$L_5^{(2)}$	50	0	100	600	5600	30600	5207622.6
$L_6^{(2)}$	100	0	100	600	10600	60600	12922703.9
$L_7^{(2)}$	150	0	100	600	15600	90600	22663204.5
$L_8^{(2)}$	200	0	100	600	20600	120600	36829147.5
$L_1^{(3)}$	1	0	2502	5000	2502	5000	94212753.2
$L_2^{(3)}$	4	0	2502	5000	15008	25000	355884986.5
$L_3^{(3)}$	8	0	2502	5000	25016	45000	128743093.6
$L_4^{(3)}$	16	0	2502	5000	45032	85000	253615755.8
$L_5^{(3)}$	50	0	227	450	11800	22950	27853327.9
$L_6^{(3)}$	100	0	227	450	23150	45450	65144564.0
$L_7^{(3)}$	150	0	227	450	34500	67950	27066715.2
$L_8^{(3)}$	200	0	227	450	45850	90450	37964963.7
$L_1^{(4)}$	1	0	976	7808	976	7808	5541980.3
$L_2^{(4)}$	4	0	976	7808	11712	39040	23223474.9
$L_3^{(4)}$	8	0	976	7808	15616	70272	61792270.7
$L_4^{(4)}$	16	0	976	7808	23424	132736	165808232.3
$L_5^{(4)}$	50	0	101	606	5656	30906	1409470.3
$L_6^{(4)}$	100	0	101	606	10706	61206	2940217.3
$L_7^{(4)}$	150	0	101	606	15756	91506	4614971.4
$L_8^{(4)}$	200	0	101	606	20806	121806	6440385.5
$L_1^{(5)}$	4	0	50	104	304	520	378009.3
$L_2^{(5)}$	8	0	300	671	3071	6039	9890447.9
$L_3^{(5)}$	16	0	300	681	5481	11577	18700864.0
$L_4^{(5)}$	16	0	400	821	7221	13957	23697345.2
$L_5^{(5)}$	31	0	300	673	9973	21536	36907046.0
$L_6^{(5)}$	48	0	300	683	15083	33467	54636652.8
$L_1^{(6)}$	4	2	37	117	267	587	−1282480.0
$L_2^{(6)}$	4	12	37	153	313	777	−571304.7
$L_3^{(6)}$	4	18	25	98	216	508	−400688.8
$L_4^{(6)}$	4	3	99	315	714	1578	−995067.1
$L_5^{(6)}$	4	3	685	2141	4884	10708	71115082.6
$L_1^{(7)}$	1	36	85	348	121	384	−16.9344
$L_2^{(7)}$	1	504	1177	4872	1681	5376	−54.43203

Table 2
Results for the linear test problems

	PPRN				MINOS			MCNF85		LoQo	
Test	It.0	It.1	It.2	CPU sec	It.1	It.2	CPU sec	It.	CPU sec	It.	CPU sec
$L_1^{(1)}$	5689	0	0	6.5*	0	4084	112.6		[d]	20	499.65
$L_2^{(1)}$	12103	13	818	98.7*	5	19864	3628.5	7809	588.5		[g]
$L_3^{(1)}$	24486	389	4510	737.9*	9	47769	15147.7	16452	1778.2		[g]
$L_4^{(1)}$	48928	6317	21024	6838.7		[a]		31178	5652*		[g]
$L_5^{(1)}$	7962	2219	5861	275.1*	6669	22077	2639.4	8719	398.6	36	3402.8
$L_6^{(1)}$	15924	5856	32175	2747.0*	20829	123619	29665.3	36717	4305.5		[g]
$L_7^{(1)}$	23729	8879	60974	8069.0*		[b]		55817	11319.0		[g]
$L_8^{(1)}$	32188	11219	88548	15415.3*		[c]		82306	26479.9		[g]
$L_1^{(2)}$	6746	0	74	6.9*	72	8007	160.3		[d]	25	410.4
$L_2^{(2)}$	15464	556	8320	537.9*	312	68429	10594.5	10358	684.3		[g]
$L_3^{(2)}$	31241	3840	40630	4962.2		[b]		43226	4833.0*		[g]
$L_4^{(2)}$	60086	12080	170041	37470.5		[c]		182343	34383.0*		[g]
$L_5^{(2)}$	7348	1313	3660	169.2*	3163	12932	1402.8	9777	466.9	49	5211.1
$L_6^{(2)}$	15053	5091	19170	1625.4*	38924	75696	18085.3	26345	3135.8		[g]
$L_7^{(2)}$	22345	12590	59436	7605.5*	144062	253256	105082.5	74806	15836.2		[g]
$L_8^{(2)}$	29930	31574	127657	22218.4*		[c]		246426	81903.5		[g]
$L_1^{(3)}$	4178	0	0	4.4*	820	2874	97.5		[d]	32	29.8
$L_2^{(3)}$	16590	2544	9429	807.7*	12561	44814	10225.6	22280	1370.7	37	874.1
$L_3^{(3)}$	26161	2500	8620	1409.2*		[b]		25426	2134.1		[g]
$L_4^{(3)}$	51537	12221	49034	14139.8*		[c]		91939	14709.9		[g]
$L_5^{(3)}$	8735	1645	5497	364.9*	3869	16580	3172.8	12533	533.9	39	3180.4
$L_6^{(3)}$	17425	6549	33218	4264.1*		[b]		42553	5381.2		[g]
$L_7^{(3)}$	25693	6052	22811	4480.5*		[a]		37013	6142.8		[g]
$L_8^{(3)}$	53524	10392	43132	11736.8*		[a]		59798	19458.0		[g]
$L_1^{(4)}$	7576	0	0	5.7*	193	4246	70.4		[d]	22	362.1
$L_2^{(4)}$	33544	625	5845	334.5*	7330	68026	9017.3		[e]		[g]
$L_3^{(4)}$	69242	3011	35263	3424.7*	57801	870882	205836.0		[e]		[g]
$L_4^{(4)}$	137543	11892	248422	40974.1*		[c]			[e]		[g]
$L_5^{(4)}$	15113	110	592	39.4*	1617	9779	918.2		[e]		[g]
$L_6^{(4)}$	30825	498	2046	192.2*	5336	31486	6147.6		[e]		[g]
$L_7^{(4)}$	44646	831	2999	415.8*	8862	46991	15236.6		[e]		[g]
$L_8^{(4)}$	59639	1528	7504	1273.4*		[c]			[e]		[g]
$L_1^{(5)}$	310	19	21	0.3*	145	162	1.6	400	4.0	14	0.8
$L_2^{(5)}$	4094	288	1190	21.7*	3125	4173	212.8	5450	112	20	119.9
$L_3^{(5)}$	11019	299	1514	46.5*	5471	6697	733.4	8743	266.2	19	763.1
$L_4^{(5)}$	14190	497	2386	95.0*	5473	7666	1108.3	10240	371.0	22	1909.5
$L_5^{(5)}$	21559	525	5413	293.2*	9570	13547	2747.2	17671	788.0	22	2058.6
$L_6^{(5)}$	34521	695	5930	526.0*	13961	18887	7140.4	23440	1707.6		[g]
$L_1^{(6)}$	184	71	82	0.5*	238	79	2.5		[f]	14	1.0
$L_2^{(6)}$	162	79	86	0.6*	154	230	3.1		[f]	13	1.1
$L_3^{(6)}$	219	276	343	0.9*	222	218	2.3		[f]	23	2
$L_4^{(6)}$	646	434	1508	6.8	542	796	13.7		[f]	18	4.7*
$L_5^{(6)}$	5200	3665	10933	348.5	2372	8333	1110.3		[f]	24	111.6*
$L_1^{(7)}$	228	106	40	0.5*	128	159	2.8		[f]	17	1.0
$L_2^{(7)}$	3750	1921	446	46.0	2777	1812	252.3		[f]	21	30.68*

[a] Too many constraints.
[b] Error during execution.
[c] Not executed (the execution would be too long).
[d] Not executed (single commodity problems were not executed with MCNF85).
[e] Feasibility error.
[f] MCNF85 cannot solve problems with side constraints.
[g] Not enough memory to run this problem.

used for all the adjustable parameters of the codes (only PPRN has been tuned to find the optimal solution of (17) at phase 0 at tests $L_i^{(j)}$, $i = 1,\ldots,8$, $j = 1,\ldots,4$, instead of finding just a feasible one, which is the default option). In all the executions all the codes reached the same optimal objective function value (in fact there were some differences but none significant). Table 2 shows the results obtained with all the codes. For PPRN the information disclosed includes columns "It.0" (number of iterations at phase 0), "It.1" (number of iterations at phase 1), "It.2"(number of iterations at phase 2) and "CPU sec."(CPU seconds spent by the execution). For MINOS the same information is given excluding column "It.0". For MCNF85 and LoQo, besides the "CPU sec." column, column "It." gives the total number of iterations required. For each test the fastest execution (of those successfully finished) is marked with an asterisk (*) in the "CPU sec." column.

In Table 2 it can be observed that there were some problems in the execution of some tests (these are mentioned in the legend at the bottom of the table). Minos could not solve problems with more than 32767 constraints, since it stores the number of rows in a two-bytes signed integer. MCNF85 stopped some executions with a message of "feasibility error", meaning that it had difficulty in finding a feasible point. It can also be seen that the interior point code LoQo required a large amount of memory to solve this kind of problems. Thus, where it would have been possible to take advantage of the interior point methodology, the big examples could not be executed.

6.2. *Tests of nonlinear problems*

Two sets of problems were employed to test the performance of the code for the nonlinear case. The first set are expensive (in computation time) artificial problems, while the second arise from the fields of long- and short-term hydrothermal coordination of electricity generation and traffic assignment.

Three different artificial objective functions were tested. The first two are simple convex functions defined by

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

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

x_{ki} being the flow of the ith arc and kth commodity. The third objective function is derived from that described in Ref. [26], and is defined as

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

where

$$F_k(X_k) = \frac{1}{c_1}\sum_{i=1}^{n} x_{ki}^2 + \frac{1}{c_2}\left[\sum_{i=1}^{n-1}\sqrt{1 + x_{ki}^2 + (x_{ki} - x_{k,i+1})^2} + \frac{1}{c_3}\left(10 + \sum_{i=1}^{n}(-1)^i x_{ki}\right)^4\right] \quad (23)$$

$c_1, c_2, c_3 \in \mathbb{R}$ (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. Three networks were used with these artificial objective functions, which are real ones and were obtained from long-term hydrothermal scheduling problems that will be introduced later. We will refer to these tests as $N_i^{(j)}$, $i = 1,\ldots,3$, $j = 1,\ldots,3$, where the subscript i identifies the network used and the superscript (j), $j = 1,\ldots,3$ refers to the artificial objective function ((6.2), (6.2) or (6.2), respectively).

As in the linear case, the second set of problems was obtained as instances of real problems. These can be divided into two types of models. The first type are long and short-term hydrothermal scheduling problems of electricity generation according to the models proposed in Refs. [24,17] (where a comprehensive explanation of the objective functions can be found). These models are the same as those used in the linear case, but consider the real function instead of a linearized one. The long-term problems will be denoted as $N_i^{(4)}$, $i = 1,\ldots,3$, whereas the short-term ones will be denoted as $N_i^{(5)}$, $i = 1,\ldots,4$. The second type of problems arise from the field of static user equilibrium traffic assignment problems with inelastic demand and

Table 3
Nonlinear test problems

Test	K	#s.c.	Nodes	Arcs	Rows $\hat{A}$	Columns $\hat{A}$
$N_1^{(j)}$ [a]	4	12	37	153	313	777
$N_2^{(j)}$ [a]	4	3	99	315	714	1578
$N_3^{(j)}$ [a]	4	3	685	2141	4884	10708
$N_1^{(5)}$	1	528	1345	4416	1873	4944
$N_2^{(5)}$	1	840	1975	6048	2815	6888
$N_3^{(5)}$	1	840	2479	8064	3319	8904
$N_4^{(5)}$	1	1848	4741	15600	2289	17448
$N_1^{(6)}$	16	0	182	309	3221	5253

[a] $j = 1, \ldots, 4$.

separable link cost functions [4]. This problem has been formulated as a multicommodity problem, considering that each commodity k_{ij} is the traffic flow leaving from origin i and arriving at destination j. A more detailed description of the model and the objective function employed can be found in Ref. [10]. Only one test problem was used with this objective function. This problem corresponds to a subnetwork of the city of Barcelona (thus being a real case network), and will be denoted as $N_1^{(6)}$. Table 3 shows the characteristics of the nonlinear tests employed. The meaning of the columns is the same as in Table 1.

For the nonlinear tests PPRN has been compared only with the general-purpose package MINOS 5.3 [23], since no other specialized code for nonlinear multicommodity flows with side constraints is known. Table 4 shows the results obtained with both codes. The information disclosed is the same as in the Table 2, considering an additional column "$h(x^*)$" with the optimal objective value found with each code. The fastest execution is also marked with an asterisk (*) in the "CPU sec" column.

Some comments should be made about the results presented:

- In the problem $N_3^{(4)}$ PPRN and MINOS obtained different solutions. This is due to the nonconvexity of the objective function for the long-term hydrothermal model.
- In all runs the optimality tolerance required was $\epsilon_{opt} = 10^{-6}$. Only for test problem $N_2^{(4)}$ could this tolerance not be achieved for both codes, due to the nonlinearities in the long-term hydrothermal objective function. In this case, the PPRN code reduced ϵ_{opt}^* more than the MINOS package ($\epsilon_{opt} = 3.1 \times 10^{-3}$ for PPRN whereas MINOS achieves a point where $\epsilon_{opt} = 1.7 \times 10^{-2}$), 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 $s \leqslant 500$ (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 affects heavily the performance in test problems $N_3^{(1)}$, $N_3^{(2)}$ and $N_3^{(3)}$, where the number of superbasic variables at the optimum is very high. 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 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 when the number of superbasic variables is very high, and that the truncated-Newton algorithm seems to be clearly much more efficient than the quasi-Newton update in such cases.

7. Conclusions

The implementation made of primal partitioning for solving multicommodity network flow problems is ef-

Table 4
Results for the nonlinear test problems

	PPRN					MINOS			
Test	It.0	It.1	It.2	$h(x^*)$	CPU sec	It.1	It.2	$h(x^*)$	CPU sec
$N_1^{(1)}$	107	63	654	285515.68	10.4*	163	687	285515.68	17.0
$N_2^{(1)}$	460	226	1459	867915.88	87.2*	617	1753	867915.87	139.7
$N_3^{(1)}$	4083	2896	9458	210894646.82	3086.7*	16598	12401	210894647.06	23579.6
$N_1^{(2)}$	107	63	845	3.9718×10^{10}	10.4*	163	1160	3.9718×10^{10}	22.5
$N_2^{(2)}$	460	226	2127	6.0465×10^{10}	83.1*	689	2297	6.0465×10^{10}	118.4
$N_3^{(2)}$	4083	2896	15463	1.9431×10^{14}	5823.3*	16598	17019	1.9431×10^{14}	16524.7
$N_1^{(3)}$	175	74	1640	971.69	30.9*	256	1638	971.69	34.4
$N_2^{(3)}$	460	226	3539	891.69	236.0*	689	3772	891.69	292.3
$N_3^{(3)}$	4083	2896	18289	212682.61	6831.2*	16598	28314	212682.61	19398.0
$N_1^{(4)}$	175	74	395	-3.7747×10^{12}	3.2*	261	202	-3.7747×10^{12}	3.9
$N_2^{(4)}$	460	226	4457	1.0792×10^{8} [a]	183.1*	574	5092	1.2228×10^{8} [a]	213.9
$N_3^{(4)}$	4083	2813	14186	-7.9171×10^{9} [b]	2504.1*	16069	16569	-6.7860×10^{9} [b]	4284.9
$N_1^{(5)}$	3419	1177	1692	0.4009	110.2*	2653	2522	0.4009	226.5
$N_2^{(5)}$	5209	915	1608	0.8715	156.7*	7044	1216	0.8715	357.8
$N_3^{(5)}$	7044	1216	1850	0.3844	217.3*	2965	2455	0.3844	485.5
$N_4^{(5)}$	11650	3341	6768	1.0920	2026.0*	12257	7507	1.0917[c]	3316.9
$N_1^{(6)}$	869	78	460	288.9697	11.6*	660	526	288.9697	69.6

[a] The required optimality tolerance $\epsilon_{opt} = 10^{-6}$ could not be achieved.
[b] Different local minima were reached.
[c] In this execution the "feasibility tolerance" parameter of the MINOS package was increased considerably to obtain a feasible solution. The different value $h(x^*)$ for MINOS and PPRN may be due to this fact.

ficient, and compares well with the alternative methods tried. Instrumental in the efficiency of the PPRN code is the three-phase procedure used and the special variable dimension update of the factorization of the working matrix.

As shown in Tables 2 and 4 in no case could the general purpose code MINOS employed as a linearly constrained optimization tool outperform the specialised code PPRN, which was to be expected since PPRN uses the same technique as MINOS for linear and for nonlinear objective functions but it can take advantage of the multicommodity network structure.

For linear problems without side constraints PPRN is generally better than the specialised linear multicommodity network flow code MCNF85. Whenever MCNF85 is better it is so by a little margin, whereas PPRN is many times faster than MCNF85 in many cases. It can be observed that MCNF85 only gets to be better than PPRN when the number of commodities is small but its performance gets worse than PPRN as the number of commodities increases.

The iteration counts of PPRN and those of MINOS and MCNF85 are not comparable since phase 0 of PPRN iterates in single-commodity problems.

The general purpose interior point code employed (LoQo) is in some cases better than PPRN but it requires a lot more of workspace. In fact, only the smaller problems could be solved due to memory limitations. In other cases LoQo did similar or quite worse than PPRN. All the same, it would be worth specialising an interior point implementation for multicommodity network flows since it would reduce the memory requirements and would make interior point iterations less time-consuming, thus comparing well with a primal partitioning specialised code.

References

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

[2] Ali, A., and Kennington, J.L., "MNETGEN program documentation", Technical Report 77003, Department of Industrial Engineering and Operations Research, Southern Methodist University, Dallas, TX 75275, USA, 1977.

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

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

[5] Bradley, G.H., Brown, G.G., and Graves, G.W., "Design and implementation of large scale primal transshipment algorithms", *Management Science* 24/1 (1977) 1–34.

[6] Castro, J., "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, Spain, 1993 (written in Catalan).

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

[8] Castro, J., and Nabona, N., "Nonlinear multicommodity network flows through primal partitioning and comparison with alternative methods", in: J. Henry and J.-P. Yvon (eds.), *System Modelling and Optimization. Proceedings of the 16th IFIP Conference*, Springer-Verlag, Berlin, 1994, 875–884.

[9] Castro, J., and Nabona, N., "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, Spain, 1994.

[10] Castro, J., and Nabona, N., "Computational tests of a nonlinear multicommodity network flow code with linear side constraints through primal partitioning", DR 94/05 Statistics and Operations Research Dept., Universitat Politècnica de Catalunya, Barcelona, Spain, 1994.

[11] Choi, I.C., and Goldfarb, D., "Solving multicommodity network flow problems by an interior point method", *SIAM Proceedings on Applied Mathematics* 46 (1990) 58–69, Philadelphia, PA, USA.

[12] Dembo, R.S., and Steihaug, T., "Truncated-Newton algorithms for large-scale unconstrained optimization", *Mathematical Programming* 26 (1983) 190–212.

[13] DIMACS, "The first DIMACS international algorithm implementation challenge: The bench-mark experiments", Technical Report, DIMACS, New Brunswick, NJ, USA, 1991.

[14] Goldfarb, D., and Grigoriadis, M. D., "A computational comparison of the Dinic and network simplex methods for maximum flow", *Annals of Operations Research* 13 (1988) 83–128.

[15] Grigoriadis, M.D., "An efficient implementation of the network simplex method", *Mathematical Programming Study* 26 (1986) 83–111.

[16] Hellerman, E., and Rarick, D., "Reinversion with the preassigned pivot procedure", *Mathematical Programming* 1 (1971) 195–216.

[17] Heredia, F.J., and Nabona, N., "Optimum short-term hydrothermal scheduling with spinning reserve through network flows", accepted for publication in *IEEE Transactions on Power Systems*, 1994.

[18] Kamath, A.P, Karmarkar, N.K., and Ramakrishnan, K.G., "Computational and complexity results for an interior point algorithm on multicommodity flow problems", Presented at *Netflow93*, San Miniato, Italy, October 3–7, 1993.

[19] Kennington, J.L., "A primal partitioning code for solving multicommodity flow problems (version 1)", Technical Report 79008, Department of Industrial Engineering and Operations Research, Southern Methodist University, Dallas, TX, 1979 .

[20] Kennington, J.L., and Helgason, R.V., *Algorithms for Network Programming*, Wiley, New York, 1980.

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

[22] Murtagh, B.A., and Saunders, M.A., "Large-scale linearly constrained optimization", *Mathematical Programming* 14 (1978) 41–72.

[23] Murtagh, B.A., and Saunders, M.A., "MINOS 5.0. User's guide", Dept. of Operations Research, Stanford University, Standord, CA, 1983.

[24] Nabona, N., "Multicommodity network flow model for long-term hydrogeneration optimization", *IEEE Transactions on Power Systems* 8/2 (1993) 395–404.

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

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

[27] Vanderbei, R.J., and Carpenter, T.J., "Symmetric indefinite systems for interior point methods", *Mathematical Programming* 58 (1993) 1–32.