Journal of the Operations Research
Society of Japan
Vol. 32, No. 2, June 1989

INTERIOR METHODS FOR NONLINEAR MINIMUM COST NETWORK FLOW PROBLEMS

Ryuji Katsura
Japan Medical Supply Co., Ltd.

Masao Fukushima
Kyoto University

Toshihide Ibaraki
Kyoto University

(Received March 24, 1988; Revised August 5, 1988)

Abstract In this paper we propose practical algorithms for solving the nonlinear minimum cost network flow problem which has many fields of application such as production-distribution systems, pipe network systems, and communication systems. Here we assume that the problem is defined on an open subset of the affine subspace corresponding to the flow conservation equations. This assumption offers great flexibility in choosing a basis to represent feasible solutions, and the conventional capacitated network flow problems can be put into this framework by exploiting an interior penalty function technique. The algorithms proposed in this paper belong to the class of feasible descent methods which successively generate search directions based on the idea of Newton method. We give some practical strategies of determining search directions which approximate solutions of Newton equations. We also discuss ways of maintaining a desirable basis which makes those strategies effective. We examined the efficiency of the algorithms by means of some computational experiments. The proposed algorithms could practically solve a problem with more than 500 nodes and 1500 arcs, which is quite large as a nonlinear optimization problem.

1. Introduction

Many important problems in engineering and economics are formulated as nonlinear minimum cost network flow problems. In particular, typical applications of such problems may be found in production-distribution systems, pipe network systems, resistive electrical network systems and communication systems. For solving such problems, there have been proposed many algorithms including the Frank-Wolfe method, the convex simplex method and the piecewise linear approximation method (see, e.g., Kennington and Helgason [16]). Among others, recently developed Newton-type algorithms (Dembo [6], Dembo and Klincewicz [7], Klincewicz [17]) are considered the most efficient from the viewpoint of the speed of convergence.

In this paper, we propose Newton-type algorithms for solving nonlinear network flow problems under a slightly different setting from the one of conventional formulation. Specifically, we assume that the flow conservation equations are the only constraints of the problem and that the domain of the cost function is open relative to the affine subspace corresponding to the equality constraints. Therefore, at any feasible point, we can choose a basis in a rather flexible manner, thereby the algorithms may be substantially simplified compared with those which have been developed as direct extentions of the network simplex method [6,7,17]. Although the above assumptions are not standard, any capacitated problem can be put into the present framework by using an interior penalty function technique (Gill et al. [12]). In particular, for problems with linear cost functions, such a transformation is closely related to Karmarkar's algorithm [15] in linear programming, as pointed out by Gill et al. [13].

The algorithms presented here are improvements of the one proposed in [10] and belong to the class of feasible descent methods which use search directions approximating the solutions of Newton equations. In this paper, we present three practical strategies of determining approximate Newton directions; the first two of them replace coefficient matrices of the equations by diagonal approximations, while the third exploits the preconditioned conjugate gradient method (Gill et al. [12]) to solve the equations rather accurately.

As is well known, every basis of the minimum cost network flow problem corresponds to a spanning tree of the graph. Although the proposed algorithms have flexibility in choosing a basis at each feasible point as mentioned above, it is also true that some bases are more desirable than others from numerical viewpoints. In fact, we may characterize a desirable basis as a spanning tree of small weight, where weights are assigned to arcs in relation to the second derivatives of the cost function evaluated at the point under consideration. Since the weights of arcs vary as the iteration proceeds, it is advisable to systematically maintain desirable bases in order for the algorithms to work efficiently. For this purpose, we present two basis updating procedures which construct a minimum spanning tree by partially modifying the spanning tree used on the previous iteration.

This paper is organized as follows. In the next section, we state the problem formulation along with some assumptions. In Section 3, we describe the basic algorithm on which the proposed algorithms are based. In Section 4, we present strategies to determine search directions used in the algorithms. In Section 5, we discuss procedures of updating a desirable basis. Finally in Section 6, we report some computational results in order to demonstrate the efficiency of the proposed algorithms.

2. Problem

In this section, we formally state the problem and make some assumptions which are the same as those in [10].

Consider a connected directed graph $G = (V, E)$ consisting of a set of nodes $V = \{1, 2, \ldots, m\}$ and a set of arcs $E = \{1, 2, \ldots, n\}$. For each arc $j \in E$, let $t(j)$ and $h(j)$ denote its "from" node (or tail) and its "to" node (or head), respectively. In the following, we consider the nonlinear minimum cost network flow problem

$$\text{(2.1)} \quad \text{minimize} \quad f(x_1, x_2, \ldots, x_n) = \sum_{j \in E} f_j(x_j)$$

$$\text{subject to} \quad \sum_{t(j)=i} x_j - \sum_{h(j)=i} x_j = b_i, \qquad i = 1, 2, \ldots, m,$$

where x_j is the flow on arc $j \in E$, and b_i is the requirement at node $i \in V$. The equality constraints in (2.1) are called the flow conservation equations. Node i is a supply node if $b_i > 0$, a transshipment node if $b_i = 0$, and a demand node if $b_i < 0$. It is assumed that the total supply equals the total demand, i.e., $\sum_{i \in V} b_i = 0$.

Let x and b denote the column vectors $x = (x_1, x_2, \ldots, x_n)^T$ and $b = (b_1, b_2, \ldots, b_m)^T$, respectively. Then problem (2.1) may be rewritten in matrix form as follows:

$$\text{(2.2)} \quad \text{minimize} \quad \sum_{j \in E} f_j(x_j)$$

$$\text{subject to} \quad Ax = b.$$

We shall make the following assumptions on the cost function f_j associated with each arc $j \in E$:

(a) The domain of f_j is an open interval (l_j, u_j), and $f_j(x_j) \uparrow \infty$ if $x_j \downarrow l_j$ or $x_j \uparrow u_j$.
(b) f_j is twice continuously differentiable and $f_j''(x_j) > 0$ on its domain.

In (a), we allow $l_j = -\infty$ and/or $u_j = +\infty$. From (b), each f_j is strictly convex on its domain. Notice that these assumptions implicitly assume that the lower and upper bounds are not equal, i.e., $l_j < u_j$. However, the subsequent discussions remain valid even if there exists an arc such that $l_j = u_j$, because the flow x_j on such an arc can be regarded as a constant. In what follows, a flow x will be called a feasible solution of problem (2.2) if it is included in the domain of the cost function f and satisfies the flow conservation equations.

The above assumption (a) is different from those which are assumed in the conventional formulation of the nonlinear network flow problems (Kennington and Helgason [16], Meyer [18]). Usually, bound constraints on the variables are explicitly included as $l_j \leq x_j \leq u_j$ and cost functions f_j are defined on the real line. If we extend the network simplex method to a method for solving such standard problems, the choice of the basis may be restricted by the status of the variables, which depends on whether they are at one of their bounds or not (Dembo and Klincewicz [7]). In our formulation, however, each variable is containeed in the interior of the bound constraint, so that such a combinatorial restriction is not imposed on the choice of the basis at least theoretically. Moreover, it is noted that the standard nonlinear capacitated minimum cost network flow problem with separable costs may be treated within our framework by utilizing a penalty function technique. In fact, the problem

$$(2.3)\quad \text{minimize} \quad \sum_{j\in E} \bar{f}_j(x_j) \qquad \text{subject to} \quad Ax = b, \quad l \leq x \leq u$$

can be transformed into a problem of the form (2.2) , in which f_j is a barrier function defined by

$$(2.4)\quad f_j(x_j) = \bar{f}_j(x_j) - \mu_j \ln(x_j - l_j) - \mu_j \ln(u_j - x_j),$$

where $\mu_j > 0$ is a penalty parameter. Of course, if $l_j = -\infty$ ($u_j = \infty$), then the second (third) term on the right-hand side of (2.4) is vacuous. If $\bar{f}_j$ is convex (not necessarily strictly convex) and twice continuously differentiable, and if at least one of l_j and u_j is finite for each j, then the barrier function f_j defined by (2.4) obviously satisfies assumptions (a) and (b). When the parameters μ_j are small enough, the optimum of the transformed problem may be regarded as a good approximation to an optimal solution of problem (2.3). (Note that in the ordinary barrier methods, the parameters μ_j are decreased to zero using an appropriate controlling scheme [12,13]. In this paper, however, the parameters will be fixed at a very small value throughout the computation.)

3. Basic Algorithm

In this section we describe a basic algorithm for minimizing a nonlinear function subject to linear equality constraints. This algorithm serves as a basis of the algorithms to be presented in the subsequent sections for the solution of the network flow problem (2.2). The algorithm belongs to a class of feasible descent methods and consists of the following steps:

Step 1: Obtain an initial feasible solution x.

Step 2: Compute a search direction p satisfying $Ap = 0$ and $\nabla f(x)^T p < 0$.

Step 3: Carry out line search to determine a steplength $\alpha > 0$ such that $f(x + \alpha p) < f(x)$.
Set $x := x + \alpha p$.

Step 4: If a convergence criterion is satisfied, terminate. Otherwise go to Step 2.

Note that the sequence of points $\{x^k\}$ generated by this algorithm is always contained in the feasible region. Since the domain of the cost function f is an open set, there surely exists a steplength $\alpha > 0$ such that $f(x + \alpha p) < f(x)$ for any search direction p satisfying the condition in Step 2. Therefore the sequence of the cost function values $\{f(x^k)\}$ decreases monotonically. Strictly speaking, we cannot guarantee convergence to an optimal solution theoretically under only the conditions in Steps 2 and 3 of the above algorithm. But under assumption (a) on the cost function, problem (2.2) is essentially equivalent to an unconstrained minimization problem in a subspace of smaller dimension. Thus we may extend a convergence theorem of unconstrained descent methods (Fletcher [9], Polak [20]) to the present algorithm for solving problem (2.2). However we will not pursue this theoretical issue any more, because the chief aim of this paper is to show how the above algorithm spcialized to the network problems performs practically.

In Step 1 of the algorithm, we have to find an initial feasible solution. To get such a solution, we may apply a method analogous to the so-called Big M method in linear programming (Bazarra and Jarvis [3]). This method has been used in Fukushima et al.[10] and is given in Appendix 1. (An alternative way of obtaining an initial feasible solution may be to find a basic feasible solution of the system $Ax = b$, $l_j + \epsilon \le x_j \le u_j - \epsilon$, for sufficiently small $\epsilon > 0$, using a combinatorial strategy [16, pp.244-248].)

There can be a number of ways of obtaining search directions satisfying the conditions of Step 2. Here we adopt a Newton-like method which will be described in detail in the next section. As for the line search in Step 3, we use a practical algorithm of Armijo type (Polak [20]), which is also described in Appendix 2.

4. Search Directions

Let $g = g(x)$ and $H = H(x)$ represent the gradient vector and the Hessian matrix, respectively, of the cost function f evaluated at a feasible solution x, i.e.,

$$g(x) = \begin{pmatrix} f_1'(x_1) \\ \vdots \\ f_n'(x_n) \end{pmatrix}, \quad H(x) = \begin{pmatrix} f_1''(x_1) & & O \\ & \ddots & \\ O & & f_n''(x_n) \end{pmatrix}.$$

Notice that H is a diagonal matrix whose elements are all positive by assumption (b) in Section 2, and hence H is positive definite.

For problem (2.2), the Newton direction at a feasible solution x is given as the optimal solution of the following quadratic programming problem.

$$(4.1) \quad \begin{array}{ll} \text{minimize} & g^T p + \frac{1}{2} p^T H p \\ \text{subject to} & Ap = 0. \end{array}$$

We partition the matrix A into a basic matrix B and a nonbasic matrix N, i.e.,

$$(4.2) \quad A = [B, N], \quad \operatorname{rank} B = m.$$

(Note that, by introducing an artificial arc, we may assume the matrix A to have full rank. See, e.g., Bazarra and Jarvis [3, p.409].) According to (4.2), the following partitions are obtained:

$$(4.3) \quad x = \begin{pmatrix} x_B \\ x_N \end{pmatrix}, \; g = \begin{pmatrix} g_B \\ g_N \end{pmatrix}, \; p = \begin{pmatrix} p_B \\ p_N \end{pmatrix}, \; H = \begin{pmatrix} H_B & O \\ O & H_N \end{pmatrix}.$$

From (4.1), (4.2) and (4.3), we get

$$p_B = -B^{-1} N p_N,$$

i.e.,

$$(4.4) \quad p = \begin{pmatrix} -B^{-1} N \\ I \end{pmatrix} p_N.$$

Substituting these expressions into (4.1), we obtain the equality

$$g^T p + \frac{1}{2} p^T H p = \overline{g}^T p_N + \frac{1}{2} p_N^T \overline{H} p_N,$$

where

$$(4.5) \quad \overline{g} = g_N - N^T B^{-T} \overline{g}_B,$$

$$(4.6) \quad \overline{H} = H_N + N^T B^{-T} H_B B^{-1} N.$$

It then follows that problem (4.1) is equivalent to the following unconstrained optimization problem including only the nonbasic variables p_N.

$$(4.7) \quad \text{minimize} \quad \overline{g}^T p_N + \frac{1}{2} p_N^T \overline{H} p_N.$$

Since the matrix H is positive definite, the matrix $\overline{H}$ is also positive definite. Therefore the solution p_N of problem (4.7) is given by solving the linear equations

$$(4.8) \quad \overline{H} p_N + \overline{g} = 0.$$

For the search direction p computed from (4.8) and (4.4), the following relation holds by the positive definiteness of $\overline{H}$.

(4.9) $g^T p = \overline{g}^T p_N = -p_N^T \overline{H} p_N < 0.$

Hence the search direction p thus obtained is a descent direction of the cost function f at x. Also it is obvious from (4.4) that p is a feasible direction. Therefore the search direction p satisfies the two conditions in Step 2 of the algorithm described in the previous section.

Note that in (4.4) and (4.5) we must solve the equations having B or B^T as the coefficient matrix. As in the network simplex method (Bazarra and Jarvis [3], Kennington and Helgason [16]), we can solve those equations efficiently by exploiting the special feature that basis B is associated with a spanning tree in graph G. In the actual computations, we can also use the data structures which are useful in the network simplex method (Ali et al.[1], Barr et al.[2]).

Now we turn our attention to practical methods of computing search directions. Bearing in mind that for large-scale problems, it may be extremely expensive to solve the equation (4.8) exactly. We shall consider three ways of computing search directions. The first two use a diagonal approximation to the coefficient matrix $\overline{H}$ in (4.8), while the third attempts to find the solution of (4.8) using an iterative method.

(i) Substituting H_N for $\overline{H}$: If the second term on the right hand side of (4.6) is sufficiently smaller than the first term, it would be valid to substitute H_N for $\overline{H}$ (Dembo and Klincewicz [7], Fukushima et al.[10]). Since H_N is diagonal, we can solve the following equation in a trivial manner.

(4.10) $H_N p_N + \overline{g} = 0.$

It is ensured in a similar way to (4.9) that the search direction p obtained from (4.10) together with (4.4) satisfies the two conditions in Step 2 of the basic algorithm.

(ii) Substituting diag $\overline{H}$ for $\overline{H}$: The approximation used in (i) completely loses the H_B part of the second order information on the cost function. Here, in order to take into account an effect of H_B, we incorporate the diagonal elements of the second term on the right hand side of (4.6) (Dembo and Klincewicz [7]). Let D denote the diagonal part of $\overline{H}$. Then the equation

(4.11) $D p_N + \overline{g} = 0$

can also be solved trivially. Since D is also positive definite, the search direction p computed from (4.11) along with (4.4) satisfies the conditions in Step 2 of the basic algorithm.

It may be worth mentioning a procedure for computing the elements of D efficiently. Let

$$Q = B^{-1} N$$

and let q_j and n_j denote the column vectors of Q and N, respectively. Then we have

$$Bq_j = n_j,$$

showing that q_j is the vector whose ith element is 1 or -1 if basic arc i is used to represent nonbasic arc j, and zero otherwise (Bazarra and Jarvis [3]). Using Q, we rewrite (4.6) as

$$\overline{H} = H_N + Q^T H_B Q,$$

from which the diagonal elements D_{jj} of $\overline{H}$ are given by

$$D_{jj} = H_{jj} + q_j^T H_B q_j.$$

Thus, we obtain the formula

$$D_{jj} = H_{jj} + \sum_{i \in S(j)} H_{ii},$$

where $S(j)$ is the set of the basic arcs which constitute the cycle connecting the "from" and "to" nodes of the nonbasic arc j (Dembo and Klincewicz [7]).

(iii) Solving (4.8) by the conjugate gradient (CG) method: We apply the preconditioned conjugate gradient (PCG) method to solve (4.8) (Dembo [6], Klincewicz [17]). When we use the CG method, the rate of convergence depends highly upon the distribution of eigenvalues of the coefficient matrix $\overline{H}$. In this respect, it is practically useful to transform (4.8) by utilizing a preconditioning matrix so that the coefficient matrix of the preconditioned equation has as many unit eigenvalues as possible (Gill et al.[12]). Here we use the diagonal matrix D given in (ii) as a preconditioning matrix.

In the CG method, the matrix $\overline{H}$ appears only in a matrix-vector product of the form

$$\overline{H}y = H_N y + N^T B^{-T} H_B B^{-1} N y,$$

where y is some vector. The first term on the right hand side is computed trivially. The complicated second term can be calculated efficiently without any additional storage by making use of the special structure of the basis B.

When the CG method is incorporated in the basic algorithm, it may be useful to terminate the CG iterations by taking into account a discrepancy between the current solution x and the optimal solution of problem (2.2). This idea is based on the observation (Dembo and Steihaug [8]) that, when far from the solution, it is not justified to solve equation (4.8) exactly with much computational effort.

5. Basis Update

In the previous section, we discussed ways of determining a search direction p by solving the equation (4.8) with $\overline{H}$ being replaced by a diagonal matrix. The validity of this substitution depends upon whether the second term on the right hand side of (4.6) is sufficiently smaller than the first term. In other words, the strategies (i) and (ii) described in Section 4 are considered reasonable if the elements of H_B are much smaller than those of H_N.

In network problems, a basis corresponds to a spanning tree of the graph (Bazarra and Jarvis [3], Kennington and Helgason [16]). Thus we may determine a "desirable" basis by constructing a minimum spanning tree of graph G in which the weight w_j of arc j is given by $f_j''(x_j)$.

In the context of solving problem (2.2) iteratively, we have to successively obtain such desirable trees in G where the weights of arcs vary as the iteration proceeds. Therefore, it seems practical to find a minimum spanning tree by partially modifying the tree that was used at the previous iteration. In fact, in the later stage of the iterations, it is expected that the number of basic arcs to be exchanged is relatively small, since the weights of arcs will be close to those at the previous iteration. So it may be advantageous to use such an updating procedure, instead of finding a minimum spanning tree from scratch at each iteration by using minimum spanning tree algorithms available in the literature (Iri et al.[14], Papadimitriou and Steiglitz [19]).

Such a procedure of updating a minimum spanning tree may be constructed on the basis of cutsets or circuits of graph G.

5.1. Procedure Based on Cutsets

Given a tree T of graph G and arc $k \in T$, we define a cutset $C^*(E-T|k)$ by

$$C^*(E-T|k) = \{j \in E | t(j) \in V_1, h(j) \in V_2\} \cup \{j \in E | t(j) \in V_2, h(j) \in V_1\},$$

where V_1 and V_2 denote the sets of nodes in the connected components of T with arc k removed (Iri et al.[14]).

We present a procedure based on the idea that basic arcs with larger weight should have higher priority of leaving the basis. This procedure, which is an improvement of the one given in Fukushima et al.[10], scans a list of basic arcs by examining the property stated in the next proposition.

Proposition 1. (Cheriton and Tarjan [4]): For any arc $k \in T$, there exists a minimum spanning tree containing an arc $s \in C^*(E-T|k)$ such that $w_s = \min \{w_j | j \in C^*(E-T|k)\}$.

Now we explicitly state the basis updating procedure based on the cutsets.

Procedure CS

```
    **Input:** A connected directed graph $G = (V, E)$, a spanning tree $T$ of $G$, and arc weights $w_j$.
    **Output:** The minimum spanning tree $T^*$ of $G$.
    **begin** $\overline{w} :=$ max $\{w_j | j \in T\}$, $\underline{w} :=$ min $\{w_j | j \in E - T\}$,
    $J_B := \{j | j \in T, \underline{w} \leq w_j \leq \overline{w}\}$, $J_N := \{j | j \in E - T, \underline{w} \leq w_j \leq \overline{w}\}$ ;
    (**comment:** initialize; $J_B$ and $J_N$ are the sets of possibly leaving and entering arcs,
    respectively.)
    **while** $J_B \neq \emptyset$ **do**
        **begin**
        let $k$ be an arc of maximum weight in $J_B$;
        **if** $J_N \cap C^*(E - T|k) \neq \emptyset$ **then**
            **begin**
            let $s$ be an arc of minimum weight in $J_N \cap C^*(E - T|k)$;
            **if** $w_s < w_k$ **then** $T := T \cup \{s\} - \{k\}, J_N := J_N - \{s\}$;
            **end**
        **end**
        $J_B := J_B - \{k\}$;
    $T^* := T$;
    **end**
```

Note that this procedure checks only those arcs which are contained in the subset $J_B \cup J_N$ of E specified in the initialization step. Because, by the difinition of $\overline{w}$ and $\underline{w}$, it is ensured in the initialization step that each arc in $T - J_B$ belongs to the minimum spanning tree T^* while any arc not in $T \cup J_N$ never belongs to T^*.

The validity of this procedure can be ascertained as follows: At each iteration, either arc k is judged to remain in T because it has the minimum weight in the cutset $C^*(E - T|k)$, or the arc of minimum weight other than arc k in $C^*(E - T|k)$ enters T in place of arc k. In any case, Proposition 1 guarantees that the arc which is decided to be included in T constitutes an arc of the minimum spanning tree T^*.

In the computational experience to be reported in the next section, we try two ways of implementing procedure CS as a subroutine of the basic algorithm. First one is to carry out the procedure completely at every iteration of the basic algorithm. The second is to truncate the procedure after examining a certain number of cutsets, except on some predetermined iterations where the procedure is executed completely. In the second method, we cannot necessarily get a minimun spanning tree, but this modification may offer significant saving of computational effort.

5.2. Procedure Based on Circuits

When a nonbasic arc s is added to a basis tree T of graph G, a cycle is determined uniquely. We define the circuit $C(T|s)$ as the set of those arcs which are contained in this cycle (Iri et al.[14]).

The procedure presented here is based on the idea that such nonbasic arcs with smaller weight should have higher priority of entering the basis. This procedure checks a list of nonbasic arcs by ascertaining the property stated in the next proposition.

Proposition 2. (Tarjan [21]): T is a minimum spanning tree of G if and only if, for each nonbasic arc s, w_s is at least as large as the weight of any arc in $C(T|s)$.

Now we present the basis updating procedure based on the circuits of the graph.

Procedure CC

```
    **Input**: A connected directed graph $G = (V, E)$, a spanning tree $T$ of $G$, and arc weights $w_j$.
    **Output**: The minimum spanning tree $T^*$ of $G$.
    **begin**
    $\overline{w} := \max \{w_j | j \in T\}$, $\underline{w} := \min \{w_j | j \in E - T\}$,
    $J_B := \{j | j \in T, \underline{w} \leq w_j \leq \overline{w}\}$, $J_N := \{j | j \in E - T, \underline{w} \leq w_j \leq \overline{w}\}$ ;
    (**comment**: initialize; $J_B$ and $J_N$ are the sets of possibly leaving and entering arcs,
    respectively.)
    **while** $J_N \neq \emptyset$ **do**
        **begin**
        let $s$ be an arc of minimum weight in $J_N$;
        **if** $J_B \cap C(T|s) \neq \emptyset$ **then**
            **begin**
            let $k$ be an arc of maximum weight in $J_B \cap C(T|s)$;
            **if** $w_s < w_k$ **then** $T := T \cup \{s\} - \{k\}, J_B := J_B - \{k\}$;
            **end**
        **end**
        $J_N := J_N - \{s\}$;
    $T^* := T$;
    **end**
```

The validity of procedure CC can be ascertained as follows: At each iteration, either arc s is judged not to enter the basis because it has the largest weight in the circuit $C(T|s)$, or the arc of maximum weight in $C(T|s)$ leaves T in place of arc s. In any case, Proposition 2 guarantees that the arc which is decided not to be included in T never constitutes an arc of the minimum spanning tree.

In the previous section, we mentioned the possibility that the basic algorithm incorporates incomplete implementation of procedure CS, which allows partial examination of the set of cutsets. By analogy, it may appear that such modification is also applicable to procedure CC. The computational experience has revealed, however, that it does not lead to any favorable result. This is because, unlike the truncated versions of procedure CS, truncated versions of procedure CC are very likely to fail to let some arcs with extremely large weight leave the basis tree. In this case, the computed directions will completely differ from the Newton directions and the computational efficiency of the algorithm will be seriously affected.

6. Computational Results

We have conducted numerical experiments with the proposed algorithms for several test problems derived from the capacitated problems of the form (2.3) through the transformation (2.4). A variety of algorithms have been proposed for the solving of problems (2.3). For the purpose of comparison, we have adopted the Frank-Wolfe (F-W) method which is one of the popular methods for nonlinear network flow problems (Kennington and Helgason [16]). The Frank-Wolfe method solves a linear minimum cost network flow problem as a subproblem at each iteration. We used the code written by Laszlo Hars as a subroutine for solving the subproblems, and the golden section algorithm for line search. All programs were coded in FORTRAN 77 and the runs were made in double precision on a FACOM M-780/30 computer.

6.1. Test Problems

For the test problems, we used lattice networks as shown in Fig. 1, where the nodes in the left-most column are supply nodes ($b_i > 0$), the nodes in the right-most column are demand nodes ($b_i < 0$), and the other nodes are transshipment nodes ($b_i = 0$). All the arc cost functions are of the form

$$f_j(x_j) = c_j x_j + d_j x_j^5 - \mu \ln x_j - \mu \ln(u_j - x_j),$$

where $\mu = 10^{-9}$. The constants b_i, c_j, d_j and u_j are random numbers uniformly generated in the intervals (1,10), (0,20), $(10^{-3}, 10^{-2})$ and (5,10), respectively.

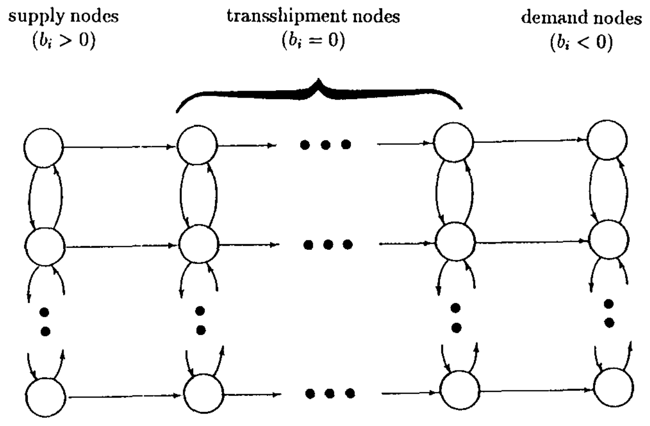

Fig. 1 Lattice network

The sizes of the test problems are as follows:

(1) 56 nodes and 146 arcs,

(2) 121 nodes and 330 arcs,

(3) 256 nodes and 720 arcs, and

(4) 529 nodes and 1518 arcs.

6.2. Convergence Criterion

The convergence criterion used for the proposed algorithms and the Frank-Wolfe method is defined on the basis of the relative error in the cost function, i.e.,

$$|f - f^*|/f^* \leq 10^{-3},$$

where f^* is the optimal value of the test problem. Since the true value of f^* is unknown, we substitute for f^* an estimate of f^* obtained by the Frank-Wolfe method. More precisely, since the Frank-Wolfe method generates a sequence of lower bounds converging to the optimum value, f^* is replaced by its lower bound which is obtained by running the Frank-Wolfe method for sufficiently long time, i.e., 5 minutes for problems (1) and (2), and 15 minutes for problems (3) and (4).

6.3. Results

The basic algorithm has been applied to each test problem using various direction finding strategies and basis updating procedures, that are described in Sections 4 and 5, respectively. The results are summarized in Tables 1(i)-4(iii). For example, Table 1(i) shows the results for problem (1), obtained by the algorithm with direction finding strategy (i) and various basis updating procedures. For comparison purposes, the tables also contain the results of the F-W method. Each table consists of the following items:

(a) the marginal/cumulative number of iterations required to achieve the levels of accuracy 10%, 1% and 0.1%;

(b) the number of basis updates required to achieve the level of accuracy 0.1%;

(c) the details of CPU time required at each step of the basic algorithm (CPU time per iteration is shown in the parentheses).

In order to help more intuitive understanding, the details of CPU time for problem (4) are also illustrated in Fig. 2.

In what follows, we first appraise basis updating procedures CS and CC, and then compare direction finding strategies (i), (ii) and (iii).

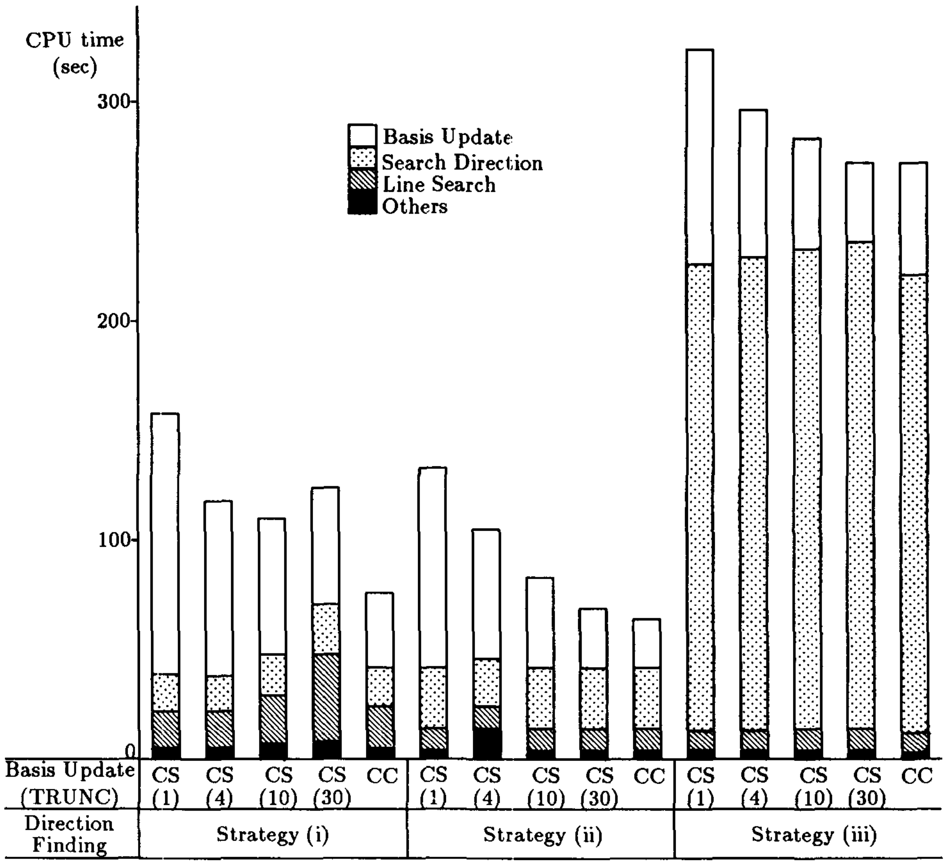

Fig. 2 Results for Problem (4)

Table 1(i). Results for Problem 1, with direction finding strategy (i)

	CS				CC	F-W
	TRUNC=1	TRUNC=4	TRUNC=10	TRUNC=30		
(a) Iterations						
phase I	54	55	56	117*	54	
10%	42 / 96	53 / 108	49 / 105		42 / 96	2
phase II 1%	28 / 124	18 / 126	15 / 120		28 / 124	12 / 14
0.1%	22 / 146	42 / 168	14 / 134		22 / 146	74 / 88
(b) Number of						
basis updates	406	366	250		406	
(c) CPU time (s)						
Basis update	0.488(0.0033)	0.424(0.0025)	0.318(0.0024)		0.312(0.0021)	
Search direction	0.187(0.0013)	0.222(0.0013)	0.177(0.0013)		0.193(0.0013)	
Line search	0.185(0.0013)	0.240(0.0014)	0.175(0.0013)		0.184(0.0013)	
Others	0.073(0.00050)	0.093(0.00055)	0.070(0.00052)		0.078(0.00053)	
Total	0.933(0.0064)	0.979(0.0058)	0.740(0.0055)		0.767(0.0053)	1.552(0.018)

Table 1(ii). Results for Problem 1, with direction finding strategy (ii)

	CS				CC	F-W
	TRUNC=1	TRUNC=4	TRUNC=10	TRUNC=30		
(a) Iterations						
phase I	53	55	55	112*	53	
10%	45 / 98	40 / 95	42 / 97		45 / 98	2
phase II 1%	23 / 121	24 / 119	19 / 116		23 / 121	12 / 14
0.1%	12 / 133	16 / 135	11 / 127		12 / 133	74 / 88
(b) Number of						
basis updates	363	332	235		363	
(c) CPU time (s)						
Basis update	0.457(0.0034)	0.372(0.0028)	0.309(0.0024)		0.300(0.0023)	
Search direction	0.237(0.0018)	0.239(0.0018)	0.228(0.0018)		0.245(0.0018)	
Line search	0.150(0.0011)	0.159(0.0012)	0.139(0.0011)		0.144(0.0011)	
Others	0.074(0.00056)	0.076(0.00056)	0.073(0.00057)		0.075(0.00056)	
Total	0.918(0.0069)	0.846(0.0063)	0.749(0.0059)		0.764(0.0057)	1.552(0.018)

Table 1(iii). Results for Problem 1, with direction finding strategy (iii)

	CS				CC	F-W
	TRUNC=1	TRUNC=4	TRUNC=10	TRUNC=30		
(a) Iterations						
phase I	65	91	61	78	65	
10%	52 / 117	52 / 143	53 / 114	38 / 116	52 / 117	2
phase II 1%	20 / 137	20 / 163	20 / 134	20 / 136	20 / 137	12 / 14
0.1%	16 / 153	16 / 179	18 / 151	5 / 141	16 / 153	74 / 88
(b) Number of						
basis updates	182	167	166	129	182	
(c) CPU time (s)						
Basis update	0.524(0.0034)	0.498(0.0028)	0.358(0.0024)	0.304(0.0022)	0.323(0.0021)	
Search direction	1.690(0.011)	2.024(0.011)	1.727(0.011)	13.555(0.096)	1.683(0.011)	
Line search	0.195(0.0013)	0.215(0.0012)	0.177(0.0012)	0.161(0.0011)	0.179(0.0012)	
Others	0.078(0.00051)	0.099(0.00055)	0.087(0.00058)	0.079(0.00056)	0.084(0.00055)	
Total	2.487(0.016)	2.836(0.016)	2.349(0.016)	14.099(0.10)	2.269(0.015)	1.552(0.018)

Table 2(i). Results for Problem 2, with direction finding strategy (i)

	CS				CC	F-W
	TRUNC=1	TRUNC=4	TRUNC=10	TRUNC=30		
(a) Iterations						
phase I	123	121	127	254*	123	
10%	113 / 236	116 / 237	116 / 243		113 / 236	3
phase II 1%	65 / 301	67 / 304	111 / 354		65 / 301	28 / 31
0.1%	121 / 422	148 / 452	201 / 555		122 / 423	216 / 247
(b) Number of						
basis updates	1050	912	811		1050	
(c) CPU time (s)						
Basis update	3.9(0.0092)	3.1(0.0069)	3.0(0.0054)		2.0(0.0047)	
Search direction	1.3(0.0031)	1.4(0.0031)	1.7(0.0031)		1.3(0.0031)	
Line search	1.5(0.0036)	1.8(0.0040)	2.5(0.0045)		1.5(0.0035)	
Others	0.4(0.00095)	0.4(0.00088)	0.6(0.0011)		0.4(0.00095)	
Total	7.1(0.0017)	6.7(0.015)	7.8(0.014)		5.2(0.012)	12.5(0.051)

Table 2(ii). Results for Problem 2, with direction finding strategy (ii)

	CS				CC	F-W
	TRUNC=1	TRUNC=4	TRUNC=10	TRUNC=30		
(a) Iterations						
phase I	126	125	123	472*	126	
10%	109 / 235	111 / 236	111 / 234		109 / 235	3
phase II 1%	38 / 273	50 / 286	55 / 289		38 / 273	28 / 31
0.1%	31 / 304	30 / 316	12 / 301		31 / 304	216 / 247
(b) Number of basis updates	943	880	678		943	
(c) CPU time (s)						
Basis update	3.2(0.011)	2.6(0.0082)	2.0(0.0066)		1.7(0.0056)	
Search direction	1.3(0.0043)	1.4(0.0044)	1.3(0.0043)		1.3(0.0043)	
Line search	0.8(0.0026)	0.8(0.0025)	0.8(0.0027)		0.8(0.0026)	
Others	0.4(0.0013)	0.3(0.00095)	0.3(0.0010)		0.3(0.00099)	
Total	5.7(0.019)	5.1(0.016)	4.4(0.015)		4.1(0.013)	12.5(0.051)

Table 2(iii). Results for Problem 2, with direction finding strategy (iii)

	CS				CC	F-W
	TRUNC=1	TRUNC=4	TRUNC=10	TRUNC=30		
(a) Iterations						
phase I	113	113	113	113	113	
10%	99 / 212	99 / 212	101 / 214	101 / 214	99 / 212	3
phase II 1%	42 / 254	42 / 254	39 / 253	39 / 253	42 / 254	28 / 31
0.1%	23 / 277	26 / 280	25 / 278	26 / 279	23 / 277	216 / 247
(b) Number of basis updates	392	388	359	339	392	
(c) CPU time (s)						
Basis update	3.2(0.012)	2.4(0.0086)	1.8(0.0065)	1.5(0.0054)	1.5(0.0054)	
Search direction	10.4(0.038)	10.3(0.037)	10.3(0.037)	87.5(0.31)	10.4(0.038)	
Line search	0.7(0.0025)	0.7(0.0025)	0.7(0.0025)	0.7(0.0025)	0.7(0.0025)	
Others	0.3(0.0011)	0.3(0.0011)	0.3(0.0011)	0.3(0.0011)	0.3(0.0011)	
Total	14.6(0.053)	13.7(0.049)	13.1 (0.047)	90.0(0.32)	12.9(0.047)	12.5(0.051)

Table 3(i). Results for Problem 3, with direction finding strategy (i)

	CS				CC	F-W
	TRUNC=1	TRUNC=4	TRUNC=10	TRUNC=30		
(a) Iterations						
phase I	264	269	267	268	264	
10%	236 / 500	229 / 498	234 / 501	242 / 510	236 / 500	4
phase II 1%	175 / 675	207 / 705	207 / 708	241 / 751	175 / 675	28 / 32
0.1%	111 / 786	234 / 939	314 / 1022	334 / 1085	131 / 806	215 / 247
(b) Number of						
basis updates	2905	2675	2254	1418	2914	
(c) CPU time (s)						
Basis update	27.7(0.035)	21.2(0.023)	16.8(0.016)	13.7(0.013)	9.6(0.012)	
Search direction	5.6(0.0071)	6.5(0.0069)	7.0(0.0068)	7.4(0.0068)	5.7(0.0071)	
Line search	5.0(0.0064)	7.4(0.0079)	8.6(0.0084)	11.6(0.011)	5.3(0.0066)	
Others	1.8(0.0023)	2.1(0.0022)	2.3(0.0023)	2.5(0.0023)	1.8(0.0022)	
Total	40.1(0.051)	37.3(0.040)	34.7(0.034)	35.2(0.032)	22.4(0.028)	38.6(0.16)

Table 3(ii). Results for Problem 3, with direction finding strategy (ii)

	CS				CC	F-W
	TRUNC=1	TRUNC=4	TRUNC=10	TRUNC=30		
(a) Iterations						
phase I	262	268	267	279	262	
10%	241 / 503	241 / 509	240 / 507	243 / 522	241 / 503	4
phase II 1%	112 / 615	120 / 629	118 / 625	111 / 633	112 / 615	28 / 32
0.1%	37 / 652	25 / 654	18 / 643	35 / 668	37 / 652	215 / 247
(b) Number of						
basis updates	2657	2483	1995	1442	2657	
(c) CPU time (s)						
Basis update	24.6(0.038)	17.7(0.027)	12.5(0.019)	9.6(0.014)	8.4(0.013)	
Search direction	6.8(0.010)	6.8(0.010)	6.7(0.010)	6.9(0.010)	6.7(0.010)	
Line search	3.6(0.0055)	3.6(0.0055)	3.5(0.0054)	3.7(0.0055)	3.6(0.0055)	
Others	1.5(0.0023)	1.5(0.0023)	1.5(0.0023)	1.6(0.0024)	1.5(0.0023)	
Total	36.5(0.056)	29.6(0.045)	24.2(0.038)	21.7(0.032)	20.2(0.031)	38.6(0.16)

Table 3(iii). Results for Problem 3, with direction finding strategy (iii)

	CS				CC	F-W
	TRUNC=1	TRUNC=4	TRUNC=10	TRUNC=30		
(a) Iterations						
phase I	272	267	268	275	272	
phase II 10%	219 / 491	218 / 485	221 / 489	221 / 496	220 / 492	4
phase II 1%	86 / 577	88 / 573	86 / 575	87 / 583	851 / 577	28 / 32
phase II 0.1%	54 / 631	56 / 628	58 / 633	60 / 643	53 / 630	215 / 247
(b) Number of basis updates	891	867	853	818	887	
(c) CPU time (s)						
Basis update	25.7(0.041)	17.3(0.028)	12.4(0.020)	9.3(0.014)	8.0(0.013)	
Search direction	68.5(0.11)	67.0(0.11)	68.1(0.11)	68.1(0.11)	69.7(0.11)	
Line search	3.5(0.0055)	3.5(0.0056)	3.6(0.0057)	3.6(0.0056)	3.5(0.0056)	
Others	1.5(0.0024)	1.5(0.0024)	1.5(0.0024)	1.5(0.0023)	1.5(0.0024)	
Total	99.2(0.16)	89.3(0.14)	85.6(0.14)	82.6(0.13)	82.7(0.13)	38.6(0.16)

Table 4(i). Results for Problem 4, with direction finding strategy (i)

	CS				CC	F-W
	TRUNC=1	TRUNC=4	TRUNC=10	TRUNC=30		
(a) Iterations						
phase I	556	558	559	564	556	
phase II 10%	553 / 1109	551 / 1109	550 / 1109	584 / 1148	551 / 1107	4
phase II 1%	374 / 1483	365 / 1474	531 / 1640	805 / 1953	372 / 1479	36 / 40
phase II 0.1%	438 / 1921	383 / 1857	539 / 2181	809 / 2762	580 / 2059	373 / 413
(b) Number of basis updates	7876	7276	5932	4172	7872	
(c) CPU time (s)						
Basis update	119.3(0.062)	79.7(0.043)	63.3(0.029)	54.8(0.020)	33.7(0.016)	
Search direction	16.8(0.0087)	16.4(0.0088)	18.6(0.0085)	22.5(0.0081)	17.7(0.0086)	
Line search	17.3(0.0090)	16.8(0.0090)	22.4(0.010)	39.7(0.014)	19.4(0.0094)	
Others	4.9(0.0026)	4.8(0.0026)	5.6(0.0026)	7.1(0.0026)	5.2(0.0025)	
Total	158.3(0.082)	117.6(0.063)	110.0(0.050)	124.1(0.045)	76.0(0.037)	139.7(0.34)

Table 4(ii). Results for Problem 4, with direction finding strategy (ii)

	CS				CC	F-W
	TRUNC=1	TRUNC=4	TRUNC=10	TRUNC=30		
(a) Iterations						
phase I	556	541	553	556	555	
10%	561 / 1116	564 / 1105	576 / 1129	559 / 1115	561 / 1116	4
phase II 1%	208 / 1324	208 / 1313	208 / 1337	212 / 1327	204 / 1320	36 / 40
0.1%	64 / 1388	70 / 1383	56 / 1393	72 / 1399	70 / 1390	373 / 413
(b) Number of						
basis updates	7508	6484	5337	3845	7491	
(c) CPU time (s)						
Basis update	97.8(0.070)	69.6(0.050)	48.0(0.034)	34.1(0.024)	28.6(0.021)	
Search direction	21.6(0.016)	21.6(0.016)	21.9(0.016)	21.7(0.016)	21.6(0.016)	
Line search	9.7(0.0070)	9.7(0.0070)	9.8(0.0070)	9.7(0.0069)	9.8(0.0071)	
Others	3.6(0.0026)	13.7(0.0099)	3.7(0.0027)	3.6(0.0026)	3.6(0.0026)	
Total	132.7(0.096)	104.6(0.076)	83.3(0.060)	69.3(0.050)	63.6(0.046)	139.7(0.34)

Table 4(iii). Results for Problem 4, with direction finding strategy (iii)

	CS				CC	F-W
	TRUNC=1	TRUNC=4	TRUNC=10	TRUNC=30		
(a) Iterations						
phase I	481	481	606	635	481	
10%	507 / 988	510 / 991	530 / 1136	522 / 1157	508 / 989	4
phase II 1%	210 / 1198	205 / 1196	195 / 1331	198 / 1355	209 / 1198	36 / 40
0.1%	105 / 1303	107 / 1303	100 / 1431	103 / 1458	100 / 1298	373 / 413
(b) Number of						
basis updates	1917	1888	1848	1771	1900	
(c) CPU time (s)						
Basis update	98.4(0.076)	67.4(0.052)	50.1(0.035)	35.6(0.024)	26.5(0.020)	
Search direction	213.2(0.16)	216.1(0.17)	219.2(0.15)	222.4(0.15)	209.0(0.16)	
Line search	9.2(0.0071)	9.3(0.0071)	10.0(0.0070)	10.2(0.0070)	9.2(0.0071)	
Others	3.5(0.0027)	3.5(0.0027)	3.7(0.0026)	3.9(0.0027)	3.4(0.0026)	
Total	324.2(0.25)	296.2(0.23)	283.1(0.20)	272.0(0.19)	248.2(0.19)	139.7(0.34)

6.3.1. Comparison of Basis Updating Procedures CS and CC

We introduce parameter TRUNC which is a key to the implimentation of procedure CS. Namely, we execute procedure CS completely every TRUNC iterations, and at the other iterations we truncate procedure CS when the number of examined cutsets exceeds $|J_B|$/TRUNC. Particularly, TRUNC = 1 indicates that we do not truncate the procedure at all.

Every table shows that CS with TRUNC = 1 always requires more CPU time than CC. This indicates that, when completely implemented, procedure CS is less effective than CC. This may be considered because, unlike fundamental cutsets, fundamental circuits can be found by utilizing a simple data structure [1,2,16].

As far as procedure CS is concerned, as TRUNC becomes large, the number of basis updates decreases and accordingly less CPU time is required for basis updates. Also, in many cases, the time saved in basis updates is almost equal to the reduction in the total running time. In particular, the running time of the algorithm using CS with large TRUNC is often comparable to that of the algorithm using CC. However, it can also happen that large TRUNC causes a considerable increase in the total number of iterations, because the search directions may significantly deviate from Newton directions during a fairly large number of iterations. In fact, Tables 1(i), 1(ii), 2(i) and 2(ii) show that for problems (1) and (2), the algorithm with CS(TRUNC = 30) could not get through with phase I, namely, could not obtain an initial feasible solution.

Note that the lattice network of Fig. 1 is sparse because the number of arcs is of the same order as the number of nodes. For a sparse network, the set J_N of nonbasic arcs that can be neglected in the search for an entering arc in general consists of a rather large portion of the entire set of nonbasic arcs, compared with dense networks such as the complete graphs. Therefore, the idea of updating the basis tree seems to work more effectively for dense networks. In fact, Fukushima et al.[11] have recently proposed a modification of procedure CC and have shown by computational experiments that dense graphs are more amenable to updating minimum spanning trees than sparse graphs.

6.3.2. Comparison of Direction Finding Strategies

The algorithm with strategy (i) usually requires a fairly large number of iterations to reach the level of accuracy 0.1% from the level of 1%. In particular, the examples shown in Table 2(i) have difficulty in achieving the level of accuracy 0.1%. This is considered because the directions obtained by strategy (i) do not approximate Newton directions satisfactorily. Moreover the algorithm with strategy (i) is somewhat sensitive to the value of TRUNC, in the sense that the total number of iterations gradually increases as TRUNC becomes larger.

The algorithm with strategy (ii) generally requires far less iterations to attain the level of accuracy 0.1% from the level of 1%, compared with the algorithm with strategy (i). Moerover, the decrease in total number of iterations well compensates for the increase in CPU time to compute search directions. Also, the number of iterations is not affected by the change of TRUNC, and hence strategy (ii) is regarded as more stable than strategy (i). These advantages of strategy (ii) seem to result from the fact that it produces search directions which approximate Newton directions satisfactorily.

The algorithm with strategy (iii) also does not require many iterations to attain the level of accuracy 0.1% from the level of 1%, and the number of iterations is not affected by the change of TRUNC as in the case of the algorithm with strategy (ii). However, the algorithm with (iii) needs much more computational effort in computing search directions than the one with (i) or (ii). In particular, if TRUNC is chosen large, strategy (iii) sometimes finds serious difficulty in solving Newton equations, as shown in the column of CS(TRUNC = 30) in Tables 1(iii) and 2(iii). Moreover, since computation time of search directions forms a large part of the total CPU time, it has relatively small merit to reduce CPU time for basis updates by changing the value of TRUNC. Consequently, as long as the same convergence criterion is used, strategy (iii) does not seem effective. When we need a more accurate solution, however, this strategy may work efficiently, because of the quadratic convergence property of Newton method.

7. Concluding Remarks

We have presented interior methods for nonlinear minimum cost network flow problems. The problem treated here is defined on an open subset relative to the subspace associated with the flow conservation equations. Exploiting this feature, the proposed algorithms have great flexibility in the choice of a basis, and this point distinguishes the proposed algorithms from those which have ever been developed. Based on Newton's method, we have proposed some practical strategies of determining search directions. We have also presented procedures of maintaining a desirable basis, which constructs a minimum spanning tree from an arbitrary tree. Numerical results reported in Section 6 indicate that the proposed algorithms are practically effective.

Although we have assumed the problem to have a separable cost function, it may be possible to modify the proposed algorithms so as to deal with problems with a nonseparable cost function. In fact, if the cost function has a positive definite Hessian matrix, we may directly apply the algorithms by using only the diagonal part of the Hessian matrix since it is also positive definite.

Appendix 1. Finding an Initial Feasible Solution

In Step 1 of the basic algorithm described in Section 3, we must obtain a feasible solution as an initial point. We apply an algorithm similar to the Big M method in linear programming (Bazarra and Jarvis [3], Fukushima et al.[10]).

To begin with, we introduce an artificial node $m+1$ with zero requirement into graph G. Next, we determine an arbitrary initial flow x_j for each arc $j \in E$ such that $x_j \in (l_j, u_j)$. For each node $i \in V$, we compute r_i by

$$r_i = b_i + \sum_{h(j)=i} x_j - \sum_{t(j)=i} x_j.$$

Let us assume temporarily that $r_i \neq 0$ for each node $i \in V$. Then we add an arc which originates at node i and terminates at node $m+1$ if $r_i > 0$, and add an arc which originates at node $m+1$ and terminates at node i if $r_i < 0$. The flows on these artificial arcs $n+i$, $i = 1, 2, \dots, m$, are given by

$$x_{n+i} = |r_i|, \qquad i = 1, 2, \dots, m.$$

The flow vector $(x_1, x_2, \dots, x_{n+m})$ thus obtained is a feasible solution of the problem

$$\text{(A.1)} \quad \text{minimize} \quad \sum_{j \in \overline{E}} f_j(x_j)$$
$$\text{subject to} \quad \overline{A}\overline{x} = \overline{b},$$

where $\overline{V} = V \cup \{m+1\}, \overline{E} = E \cup \{n+1, \dots, n+m\}$, $\overline{x}$ and $\overline{b}$ are the column vectors whose elements are x_j, $j \in \overline{E}$, and b_i, $i \in \overline{V}$, respectively. $\overline{A}$ is the node-arc incidence matrix of graph $\overline{G} = (\overline{V}, \overline{E})$. The cost functions f_j for the artificial variables are defined as

$$f_j(x_j) = Mx_j - \mu_j \ln x_j, \qquad j = n+1, \dots, n+m,$$

where M and μ_j are positive constants. If M is sufficiently large and μ_j are sufficiently small, then the solution of problem (A.1) is supposed to solve the original problem (2.2) approximately. In the computational experiments in Section 6, we set $M = 10^9$ and $\mu_j = 10^{-9}$ for all j.

With respect to problem (A.1), we can determine an initial basis (an initial spanning tree of $\overline{G}$) by selecting the newly introduced artificial arcs. Though we have assumed that $r_i \neq 0$ for each node $i \in V$, this assumption can be easily relaxed. In fact, if $r_i = 0$ for some i, then we can pivot the artificial variable x_{n+i} out of the basis, thereby we can make the values of all basic variables lie in the domain of the corresponding cost functions. By using the solution thus obtained as an initial solution, we can start the basic algorithm of Section 3.

Note that, if the original problem (2.2) is feasible, the flow x_{n+i} on each artificial arc approaches zero as the iteration proceeds. So we remove from graph $\overline{G}$ those nonbasic artificial arcs of which flows are sufficiently close to zero. When the last artificial arc is removed, we also have to remove the artificial node $m+1$ from graph $\overline{G}$.

Appendix 2. Line Search

In Step 3 of the basic algorithm, given a search direction p at the current iterate x, we have to find a new iterate $x + \alpha p$ having a smaller function value. Since the domain of the cost function consists of the intervals (l_j, u_j), $j \in E$, the maximum steplength $\overline{\alpha} > 0$ is given by

$$\overline{\alpha} = \sup\{\alpha | l_j < x_j + \alpha p_j < u_j, j \in E\}.$$

The following procedure determines a steplength using Armijo rule (Polak [19]).

Step 1: Choose a sufficiently small $\epsilon > 0$ and a constant $\gamma \in (0,1)$. Calculate the maximum steplength $\overline{\alpha}$, and set $\alpha := (1-\epsilon)\overline{\alpha}$.
Step 2: If $f(x+\alpha p) < f(x)$, then terminate with steplength α.
Step 3: Set $\alpha := \gamma\alpha$, and go to Step 2.

Because p is a descent direction of the cost function f at x as shown in Section 4, we can determine in a finite number of iterations such a positive α that satisfies $f(x+\alpha p) < f(x)$. The above algorithm uses two parameters ϵ and γ. In the computational experiments reported in Section 6, we set $\epsilon = 10^{-10}$ and $\gamma = 0.5$.

Acknowledgment. We wish to thank Satoru Ibaraki for assisting us in the computational experiments and typing the manuscript so carefully.

References

[1] Ali, A.I., Helgason, R.V., Kennington, J.L. and Lall, H.S.: Primal Simplex Network Codes; State-of-the-Art Implementations Technology. *Networks,* Vol. 8 (1978), 315-339.

[2] Barr, R., Glover, F. and Klingman, D.: Enhancements of Spanning Tree Labelling Procedures for Network Optimization. *INFOR,* Vol. 17 (1979), 16-34.

[3] Bazarra, M.S. and Jarvis, J.J.: *Linear Programming and Network Flows.* John Wiley & Sons, New York, 1977.

[4] Cheriton, D. and Tarjan, R.E.: Finding Minimum Spanning Trees. *SIAM Journal on Computing*, Vol. 5 (1976), 724-742.

[5] Chin, F. and Houck, D.: Algorithms for Updating Minimal Spanning Trees. *Journal of Computer and System Sciences*, Vol. 16 (1978), 333-344.

[6] Dembo, R.S.: A Primal Truncated Newton Algorithm with Application to Large-Scale Nonlinear Network Optimization. *Mathematical Programming Study*, No. 31 (1987), 43-71.

[7] Dembo, R.S. and Klincewicz, J.G.: A Scaled Reduced Gradient Algorithm for Network Problems with Convex Separable Costs. *Mathematical Programming Study*, No. 15 (1981), 125-147.

[8] Dembo, R.S. and Steihaug, T.: Truncated-Newton Algorithms for Large-Scale Unconstrained Optimization. *Mathematical Programming*, Vol. 26 (1983), 190-212.

[9] Fletcher, R.: *Practical Methods of Optimization, Volume 1: Unconstrained Optimization.* John Wiley & Sons, Chichester, 1981.

[10] Fukushima, M., Arai, N. and Ibaraki, T.: An Interior Method for Nonlinear Minimum Cost Network Flow Problems (in Japanese). *Systems and Control*, Vol. 31 (1987), 837-843.

[11] Fukushima, M., Katsura, R. and Ibaraki, T.: On Updating Minimum Spanning Trees (in Japanese). *Transactions of the Institute of Electronics, Information and Communication Engineers of Japan, Section A*, Vol. J71-A (1988), 1979-1982.

[12] Gill, P.E., Murray, W. and Wright, M.H.: *Practical Optimization.* Academic Press, London, 1981.

[13] Gill, P.E., Murray, W., Saunders, M.A., Tomlin, J.A. and Wright, M.H.: On Projected Newton Barrier Methods for Linear Programming and an Equivalence to Karmarkar's Projective Method. *Mathematical Programming*, Vol. 36 (1986), 183-209.

[14] Iri, M., Fujishige, S. and Ohyama, T.: *Graph, Network and Matroid* (in Japanese). Sangyo-Tosho, Tokyo, 1986.

[15] Karmarkar, N.: A New Polynomial-Time Algorithm for Linear Programming. *Combinatorica*, Vol. 4 (1984), 373-395.

[16] Kennington, J.L. and Helgason, R.V.: *Algorithms for Network Programming.* John Wiley & Sons, New York, 1980.

[17] Klincewicz, J.G.: A Newton Method for Convex Separable Network Flow Problems. *Networks,* Vol. 13 (1983), 427-442.

[18] Meyer, R.R.: Network Optimization. *Computational Mathematical Programming* (ed. K. Schittkowski). Springer-Verlag, Berlin, 1985, 125-139.

[19] Papadimitriou, C.H. and Steiglitz, K.: *Combinatorial Optimization; Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[20] Polak, E.: *Computational Methods in Optimization.* Academic Press, New York, 1971.

[21] Tarjan, R.E.: Sensitivity Analysis of Minimum Spanning Trees and Shortest Path Trees. *Information Processing Letters*, Vol. 14 (1982), 30-33.

Masao FUKUSHIMA: Department of
Applied Mathematics and Physics,
Faculty of Engineering, Kyoto University,
Kyoto 606, Japan