Journal of the Operations Research
Society of Japan
Vol. 34, No. 3, September 1991

DUAL-BASED NEWTON METHODS FOR NONLINEAR MINIMUM COST NETWORK FLOW PROBLEMS

Satoru Ibaraki Masao Fukushima Toshihide Ibaraki
Kyoto University

(Received April 23, 1990; Revised January 30, 1991)

Abstract Nonlinear network optimization is of great importance not only in theory but also in practical applications. The range of its applications covers a variety of problems which arise in transportation systems, water distribution systems, resistive electrical networks, and so on. There are various methods to solve nonlinear network flow problems, and many of them belong to the class of descent methods which successively generate search directions and perform line searches. In this paper we propose an algorithm, based on the Newton method, which exploits the network structure of the problems. The algorithm directly solves the dual problem which, under appropriate conditions, can be formulated as an unconstrained convex minimization problem with a continuously differentiable objective function. We give a global convergence theorem of the algorithm and present practical strategies for computing search directions and finding steplengths. Some computational results for test problems of up to 4900 nodes and 14490 arcs show the practical efficiency of the proposed algorithm.

1. Introduction

Nonlinear network optimization is of great importance not only in theory but also in practical applications. The range of its application is so wide as to cover a variety of problems which arise in transportation systems, water distribution systems, resistive electrical networks, and so on.

There are various methods to solve nonlinear network flow problems, and many of them belong to the class of descent methods which successively generate search directions and perform line searches [2, 6, 7, 10, 11, 12, 13, 15]. Among others, the algorithms presented in [7, 10, 12, 13] are adaptations of Newton method, which effectively utilize the network structure of the problems. These algorithms are expected to have good convergence properties because Newton method is a very efficient nonlinear optimization method whose convergence rate is normally superlinear.

In this paper we propose an algorithm, based on Newton method, for separable nonlinear minimum cost network flow problems. Unlike the above mentioned algorithms, it directly solves the *dual* problem which, under appropriate conditions, can be formulated as an unconstrained convex minimization problem with a continuously differentiable objective function. We use the conjugate gradient method to solve Newton equations. The algorithm

can be efficiently implemented using the network structure, though the Hessian matrix of the dual objective function is composed of that of the primal cost functions and the node-arc incident matrix of the underlying graph.

The idea of solving the dual problem has also been presented by Hager and Hearn [9] and Tseng and Bertsekas [17] (see also [4]). In particular, Tseng and Bertsekas [17] propose Gauss-Seidel relaxation procedures which successively minimize the dual function along each coordinate. Their algorithms have a linear convergence rate but are well suited for parallel computation.

This paper consists of eight sections. In the next section we formulate a nonlinear minimum cost network flow problem and make two fundamental assumptions. In Section 3 we derive the dual problem and examine differential properties of its objective function. In Section 4 we describe a basic algorithm of descent type and establish a global convergence theorem. We propose a direction finding procedure based on Newton method in Section 5, and describe a practical line search technique in Section 6. We present computational results in Section 7. Finally Section 8 concludes the paper.

2. Network Flow Problems

We describe a nonlinear network flow problem, with a single commodity, which has a convex and separable cost function.

Let Γ be a directed graph which has the set $\mathcal{N} = \{1, 2, \ldots, m\}$ of nodes and the set $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ of arcs, i.e. $\Gamma = (\mathcal{N}, \mathcal{A})$. Let $a_j = (i, k)$ denote that the start and end nodes of arc a_j are i and k, respectively. A flow of arc a_j is denoted by $x_j \in R$ and each arc has a cost function $f_j : R \to R \cup \{+\infty\}$.

Then the minimum cost flow problem is stated as

$$\text{(P)} \qquad \text{minimize} \quad f(x) = \sum_{j=1}^{n} f_j(x_j)$$

$$\text{(2.1)} \qquad \text{subject to} \quad \sum_{j=1}^{n} e_{ij} x_j = b_i, \quad \forall i \in \mathcal{N} - \{m\},$$

where e_{ij} are the elements of the incident matrix denoted by $E \in R^{(m-1)\times n}$, i.e.,

$$e_{ij} = \begin{cases} 1 & \text{if } i \text{ is the start node of arc } a_j, \\ -1 & \text{if } i \text{ is the end node of arc } a_j, \\ 0 & \text{otherwise,} \end{cases}$$

and the equality constraints (2.1) are called the flow conservation equations. Except the columns corresponding to the arcs incident to node m, the number of nonzero elements in each column is two; an entry $+1$ is in the row corresponding to the node where the arc originates and an entry -1 is in the row corresponding to the node where the arc terminates.

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., $\sum_{i \in N} b_i = 0$. As this implies that the flow conservation equation for node m is redundant, that equation has been ignored in (2.1). We assume that the graph Γ is connected, so that the matrix E has full rank [1].

Since the range of each cost function f_j is $R \cup \{+\infty\}$, we may define u_j and l_j such that

$$u_j = \sup\{x_j | f_j(x_j) < +\infty\}$$

$$l_j = \inf\{x_j | f_j(x_j) < +\infty\},$$

which are regarded as the upper and lower bounds of the flow x_j, respectively. It is allowed that $u_j = +\infty$ or $l_j = -\infty$ or both.

Conventionally, a minimum cost flow problem contains real valued cost functions $\tilde{f}_j : R \to R$ and, besides the flow conservation equations, the explicit upper and lower bound constraints $l_j \leq x_j \leq u_j$, that is,

$$\text{(P')} \qquad \text{minimize } \tilde{f}(x) = \sum_{j=1}^{n} \tilde{f}_j(x_j)$$

$$\text{subject to } \sum_{j=1}^{n} e_{ij} x_j = b_i, \quad \forall i$$

$$l_j \leq x_j \leq u_j, \quad \forall j. \tag{2.2}$$

In this case, problem (P') can be reformulated as problem (P) with the cost functions

$$f_j(x_j) = \begin{cases} \tilde{f}_j(x_j) & \text{if } x_j \in [l_j, u_j], \\ +\infty & \text{otherwise.} \end{cases} \tag{2.3}$$

Clearly, the convexity of the cost function is retained by (2.3).

We may also transform problem (P') into problem (P) using penalty functions. In particular, we may introduce the barrier function

$$f_j(x_j) = \tilde{f}_j(x_j) - \mu_j \log(u_j - x_j) - \mu_j \log(x_j - l_j), \tag{2.4}$$

where $\mu_j > 0$, so as to include the upper and lower bound constraints (2.2) in the cost functions. Of course, if $l_j = -\infty$ ($u_j = +\infty$), then the second (third) term of the right-hand side of (2.4) is vacuous. When the parameters μ_j are small enough, the optimal solution of the transformed problem (P) may be regarded as a good approximation to an optimal solution of problem (P'). Note that the domain of the function $f_j(x_j)$ defined by (2.4) is (l_j, u_j) and hence the feasible region of the problem is open relative to the affine subspace corresponding to the flow conservation equations. In fact, the methods proposed in [7, 10] take advantage of this property.

The following two assumptions on the cost function $f_j(x_j)$ of problem (P) will play a crucial role in the subsequent sections.

Assumption 1. On the interval $\{x_j | f_j(x_j) < +\infty\} \subset R$, the function $f_j(x_j)$ is strictly convex.

Assumption 2. $f_j(x_j)$ is *co-finite* [16], i.e.,

$$\lim_{x_j \to \pm\infty} \frac{f_j(x_j)}{|x_j|} = +\infty.$$

Note that Assumption 2 is satisfied whenever the interval $\{x_j | f_j(x_j) < +\infty\}$ is bounded.

3. Dual Problem

In this section we will formulate the dual of problem (P). Let us associate a Lagrange multiplier, or a *potential*, p_i with the flow conservation equation (2.1) for node i, and define the Lagrangian function by

$$L(x,p) = \sum_{j=1}^{n} f_j(x_j) - \sum_{i=1}^{m-1} p_i(\sum_{j=1}^{n} e_{ij}x_j - b_i),$$

where p is the vector with elements $p_i, i \in \mathcal{N} - \{m\}$. Then the dual problem can be written as

(D) minimize $q(p)$

where the dual function $q(p)$ is given by

$$\begin{aligned} q(p) &= -\inf_x L(x,p) \\ &= \sup_x \sum_j \{x_j \sum_i e_{ij}p_i - f_j(x_j)\} - \sum_i b_i p_i. \end{aligned}$$

For the subsequent discussions, it is convenient to define t_j by

$$t_j = \sum_i e_{ij}p_i = p_l - p_k, \quad \forall j = 1,2,\ldots,n \tag{3.1}$$

where $a_j = (l,k)$. We call t_j the *tension* of arc a_j. Furthermore, the conjugate function of $f_j(x_j)$ is defined by

$$f_j^*(t_j) = \sup_{x_j}\{x_j t_j - f_j(x_j)\}. \tag{3.2}$$

An example of a conjugate pair of the functions f_j and f_j^* is illustrated in Figure 1. Then the above dual function can be rewritten as

$$\begin{aligned} q(p) &= \sum_j f_j^*(\sum_i e_{ij}p_i) - \sum_i b_i p_i \\ &= \sum_j f_j^*(t_j) - \sum_i b_i p_i, \end{aligned} \tag{3.3}$$

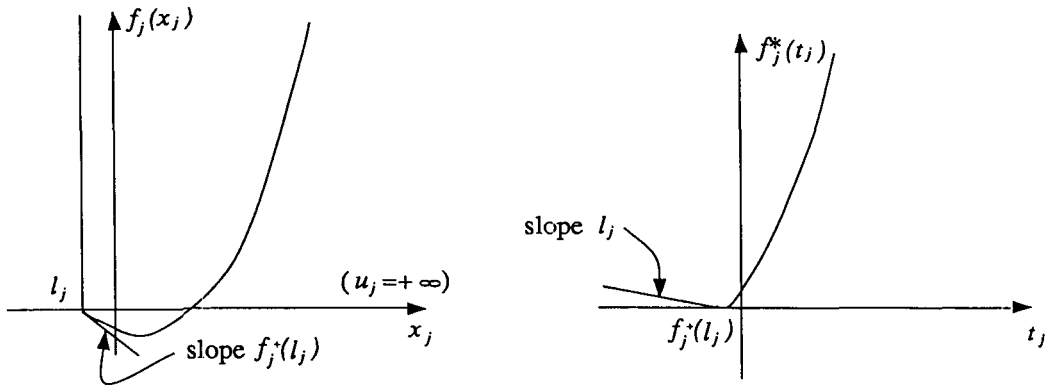

Figure 1. An example of the conjugate pair f_j and f_j^*.

where t_j is given by (3.1). The conjugate functions $f_j^*(t_j)$ are convex and satisfy $f_j^*(t_j) < +\infty$ for all $t_j \in (-\infty, +\infty)$ by Assumption 2 in the previous section [16]. Hence the dual problem (D) is an unconstrained convex minimization problem.

In the remainder of this section, let us consider differential properties of the dual function $q(p)$. For simplicity, we assume here that each function $f_j(x_j)$ is closed [16, p.52]. Then, Assumption 1 implies that, for every t_j, there exists a unique $\overline{x}_j$ such that

$$(3.4) \quad \overline{x}_j = \arg\max_{z_j}\{t_j z_j - f_j(z_j)\}, \quad \forall j = 1, 2, \ldots, n.$$

(This means that $\overline{x}_j$ is the value of z_j which uniquely attains the maximum in (3.4)). Moreover, it can be shown [16, Theorems 23.5 and 25.1] that $\overline{x}_j$ is the gradient of f_j^* at t_j, i.e.,

$$(3.5) \quad f_j^{*\prime}(t_j) = \overline{x}_j, \quad \forall j = 1, 2, \ldots, n.$$

From (3.2), (3.3) and (3.5), we see that for a given potential vector p the partial derivatives of the dual function $q(p)$ are given by

$$\begin{aligned} \frac{\partial q(p)}{\partial p_i} &= \sum_j e_{ij} f_j^{*\prime}(t_j) - b_i \\ &= \sum_j e_{ij}\overline{x}_j - b_i, \quad \forall i = 1, 2, \ldots, m-1, \end{aligned}$$

where $\overline{x}_j$ is given by (3.4). Note that the dual function $q(p)$ is differentiable and convex, so that $\partial q(p)/\partial p_i$ is continuous and monotonically nondecreasing in p_i. Consequently the gradient vector $\nabla q(p)$ is given by

$$\begin{aligned} \nabla q(p) &= \left(\frac{\partial q(p)}{\partial p_1}, \cdots, \frac{\partial q(p)}{\partial p_{m-1}}\right)^T \\ &= E\overline{x} - b. \end{aligned}$$

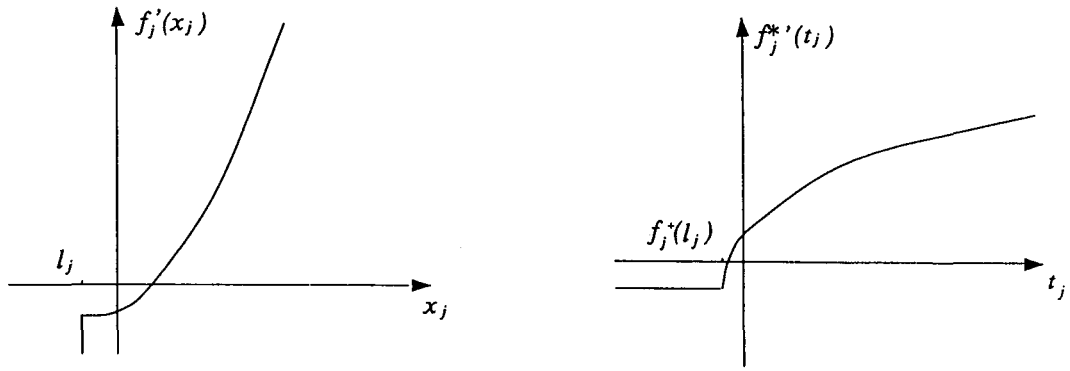

Figure 2. Relationship between the first derivatives of the pair f_j and f_j^* exemplified in Figure 1.

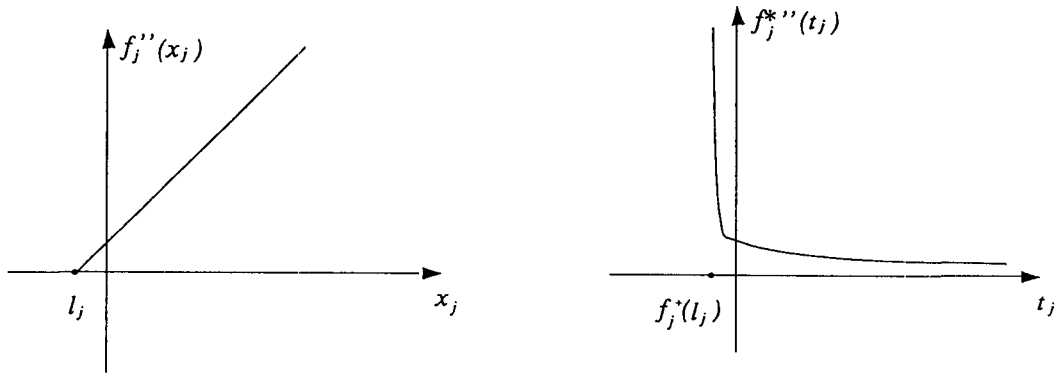

Figure 3. Relationship between the second derivatives of the pair f_j and f_j^* exemplified in Figure 1.

Now we turn our attention to the second derivatives of function $q(p)$. As seen in (3.3), the dual function $q(p)$ consists of the conjugate functions $f_j^*(t_j)$. Therefore, if each $f_j^*(t_j)$ is twice differentiable, then so is $q(p)$ and its Hessian matrix can be written as

$$\nabla^2 q(p) = EH(t)E^T, \tag{3.6}$$

where tension t is given by (3.1) and $H(t)$ is the diagonal matrix defined by

$$H(t) = \mathrm{diag}(f_j^{*\prime\prime}(t_j)). \tag{3.7}$$

The following theorem establishes a condition under which the second derivative of $f_j^*(t_j)$ exists and is positive for a given t_j.

Theorem 1 *Assume that for a given t_j the corresponding $\overline{x}_j$ is uniquely determined by (3.5). Moreover, suppose that the function $f_j(x_j)$ is twice differentiable at $\overline{x}_j$ and that $f_j''(\overline{x}_j) > 0$. Then the conjugate function $f_j^*(t_j)$ is also twice differentiable at t_j and the second derivative is given by*

$$f_j^{*\prime\prime}(t_j) = \frac{1}{f_j''(\bar{x}_j)} > 0. \tag{3.8}$$

Proof. Under the given assumptions, the function $f_j(x_j)$ is differentiable at $\bar{x}_j$ and it follows from [16, Theorem 23.5] that

$$f_j'(\bar{x}_j) = t_j. \tag{3.9}$$

Therefore (3.5) and (3.9) imply that f_j' and $f_j^{*\prime}$ are the inverse functions of each other. Since f_j' is continuous and strictly monotone at $\bar{x}_j$, so is the inverse $f_j^{*\prime}(t_j)$ and (3.8) holds. □

Figures 2 and 3 show the relationship between the first and second derivatives of the pair f_j and f_j^* exemplified in Figure 1.

The algorithm proposed in this paper does not utilize explicit representation of $q(p)$ or $f_j^*(t_j)$. We only require that for each given p, the values of $q(p)$, $\nabla q(p)$, and possibly $\nabla^2 q(p)$ are computed. Furthermore, we comment that the value of $f_j^*(t_j)$ is easily evaluated by solving a one-dimensional minimization problem that appears on the right-hand side of (3.5).

Though problem (P) has the unique solution owing to the strictly convexity of the cost function, an optimal solution of the dual problem (D) is not necessarily unique. The next theorem gives a sufficient condition which guarantees the uniqueness of the optimal solution of problem (D).

Theorem 2 *Let $\bar{p}$ be an optimal solution of problem (D) and $\bar{t}$ be the tension vector associated with $\bar{p}$, i.e., $\bar{t} = E^T\bar{p}$. Suppose that $f_j^{*\prime\prime}(\bar{t}_j)$ exist for all j. Then $\bar{p}$ is the unique solution of problem (D), provided that the subgraph $(\mathcal{N}, \hat{\mathcal{A}})$ of Γ is connected, where $\hat{\mathcal{A}}$ is the set of arcs defined by*

$$\hat{\mathcal{A}} = \{a_j \in \mathcal{A} | f_j^{*\prime\prime}(\bar{t}_j) > 0\}.$$

Proof. The existence of $f_j^{*\prime\prime}(\bar{t}_j)$ implies that the dual objective function $q(p)$ is twice differentiable at $\bar{p}$. Moreover, by (3.6), we have

$$\nabla^2 q(\bar{p}) = EH(\bar{t})E^T. \tag{3.10}$$

To prove the theorem, it then suffices to show that $\nabla^2 q(\bar{p})$ is positive definite. Let $\hat{E}$ be the node-arc incidence matrix of the graph $(\mathcal{N}, \hat{\mathcal{A}})$ and $\hat{H}(\bar{t})$ be the diagonal submatrix of $H(\bar{t})$ which consists of the elements $f_j^{*\prime\prime}(\bar{t}_j)$ such that $a_j \in \hat{\mathcal{A}}$. Then the matrix $\hat{H}(\bar{t})$ is obviously positive definite. On the other hand, since the convexity of $f_j^*(\bar{t}_j)$ implies $f_j^{*\prime\prime}(\bar{t}_j) \geq 0$, it follows from the definition of $\hat{\mathcal{A}}$ that $f_j^{*\prime\prime}(\bar{t}_j) = 0$ for all $a_j \notin \hat{\mathcal{A}}$. Therefore (3.10) can be rewritten as

$$\nabla^2 q(\bar{p}) = \hat{E}\hat{H}(\bar{t})\hat{E}^T.$$

Since the connectedness of the subgraph $(\mathcal{N}, \hat{\mathcal{A}})$ implies that the incidence matrix $\hat{E}$ has full rank, the matrix $\nabla^2 q(\bar{p})$ is positive definite. □

4. Basic Algorithm

Since the dual problem (D) is formulated as an unconstrained minimization problem with a differentiable objective function, we can adopt the following algorithm which belongs to the class of descent methods.

Basic Algorithm

Step 0: Choose an initial solution p.

Step 1: Choose a symmetric positive definite matrix $Q(p) \in R^{(m-1)\times(m-1)}$, and solve the following system of linear equations so as to obtain a search direction s satisfying the descent property $\nabla q(p)^T s < 0$:

$$(4.1) \quad Q(p)s = -\nabla q(p).$$

Step 2: Determine a steplength $\delta > 0$ such that

$$(4.2) \quad q(p + \delta s) < q(p)$$

by approximately solving the one-dimensional problem

$$(4.3) \quad \min_{\delta > 0} q(p + \delta s).$$

Step 3: Set $p := p + \delta s$ and go to Step 1.

Note that if $\nabla q(p)$ does not vanish and the search direction s is determined by (4.1), then the descent property is satisfied by the positive definiteness of $Q(p)$, i.e.,

$$\nabla q(p)^T s = -\nabla q(p)^T Q(p)^{-1} \nabla q(p) < 0.$$

If $Q(p) \equiv I$, the direction s is the steepest descent direction, while if $Q(p)$ is chosen as the Hessian matrix of function $q(p)$, the direction s is the Newton direction.

From the descent property, there exists a $\overline{\delta} > 0$ such that the inequality (4.2) holds for all $\delta \in (0, \overline{\delta}]$. In practice we solve (4.3) only approximately because it is expensive to perform exact line search.

Basic Algorithm generates a sequence of points whose objective function values decrease monotonically. Strictly speaking, however, this property is not sufficient to ensure global convergence to an optimal solution. Specifically, sufficient reduction in $q(p)$ may not be obtained if the directions s tend to be orthogonal to the gradients $\nabla q(p)$, or if steplengths approach zero. In the rest of this section, we shall consider general conditions under which the global convergence of Basic Algorithm is guaranteed. We will explain in detail practical implementation of the algorithm in Sections 5 and 6.

First, we set up a condition which the matrices $Q(p)$ should satisfy in order to yield sufficient reduction in $q(p)$. Specifically, we require $Q(p)$ to satisfy the inequalities

$$(4.4) \quad 0 < \lambda \leq \frac{y^T Q(p) y}{y^T y} \leq \Lambda, \quad \forall y \neq 0,$$

where λ and Λ are positive constants independent of p. In particular, the inequalities (4.4) are satisfied if the matrices $Q(p)$ are determined as $\tilde{E} D(p) \tilde{E}^T$, where $\tilde{E}$ is a matrix of full rank and $D(p)$ are diagonal matrices whose diagonal elements are uniformly bounded above and away from zero.

When the matrices $Q(p)$ are chosen to satisfy the above condition (4.4), the directions s defined by (4.1) have the property that the angle θ between s and $-\nabla q(p)$ is uniformly bounded away from orthogonality [5, p.31], that is,

$$\theta \leq \frac{\pi}{2} - \frac{\lambda}{\Lambda},$$

where $\theta \in [0, \pi/2)$ is given by

$$\cos\theta = -\frac{\nabla q(p)^T s}{\|\nabla q(p)\| \|s\|}.$$

Next, let us consider how to select steplengths δ which give a sufficient reduction in $q(p)$. For convenience, we write

$$\phi(\delta) = q(p + \delta s).$$

Then the gradient of $\phi(\delta)$ is given by

$$\phi'(\delta) = \nabla q(p + \delta s)^T s.$$

Note that $\phi(0) = q(p)$ and the descent property is equivalent to $\phi'(0) < 0$.

Following Fletcher [5, p.27], we call a steplength δ *acceptable* if it satisfies the conditions

$$(4.5) \quad \phi(\delta) \leq \phi(0) + \delta\rho\phi'(0)$$

$$(4.6) \quad \phi'(\delta) \geq \sigma\phi'(0),$$

where $\rho \in (0, \frac{1}{2})$ and $\sigma \in (\rho, 1)$ are preset parameters. These conditions are illustrated in Figure 4. It is known that the interval of acceptable steplengths δ is nonempty whenever the dual function is bounded below (see Fletcher [5, Lemma 2.5.1]).

Finally, we can state a global convergence theorem for Basic Algorithm. The proof follows directly from Fletcher [5, Theorem 2.5.1] and is omitted here.

Theorem 3 *Assume that the dual problem (D) has an optimal solution. Let the search directions s be obtained by solving (4.1) in which the matrices $Q(p)$ satisfy the condition (4.4), and let the steplengths δ be determined by inexact line search based on (4.5) and (4.6). Then Basic Algorithm generates a sequence which is convergent to an optimal solution of problem (D).*

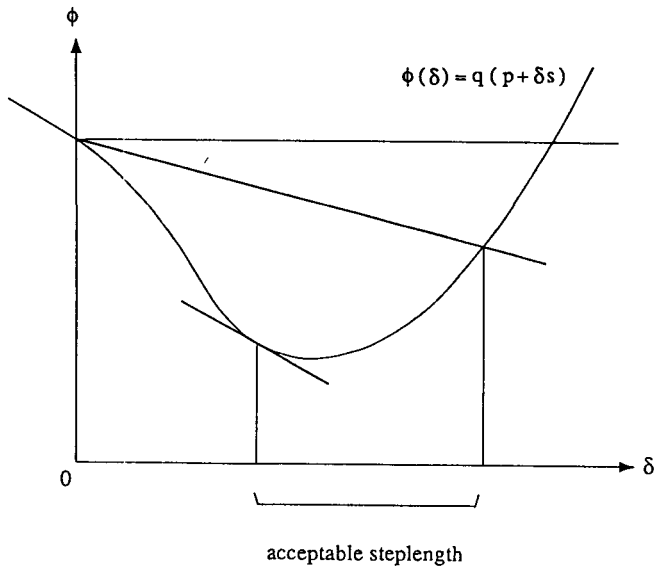

Figure 4. Conditions for line search.

5. Search Directions

As described in the previous section, search directions are determined by solving (4.1). If it were possible to set $Q(p) = \nabla^2 q(p)$ for any p, the algorithm would be regarded as Newton method. In practice, however, the second derivatives of $f_j^*(t_j)$ may not be defined everywhere. So we propose a practical strategy for constructing matrices $Q(p)$, which attempts to incorporate second-order information on $f_j^*(t_j)$ as much as possible. In view of (3.6) and (3.7), we let $Q(p)$ be given as

$$Q(p) = EH(t)E^T, \tag{5.1}$$

where E is the node-arc incidence matrix and $H(t)$ is a diagonal matrix such that

$$H(t) = \mathrm{diag}(h_j(t_j)). \tag{5.2}$$

To simplify the discussion, we assume that the number of points t_j where $f_j^{*\prime\prime}(t_j)$ is undefined is finite. This assumption is not restrictive and usually satisfied in practical applications. Then we determine $h_j(t_j)$ in (5.2) as follows. For a given t_j, if $f_j^{*\prime\prime}(t_j)$ exists, then define

$$h_j(t_j) = \begin{cases} \underline{c} & f_j^{*\prime\prime}(t_j) < \underline{c} \\ f_j^{*\prime\prime}(t_j) & \underline{c} \le f_j^{*\prime\prime}(t_j) \le \overline{c} \\ \overline{c} & \overline{c} < f_j^{*\prime\prime}(t_j), \end{cases}$$

where $\underline{c}$ and $\bar{c}$ are constants such that $\bar{c} > \underline{c} > 0$, while if $f_j^{*\prime\prime}(t_j)$ is undefined, then $h_j(t_j)$ is set to be a real number arbitrarily chosen from the interval $[\liminf_{z \to t_j} h_j(z), \limsup_{z \to t_j} h_j(z)]$. Note that matrices $Q(p)$ are symmetric and positive definite. Moreover, since $h_j(t_j)$ always lies in the interval $[\underline{c}, \bar{c}]$, $Q(p)$ satisfy condition (4.4) which is necessary to guarantee global convergence of Basic Algorithm. In numerical experiments reported below, the constants $\underline{c}$ and $\bar{c}$ are set 10^{-5} and 10^5, respectively.

Now we consider a procedure for solving the system (4.1) of linear equations. For large-scale problems, it seems difficult to apply direct methods such as Gaussian elimination because they require large memory space. Here we employ the conjugate gradient (CG) method to solve (4.1). We deal with matrices $Q(p)$ as the product form (5.1) and hence the most time-consuming matrix-vector multiplication can be efficiently calculated using one-dimensional arrays representing the network structure. Since the speed of convergence of a CG algorithm is dependent on the condition number of $Q(p)$, a preconditioning is recommended in order to improve the condition of $Q(p)$. Here we use the diagonal part of $Q(p)$ as the preconditioner matrix M, because its inverse is easily available. Note that the entries of matrix M can be obtained by

$$M_{ii} = \sum_{a_j=(i,k)} h_j(t_j) + \sum_{a_j=(k,i)} h_j(t_j).$$

Now we can state the CG algorithm for solving (4.1).

CG algorithm
begin
$k := 0; s_0 := 0; r_0 := -\nabla q(p)$
while a convergence criterion is not satisfied **do**
begin
begin
$\tilde{r}_k := M^{-1} r_k;$
$k := k + 1$
end
if $k = 1$ **then**
$\tilde{s}_1 := \tilde{r}_0$
else
begin
$\beta_k := r_{k-1}^T \tilde{r}_{k-1} / r_{k-2}^T \tilde{r}_{k-2};$
$\tilde{s}_k := \tilde{r}_{k-1} + \beta_k \tilde{s}_{k-1}$
end
end if
begin
$\alpha_k := r_{k-1}^T \tilde{r}_{k-1} / \tilde{s}_k^T Q(p) \tilde{s}_k;$
$s_k := s_{k-1} - \alpha_k \tilde{s}_k;$
$r_k := r_{k-1} - \alpha_k Q(p) \tilde{s}_k$
end
end

$s := s_k$
end

Let us mention the convergence criterion of the above algorithm. It is well known that CG algorithms finitely terminate under exact arithmetic. In practice, however, this property fails to hold because of numerical error. Therefore we terminate the procedure if the residual r_k satisfies

(5.3) $\|r_k\| < \epsilon_{CG}\|r_0\|$,

where r_0 is the initial residual and $\epsilon_{CG} > 0$ is a small constant.

6. Line Search

We describe an iterative method for finding an acceptable steplength δ that satisfies the inequalities (4.5) and (4.6). This is accomplished by bracketing an interval of the acceptable steplengths, and then sectioning this bracket to find a suitable steplength. The procedure is an adaptation of the one presented in [5, §2.6].

[Bracketing Phase]
choose a sufficiently large integer $imax$;
choose $\rho \in (0, 1/2), \sigma \in (\rho, 1), \gamma \in (1, +\infty)$ and $\delta > 0$;
$i := 1$
while $i \leq imax$ **do**
begin
evaluate $\phi(\delta)$;
if $\phi(\delta) \geq \phi(0) + \rho\delta\phi'(0)$ **then**
begin
$\delta_u := \delta$; $\delta_l := 0$; go to Sectioning Phase
end
end if
evaluate $\phi'(\delta)$;
if $\phi'(\delta) \geq \sigma\phi'(0)$ **then** terminate;
set $\delta := \gamma\delta$
$i := i + 1$
end
[Sectioning Phase]
while $i \leq imax$ **do**
begin
choose $\delta \in (\delta_l, \delta_u)$
evaluate $\phi(\delta)$;
if $\phi(\delta) > \phi(0) + \rho\delta\phi'(0)$ **then** $\delta_u := \delta$
else
begin

```
        evaluate φ'(δ);
        if φ'(δ) ≥ σφ'(0) then terminate;
        δ_l := δ
      end
    end if
    i := i + 1
  end
```

Various methods are available to choose $\delta_j \in (\delta_l, \delta_u)$ in the Sectioning Phase. Here we shall employ a curve fitting technique based on cubic interpolation [14]. To start the procedure, we have to specify the values of several parameters. In the numerical experiments reported below, we set $\rho = 0.01, \sigma = 0.7, \gamma = 10$ and $\delta_1 = 1$.

7. Numerical Results

In this section we report some computational results with the proposed algorithm. The computer codes were written entirely in FORTRAN77, and run in double precision on a FACOM M780-30.

In addition to the proposed algorithm, we have coded two other methods. One is the relaxation method presented in [17], and the other is a primal Newton method proposed in [10]. In particular, the latter method is an interior method which is designed to solve problems whose objective function is defined on an open set like barrier or penalty functions.

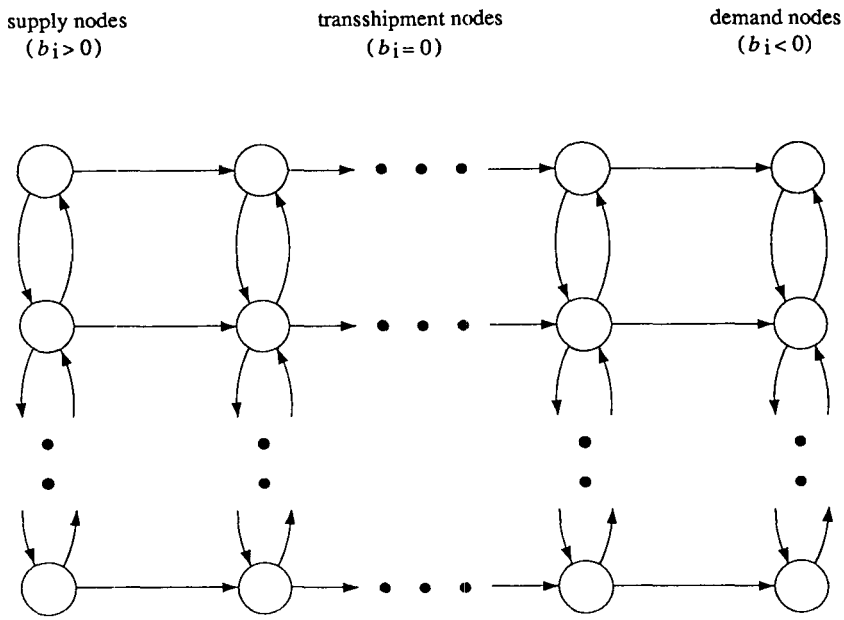

Figure 5. Lattice network.

7.1. Test Problems

We have used lattice networks as illustrated in Figure 5 to construct test problems. In those networks, supply nodes ($b_i > 0$) are all located in the left-most column, while demand nodes ($b_i < 0$) are in the right-most column. The nodes in the intermediate columns are all transshipment nodes ($b_i = 0$). The size of the test problems varies from $(m, n) = (30, 73)$ to $(m, n) = (4900, 14490)$, where m and n are the numbers of nodes and arcs, respectively.

The cost functions are supposed to be one of the following two types.

$$(7.1) \quad f_j(x_j) = \begin{cases} c_j x_j + \frac{1}{2} d_j x_j^2, & \text{if } x_j \in [0, u_j] \\ +\infty, & \text{otherwise,} \end{cases}$$

$$(7.2) \quad f_j(x_j) = \begin{cases} c_j x_j + \frac{1}{3} d_j x_j^3, & \text{if } x_j \in [0, u_j] \\ +\infty, & \text{otherwise.} \end{cases}$$

In each test problem, b_i, c_j and u_j are randomly chosen from the intervals (1, 10), (1, 20) and (5, 10), respectively. As to the coefficient d_j in (7.1) and (7.2), we have tested two cases, i.e., d_j are randomly chosen from the intervals (1, 10) and (0.1, 2). In the former case, the effect of the nonlinear term is stronger than that in the latter. In the rest of this section, we shall refer these two cases to as type I and type II, respectively.

7.2. Results

In the tests, the convergence of Basic Algorithm is checked using the ratio of the norm of the gradient $\nabla q(p)$. Specifically, we terminate the iteration if the condition

$$(7.3) \quad \frac{\|\nabla q(p)\|}{\|\nabla q(p_0)\|} < \epsilon$$

is satisfied, where p and p_0 are the current and initial values of the dual variables, respectively, and ϵ is a small positive number.

Tables 1 through 4 summarize the numerical results. Tables 1 and 2 respectively show the results for the type I and the type II problems, in which the cost functions are given by (7.1). Tables 3 and 4 respectively show the results for the type I and the type II problems, in which the cost functions are given by (7.2). In all cases, the initial point p_0 was chosen to be $p_0 = 0$. The tolerance ϵ in (7.3) was set equal to 10^{-3}.

As mentioned in Section 6, the accuracy of the computed solutions of Newton equation (4.1) depends on the tolerance ϵ_{CG} used in condition (5.3). Tables 1 and 2, which contain the results obtained by setting $\epsilon_{CG} = 10^{-1}$ and $\epsilon_{CG} = 10^{-3}$, show that computing an accurate solution of Newton equations on every iteration reduces the number of the major iterations of Basic Algorithm but tends to spend more CPU time in total.

Making a comparison between Tables 1 and 2, it is evident that the total CPU time and the total number of iterations for the type I problems are considerably less than those for the

type II problems. Similar relation may be observed in Tables 3 and 4. This phenomenon may be explained as follows: When the coefficients d_j are relatively small, the function $f_j(x_j)$ is almost linear, and hence the gradients of $f_j(x_j)$ are nearly constant, on its domain $[0, u_j]$. This implies that the first derivatives of $f_j^*(t_j)$ behave like a step function, because $f_j^{*\prime}(t_j)$ is the inverse function of $f_j'(x_j)$. In other words, the objective function $q(p)$ of the dual problem (D) may be regarded as practically nondifferentiable from the numerical standpoint. The algorithm requires a large number of iterations to achieve the termination condition (7.3) for such problems.

Detailed results for test problems I-1-11, I-1-12, II-1-11, II-1-12, I-2-11, I-2-12 and II-2-11, II-2-12 are shown in Tables 5, 6, 7 and 8, which consist of the following items:

(a) the marginal/cumulative number of iterations required to achieve the levels of accuracy $\epsilon = 10^{-1}, 10^{-2}, 10^{-3}$, and 10^{-4};

(b) the marginal/cumulative CPU time spent to achieve the same levels of accuracy as above.

It is recognized that the speed of convergence is very fast near the optimal solution not only for the type I test problems but also for the type II problems. This phenomenon, which is typical in Newton-type methods, has also been observed for other test problems. We add that, for all test problems, Basic Algorithm spent more than 90% of the total CPU time in Step 1 to find search directions.

We have also solved the same test problems using the relaxation method presented in [3]. Figure 6 compares the behavior of this method with that of the proposed algorithm for the test problems I-1-7 and II-1-7. From these numerical results, we see that the proposed algorithm converges faster than the relaxation method, in particular for the the type II problems. However, it may be worth mentioning that, as pointed out in [3, 4, 17], the latter method is suited for parallel computation and hence its efficiency would be much improved if implemented using several processors.

Figure 7 illustrates the behavior of the proposed algorithm and the primal Newton method presented in [10], for the test problem I-1-7. The latter is an interior method which solves the problem constructed from the primal problem (P) using barrier functions (2.4). Note that since problem (D) has been formulated as a minimization problem, its optimal value is the negative of that of problem (P). In order to clarify the fact that the proposed method actually solves the dual problem, we have plotted in Figure 7 the curve corresponding to the negative of the objective values attained by the proposed algorithm. The curve is thus ascending and approaches the optimal value of the primal problem from below. It is seen that the proposed algorithm produces a near optimal value much faster than the primal Newton method. The main reason for this is that the latter has to work on an artificial problem first in order to find an initial feasible solution of the primal problem. This is in contrast with the proposed algorithm in which the initial solution can be chosen arbitrarily.

Table 1: Results for the type I problems with cost functions defined by (7.1): $\epsilon = 10^{-3}$

Problem	Number	Number	$\epsilon_{CG} = 10^{-1}$		$\epsilon_{CG} = 10^{-3}$	
number	of nodes	of arcs	CPU(s)	Iterations	CPU(s)	Iterations
I-1-1	30	73	0.04	10	0.05	10
I-1-2	30	73	0.04	14	0.04	10
I-1-3	56	146	0.13	16	0.19	16
I-1-4	56	146	0.11	14	0.20	15
I-1-5	121	330	0.43	25	1.02	20
I-1-6	121	330	0.53	26	0.91	20
I-1-7	256	720	2.02	25	3.84	22
I-1-8	256	720	1.92	26	3.66	21
I-1-9	529	1518	4.50	22	12.6	22
I-1-10	529	1518	6.08	27	14.3	25
I-1-11	1024	2976	21.6	40	60.7	37
I-1-12	1024	2976	18.3	36	46.3	28
I-1-13	2025	5940	53.8	40	165	36
I-1-14	3025	8910	111	54	388	53

Table 2: Results for the type II problems with cost functions defined by (7.1): $\epsilon = 10^{-3}$

Problem	Number	Number	$\epsilon_{CG} = 10^{-1}$		$\epsilon_{CG} = 10^{-3}$	
number	of nodes	of arcs	CPU(s)	Iterations	CPU(s)	Iterations
II-1-1	30	73	0.06	16	0.09	16
II-1-2	30	73	0.08	16	0.08	15
II-1-3	56	146	0.24	26	0.29	18
II-1-4	56	146	0.17	17	0.38	21
II-1-5	121	330	1.34	33	2.44	34
II-1-6	121	330	1.04	30	2.17	33
II-1-7	256	720	4.54	42	9.63	38
II-1-8	256	720	4.84	38	11.7	44
II-1-9	529	1518	15.7	53	46.9	55
II-1-10	529	1518	15.2	47	41.5	47
II-1-11	1024	2976	69.9	77	224	74
II-1-12	1024	2976	64.2	61	181	60
II-1-13	2025	5940	248	108	829	76
II-1-14	3025	8910	510	159	$> 900^*$	-

* The convergence criterion (7.4) was not satisfied in 900 seconds.

Table 3: Results for the type I problems with cost functions defined by (7.2): $\epsilon = 10^{-3}$

Problem	Number	Number	$\epsilon_{CG} = 10^{-1}$		$\epsilon_{CG} = 10^{-3}$	
number	of nodes	of arcs	CPU(s)	Iterations	CPU(s)	Iterations
I-2-1	30	73	0.06	20	0.08	16
I-2-2	30	73	0.04	12	0.07	14
I-2-3	56	146	0.10	13	0.19	16
I-2-4	56	146	0.10	16	0.18	15
I-2-5	121	330	0.52	26	0.86	19
I-2-6	121	330	0.34	18	0.74	18
I-2-7	256	720	1.90	35	3.51	24
I-2-8	256	720	1.33	30	3.46	25
I-2-9	529	1518	4.79	43	11.0	30
I-2-10	529	1518	9.50	69	12.8	29
I-2-11	1024	2976	17.1	55	37.8	28
I-2-12	1024	2976	14.9	48	34.0	33
I-2-13	2025	5940	36.2	54	101	41
I-2-14	3025	8910	49.7	52	193	36
I-2-15	3422	10090	91.4	64	200	29
I-2-16	4900	14490	159	58	434	36

Table 4: Results for the type II problems with cost functions defined by (7.2): $\epsilon = 10^{-3}$

Problem	Number	Number	$\epsilon_{CG} = 10^{-1}$		$\epsilon_{CG} = 10^{-3}$	
number	of nodes	of arcs	CPU(s)	Iterations	CPU(s)	Iterations
II-2-1	30	73	0.04	14	0.06	13
II-2-2	30	73	0.06	17	0.08	15
II-2-3	56	146	0.19	24	0.24	20
II-2-4	56	146	0.21	24	0.32	22
II-2-5	121	330	0.96	35	1.83	32
II-2-6	121	330	0.60	26	1.00	19
II-2-7	256	720	2.90	37	6.12	31
II-2-8	256	720	2.51	38	5.74	28
II-2-9	529	1518	8.75	47	21.5	34
II-2-10	529	1518	10.7	55	26.7	42
II-2-11	1024	2976	23.4	48	98.5	43
II-2-12	1024	2976	26.7	57	88.0	48
II-2-13	2025	5940	103	87	318	54
II-2-14	3025	8910	181	115	551	65
II-2-15	3422	10090	206	94	619	54
II-2-16	4900	14490	623	144	$> 900^*$	-

* The convergence criterion (7.4) was not satisfied in 900 seconds.

Table 5: Detailed results for problems I-1-11 and I-1-12

	problem I-1-11		problem I-1-12	
	$\epsilon_{CG} = 10^{-1}$	$\epsilon_{CG} = 10^{-3}$	$\epsilon_{CG} = 10^{-1}$	$\epsilon_{CG} = 10^{-3}$
(a) Iterations				
$\epsilon = 10^{-1}$	33	33	30	22
10^{-2}	4/37	3/36	3/33	4/26
10^{-3}	3/40	1/37	3/36	2/28
10^{-4}	2/42	0/37	4/40	0/28
(b) CPU (sec)				
$\epsilon = 10^{-1}$	20.3	58.2	17.2	42.7
10^{-2}	0.6/20.9	1.9/60.1	0.6/17.8	2.4/45.1
10^{-3}	0.7/21.6	0.6/60.7	0.5/18.3	1.2/46.3
10^{-4}	0.4/22.0	0/60.7	0.9/19.2	0/46.3

Table 6: Detailed results for problems II-1-11 and II-1-12

	problem II-1-11		problem II-1-12	
	$\epsilon_{CG} = 10^{-1}$	$\epsilon_{CG} = 10^{-3}$	$\epsilon_{CG} = 10^{-1}$	$\epsilon_{CG} = 10^{-3}$
(a) Iterations				
$\epsilon = 10^{-1}$	55	65	46	53
10^{-2}	11/66	4/69	8/54	3/56
10^{-3}	11/77	5/74	7/61	4/60
10^{-4}	2/79	2/76	3/65	2/62
(b) CPU (sec)				
$\epsilon = 10^{-1}$	60.7	215.9	58.9	175.9
10^{-2}	4.6/65.3	3.4/219.3	3.4/62.3	2.6/178.5
10^{-3}	4.6/69.9	5.4/224.7	1.9/64.2	2.3/180.8
10^{-4}	0.4/70.3	1.8/226.5	0.4/64.6	1.7/182.5

Table 7: Detailed results for problems I-2-11 and I-2-12

	problem I-2-11		problem I-2-12	
	$\epsilon_{CG} = 10^{-1}$	$\epsilon_{CG} = 10^{-3}$	$\epsilon_{CG} = 10^{-1}$	$\epsilon_{CG} = 10^{-3}$
(a) Iterations				
$\epsilon = 10^{-1}$	21	16	25	16
10^{-2}	15/36	5/21	7/32	8/24
10^{-3}	19/55	7/28	16/48	9/33
10^{-4}	18/73	2/30	12/60	4/37
(b) CPU (sec)				
$\epsilon = 10^{-1}$	7.5	26.4	9.4	21.6
10^{-2}	3.5/11.0	4.9/31.3	2.0/11.4	6.0/27.6
10^{-3}	6.2/17.2	6.5/37.8	3.5/14.9	6.4/34.0
10^{-4}	3.4/20.6	1.8/39.6	5.6/20.5	4.1/38.1

Table 8: Detailed results for problems II-2-11 and II-2-12

	problem II-2-11		problem II-2-12	
	$\epsilon_{CG} = 10^{-1}$	$\epsilon_{CG} = 10^{-3}$	$\epsilon_{CG} = 10^{-1}$	$\epsilon_{CG} = 10^{-3}$
(a) Iterations				
$\epsilon = 10^{-1}$	24	27	26	29
10^{-2}	13/37	9/36	11/37	10/39
10^{-3}	11/48	6/43	20/57	9/48
10^{-4}	12/60	3/46	14/71	5/52
(b) CPU (sec)				
$\epsilon = 10^{-1}$	18.2	80.9	20.9	72.0
10^{-2}	2.6/20.8	9.9/90.8	1.7/22.6	8.0/80.0
10^{-3}	2.6/23.4	7.7/98.5	4.1/26.8	7.9/87.9
10^{-4}	2.2/26.7	3.1/101.6	4.7/31.4	3.7/91.7

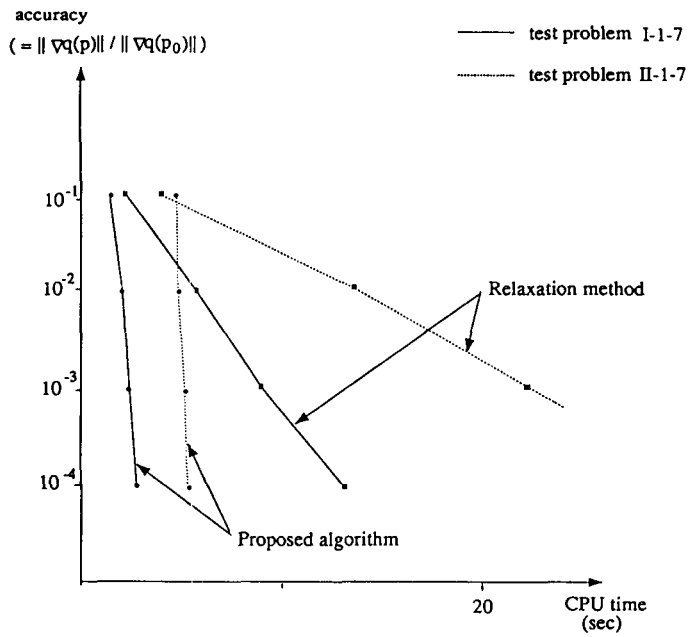

Figure 6. Comparison of the proposed algorithm and relaxation method for test problems I-1-7 and II-1-7.

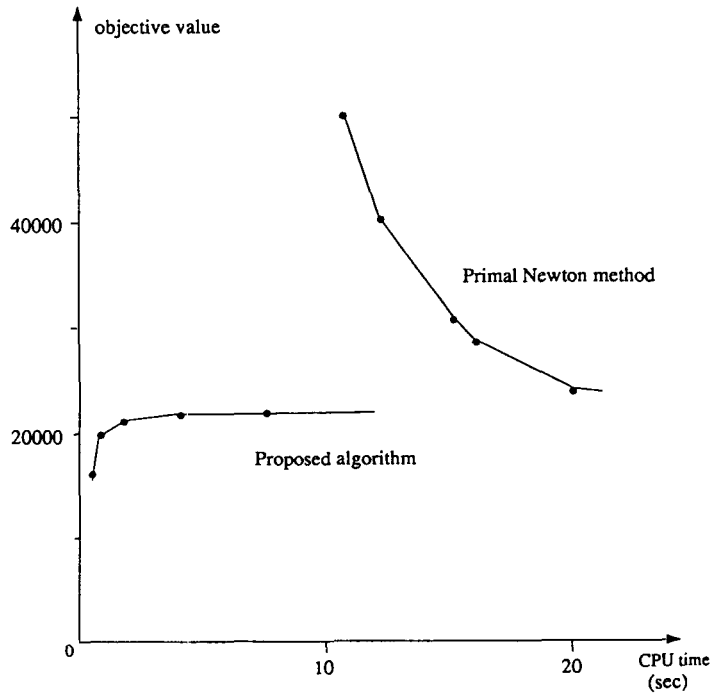

Figure 7. Comparison of the proposed algorithm with the primal Newton method for test problem I-1-7. The curve for the proposed algorithm corresponds to the negative of the objective values of the dual problem (D).

8. Conclusion

We have proposed a globally convergent dual-based Newton method for nonlinear minimum cost network flow problems. The method can effectively be applied to problems whose cost functions are strictly convex and co-finite. We have solved various test problems of up to 4900 nodes and 14490 arcs, and obtained very encouraging results in term of the speed of convergence and the accuracy of the computed solutions.

Acknowledgement

The authors are grateful to Dr. Paul Tseng for his helpful comments on the relaxation method. The implementation of the relaxation method reported in Section 7 is, however, entirely responsible to the authors of this paper.

References

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

[2] Beck, P., Lasdon, L. and Engquist, M.: "A reduced gradient algorithm for nonlinear network flow problems," *ACM Transactions on Mathematical Software*, Vol. 9 (1983), 57-70.

[3] Bertsekas, D.P. and El Baz, D.: "Distributed asynchronous relaxation methods for convex network flow problems," *SIAM Journal on Control and Optimization*, Vol. 25 (1987), 74-85.

[4] Bertsekas, D.P., Hossein, P. and Tseng, P.: "Relaxation methods for network flow problems with convex arc costs," *SIAM Journal on Control and Optimization*, Vol. 25 (1987), 1219-1243.

[5] Fletcher, R.: *Practical Methods of Optimization, Second Edition.* John Wiley & Sons, Chichester, 1987.

[6] Florian, M., Guelat, J. and Spiess, H.: "An efficient implementation of the "PARTAN" variant of the linear approximation method for the network equilibrium problem," *Networks*, Vol. 17 (1987), 319-339.

[7] 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.

[8] Golub, G.H. and Van Loan, C.F.: *Matrix Computations, Second Edition*, Johns Hopkins University Press, Baltimore, 1989.

[9] Hager, W.W. and Hearn, D.W.: "The dual active set algorithm and quadratic networks," Research Report No. 90-7, Department of Industrial and Systems Engineering, University of Florida (1990).

[10] Katsura, R., Fukushima, M. and Ibaraki, T.: "Interior methods for nonlinear minimum cost network flow problems," *Journal of the Operations Research Society of Japan*, Vol. 32 (1989), 174-199.

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

[12] Klincewicz, J.G.: "A Newton method for convex separable network flow problems," *Networks*, Vol. 13 (1983), 427-442.

[13] Klincewicz, J.G.: "Implementing an "exact" Newton method for separable convex transeportation problems," *Networks*, Vol. 19 (1989), 95-105.

[14] Luenberger, D.G.: *Introduction to Linear and Nonlinear Programming, Second Edition*. Addison-Wesley, Reading, Massachusetts, 1984.

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

[16] Rockafellar, R.T.: *Convex Analysis*. Princeton University Press, Princeton, New Jersey, 1970.

[17] Tseng, P. and Bertsekas, D.P.: "Relaxation methods for problems with strictly convex separable costs and linear constraints," *Mathematical Programming*, Vol. 38 (1987), 303-321.

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