Annals of Operations Research 13(1988)125 – 190 

# THE RELAX CODES FOR LINEAR MINIMUM COST NETWORK FLOW PROBLEMS*

Dimitri P. BERTSEKAS and Paul TSENG

*Laboratory for Information and Decision Systems and the Operations Research Center, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA*

## Abstract

We describe a relaxation algorithm [1,2] for solving the classical minimum cost network flow problem. Our implementation is compared with mature state-of-the-art primal simplex and primal-dual codes and is found to be several times faster on all types of randomly generated network flow problems. Furthermore, the speed-up factor increases with problem dimension. The codes, called RELAX-II and RELAXT-II, have a facility for efficient reoptimization and sensitivity analysis, and are in the public domain.

## 1. Introduction

Consider a directed graph with a set of nodes $\mathcal{N}$ and a set of arcs $\mathcal{A}$. Each arc $(i, j)$ has associated with it an integer $a_{ij}$ referred to as the *cost* of $(i, j)$. We denote by $f_{ij}$ the *flow* of the arc $(i, j)$ and consider the classical minimum cost flow problem

$$\text{minimize} \quad \sum_{(i,j)\in\mathcal{A}} a_{ij} f_{ij} \qquad \text{(MCF)}$$

$$\text{subject to} \quad \sum_{\substack{m \\ (m,i)\in\mathcal{A}}} f_{mi} - \sum_{\substack{m \\ (i,m)\in\mathcal{A}}} f_{im} = 0, \;\; \forall i \in \mathcal{N} \quad \text{(conservation of flow)} \qquad (1)$$

$$\ell_{ij} \leqslant f_{ij} \leqslant c_{ij}, \;\; \forall (i,j) \in \mathcal{A} \qquad \text{(capacity constraint)} \qquad (2)$$

where $\ell_{ij}$ and $c_{ij}$ are given integers. We assume throughout that there exists at least one feasible solution of (MCF). We formulate a dual problem to (MCF).

We associate a Lagrange multiplier $p_i$ (referred to as the *price* of node $i$) with the $i$th conservation of flow constraint (1). By denoting by $f$ and $p$ the vectors with elements $f_{ij}$, $(i, j) \in \mathcal{A}$ and $p_i$, $i \in \mathcal{N}$, respectively, we can write the corresponding Lagrangian function

*This work has been supported by the National Science Foundation under Grant NSF-ECS-8217668.

$$L(f,p) = \sum_{(i,j)\in\mathcal{A}} (a_{ij} + p_j - p_i) f_{ij}\,.$$

The dual problem is

$$\begin{aligned} &\text{maximize} \quad q(p) \\ &\text{subject to} \quad \text{no constraints on } p, \end{aligned} \tag{3}$$

where the dual functional $q$ is given by

$$\begin{aligned} q(p) &= \min_{\ell_{ij} \leqslant f_{ij} \leqslant c_{ij}} L(f,p) \\ &= \sum_{(i,j)\in\mathcal{A}} \min_{\ell_{ij} \leqslant f_{ij} \leqslant c_{ij}} \{(a_{ij} + p_j - p_i) f_{ij}\} \triangleq \sum_{(i,j)\in\mathcal{A}} q_{ij}(p_i - p_j). \end{aligned} \tag{4}$$

The form of the dual arc cost functions $q_{ij}$ is shown in fig. 1.

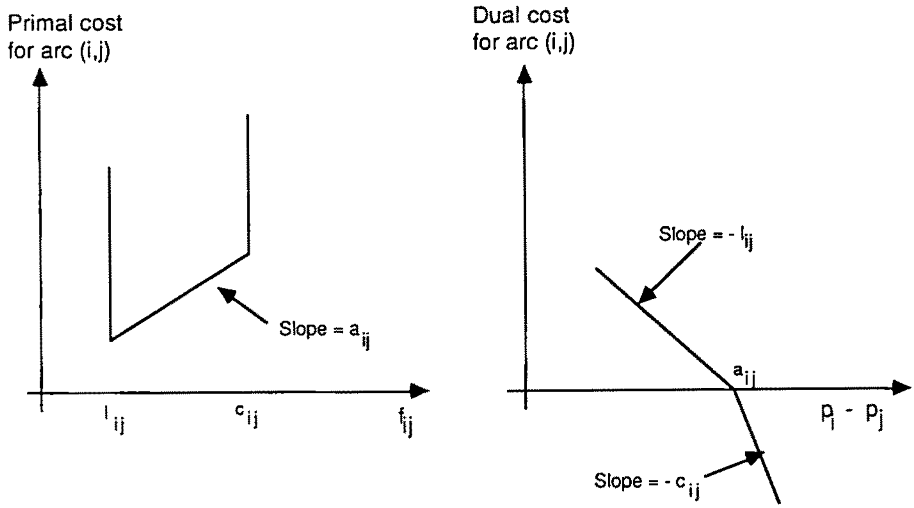



Fig. 1. Primal and dual costs for arc $(i,j)$.

Given any price vector $p$, we consider the corresponding *tension vector* $t$ having elements $t_{ij}, (i,j) \in \mathcal{A}$ defined by

$$t_{ij} = p_i - p_j, \quad \forall (i,j) \in \mathcal{A}. \tag{5}$$

Since the dual functional as well as subsequent definitions, optimality conditions and algorithms depend on the price vector $p$ only through the corresponding tension vector $t$, we will often make no distinction between $p$ and $t$ in what follows.

For any price vector $p$, we say that an arc $(i, j)$ is:

$$\text{Inactive} \quad \text{if} \quad t_{ij} < a_{ij} \tag{6}$$

$$\text{Balanced} \quad \text{if} \quad t_{ij} = a_{ij} \tag{7}$$

$$\text{Active} \quad \text{if} \quad t_{ij} > a_{ij}. \tag{8}$$

For any flow vector $f$, the scalar

$$d_i = \sum_{\substack{m \\ (i,m) \in \mathcal{A}}} f_{im} - \sum_{\substack{m \\ (m,i) \in \mathcal{A}}} f_{mi} \quad \forall i \in \mathcal{N} \tag{9}$$

will be referred to as the *deficit* of node $i$. It represents the difference of total flow exported and total flow imported by the node.

The optimality conditions in connection with (MCF) and its dual given by (3) and (4) state that $(f, p)$ is a primal and dual optimal solution pair if and only if

$$f_{ij} = \ell_{ij} \qquad \text{for all inactive arcs } (i, j) \tag{10}$$

$$\ell_{ij} \leqslant f_{ij} \leqslant c_{ij} \qquad \text{for all balanced arcs } (i, j) \tag{11}$$

$$f_{ij} = c_{ij} \qquad \text{for all active arcs } (i, j) \tag{12}$$

$$d_i = 0 \qquad \text{for all nodes } i. \tag{13}$$

Relations (10)–(12) are known as the *complementary slackness conditions.*

Our approach is based on iterative ascent of the dual functional. The price vector $p$ is updated while simultaneously maintaining a flow vector $f$ satisfying complementary slackness with $p$. The algorithms proposed terminate when $f$ satisfies primal feasibility (deficit of each node equals zero). The main feature of the algorithms, which distinguishes them from classical primal-dual methods, is that the choice of ascent directions is very simple. At a given price vector $p$, a node $i$ with nonzero deficit is chosen, and an ascent is attempted along the coordinate $p_i$. If such an ascent is not possible and a reduction of the total absolute deficit $\Sigma_m |d_m|$ cannot be effected through flow augmentation, an adjacent node of $i$, say $i_1$, is chosen and an ascent is attempted along the sum of the coordinate vectors corresponding to $i$ and $i_1$. If such an ascent is not possible, and flow augmentation is not possible either, an adjacent node of either $i$ or $i_1$ is chosen and the process is continued. In practice, most of the ascent directions are single coordinate directions, leading to the interpretation of the algorithms as *coordinate ascent* or *relaxation methods.* This is an important characteristic, and a key factor in the algorithms' efficiency. We have found through experiment that, for ordinary networks, the ascent directions used by our algorithms lead to comparable improvement per iteration as the direction of maximal rate of ascent (the one used by the classical primal-dual method), but are computed with considerably less overhead.

In the next section, we characterize the ascent directions used in the algorithms. In sect. 3, we describe our relaxation methods. In sect. 4, we describe the codes and give results of computational experimentation.

## 2. Characterization of ascent directions

Each ascent direction used by the algorithm is associated with a connected strict subset $S$ of $\mathcal{N}$, and has the form $v = \{v_{ij} | (i, j) \in \mathcal{A}\}$, where

$$v_{ij} = \begin{cases} 1 & \text{if } i \notin S,\ j \in S \\ -1 & \text{if } i \in S,\ j \notin S \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

Changing any tension vector $t$ in the direction $v$ of (14) corresponds to decreasing the prices of all nodes in $S$ by an equal amount while leaving the prices of all other nodes unchanged. It is seen from (4) that the directional derivative at $t$ of the dual cost in the direction $v$ is $C(v, t)$, where

$$C(v, t) = \sum_{(i,j) \in \mathcal{A}} \lim_{\alpha \to 0^+} \frac{q_{ij}(t_{ij} + \alpha v_{ij}) - q_{ij}(t_{ij})}{\alpha}$$

$$= \sum_{(i,j) \in \mathcal{A}} e_{ij}(v_{ij}, t_{ij}) \tag{15}$$

and

$$e_{ij}(v_{ij}, t_{ij}) = \begin{cases} -v_{ij}\ell_{ij} & \text{if } (i,j) \text{ is inactive or if } (i,j) \\ & \quad \text{is balanced and } v_{ij} \leqslant 0 \\ -v_{ij}c_{ij} & \text{if } (i,j) \text{ is active or if } (i,j) \\ & \quad \text{is balanced and } v_{ij} \geqslant 0. \end{cases} \tag{16}$$

Note that $C(v, t)$ is the difference of outflow and inflow across $S$ when the flows of inactive and active arcs are set at their lower and upper bounds, respectively, while the flow of each balanced arc incident to $S$ is set to its lower or upper bound depending on whether the arc is going out of $S$ or coming into $S$, respectively. We have the following proposition.

PROPOSITION 1

For every non-empty strict subset $S$ of $\mathcal{N}$ and every tension vector $t$, there holds

$$w(t + \gamma v) = w(t) + \gamma C(v, t), \quad \forall \gamma \in [0, \delta), \tag{17}$$

where $w(\cdot)$ is the dual cost as a function of $t$

$$w(t) = \sum_{(i,j)} q_{ij}(t_{ij}). \tag{18}$$

Here, $v$ is given by (14) and $\delta$ is given by

$$\delta = \inf\{\{t_{im} - a_{im} \mid i \in S, m \notin S, (i, m) : \text{active}\},$$
$$\{a_{mi} - t_{mi} \mid i \in S, m \notin S, (m, i) : \text{inactive}\}\}. \tag{19}$$

(We use the convention $\delta = +\infty$ if the set over which the infimum above is taken is empty.)

*Proof*

It was seen [cf. (15)] that the rate of change of the dual cost $w$ at $t$ along $v$ is $C(v, t)$. Since $w$ is piecewise linear, the actual change of $w$ along the direction $v$ is linear in the stepwise $\gamma$ up to the point where $\gamma$ becomes large enough so that the pair $[w(t + \gamma v), t + \gamma v]$ meets a new face of the graph of $w$. This value of $\gamma$ is the one for which a new arc incident to $S$ becomes balanced and it equals the scalar $\delta$ of (19). Q.E.D.

## 3. The relaxation method

The relaxation algorithm maintains complementary slackness at all times. At each iteration, it starts from a single node with nonzero deficit and checks whether changing its price can improve the value of the dual cost. If not, it gradually builds up, via a labeling procedure, either a flow augmenting path or a cutset associated with a direction of ascent. The main difference from the classical primal-dual method is that instead of continuing the labeling process until a maximal set of nodes is labeled, we stop at the *first* possible direction of ascent – frequently the direction associated with just the starting node.

TYPICAL RELAXATION ITERATION FOR AN ORDINARY NETWORK

At the beginning of each iteration, we have a pair $(f, t)$ satisfying complementary slackness. The iteration determines a new pair $(f, t)$ satisfying complementary slackness by means of the following process.

*Step 1:* Choose a node $s$ with $d_s > 0$. (The iteration can be started also from a node $s$ with $d_s < 0$ – the steps are similar.) If no such node can be found, terminate the algorithm. Else give the label "0" to $s$, set $S = \emptyset$, and go to step 2. Nodes in $S$ are said to be *scanned.*

*Step 2:* Choose a labeled but unscanned node $k$, set $S = S \cup \{k\}$, and go to step 3.

*Step 3:* Scan the label of the node $k$ as follows: Give the label "$k$" to all unlabeled nodes $m$ such that $(m, k)$ is balanced and $f_{mk} < c_{mk}$, and to all unlabeled $m$ such that $(k, m)$ is balanced and $\ell_{km} < f_{km}$. If $v$ is the vector corresponding to $S$ as in (14) and

$$C(v, t) > 0, \tag{20}$$

go to step 5. Else if for any of the nodes $m$ labeled from $k$ we have $d_m < 0$, go to step 4. Else go to step 2.

*Step 4 (flow augmentation):* A directed path $P$ has been found that begins at the starting node $s$ and ends at the node $m$ with $d_m < 0$ identified in step 3. The path is constructed by tracing labels backwards starting from $m$, and consists of balanced arcs such that we have $\ell_{kn} < f_{kn}$ for all $(k, n) \in P^+$ and $f_{kn} < c_{kn}$ for all $(k, n) \in P^-$, where

$$P^+ = \{(k, n) \in P \mid (k, n) \text{ is oriented in the direction from } s \text{ to } m\} \tag{21}$$

$$P^- = \{(k, n) \in P \mid (k, n) \text{ is oriented in the direction from } m \text{ to } s\}. \tag{22}$$

Let

$$\epsilon = \min\{d_s, -d_m, \{f_{kn} - \ell_{kn} \mid (k, n) \in P^+\}, \{c_{kn} - f_{kn} \mid (k, n) \in P^-\}\}. \tag{23}$$

Decrease by $\epsilon$ the flows of all arcs $(k, n) \in P^+$, increase by $\epsilon$ the flows of all arcs $(k, n) \in P^-$, and go to the next iteration.

*Step 5 (price adjustment):* Let

$$\delta = \min\{\{t_{km} - a_{km} \mid k \in S,\ m \notin S,\ (k, m)\text{: active}\},\ \{a_{mk} - t_{mk} \mid k \in S,\ m \notin S,\ (m, k)\text{: inactive}\}\}, \tag{24}$$

where $S$ is the set of scanned nodes constructed in step 2. Set

$$f_{km} := \ell_{km},\ \forall \text{ balanced arcs } (k, m) \text{ with } k \in S,\ m \in L,\ m \notin S$$

$$f_{mk} := c_{mk},\ \forall \text{ balanced arcs } (m, k) \text{ with } k \in S,\ m \in L,\ m \notin S,$$

where $L$ is the set of labeled nodes. Set

$$t_{km} := \begin{cases} t_{km} + \delta & \text{if} \quad k \notin S,\ m \in S \\ t_{km} - \delta & \text{if} \quad k \in S,\ m \notin S \\ t_{km} & \text{otherwise.} \end{cases}$$

Go to the next iteration.

The relaxation iteration terminates with either a flow augmentation (via step 4) or with a dual cost improvement (via step 5). In order for the procedure to be well defined, however, we must show that whenever we return to step 2 from step 3, there is still some labeled node which is unscanned. Indeed, when all labeled nodes are scanned (i.e. the set $S$ coincides with the labeled set), there is no balanced arc $(m, k)$ such that $m \notin S$, $k \in S$ and $f_{mk} < c_{mk}$ or a balanced arc $(k, m)$ such that $k \in S$, $m \notin S$ and $f_{km} > \ell_{km}$. It follows from the definition (15), (16) [see also the following equation (25)] that

$$C(v, t) = \sum_{k \in S} d_k\ .$$

Under the above circumstances, all nodes in $S$ have nonnegative deficit and at least one node in $S$ (the starting node $s$) has strictly positive deficit. Therefore, $C(v, t) > 0$ and it follows that the procedure switches from step 3 to step 5 rather than switch back to step 2.

If $a_{ij}$, $\ell_{ij}$, and $c_{ij}$ are integer for all $(i, j) \in \mathcal{A}$ and the starting $t$ is integer, then $\delta$ as given by (24) will also be a positive integer and the dual cost is increased by an integer amount each time step 5 is executed. Each time a flow augmentation takes place via step 4, the dual cost remains unchanged. If the starting $f$ is integer, all successive $f$ will be integer, so the amount of flow augmentation $\epsilon$ in step 4 will be a positive integer. Therefore, there can be only a finite number of flow augmentations between successive reductions of the dual cost. It follows that the algorithm will finitely terminate at an integer optimal pair $(f, t)$ if the starting pair $(f, t)$ is integer.

It can be seen that the relaxation iteration involves a comparable amount of computation per node scanned as the usual primal-dual method [3]. The only additional computation involves maintaining the quantity $C(v, t)$, but it can be seen that this can be computed *incrementally* in step 3 rather than recomputed each time the set $S$ is enlarged in step 2. As a result, this additional computation is insignificant. To compute $C(v, t)$ incrementally in the context of the algorithm, it is helpful to use the identity

$$C(v, t) = \sum_{i \in S} d_i - \sum_{\substack{(i,j):\text{ balanced} \\ i \in S,\, j \notin S}} (f_{ij} - \ell_{ij}) - \sum_{\substack{(i,j):\text{ balanced} \\ i \notin S,\, j \in S}} (c_{ij} - f_{ij}). \tag{25}$$

We note that a similar iteration can be constructed starting from a node with negative deficit. Here, the set $S$ consists of nodes with nonpositive deficit, and in step 5, the prices of the nodes in $S$ are increased rather than decreased. The straightforward details are left to the reader. Computational experience suggests that termination is typically accelerated when ascent iterations are initiated from nodes with negative as well as positive deficit.

## LINE SEARCH

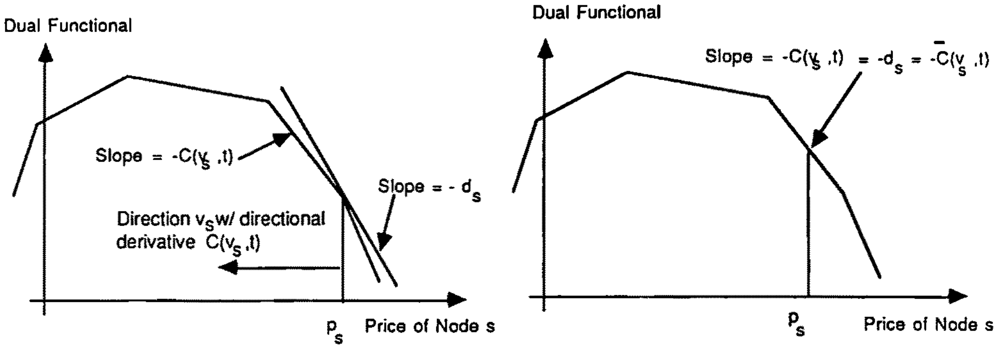
The stepsize $\delta$ of (24) corresponds to the first break point of the (piecewise linear) dual functional along the ascent direction. It is possible to instead use an optimal stepsize that maximizes the dual functional along the ascent direction. Such a stepsize can be calculated quite efficiently by testing the sign of the directional derivative of the dual cost at successive break points along the ascent direction. Computational experimentation showed that this type of line search is beneficial, and was implemented in the relaxation codes.

## SINGLE NODE ITERATIONS

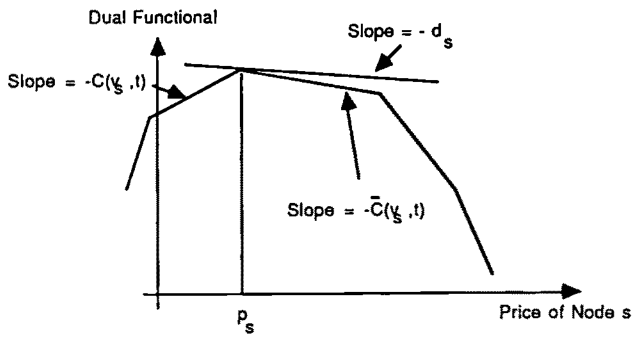
The case where the relaxation iteration scans a single node (the starting node $s$ having positive deficit $d_s$), finds the corresponding direction $v_s$ to be an ascent direction, i.e.

$$C(v_s, t) = d_s - \sum_{(s,m):\text{ balanced}} (f_{sm} - \ell_{sm}) - \sum_{(m,s):\text{ balanced}} (c_{ms} - f_{ms}) > 0, \tag{26}$$

reduces the price $p_s$ (perhaps repeatedly via the line search mentioned earlier) and terminates is particularly important for the conceptual understanding of the algorithm.




CASES WHERE A SINGLE NODE ITERATION IS POSSIBLE




CASE WHERE A SINGLE NODE ITERATION IS NOT POSSIBLE

Fig. 2. Illustration of dual functional and its directional derivatives along the price coordinate $p_s$. Break points correspond to values of $p_s$ where one or more arcs incident to node $s$ are balanced.

We believe that much of the success of the algorithm is owed to the relatively large number of single node iterations for many classes of problems.

When only the price of a single node $s$ is changed, the absolute value of the deficit of $s$ is decreased at the expense of possibly increasing the absolute value of the deficit of its neighboring nodes. This is reminiscent of *relaxation methods* where a change of a single variable is effected with the purpose of satisfying a single constraint at the expense of violating others.

A dual viewpoint, reminiscent of *coordinate ascent methods*, is that a single (the $s$th) coordinate direction is chosen and a line search is performed along this direction. Figure 2 shows the form of the dual function along the direction of the coordinate $p_s$ for a node with

$$d_s > 0.$$

The left-hand slope at $p_s$ is

$$-C(v_s, t),$$

while the right-hand slope is

$$-\bar{C}(v_s, t) = -\sum_{\substack{(s,m)\in\mathcal{A}\\(s,m):\,\text{active}\\\text{or balanced}}} c_{sm} - \sum_{\substack{(s,m)\in\mathcal{A}\\(s,m):\,\text{inactive}}} \ell_{sm}$$

$$+\sum_{\substack{(m,s)\in\mathcal{A}\\(m,s):\,\text{active}}} c_{ms} + \sum_{\substack{(m,s)\in\mathcal{A}\\(m,s):\,\text{inactive}\\\text{or balanced}}} \ell_{ms}\,.$$

We have

$$-\bar{C}(v_s, t) \leqslant -d_s \leqslant -C(v_s, t), \tag{27}$$

so $-d_s$ is a subgradient of the dual functional at $p_s$ in the $s$th coordinate direction.

A single node iteration will be possible if and only if the right-hand slope is negative or equivalently

$$C(v_s, t) > 0.$$

This will always be true if we are not at a corner and hence equality holds throughout in (27). However, if the dual cost is nondifferentiable at $p_s$ along the $s$th coordinate, it may happen that (see fig. 2)

$$-\bar{C}(v_s, t) \leqslant -d_s < 0 \leqslant -C(v_s, t),$$

in which case the single node iteration fails to make progress and we must resort to scanning more than one node.

Figure 3 illustrates a single node iteration for the case where $d_s > 0$. It is seen that the break points of the dual functional along the coordinate $p_s$ are the values of $p_s$ for which one or more arcs incident to node $s$ are balanced. The single node iteration shown starts with arcs $(1, s)$ and $(3, s)$ inactive, and arcs $(s, 2)$ and $(s, 4)$ active. To reduce $p_s$ beyond the first break point $p_4 + a_{s4}$. the flow of arc $(s, 4)$ must be pulled back from $f_{s4} = 30$ to $f_{s4} = 0$. At the level $p_3 - a_{3s}$, the dual cost is maximized because if the flow of arc $(3, s)$ is set to the lower bound of zero, the deficit $d_s$ switches from positive $(+10)$ to negative $(-10)$. Figure 4 illustrates a single node iteration for the same node when $d_s < 0$. The difference to the case $d_s > 0$ is that

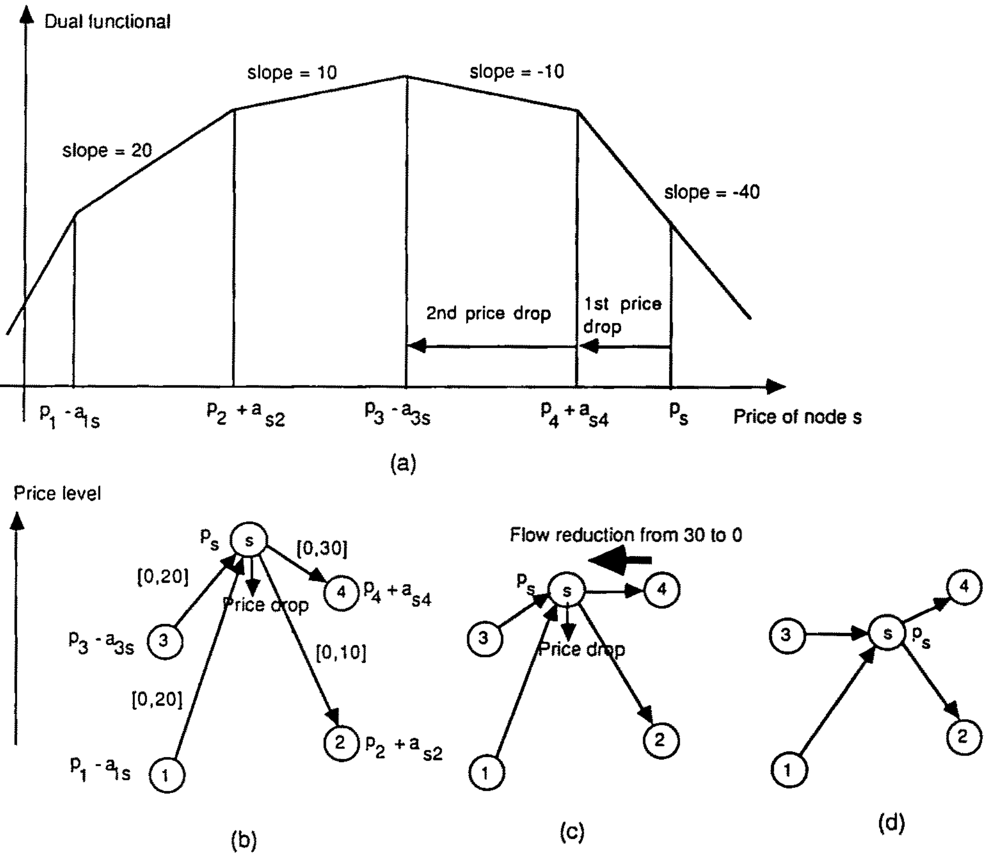



Fig. 3. Illustration of an iteration involving a single node $s$ with four adjacent arcs $(1, s)$, $(3, s)$, $(s, 2)$, $(s, 4)$ with feasible arc flow ranges $[1, 20]$, $[0, 20]$, $[0, 10]$, $[0,30]$, respectively. (a) Form of the dual functional along $p_s$ for given values of $p_1$, $p_2$, $p_3$, and $p_4$. The break points correspond to the levels of $p_s$ for which the corresponding arcs become balanced. (b) Illustration of a price drop of $p_s$ from a value higher than all break points to the break point at which arc $(s, 4)$ becomes balanced. (c) Price drop of $p_s$ to the break point at which arc $(3, s)$ becomes balanced. When this is done, arc $(s, 4)$ becomes inactive from balanced and its flow is reduced from 30 to 0 to maintain complementary slackness. (d) $p_s$ is now at the break point $p_3 - a_{3s}$ that maximizes the dual cost. Any further price drop makes arc $(3, s)$ active, increases its flow from 0 to 20, and changes the sign of the deficit $d_s$ from positive $(+10)$ to negative $(-10)$.

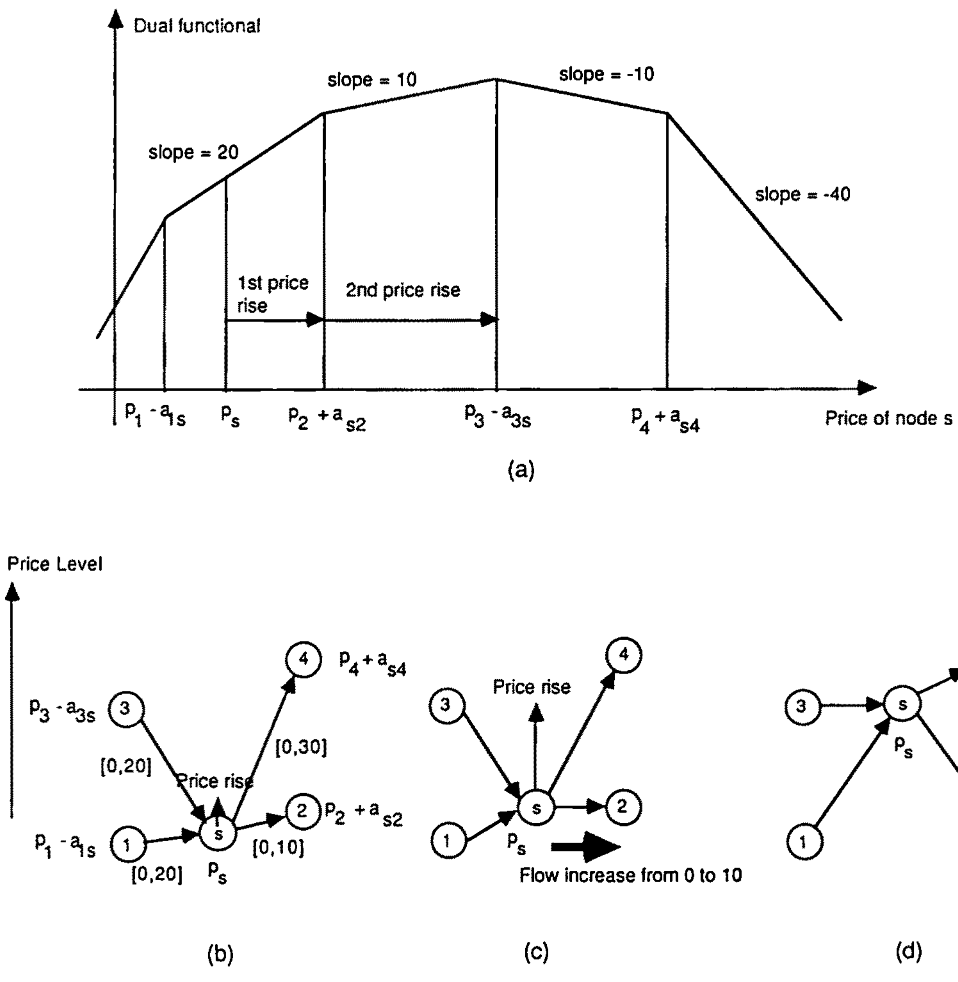



Fig. 4. Illustration of a price rise involving the single node $s$ for the example of fig. 3. Here, the initial price $p_s$ lies between the two leftmost break points corresponding to the arcs $(1, s)$ and $(s, 2)$. Initially, arcs $(1, s)$, $(s, 2)$, and $(s, 4)$ are inactive, and arc $(3, s)$ is active.

the price $p_s$ is increased, instead of decreased, and as $p_s$ moves beyond a break point, the flow of the corresponding balanced arc is pushed to the lower bound (for incoming arcs) and to the upper bound (for outgoing arcs), rather than pulled to the upper bound and lower bound, respectively.

DEGENERATE ASCENT ITERATIONS

If, for a given $t$, we can find a connected subset $S$ of $\mathcal{N}$ such that the corresponding vector $(u, v)$ satisfies

$$C(v, t) = 0,$$

then from proposition 1 we see that the dual cost remains constant as we start moving along the vector $v$. i.e.

$$w(t + \gamma v) = w(t), \qquad \forall \gamma \in [0, \delta),$$

where $w$, $v$, and $\delta$ are given by (14), (18), (19). We refer to such incremental changes in $t$ as *degenerate ascent iterations*. If the ascent condition $C(v, t) > 0$ [cf. (20)] is replaced by $C(v, t) \geqslant 0$, then we obtain an algorithm that produces at each iteration either a flow augmentation, or a strict dual cost improvement, or a degenerate ascent step. This algorithm has the same convergence properties as the one without degenerate steps under the following condition:

(C) For each degenerate ascent iteration, the starting node $s$ has positive deficit $d_s$, and at the end of the iteration, all nodes in the scanned set $S$ have nonnegative deficit.

We refer the reader to [1] for a proof of this fact. It can be easily seen that condition (C) always holds when the set $S$ consists of just the starting node $s$. For this reason, if the ascent iteration is modified so that a price adjustment at step 5 is made not only when $C(v, t) > 0$ but also when $d_s > 0$, $S = \{s\}$ and $C(v_s, t) = 0$, the algorithm maintains its termination properties. This modification was implemented in the relaxation codes and can have an important beneficial effect for special classes of problems such as assignment and transportation problems. We have no clear explanation for this phenomenon. For the assignment problem, condition (C) is guaranteed to hold even if $S$ contains more than one node. The assignment algorithm of [4] makes extensive use of degenerate ascent steps.

## 4. Code description and computational results

The relaxation codes RELAX-II and RELAXT-II solve the problem

$$\begin{aligned}
&\text{minimize} && \sum_{(i,j)\in\mathcal{A}} a_{ij} f_{ij} \\
&\text{subject to} && \sum_{(m,i)\in\mathcal{A}} f_{mi} - \sum_{(i,m)\in\mathcal{A}} f_{im} = b_i, && \forall i \in \mathcal{N} \\
& && 0 \leqslant f_{ij} \leqslant c_{ij}, && \forall (i,j) \in \mathcal{A}.
\end{aligned}$$

This form has become standard in network codes as it does not require storage and use of the array of lower bounds $\{\ell_{ij}\}$. Instead, the smaller size array $\{b_i\}$ is stored and used. The problem (MCF) of sect. 1 can be reduced to the above form by making the transformation of variables $f_{ij} := f_{ij} - \ell_{ij}$. The method for representing the problem is the linked list structure suggested by Aashtiani and Magnanti [5] and used in their KILTER code (see also Magnanti [6]). Briefly, during solution of the problem, we store for each arc its start and end node, its capacity, its reduced cost $(a_{ij} - t_{ij})$, its flow $f_{ij}$, the next arc with the same start node, and the next arc with the same end node. An additional array of length equal to half the number of arcs is used for internal calculations. This array could be eliminated at the expense of a modest increase in computation time. The total storage of RELAX-II for arc length arrays is 7.5 $|\mathcal{A}|$. RELAXT-II is a code that is similar to RELAX-II but employs two additional arc length arrays that essentially store the set of all balanced arcs. This code, written with the assistance of Jon Eckstein, is faster than RELAX-II, but requires 9.5 $|\mathcal{A}|$ total storage for arc length arrays. There is additional storage needed for node length arrays, but this is relatively insignificant for all but extremely sparse problems. This compares unfavorably with primal simplex codes, which can be implemented with four arc length arrays.

The RELAX-II and RELAXT-II codes implement with minor variations the relaxation algorithm of sect. 3. Line search and degenerate ascent steps are implemented as discussed in sect. 3.

The codes assume no prior knowledge about the structure of the problem or the nature of the solution. Initial prices are set to zero and initial arc flows are set to zero or the upper bound, depending on whether the arc cost is nonnegative or negative, respectively. RELAX-II and RELAXT-II include a preprocessing phase (included in the CPU time reported) whereby arc capacities are reduced to as small a value as possible without changing optimal solutions of the problem. Thus, for transportation problems, the capacity of each arc is set at the minimum of the supply and demand at the start and end nodes of the arc. We found experimentally that this preprocessing can markedly improve the performance of relaxation methods, particularly for transportation problems. We do not fully understand the nature of this phenomenon, but it is apparently related to the fact that tight arc capacities tend to make the shape of the isocost surfaces of the dual functional more "round". Generally speaking, tight arc capacity bounds increase the frequency of single node iterations. This behavior is in sharp contrast with that of primal simplex, which benefits from loose arc capacity bounds (fewer extreme points to potentially search over), and appears to be one of the main reasons for the experimentally observed superiority of relaxation over primal simplex for heavily capacitated problems.

It is possible to reduce the memory requirements of the codes by ordering the arc list of the network by head node, i.e. the outgoing arcs of the first node are listed first, followed by the outgoing arcs of the second node, etc. (forward star representation). If this is done, one arc length array becomes unnecessary, thereby reducing the memory requirements of RELAX-II to 6.5 arc length arrays, and of RELAXT-II to 8.5 arc length arrays. The problem solution time remains essentially unaffected by this device, but the time needed to prepare (or alter) the problem data will be increased. The same technique can also be used to reduce the memory requirements of the primal simplex method to three arc length arrays.

We have compared RELAX-II and RELAXT-II under identical test conditions with the primal-dual code KILTER (Aashtiani and Magnanti [5]) and the primal simplex code RNET (Grigoriadis and Hsu [7]). It is generally recognized that the performance of RNET is representative of the best that can be achieved with presently available simplex network codes written in FORTRAN. For example, Kennington and Helgason in their 1980 book [8] (p. 255) compare RNET with their own primal simplex code NETFLO on the first 35 NETGEN benchmarks [9] and conclude that "RNET . . . produced the shortest times that we have seen on these 35 test problems". Our computational results with these benchmarks are given in table 1 and show substantially faster computation times for the relaxation codes over both KILTER and RNET.

An important and intriguing property of RELAX-II and RELAXT-II is that their speedup factor over RNET apparently increases with the size of the problem. This can be seen by comparing the results for the small problems 1–35 with the results for the larger problems 37–40 of table 1. The comparison shows an improvement in the speedup factor that is not spectacular, but is noticeable and consistent. Table 2 shows that for even larger problems, the speedup factor increases further with problem dimension, and reaches or exceeds an order of magnitude (see fig. 5). This is particularly true for assignment problems where, even for relatively small problems, the speedup factor is of the order of 20 or more.

We note that there was some difficulty in generating the transportation problems of this table with NETGEN. Many of the problems generated were infeasible because some node supplies and demands were coming out zero or negative. This was resolved by adding the same number (usually 10) to all source supplies and all sink demands, as noted in table 2. Note that the transportation problems of the table are divided into groups and each group has the same average degree per node (8 for problems 6–15, and 20 for problems 16–20).

Table 1

Standard Benchmark Problems 1 – 40 of [9] obtained using NETGEN. All times are in secs on a VAX 11/750. All codes compiled by FORTRAN in OPTIMIZE mode under VMS version 3.7, and under VMS version 4.1, as indicated. All codes run on the same machine under identical conditions. Problem 36 could not be generated with our version of NETGEN

| Problem type | Problem no. | No. of nodes | No. of arcs | RELAX-II (VMS 3.7/ VMS 4.1) | RELAXT-II (VMS 3.7/ VMS 4.1) | KILTER VMS 3.7 | RNET VMS 3.7 |
|---|---|---|---|---|---|---|---|
| Transportation | 1 | 200 | 1300 | 2.07/1.75 | 1.47/1.22 | 8.81 | 3.15 |
| | 2 | 200 | 1500 | 2.12/1.76 | 1.61/1.31 | 9.04 | 3.72 |
| | 3 | 200 | 2000 | 1.92/1.61 | 1.80/1.50 | 9.22 | 4.42 |
| | 4 | 200 | 2200 | 2.52/2.12 | 2.38/1.98 | 10.45 | 4.98 |
| | 5 | 200 | 2900 | 2.97/2.43 | 2.53/2.05 | 16.48 | 7.18 |
| | 6 | 300 | 3150 | 4.37/3.66 | 3.57/3.00 | 25.08 | 9.43 |
| | 7 | 300 | 4500 | 5.46/4.53 | 3.83/3.17 | 35.55 | 12.60 |
| | 8 | 300 | 5155 | 5.39/4.46 | 4.30/3.57 | 46.30 | 15.31 |
| | 9 | 300 | 6075 | 6.38/5.29 | 5.15/4.30 | 43.12 | 18.99 |
| | 10 | 300 | 6300 | 4.12/3.50 | 3.78/3.07 | 47.80 | 16.44 |
| Total (problems 1 – 10) | | | | 37.32/31.11 | 30.42/25.17 | 251.85 | 96.22 |
| Assignment | 11 | 400 | 1500 | 1.23/1.03 | 1.35/1.08 | 8.09 | 4.92 |
| | 12 | 400 | 2250 | 1.38/1.16 | 1.54/1.25 | 10.76 | 6.43 |
| | 13 | 400 | 3000 | 1.68/1.42 | 1.87/1.54 | 8.99 | 8.92 |
| | 14 | 400 | 3750 | 2.43/2.07 | 2.67/2.16 | 14.52 | 9.90 |
| | 15 | 400 | 4500 | 2.79/2.34 | 3.04/2.46 | 14.53 | 10.20 |
| Total (problems 11 – 15) | | | | 9.51/8.02 | 10.47/8.49 | 56.89 | 40.37 |
| Uncapacitated & lightly capacitated problems | 16 | 400 | 1306 | 2.79/2.40 | 2.60/2.57 | 13.57 | 2.76 |
| | 17 | 400 | 2443 | 2.67/2.29 | 2.80/2.42 | 16.89 | 3.42 |
| | 18 | 400 | 1306 | 2.56/2.20 | 2.74/2.39 | 13.05 | 2.56 |
| | 19 | 400 | 2443 | 2.73/2.32 | 2.83/2.41 | 17.21 | 3.61 |
| | 20 | 400 | 1416 | 2.85/2.40 | 2.66/2.29 | 11.88 | 3.00 |
| | 21 | 400 | 2836 | 3.80/3.23 | 3.77/3.23 | 19.06 | 4.48 |
| | 22 | 400 | 1416 | 2.56/2.18 | 2.82/2.44 | 12.14 | 2.86 |
| | 23 | 400 | 2836 | 4.91/4.24 | 3.83/3.33 | 19.65 | 4.58 |
| | 24 | 400 | 1382 | 1.27/1.07 | 1.47/1.27 | 13.07 | 2.63 |
| | 25 | 400 | 2676 | 2.01/1.68 | 2.13/1.87 | 26.17 | 5.84 |
| | 26 | 400 | 1382 | 1.79/1.57 | 1.60/1.41 | 11.31 | 2.48 |
| | 27 | 400 | 2676 | 2.15/1.84 | 1.97/1.75 | 18.88 | 3.62 |
| Total (problems 16 – 27) | | | | 32.09/27.42 | 31.22/27.38 | 192.88 | 41.94 |

Table 1 (continued)

| Problem type | Problem no. | No. of nodes | No. of arcs | RELAX-II (VMS 3.7/ VMS 4.1) | RELAXT-II (VMS 3.7/ VMS 4.1) | KILTER VMS 3.7 | RNET VMS 3.7 |
|---|---|---|---|---|---|---|---|
| Uncapacitated and lightly capacitated problems | 28 | 1000 | 2900 | 4.90/4.10 | 5.67/5.02 | 29.77 | 8.60 |
| | 29 | 1000 | 3400 | 5.57/4.76 | 5.13/4.43 | 32.36 | 12.01 |
| | 30 | 1000 | 4400 | 7.31/6.47 | 7.18/6.26 | 42.21 | 11.12 |
| | 31 | 1000 | 4800 | 5.76/4.95 | 7.14/6.30 | 39.11 | 10.45 |
| | 32 | 1500 | 4342 | 8.20/7.07 | 8.25/7.29 | 69.28 | 18.04 |
| | 33 | 1500 | 4385 | 10.39/8.96 | 8.94/7.43 | 63.59 | 17.29 |
| | 34 | 1500 | 5107 | 9.49/8.11 | 8.88/7.81 | 72.51 | 20.50 |
| | 35 | 1500 | 5730 | 10.95/9.74 | 10.52/9.26 | 67.49 | 17.81 |
| Total (problems 28 – 35) | | | | 62.57/54.16 | 61.71/53.80 | 356.32 | 115.82 |
| Large uncapacitated problems | 37 | 5000 | 23000 | 87.05/73.64 | 74.67/66.66 | 681.94 | 281.87 |
| | 38 | 3000 | 35000 | 68.25/57.84 | 55.84/47.33 | 607.89 | 274.46 |
| | 39 | 5000 | 15000 | 89.83/75.17 | 66.23/58.74 | 558.60 | 151.00 |
| | 40 | 3000 | 23000 | 50.42/42.73 | 35.91/30.56 | 369.40 | 174.74 |
| Total (problems 37 – 40) | | | | 295.55/249.38 | 232.65/203.29 | 2 217.83 | 882.07 |

Table 2

Large Assignment and Transportation Problems. Times in secs on VAX 11/750. All problems obtained using NETGEN, as described in the text. RELAX-II and RELAXT-II compiled under VMS 4.1; RNET compiled under VMS 3.7. Problems marked with * were obtained by NETGEN, and then, to make to problem feasible, an increment of 2 was added to the supply of each source node, and the demand of each sink node. Problems marked with + were similarly obtained, but the increment was 10

| No. | Problem type | No. of sources | No. of sinks | No. of arcs | Cost range | Total supply | RELAX-II | RELAXT-II | RNET |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Assignment | 1 000 | 1 000 | 8 000 | 1–10 | 1 000 | 4.68 | 4.60 | 79.11 |
| 2 | | 1 500 | 1 500 | 12 000 | 1–10 | 1 500 | 7.23 | 7.03 | 199.44 |
| 3 | | 2 000 | 2 000 | 16 000 | 1–10 | 2 000 | 12.65 | 9.95 | 313.64 |
| 4 | | 1 000 | 1 000 | 8 000 | 1–1 000 | 1 000 | 9.91 | 10.68 | 118.60 |
| 5 | | 1 500 | 1 500 | 12 000 | 1–1 000 | 1 500 | 17.82 | 14.58 | 227.57 |
| 6 | Transportation | 1 000 | 1 000 | 8 000 | 1–10 | 100 000 | 31.43 | 27.83 | 129.95 |
| 7* | | 1 500 | 1 500 | 12 000 | 1–10 | 153 000 | 60.86 | 56.20 | 300.79 |
| 8+ | | 2 000 | 2 000 | 16 000 | 1–10 | 220 000 | 127.73 | 99.69 | 531.14 |
| 9+ | | 2 500 | 2 500 | 20 000 | 1–10 | 275 000 | 144.66 | 115.65 | 790.57 |
| 10+ | | 3 000 | 3 000 | 24 000 | 1–10 | 330 000 | 221.81 | 167.49 | 1 246.45 |
| 11 | Transportation | 1 000 | 1 000 | 8 000 | 1–1 000 | 100 000 | 32.60 | 31.99 | 152.17 |
| 12* | | 1 500 | 1 500 | 12 000 | 1–1 000 | 153 000 | 53.84 | 54.32 | 394.12 |
| 13+ | | 2 000 | 2 000 | 16 000 | 1–1 000 | 220 000 | 101.97 | 71.85 | 694.32 |
| 14+ | | 2 500 | 2 500 | 20 000 | 1–1 000 | 275 000 | 107.93 | 96.71 | 1 030.35 |
| 15+ | | 3 000 | 3 000 | 24 000 | 1–1 000 | 330 000 | 133.85 | 102.93 | 1 533.50 |
| 16+ | Transportation | 500 | 500 | 10 000 | 1–100 | 15 000 | 16.44 | 11.43 | 84.04 |
| 17+ | | 750 | 750 | 15 000 | 1–100 | 22 500 | 28.30 | 18.12 | 176.55 |
| 18+ | | 1 000 | 1 000 | 20 000 | 1–100 | 30 000 | 51.01 | 31.31 | 306.97 |
| 19+ | | 1 250 | 1 250 | 25 000 | 1–100 | 37 500 | 71.61 | 38.96 | 476.57 |
| 20+ | | 1 500 | 1 500 | 30 000 | 1–100 | 45 000 | 68.09 | 41.03 | 727.38 |

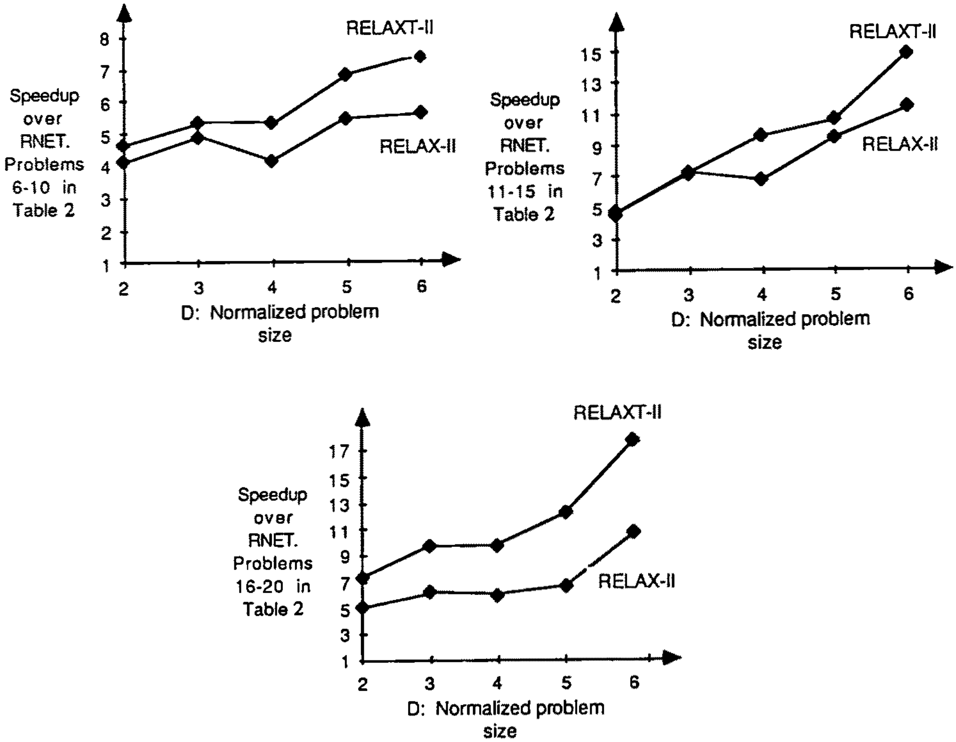



Fig. 5. Speedup factor of RELAX-II and RELAXT-II over RNET for the transportation problems of table 2. The normalized dimension $D$ gives the number of nodes $\mathcal{N}$ and arcs $\mathcal{A}$ as follows:

$$|\mathcal{N}| = 1000 * D,\ |\mathcal{A}| = 4000 * D \text{ for problems } 6-15$$
$$|\mathcal{N}| = 500 * D,\ |\mathcal{A}| = 5000 * D \text{ for problems } 16-20.$$

To corroborate the results of table 2, the random seed number of NETGEN was changed, and additional problems were solved using some of the problem data of the table. The results were qualitatively similar to those of table 2. We also solved a set of transhipment problems of increasing size generated by our random problem generator called RANET. The comparison between RELAX-II, RELAXT-II and RNET is given in fig. 6. More experimentation and/or analysis is needed to establish conclusively the computational complexity implications of these experiments.

## 8. Conclusions

Relaxation methods adapt nonlinear programming ideas to solve linear network flow problems. They are much faster than classical methods on standard benchmark problems and a broad range of randomly generated problems. They are also better

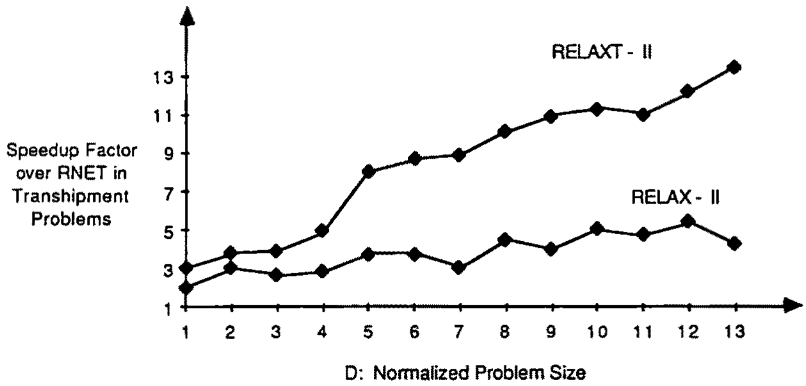



Fig. 6. Speedup factor of RELAX-II and RELAXT-II over RNET in lightly capacitated transhipment problems generated by our own random problem generator RANET. Each node is a transhipment node, and it is either a source or a sink. The normalized problem size $D$ gives the number of nodes and arcs as follows

$$|\mathcal{N}| = 200 * D, \ |\mathcal{A}| = 3000 * D.$$

The node supplies and demands were drawn from the interval $[-1000, 1000]$ according to a uniform distribution. The arc costs were drawn from the interval $[1, 100]$ according to a uniform distribution. The arc capacities were drawn from the interval $[500, 3000]$ according to a uniform distribution.

suited for post optimization analysis than primal-simplex. For example, suppose a problem is solved, and then is modified by changing a few arc capacities and/or node supplies. To solve the modified problem by the relaxation method, we use as starting node prices the prices obtained from the earlier solution, and we change the arc flows that violate the new capacity constraints to their new capacity bounds. Typically, this starting solution is close to optimal and solution of the modified problem is extremely fast. By contrast, to solve the modified problem using primal-simplex, one must provide a starting basis. The basis obtained from the earlier solution will typically not be a basis for the modified problem. As a result, a new starting basis has to be constructed, and there are no simple ways to choose this basis to be nearly optimal.

The main disadvantage of relaxation methods relative to primal-simplex is that they require more computer memory. However, technological trends are such that this disadvantage should become less significant in the future.

Our computational results provided some indication that relaxation has a superior average computational complexity over primal-simplex. Additional experimentation with large problems and/or analysis are needed to provide an answer to this important question.

The relaxation approach applies to a broad range of problems beyond the class considered in this paper (see [10--13]), including general linear programming problems. It also lends itself to distributed or parallel computation (see [10,13–16]).

The relaxation codes RELAX-II and RELAXT-II together with other support programs, including a reoptimization and sensitivity analysis capacity, are in the public domain with no restrictions, and can be obtained from the authors at no cost on IBM-PC or Macintosh diskette.

## References

[1] D.P. Bertsekas, A unified framework for minimum cost network flow problems, LIDS Report LIDS-P-1245-A, M.I.T. (1982); also Math. Progr. 32(1985)125.

[2] D.P. Bertsekas and P. Tseng, Relaxation methods for minimum cost – ordinary and generalized network flow problems, LIDS Report LIDS-P-1462, M.I.T. (1985); also Oper. Res. Journal, to appear.

[3] L.R. Ford, Jr. and D.R. Fulkerson, *Flows in Networks* (Princeton University Press, New Jersey, 1962).

[4] D.P. Bertsekas, A new algorithm for the assignment problem, Math. Progr. 21(1982)152.

[5] H.A. Aashtiani and T.L. Magnanti, Implementing primal-dual network flow algorithms, Oper. Res. Center Report 055-76, M.I.T. (1976).

[6] T. Magnanti, Optimization for sparse systems, in: *Sparse Matrix Computations*, ed. J.R. Bunch and D.J. Rose (Academic Press, New York, 1976) pp. 147 – 176.

[7] M.D. Grigoriadis and T. Hsu, The Rutgers minimum cost network flow subroutines (RNET documentation), Dept. of Computer Science, Rutgers University (1980).

[8] J. Kennington and R. Helgason, *Algorithms for Network Programming* (Wiley, New York, 1980).

[9] D. Klingman, A. Napier and J. Stutz, NETGEN – A program for generating large scale (un)capacitated assignment, transportation and minimum cost flow network problems Management Science 20(1974)814.

[10] D.P. Bertsekas, P. Hosein and P. Tseng, Relaxation methods for network flow problems with convex arc costs, SIAM J. Control and Optimization 25(1987).

[11] P. Tseng, Relaxation methods for monotropic programs, Ph.D. Thesis, M.I.T. (1986).

[12] P. Tseng and D.P. Bertsekas, Relaxation methods for linear programs, LIDS Report LIDS-P-1553, M.I.T. (1986); also Math. of Oper. Res. 12(1987).

[13] P. Tseng and D.P. Bertsekas, Relaxation methods for problems with strictly convex separable costs and linear constraints, LIDS Report LIDS-P-1567, M.I.T. (1986); also Math. Progr. 38(1987).

[14] D.P. Bertsekas, Distributed relaxation methods for linear network flow problems, *Proc. 25th IEEE Conf. on Decision and Control*, Athens, Greece (1986).

[15] D.P. Bertsekas and D. El Baz, Distributed asynchronous relaxation methods for convex network flow problems, LIDS Report LIDS-P-1417, M.I.T. (1984); also SIAM J. Control and Optimization 25(1987)74.

[16] D.P. Bertsekas and J. Eckstein, Distributed asynchronous relaxation methods for linear network flow problems, *Proc. of IFAC '87*, Munich, Germany (1987) (Pergamon Press, Oxford).

THE BASIC ALGORITHM

```
/* Read in problem data. */
nn := number of nodes in network
na := number of arcs in network
/* The nodes are numbered from 1 to nn and the arcs from 1 to na.*/
for arc := 1 to na do
    cost(arc) := cost of arc
    upbd(arc) := flow upper bound of arc
    head(arc) := head node of arc
    tail(arc) := tail node of arc
end do
for node := 1 to nn do
    dfct(node) := extraneous flow supply out of node
end do

/* Initialize dual prices to 0 and then assign flow to arcs to satisfy complementary slackness. */
for arc := 1 to na do
    rdcost(arc) := cost(arc)
    if rdcost(arc) ≥ 0 then
        flow(arc) := 0
    else
        flow(arc) := upbd(arc)
        dfct(head(arc)) := dfct(head(arc)) + upbd(arc)
        dfct(tail(arc)) := dfct(tail(arc)) - upbd(arc)
end do

/* Start relaxation iterations. */
while dfct(i) ≠ 0 for some i do
    for node := 1 to nn do
        if dfct(node) > 0 then
            pred(node) := 0
            labelset := {node}
            scanset := {∅}
            augnode := 0
            ascent := false
```

```
        while augnode = 0 and not ascent do
            Choose a node1 ∈ labelset \ scanset
            scanset := scanset ∪ {node1}
            /* Start scanning step. */
            scanning(node1,augnode)
            /* Check if scanset corresponds to a dual ascent direction. */
            if

                   Σ        dfct(node1)   >       Σ          flow(arc)   +         Σ          upbd(arc)-flow(arc)
          node1 ∈ scanset                 rdcost(arc) = 0                 rdcost(arc) = 0
                                          head(arc) ∈ scanset             head(arc) ∉ scanset
                                          tail(arc) ∉ scanset             tail(arc) ∈ scanset

            then ascent := true
        end do
        if ascent then
            doascent
        else
            augflow(augnode,node)
    end do
end do


procedure scanning(node1,augnode)
/* This procedure performs a scanning step at node1. */
    for all arc such that head(arc) = node1 do
        if rdcost(arc) = 0 and flow(arc) > 0 then
            node2 := tail(arc)
            if node2 ∉ labelset then
                pred(node2) := arc
                labelset := labelset ∪ {node2}
                if dfct(node2) < 0 then augnode := node2
    end do
    for all arc such that tail(arc) = node1 do
        if rdcost(arc) = 0 and flow(arc) < upbd(arc) then
```

```
            node2 := head(arc)
            if node2 ∉ labelset then
                pred(node2) := -arc
                labelset := labelset ∪ {node2}
                if dfct(node2) < 0 then augnode := node2
    end do
end;

procedure doascent
/* This procedure performs dual ascent by line-minimization and updates the flow accordingly to satisfy
complementary slackness. */
    while
                 Σ dfct(node1)    >      Σ       flow(arc)        +       Σ       upbd(arc)-flow(arc)
        node1 ∈ scanset               rdcost(arc) = 0                  rdcost(arc) = 0
                                    head(arc) ∈ scanset              head(arc) ∉ scanset
                                    tail(arc) ∉ scanset              tail(arc) ∈ scanset
    do
    /* Compute the stepsize to the next breakpoint in the dual cost and decrease the price of all nodes in
    scanset by the stepsize. Adjust the arc flow accordingly to maintain complementary slackness. */
        pricechange := very large positive number
        for all arc such that head(arc) ∈ scanset and tail(arc) ∉ scanset do
            if rdcost(arc) = 0 then
                dfct(head(arc)) := dfct(head(arc)) - flow(arc)
                dfct(tail(arc)) := dfct(tail(arc)) + flow(arc)
                flow(arc) := 0
            if 0 < -rdcost(arc) < pricechange then pricechange := -rdcost(arc)
        end do
        for all arc such that head(arc) ∉ scanset and tail(arc) ∈ scanset do
            if rdcost(arc) = 0 then
                dfct(head(arc)) := dfct(head(arc)) + (upbd(arc)- flow(arc))
                dfct(tail(arc)) := dfct(tail(arc)) - (upbd(arc)- flow(arc))
                flow(arc) := upbd(arc)
            if 0 < rdcost(arc) < pricechange then pricechange := rdcost(arc)
        end do
        for all arc such that head(arc) ∈ scanset and tail(arc) ∉ scanset do
```

```
            rdcost(arc) := rdcost(arc) + pricechange
        end do
        for all arc such that head(arc) ∉ scanset and tail(arc) ∈ scanset do
            rdcost(arc) := rdcost(arc) – pricechange
        end do
    end do
end;

procedure augflow(augnode,node)
/* This procedure adjusts the flow on arcs to decrease the total deficit, while maintaining complement
slackness. */
    flowchange := min{ dfct(node), -dfct(augnode) }
    node1 := augnode
    while node1 ≠ node do
        arc := pred(node1)
        if arc > 0 then
            flowchange := min{ flowchange, flow(arc) }
            node1 := head(arc)
        else
            flowchange := min{ flowchange, upbd(-arc)–flow(-arc) }
            node1 := tail(-arc)
    end do
    dfct(node) := dfct(node) – flowchange
    dfct(augnode) := dfct(augnode) + flowchange
    node1 := augnode
    while node1 ≠ node do
        arc := pred(node1)
        if arc > 0 then
            flow(arc) := flow(arc) – flowchange
            node1 := head(arc)
        else
            flow(-arc) := flow(-arc) + flowchange
            node1 := tail(-arc)
    end do
end;
```

# Appendix

```
C     ******* SAMPLE CALLING PROGRAM FOR SUBROUTINE RELAXT *******
C     ***     (MINIMUM COST NETWORK FLOW PROBLEM)                ***
C     ***                                                        ***
C     ***     THE PROGRAM IS BASED ON THE PAPER                  ***
C     ***     D.P. BERTSEKAS, P.TSENG "THE RELAX CODES FOR       ***
C     ***       LINEAR MINIMUM COST NETWORK FLOW PROBLEMS",      ***
C     ***      ANNALS OF OPERATIONS RESEARCH, THIS VOLUME        ***
C     ***                                                        ***
C     ***     ALL THE SUBROUTINES ARE WRITTEN IN STANDARD        ***
C     ***       FORTRAN77.                                       ***
C     ***                                                        ***
C     ***     QUESTIONS AND COMMENTS SHOULD BE DIRECTED TO       ***
C     ***     DIMITRI BERTSEKAS AND PAUL TSENG                   ***
C     ***     DEPARTMENT OF ELECTRICAL ENGINEERING &             ***
C     ***       COMPUTER SCIENCE                                 ***
C     ***     LABORATORY FOR INFORMATION AND DECISION SYSTEMS    ***
C     ***     M.I.T.  CAMBRIDGE, MASSACHUSETTS, 02139, U.S.A.    ***
C     ************************************************************
C
C     THIS PROGRAM WILL READ AND SOLVE A PROBLEM FILE CREATED VIA
C     THE RANDOM PROBLEM GENERATOR NETGEN OR ANY GENERATOR THAT
C     USES THE NETGEN FORMAT.
C
C     *********************************************
C
      DOUBLE PRECISION TCOST
      INTEGER C(70000),X(70000),U(70000),RC(70000),B(6000)
      INTEGER CAP(70000),STARTN(70000),ENDN(70000)
      INTEGER I1(6000),I2(6000),I3(6000),I4(70000),I5(6000)
      INTEGER I6(70000),I7(70000)
      LOGICAL L1(6000),L2(6000),TEST,REPEAT
      COMMON /ARRAYS/STARTN/ARRAYE/ENDN/ARRAYU/U/ARRAYX/X
     */ARRAY9/RC/ARRAYB/B/BLK1/I1/BLK2/I2/BLK3/I3/BLK4/I4/BLK5/I5/BLK6
     */I6/BLK7/I7/BLK8/L1/BLK9/L2/L/N,NA,LARGE
      COMMON/BLKR/REPEAT/ARRAYC/C/BLKCAP/CAP
      COMMON /BLK10/TFSTOU/BLK11/TNXTOU/BLK12/TFSTIN/BLK13/TNXTIN
      DIMENSION TFSTOU(6000),TNXTOU(70000),TFSTIN(6000),TNXTIN(70000)
C
      READ(5,*)N,IA
      READ(5,*)(STARTN(I),I=1,IA)
      READ(5,*)(ENDN(I),I=1,IA)
      READ(5,*)(C(I),I=1,IA)
      READ(5,*)(U(I),I=1,IA)
      DO 10 I=1,N
   10 B(I)=0
      M=0
      DO 20 I=1,IA
        IF (STARTN(I).EQ.N+1) THEN
          B(ENDN(I))=-U(I)
        ELSE IF (ENDN(I).EQ.N+2) THEN
          B(STARTN(I))=U(I)
        ELSE
          M=M+1
          C(M)=C(I)
```

```
          U(M)=U(I)
          STARTN(M)=STARTN(I)
          ENDN(M)=ENDN(I)
        END IF
   20 CONTINUE
      NA=M
      LARGE=20000000
      REPEAT=.FALSE.
      DO 30 I=1,NA
   30   CAP(I)=U(I)
      CALL INIDAT
C     ***** Set initial dual prices to zero *****
      DO 40 I=1,NA
   40   RC(I)=C(I)
      CALL RELAXT
C     ***** Display previous optimal cost *****
      IF (REPEAT) WRITE(6,50)TCOST
   50 FORMAT(' ','PREVIOUS OPTIMAL COST=',F14.2)
      TCOST=DFLOAT(0)
      DO 60 I=1,NA
   60   TCOST=TCOST+DFLOAT(X(I)*C(I))
      WRITE(6,70) TCOST
   70 FORMAT(' ','OPTIMAL COST         =',F14.2)
      END


C
C     ***************************************************
C
      SUBROUTINE INIDAT
C     ***** This subroutine uses the data arrays STARTN and ENDN
C     to construct auxiliary data arrays FOU, NXTOU, FIN, and
C     NXTIN that are required by RELAXT. In this subroutine we
C     arbitrarily order the arcs leaving each node and store
C     this information in FOU and NXTOU.  Similarly, we arbitra-
C     rilly order the arcs entering each node and store this
C     information in FIN and NXTIN.  At the completion of the
C     construction, we have that
C
C         FOU(I)     = First arc leaving node I.
C         NXTOU(J)   = Next arc leaving the head node of arc J.
C         FIN(I)     = First arc entering node I.
C         NXTIN(J)   = Next arc entering the tail node of arc J.
C
      COMMON /ARRAYS/STARTN/ARRAYE/ENDN/BLK1/TEMPIN/BLK2/TEMPOU
     */BLK3/FOU/BLK4/NXTOU/BLK5/FIN/BLK6/NXTIN
     */L/N,NA
      INTEGER STARTN(1),ENDN(1),TEMPIN(1),TEMPOU(1),FOU(1)
      INTEGER NXTOU(1),FIN(1),NXTIN(1)
      LOGICAL L
C     ******* construct data structure required by RELAXT ***********
      DO 10 I=1,N
        FIN(I)=0
        FOU(I)=0
        TEMPIN(I)=0
   10   TEMPOU(I)=0
      DO 20 I=1,NA
        NXTIN(I)=0
        NXTOU(I)=0
```

```
      I1=STARTN(I)
      I2=ENDN(I)
      IF (FOU(I1).NE.0) THEN
        NXTOU(TEMPOU(I1))=I
      ELSE
        FOU(I1)=I
      END IF
      TEMPOU(I1)=I
      IF (FIN(I2).NE.0) THEN
        NXTIN(TEMPIN(I2))=I
      ELSE
        FIN(I2)=I
      END IF
20    TEMPIN(I2)=I
      RETURN
      END


      SUBROUTINE RELAXT
C ***********************************************************************
C
C                              SUBROUTINE RELAXT
C                              RELEASE  APR. 1988
C
C ***********************************************************************
C
C     This subroutine solves the minimum (linear) cost ordinary
C     network flow problem.
C     The routine implements the relaxation method of
C     Bertsekas, D. P., "A Unified Framework for Primal-Dual Methods .."
C              Math. Programming, Vol. 32, 1985, pp. 125-145
C     Bertsekas, D. P., & Tseng, P., "Relaxation Methods for Minimum .."
C              Operations Research J., 1987 (to appear)
C     Bertsekas, D. P., & Tseng, P., "The RELAX Codes for Linear Minimum
C              Cost Network Flow Problems", ANNALS OF OPERATIONS
C              RESEARCH, THIS VOLUME
C     The routine was written by Dimitri Bertsekas and Paul Tseng with
C     contributions by Jonathan Eckstein.
C
C     This code is in the public domain and can be used for any
C     purpose. It can be distributed freely.
C     Users are requested to acknowledge the authorship
C     of the code, and the relaxation algorithm. No modifications
C     should be made to this code other than the minimal necessary
C     to make it compatible with the FORTRAN compilers of specific
C     machines. When reporting computational results please be sure
C     to describe the memory limitations of your machine. Generally
C     RELAXT requires more memory than primal simplex codes and may
C     penalized severely by limited machine memory.
C
C     The difference between this routine and the similar code RELAX is
C     that it maintains a data structure that gives all the balanced
C     arcs in the network.  This structure is called the "tree" for
C     historical reasons, even though it describes a subnetwork that
C     will generally be neither acyclic nor connected.  Also, the tree
C     may contain some arcs that are not balanced:  it turns out to be
C     cheaper to purge arcs that have become unbalanced only when their
C     end nodes are being scanned, as opposed to always maintaining an
C     exact set of balanced arcs.
```

```
C
C   ****************************************************************
C
C      The user must supply the following inputs to the subroutine:
C      All data should be in INTEGER*4. To run in limited memory systems
C       the arrays STARTN, ENDN, NXTIN, NXTOU, SAVE, FIN, FOU, LABEL,
C       PRDCSR may be declared as INTEGER*2.
C           N (the number of nodes)
C           NA (the number of arcs)
C           LARGE (a very large positive integer to represent infinity.
C            All problem data should be less than LARGE in magnitude,
C            and LARGE should be less than,say, 1/4 the largest INTEGER*4
C            of the machine used. This will guard primarily against
C            overflow in uncapacitated problems where the arc capacities
C            are taken finite but very large.)
C           STARTN(NA) (the head node array)
C           ENDN(NA) (the tail node array)
C           RC(NA) (the reduced cost array)
C           X(NA) (the arc flow array)
C           U(NA) (the arc flow capacity array)
C           DFCT(N) (the deficit array)
C           FOU(N) (the first arc out array)
C           FIN(N) (the first arc in array)
C           NXTOU(NA) (the next arc out array)
C           NXTIN(NA) (the next arc in array)
C
C      This subroutine places the optimal flow in the array X
C      and the corresponding reduced cost vector in the array RC.
C
C      ****************************************************************
C
       IMPLICIT INTEGER (A-Z)
       LOGICAL REPEAT,FEASBL,QUIT,SCAN,SWITCH,MARK,POSIT,PCHANG
       COMMON/ARRAYS/STARTN/ARRAYE/ENDN/ARRAYU/U/ARRAYX/X/ARRAY9/RC
      */ARRAYB/DFCT/BLK1/LABEL/BLK2/PRDCSR/BLK3/FOU/BLK4/NXTOU/BLK5/FIN
      */BLK6/NXTIN/BLK7/SAVE/BLK8/SCAN/BLK9/MARK/L/N,NA,LARGE
      */BLKR/REPEAT
       COMMON /BLK10/TFSTOU/BLK11/TNXTOU/BLK12/TFSTIN/BLK13/TNXTIN
C
C      Each common block contains just one array, so the arrays in RELAXT
C      can be dimensioned to 1 element and take their dimension from the
C      main calling routine.With this trick RELAXT need not be recompiled
C      if the problem dimension changes. If your FORTRAN does not support
C      this feature change the dimensions below to be the same as the
C      ones declared in your main calling program.
C
       DIMENSION TFSTOU(1),TNXTOU(1),TFSTIN(1),TNXTIN(1)
       DIMENSION STARTN(1),ENDN(1),U(1),X(1),RC(1),DFCT(1)
       DIMENSION LABEL(1),PRDCSR(1),SCAN(1),FOU(1),NXTOU(1)
       DIMENSION FIN(1),NXTIN(1),SAVE(1),MARK(1)
C
C      DDPOS and DDNEG are arrays that give the directional derivatives
C      for all positive and negative single-node price changes.  These
C      are used only in the initial phase of the algorithm, before the
C      "tree" datastructure comes into play.  Therefore, they are
C      equivalenced to TFSTOU and TFSTIN, which are the same size (number
C      of nodes) and are only used after the tree comes into use.
C
       DIMENSION   DDPOS(1),DDNEG(1)
       EQUIVALENCE (DDPOS(1),TFSTOU(1)),(DDNEG(1),TFSTIN(1))
```

```
C
C
C * reduce arc capacity as much as possible w/out changing the problem *
C * If this is a sensitivity run via routine SENSTV skip the
C   initialization *
C
      IF (REPEAT) GO TO 190
      DO 50 NODE=1,N
C
C     Note that we also set up the initial DDPOS and DDNEG for each node
C     (this is not necessary in RELAX).
C
        DDPOS(NODE)=DFCT(NODE)
        DDNEG(NODE)=-DFCT(NODE)
        SCAPOU=0
        ARC=FOU(NODE)
  10    IF (ARC.GT.0) THEN
          SCAPOU=MINO(LARGE,SCAPOU+U(ARC))
          ARC=NXTOU(ARC)
          GO TO 10
        END IF
        CAPOUT=MINO(LARGE,SCAPOU+DFCT(NODE))
        IF (CAPOUT.LT.0) THEN
C
C  ** PROBLEM IS INFEASIBLE - EXIT
C
       WRITE(6,*)'EXIT DURING INITIALIZATION'
       WRITE(6,*)'EXOGENOUS FLOW INTO NODE',NODE,' EXCEEDS OUT CAPACITY'
       CALL PRFLOW(NODE)
       GO TO 640
       END IF
C
        SCAPIN=0
        ARC=FIN(NODE)
  20    IF (ARC.GT.0) THEN
          U(ARC)=MINO(U(ARC),CAPOUT)
          SCAPIN=MINO(LARGE,SCAPIN+U(ARC))
          ARC=NXTIN(ARC)
          GO TO 20
        END IF
  30    CAPIN=MINO(LARGE,SCAPIN-DFCT(NODE))
        IF (CAPIN.LT.0) THEN
C
C  *** PROBLEM IS INFEASIBLE - EXIT
C
      WRITE(6,*)'EXIT DURING INITIALIZATION'
      WRITE(6,*)'EXOGENOUS FLOW OUT OF NODE',NODE,
     *          ' EXCEEDS IN CAPACITY'
      CALL PRFLOW(NODE)
      GO TO 640
      END IF
C
        ARC=FOU(NODE)
  40    IF (ARC.GT.0) THEN
          U(ARC)=MINO(U(ARC),CAPIN)
          ARC=NXTOU(ARC)
          GO TO 40
        END IF
  50    CONTINUE
C
```

```
C     ******* initialize the arc flows and the nodal deficits ********
C     *** note that U(ARC) is redefined as the residual capacity of ARC
C
C     Now compute the directional derivatives for each coordinate
C     exactly.
C     As well as computing the actual defecits.  U(ARC) is the residual
C     capacity on ARC, and X(ARC) is the flow.These always add up to the
C     total capacity.
C
      DO 60 ARC=1,NA
         X(ARC) = 0
         IF (RC(ARC) .LE. 0) THEN
            T  = U(ARC)
            T1 = STARTN(ARC)
            T2 = ENDN(ARC)
            DDPOS(T1) = DDPOS(T1) + T
            DDNEG(T2) = DDNEG(T2) + T
            IF (RC(ARC) .LT. 0) THEN
               X(ARC) = T
               U(ARC) = 0
               DFCT(T1) = DFCT(T1) + T
               DFCT(T2) = DFCT(T2) - T
               DDNEG(T1) = DDNEG(T1) - T
               DDPOS(T2) = DDPOS(T2) - T
            END IF
         END IF
   60 CONTINUE
C
C     Adaptive strategy:  the number of strictly single-node iteration
C     passes attempted is a function of the average density of the
C     network.
C
      IF (NA.GT.N*10) THEN
        NPASS=2
      ELSE
        NPASS=3
      END IF
C
C     We now do 2 or 3 passes through all the nodes. This is the initial
C     phase:if a single node iteration is not possible, we just go on to
C     the next node.
C
      DO 180 PASSES = 1,NPASS
      DO 170 NODE=1,N
       IF (DFCT(NODE) .NE. 0) THEN
C       Price rise or price drop? (Note: it is impossible to have both.)
C
        IF (DDPOS(NODE) .LE. 0) THEN

C     Price rise.  Loop over breakpoints in +Price(NODE) direction.
C     On outgoing arcs, tension will rise and reduced cost will fall
C        -- so, next break comes at smallest positive reduced cost.
C     On incoming arcs, tension will fall and reduced cost will rise
C        -- so, next break comes at smallest negative reduced cost.
C
           DELPRC = LARGE
           ARC = FOU(NODE)
   70      IF (ARC .GT. 0) THEN
              TRC = RC(ARC)
              IF ((TRC .GT. 0) .AND. (TRC .LT. DELPRC)) THEN
```

```
            DELPRC = TRC
         END IF
         ARC = NXTOU(ARC)
         GOTO 70
      END IF
      ARC = FIN(NODE)
 80   IF (ARC .GT. 0) THEN
         TRC = RC(ARC)
         IF ((TRC .LT. 0) .AND. (-TRC .LT. DELPRC)) THEN
            DELPRC = -TRC
         END IF
         ARC = NXTIN(ARC)
         GOTO 80
      END IF
C     If no breakpoints left and ascent still possible, the problem
C     is infeasible.
C
      IF (DELPRC .GE. LARGE) THEN
         IF (DDPOS(NODE) .EQ. 0) GOTO 170
         GOTO 640
      ENDIF
C     We have an actual breakpoint.Increase price by that quantity.
C     First check the effect on all outbound arcs,which will have a
C     tension increase and reduced cost drop.
C
 90   NXTBRK = LARGE
      ARC = FOU(NODE)
 100  IF (ARC .GT. 0) THEN
         TRC = RC(ARC)
         IF (TRC .EQ. 0) THEN
            T1 = ENDN(ARC)
            T  = U(ARC)
            IF (T .GT. 0) THEN
               DFCT(NODE) = DFCT(NODE) + T
               DFCT(T1) = DFCT(T1) - T
               X(ARC) = T
               U(ARC) = 0
            ELSE
               T = X(ARC)
            END IF
            DDNEG(NODE) = DDNEG(NODE) - T
            DDPOS(T1) = DDPOS(T1) - T
         END IF
C        For all outgoing arcs tension rises,and reduced cost drops
         TRC = TRC - DELPRC
         IF ((TRC .GT. 0) .AND. (TRC .LT. NXTBRK)) THEN
            NXTBRK = TRC
         ELSE IF (TRC .EQ. 0) THEN
C           Arc goes from inactive to balanced. Just change tension
C           increase derivatives, and check for status change at
C           other end.
C
            DDPOS(NODE) = DDPOS(NODE) + U(ARC)
            DDNEG(ENDN(ARC)) = DDNEG(ENDN(ARC)) + U(ARC)
         END IF
         RC(ARC) = TRC
         ARC =  NXTOU(ARC)
         GOTO 100
      END IF
C     Time to check the incoming arcs into the node.
```

```
C           These arcs will have an tension decrease and a reduced cost
C           rise.
C
            ARC = FIN(NODE)
  110       IF (ARC .GT. 0) THEN
               TRC = RC(ARC)
               IF (TRC .EQ. 0) THEN
                  T1 = STARTN(ARC)
                  T  = X(ARC)
                  IF (T .GT. 0) THEN
                     DFCT(NODE) = DFCT(NODE) + T
                     DFCT(T1) = DFCT(T1) - T
                     U(ARC) = T
                     X(ARC) = 0
                  ELSE
                     T = U(ARC)
                  END IF
                  DDPOS(T1) = DDPOS(T1) - T
                  DDNEG(NODE) = DDNEG(NODE) - T
               END IF
C              Note the reduced cost rise for every arc.
               TRC = TRC + DELPRC
               IF ((TRC .LT. 0) .AND. (-TRC .LT. NXTBRK)) THEN
                  NXTBRK = -TRC
               ELSE IF (TRC .EQ. 0) THEN
C                 Now check for movement from active to balanced.
C                 If so, tension decrease derivatives increase.
                  DDNEG(STARTN(ARC)) = DDNEG(STARTN(ARC)) + X(ARC)
                  DDPOS(NODE) = DDPOS(NODE) + X(ARC)
               END IF
               RC(ARC) = TRC
               ARC = NXTIN(ARC)
               GOTO 110
            END IF
C           We are now done with the iteration.  If the current direction
C           is still a (degenerate) descent direction, push onward.
C
            IF ((DDPOS(NODE) .LE. 0) .AND. (NXTBRK .LT. LARGE)) THEN
               DELPRC = NXTBRK
               GOTO 90
            END IF
C       Now comes the code for a price decrease at NODE.
C       On outgoing arcs, tension will drop and reduced cost will increase
C          -- so, next break comes at smallest negative reduced cost.
C       On incoming arcs, tension will increase and reduced cost will fall
C          -- so, next break comes at smallest positive reduced cost.
C
        ELSE IF (DDNEG(NODE) .LE. 0) THEN
           DELPRC = LARGE
           ARC = FOU(NODE)
  120      IF (ARC .GT. 0) THEN
              TRC = RC(ARC)
              IF ((TRC .LT. 0) .AND. (-TRC .LT. DELPRC)) THEN
                 DELPRC = -TRC
              ENDIF
              ARC = NXTOU(ARC)
              GOTO 120
           ENDIF
           ARC = FIN(NODE)
  130      IF (ARC .GT. 0) THEN
```

```
            TRC = RC(ARC)
            IF ((TRC .GT. 0) .AND. (TRC .LT. DELPRC)) THEN
               DELPRC = TRC
            END IF
            ARC = NXTIN(ARC)
            GOTO 130
         END IF
C        If there is no breakpoint, the problem is infeasible,
C        unless we are making a degenerate step.
C
         IF (DELPRC .EQ. LARGE) THEN
            IF (DDNEG(NODE) .EQ. 0) GOTO 170
            GOTO 640
         END IF
C        Now we make the step to the next breakpoint.  We start with the
C        outbound arcs.  These have a tension decrease and reduced cost
C        rise.  Therefore, the possible transitions are from balanced to
C        inactive or active to balanced.
C
  140    NXTBRK = LARGE
         ARC = FOU(NODE)
  150    IF (ARC .GT. 0) THEN
            TRC = RC(ARC)
            IF (TRC .EQ. 0) THEN
               T1 = ENDN(ARC)
               T  = X(ARC)
               IF (T .GT. 0) THEN
                  DFCT(NODE) = DFCT(NODE) - T
                  DFCT(T1) = DFCT(T1) + T
                  U(ARC) = T
                  X(ARC) = 0
               ELSE
                  T = U(ARC)
               END IF
               DDPOS(NODE) = DDPOS(NODE) - T
               DDNEG(T1) = DDNEG(T1) - T
            END IF
C           Log the reduced cost rise for all arcs.
            TRC = TRC + DELPRC
            IF ((TRC .LT. 0) .AND. (-TRC .LT. NXTBRK)) THEN
               NXTBRK = -TRC
            ELSE IF (TRC .EQ. 0) THEN
C              Active to balanced.  Tension decrease derivs go up.
               DDNEG(NODE) = DDNEG(NODE) + X(ARC)
               DDPOS(ENDN(ARC)) = DDPOS(ENDN(ARC)) + X(ARC)
            END IF
            RC(ARC) = TRC
            ARC = NXTOU(ARC)
            GOTO 150
         END IF
C        Now do the incoming arcs.  These have a tension increase and
C        therefore a reduced cost drop.  The possible transitions are
C        from inactive to balanced and from balanced to active.
C
         ARC = FIN(NODE)
  160    IF (ARC .GT. 0) THEN
            TRC = RC(ARC)
            IF (TRC .EQ. 0) THEN
               T1 = STARTN(ARC)
               T  = U(ARC)
```

```
              IF (T .GT. 0) THEN
                 DFCT(NODE) = DFCT(NODE) - T
                 DFCT(T1) = DFCT(T1) + T
                 X(ARC) = T
                 U(ARC) = 0
              ELSE
                 T = X(ARC)
              END IF
              DDNEG(T1) = DDNEG(T1) - T
              DDPOS(NODE) = DDPOS(NODE) - T
           END IF
           TRC = TRC - DELPRC
           IF ((TRC .GT. 0) .AND. (TRC .LT. NXTBRK)) THEN
              NXTBRK = TRC
           ELSE IF (TRC .EQ. 0) THEN
              DDPOS(STARTN(ARC)) = DDPOS(STARTN(ARC)) + U(ARC)
              DDNEG(NODE) = DDNEG(NODE) + U(ARC)
           END IF
           RC(ARC) = TRC
           ARC = NXTIN(ARC)
           GOTO 160
        END IF
C       OK.  Movement is done.  Is this direction still a (degenerate)
C       descent direction.  If so, keep going.
C
        IF ((DDNEG(NODE) .LE. 0) .AND. (NXTBRK .LT. LARGE)) THEN
           DELPRC = NXTBRK
           GOTO 140
        END IF
       END IF
      END IF
  170 CONTINUE
  180 CONTINUE
C     ******* initialize the tree ************************************
  190 DO 200 I=1,N
        TFSTOU(I)=0
  200   TFSTIN(I)=0
      DO 210 I=1,NA
        TNXTIN(I)=-1
        TNXTOU(I)=-1
        IF (RC(I).EQ.0) THEN
          TNXTOU(I)=TFSTOU(STARTN(I))
          TFSTOU(STARTN(I))=I
          TNXTIN(I)=TFSTIN(ENDN(I))
          TFSTIN(ENDN(I))=I
        END IF
  210 CONTINUE
C
C     *********** Initialize other variables ***********
C
      FEASBL=.TRUE.
      NDFCT=N
      NNONZ=0
      SWITCH=.FALSE.
      DO 220 I=1,N
        MARK(I)=.FALSE.
        SCAN(I)=.FALSE.
  220 CONTINUE
      NLABEL=0
C     ******* Set threshold for SWITCH *******************************
```

```
C     RELAXT uses an adaptive strategy for deciding whether to
C     continue the scanning process after a price change.
C     The threshold parameters tp and ts that control
C     this strategy are set in the next few lines.
C
      TP=10
      TS=INT(N/15)
C
C     **** start relaxation algorithm *************
C
  230 CONTINUE
C
        DO 630 NODE=1,N
          DEFCIT=DFCT(NODE)
          IF (DEFCIT.EQ.0) THEN
            GO TO 630
          ELSE
            POSIT = (DEFCIT .GT. 0)
            NNONZ=NNONZ+1
          END IF
C
C         ***** ATTEMPT A SINGLE NODE ITERATION FROM NODE ****
C
      IF (POSIT) THEN
C
C    ************* CASE OF NODE W/ POSITIVE DEFICIT ********
C
      PCHANG = .FALSE.
      INDEF=DEFCIT
      DELX=0
      NB=0
C
C     Check outgoing (probably) balanced arcs from NODE.
C
      ARC=TFSTOU(NODE)
  240 IF (ARC .GT. 0) THEN
         IF ((RC(ARC) .EQ. 0) .AND. (X(ARC) .GT. 0)) THEN
            DELX = DELX + X(ARC)
            NB = NB + 1
            SAVE(NB) = ARC
         ENDIF
         ARC = TNXTOU(ARC)
         GOTO 240
      END IF
C
C     Check incoming arcs now.
C
      ARC = TFSTIN(NODE)
  250 IF (ARC .GT. 0) THEN
         IF ((RC(ARC) .EQ. 0) .AND. (U(ARC) .GT. 0)) THEN
            DELX = DELX + U(ARC)
            NB = NB + 1
            SAVE(NB) = -ARC
         ENDIF
         ARC = TNXTIN(ARC)
         GOTO 250
      END IF
C
C   ***** end of initial node scan *******
C
```

```
  260  CONTINUE
C
C    ********* IF no price change is possible exit **********
C
      IF (DELX.GT.DEFCIT) THEN
        QUIT = (DEFCIT'.LT. INDEF)
        GO TO 330
      END IF
C
C     Now compute distance to next breakpoint.
C
      DELPRC = LARGE
      ARC = FOU(NODE)
  270 IF (ARC .GT. 0) THEN
         RDCOST = RC(ARC)
         IF ((RDCOST .LT. 0) .AND. (-RDCOST .LT. DELPRC)) THEN
            DELPRC = -RDCOST
         ENDIF
         ARC = NXTOU(ARC)
         GOTO 270
      END IF
      ARC = FIN(NODE)
  280 IF (ARC .GT. 0) THEN
         RDCOST = RC(ARC)
         IF ((RDCOST .GT. 0) .AND. (RDCOST .LT. DELPRC)) THEN
            DELPRC = RDCOST
         ENDIF
         ARC = NXTIN(ARC)
         GOTO 280
      END IF
C
C     ******* check if the problem is infeasible *******
C
      IF ((DELX.LT.DEFCIT).AND.(DELPRC.EQ.LARGE)) THEN
C       ***** The dual cost can be decreased without bound  *****
        GO TO 640
      END IF
C
C    **** SKIP FLOW ADJUSTEMT IF THERE IS NO FLOW TO MODIFY ***
C
      IF (DELX.EQ.0) GO TO 300
C
C     *****  Adjust the flow on balanced arcs incident of NODE to
C     maintain complementary slackness after the price change *****
C
      DO 290 J=1,NB
        ARC=SAVE(J)
        IF (ARC.GT.0) THEN
          NODE2=ENDN(ARC)
          T1=X(ARC)
          DFCT(NODE2)=DFCT(NODE2)+T1
          U(ARC)=U(ARC)+T1
          X(ARC)=0
        ELSE
          NARC=-ARC
          NODE2=STARTN(NARC)
          T1=U(NARC)
          DFCT(NODE2)=DFCT(NODE2)+T1
          X(NARC)=X(NARC)+T1
          U(NARC)=0
```

```
          END IF
  290    CONTINUE
        DEFCIT=DEFCIT-DELX
  300   IF (DELPRC.EQ.LARGE) THEN
          QUIT=.TRUE.
          GO TO 350
        END IF
C
C       ***** NODE corresponds to a dual ascent direction.  Decrease
C       the price of NODE by DELPRC and compute the stepsize to the
C       next breakpoint in the dual cost *****
C
        NB=0
        PCHANG = .TRUE.
        DP=DELPRC
        DELPRC=LARGE
        DELX=0
        ARC=FOU(NODE)
  310   IF (ARC.GT.0) THEN
          RDCOST=RC(ARC)+DP
          RC(ARC)=RDCOST
          IF (RDCOST.EQ.0) THEN
            NB=NB+1
            SAVE(NB)=ARC
            DELX=DELX+X(ARC)
          END IF
          IF ((RDCOST.LT.0).AND.(-RDCOST.LT.DELPRC)) DELPRC=-RDCOST
          ARC=NXTOU(ARC)
          GOTO 310
        END IF
        ARC=FIN(NODE)
  320   IF (ARC.GT.0) THEN
          RDCOST=RC(ARC)-DP
          RC(ARC)=RDCOST
          IF (RDCOST.EQ.0) THEN
            NB=NB+1
            SAVE(NB)=-ARC
            DELX=DELX+U(ARC)
          END IF
          IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
          ARC=NXTIN(ARC)
          GOTO 320
        END IF
C
C     ***** return to check if another price change is possible ******
C
        GO TO 260
C
C       ******* perform flow augmentation at NODE ****
C
  330   DO 340 J=1,NB
          ARC=SAVE(J)
          IF (ARC.GT.0) THEN
C           *** ARC is an outgoing arc from NODE ******************
            NODE2=ENDN(ARC)
            T1=DFCT(NODE2)
            IF (T1.LT.0) THEN
C             ***** Decrease the total deficit by decreasing flow of ARC
              QUIT=.TRUE.
              T2=X(ARC)
```

```
               DX=MIN0(DEFCIT,-T1,T2)
               DEFCIT=DEFCIT-DX
               DFCT(NODE2)=T1+DX
               X(ARC)=T2-DX
               U(ARC)=U(ARC)+DX
               IF (DEFCIT.EQ.0) GO TO 350
             END IF
           ELSE
C            *** -ARC is an incoming arc to NODE ********************
             NARC=-ARC
             NODE2=STARTN(NARC)
             T1=DFCT(NODE2)
             IF (T1.LT.0) THEN
C              ***** Decrease the total deficit by increasing flow of -ARC
               QUIT=.TRUE.
               T2=U(NARC)
               DX=MIN0(DEFCIT,-T1,T2)
               DEFCIT=DEFCIT-DX
               DFCT(NODE2)=T1+DX
               X(NARC)=X(NARC)+DX
               U(NARC)=T2-DX
               IF (DEFCIT.EQ.0) GO TO 350
             END IF
           END IF
  340   CONTINUE
  350   DFCT(NODE)=DEFCIT
C
C       Reconstruct the list of balanced arcs adjacent to this node.
C       First, the list at this node is now totally different.  Eat
C       the old lists of incoming and outgoing balanced arcs, and create
C       a whole new one. This way we get the in and out lists of balanced
C       arcs for NODE to be exactly correct.  For the adjacent nodes, we
C       add in all the newly balanced arcs, but do not bother getting rid
C       of formerly balanced ones (they will be purged the next time the
C       adjacent node is scanned).
C
        IF (PCHANG) THEN
           ARC = TFSTOU(NODE)
           TFSTOU(NODE) = 0
  360      IF (ARC .GT. 0) THEN
              NXTARC = TNXTOU(ARC)
              TNXTOU(ARC) = -1
              ARC = NXTARC
              GOTO 360
           END IF
           ARC = TFSTIN(NODE)
           TFSTIN(NODE) = 0
  370      IF (ARC .GT. 0) THEN
              NXTARC = TNXTIN(ARC)
              TNXTIN(ARC) = -1
              ARC = NXTARC
              GOTO 370
           END IF
C
C          *** Now add the currently balanced arcs to the list for this
C          *** node(which is now empty),and the appropriate adjacent ones
C
           DO 380 J=1,NB
              ARC = SAVE(J)
              IF (ARC.LE.0) ARC=-ARC
```

```
            IF (TNXTOU(ARC) .LT. 0) THEN
               TNXTOU(ARC) = TFSTOU(STARTN(ARC))
               TFSTOU(STARTN(ARC)) = ARC
            END IF
            IF (TNXTIN(ARC) .LT. 0) THEN
               TNXTIN(ARC) = TFSTIN(ENDN(ARC))
               TFSTIN(ENDN(ARC)) = ARC
            END IF
  380     CONTINUE
        END IF
C
C    *** end of single node iteration for a positive deficit node ***
C
      ELSE
C
C     ******* single node iteration for a negative deficit node ******
C
      PCHANG = .FALSE.
      DEFCIT=-DEFCIT
      INDEF=DEFCIT
      DELX=0
      NB=0
C
      ARC = TFSTIN(NODE)
  390 IF (ARC .GT. 0) THEN
         IF ((RC(ARC) .EQ. 0) .AND. (X(ARC) .GT. 0)) THEN
            DELX = DELX + X(ARC)
            NB = NB + 1
            SAVE(NB) = ARC
         ENDIF
         ARC = TNXTIN(ARC)
         GOTO 390
      END IF
      ARC=TFSTOU(NODE)
  400 IF (ARC .GT. 0) THEN
         IF ((RC(ARC) .EQ. 0) .AND. (U(ARC) .GT. 0)) THEN
            DELX = DELX + U(ARC)
            NB = NB + 1
            SAVE(NB) = -ARC
         ENDIF
         ARC = TNXTOU(ARC)
         GOTO 400
      END IF
C
  410  CONTINUE
      IF (DELX.GT.DEFCIT) THEN
        QUIT = (DEFCIT .LT. INDEF)
        GO TO 480
      END IF
C
C     Now compute distance to next breakpoint.
C
      DELPRC = LARGE
      ARC = FIN(NODE)
  420 IF (ARC .GT. 0) THEN
         RDCOST = RC(ARC)
         IF ((RDCOST .LT. 0) .AND. (-RDCOST .LT. DELPRC)) THEN
            DELPRC = -RDCOST
         ENDIF
         ARC = NXTIN(ARC)
```

```
            GOTO 420
         END IF
         ARC = FOU(NODE)
  430    IF (ARC .GT. 0) THEN
            RDCOST = RC(ARC)
            IF ((RDCOST .GT. 0) .AND. (RDCOST .LT. DELPRC)) THEN
               DELPRC = RDCOST
            ENDIF
            ARC = NXTOU(ARC)
            GOTO 430
         END IF
C        ******* check if problem is infeasible ************************
         IF ((DELX.LT.DEFCIT).AND.(DELPRC.EQ.LARGE)) THEN
           GO TO 640
         END IF
         IF (DELX.EQ.0) GO TO 450
C
C        ******* flow augmentation is possible *************************
         DO 440 J=1,NB
           ARC=SAVE(J)
           IF (ARC.GT.0) THEN
             NODE2=STARTN(ARC)
             T1=X(ARC)
             DFCT(NODE2)=DFCT(NODE2)-T1
             U(ARC)=U(ARC)+T1
             X(ARC)=0
           ELSE
             NARC=-ARC
             NODE2=ENDN(NARC)
             T1=U(NARC)
             DFCT(NODE2)=DFCT(NODE2)-T1
             X(NARC)=X(NARC)+T1
             U(NARC)=0
           END IF
  440     CONTINUE
         DEFCIT=DEFCIT-DELX
  450    IF (DELPRC.EQ.LARGE) THEN
           QUIT=.TRUE.
           GO TO 500
         END IF
C        ******* price increase at NODE is possible *******************
         NB=0
         PCHANG = .TRUE.
         DP=DELPRC
         DELPRC=LARGE
         DELX=0
         ARC=FIN(NODE)
  460    IF (ARC.GT.0) THEN
           RDCOST=RC(ARC)+DP
           RC(ARC)=RDCOST
           IF (RDCOST.EQ.0) THEN
             NB=NB+1
             SAVE(NB)=ARC
             DELX=DELX+X(ARC)
           END IF
           IF ((RDCOST.LT.0).AND.(-RDCOST.LT.DELPRC)) DELPRC=-RDCOST
           ARC=NXTIN(ARC)
           GOTO 460
         END IF
         ARC=FOU(NODE)
```

```
  470 IF (ARC.GT.0) THEN
        RDCOST=RC(ARC)-DP
        RC(ARC)=RDCOST
        IF (RDCOST.EQ.0) THEN
          NB=NB+1
          SAVE(NB)=-ARC
          DELX=DELX+U(ARC)
        END IF
        IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
        ARC=NXTOU(ARC)
        GOTO 470
      END IF
      GO TO 410
C
C     ******* perform flow augmentation at NODE ****
C
  480 DO 490 J=1,NB
        ARC=SAVE(J)
        IF (ARC.GT.0) THEN
C         *** ARC is an incoming arc to NODE *******************
          NODE2=STARTN(ARC)
          T1=DFCT(NODE2)
          IF (T1.GT.0) THEN
            QUIT=.TRUE.
            T2=X(ARC)
            DX=MIN0(DEFCIT,T1,T2)
            DEFCIT=DEFCIT-DX
            DFCT(NODE2)=T1-DX
            X(ARC)=T2-DX
            U(ARC)=U(ARC)+DX
            IF (DEFCIT.EQ.0) GO TO 500
          END IF
        ELSE
C         *** -ARC is an outgoing arc from NODE ********************
          NARC=-ARC
          NODE2=ENDN(NARC)
          T1=DFCT(NODE2)
          IF (T1.GT.0) THEN
            QUIT=.TRUE.
            T2=U(NARC)
            DX=MIN0(DEFCIT,T1,T2)
            DEFCIT=DEFCIT-DX
            DFCT(NODE2)=T1-DX
            X(NARC)=X(NARC)+DX
            U(NARC)=T2-DX
            IF (DEFCIT.EQ.0) GO TO 500
          END IF
        END IF
  490   CONTINUE
  500   DFCT(NODE)=-DEFCIT
C
C       Reconstruct the list of balanced arcs adjacent to this node.
C       First, the list at this node is now totally different.  Eat
C       the old lists of incoming and outgoing balanced arcs.
C
        IF (PCHANG) THEN
          ARC = TFSTOU(NODE)
          TFSTOU(NODE) = 0
  510     IF (ARC .GT. 0) THEN
             NXTARC = TNXTOU(ARC)
```

```
            TNXTOU(ARC) = -1
            ARC = NXTARC
            GOTO 510
         END IF
         ARC = TFSTIN(NODE)
         TFSTIN(NODE) = 0
  520    IF (ARC .GT. 0) THEN
            NXTARC = TNXTIN(ARC)
            TNXTIN(ARC) = -1
            ARC = NXTARC
            GOTO 520
         END IF
C
C        *** Now add the currently balanced arcs to the list for this
C        *** node(which is now empty),and the appropriate adjacent ones
C
         DO 530 J=1,NB
            ARC = SAVE(J)
            IF (ARC.LE.0) ARC=-ARC
            IF (TNXTOU(ARC) .LT. 0) THEN
               TNXTOU(ARC) = TFSTOU(STARTN(ARC))
               TFSTOU(STARTN(ARC)) = ARC
            END IF
            IF (TNXTIN(ARC) .LT. 0) THEN
               TNXTIN(ARC) = TFSTIN(ENDN(ARC))
               TFSTIN(ENDN(ARC)) = ARC
            END IF
  530    CONTINUE
      END IF
C
C    ***** end of single node iteration for a negative deficit node ***
C
      END IF
C
      IF (QUIT) GO TO 630
C
C     ******* do a multi-node operation from NODE *****************
C
      SWITCH = (NDFCT .LT. TP)
C
C     ******* UNMARK NODES LABELED EARLIER *******
C
         DO 540 J=1,NLABEL
           NODE2=LABEL(J)
           MARK(NODE2)=.FALSE.
           SCAN(NODE2)=.FALSE.
  540    CONTINUE
C
C     ******* INITIALIZE LABELING ******
C
         NLABEL=1
         LABEL(1)=NODE
         MARK(NODE)=.TRUE.
         PRDCSR(NODE)=0
C
C     ******** SCAN STARTING NODE *********
C
         SCAN(NODE)=.TRUE.
         NSCAN=1
         DM=DFCT(NODE)
```

```
          DELX=0
          DO 550 J=1,NB
           ARC=SAVE(J)
           IF (ARC.GT.0) THEN
            IF (POSIT) THEN
              NODE2=ENDN(ARC)
            ELSE
              NODE2=STARTN(ARC)
            END IF
            IF (.NOT.MARK(NODE2)) THEN
               NLABEL=NLABEL+1
               LABEL(NLABEL)=NODE2
               PRDCSR(NODE2)=ARC
               MARK(NODE2)=.TRUE.
               DELX=DELX+X(ARC)
            END IF
           ELSE
            NARC=-ARC
            IF (POSIT) THEN
              NODE2=STARTN(NARC)
            ELSE
              NODE2=ENDN(NARC)
            END IF
            IF (.NOT.MARK(NODE2)) THEN
              NLABEL=NLABEL+1
              LABEL(NLABEL)=NODE2
              PRDCSR(NODE2)=ARC
              MARK(NODE2)=.TRUE.
              DELX=DELX+U(NARC)
            END IF
           END IF
 550     CONTINUE
C
C        **** start scanning labeled nodes ****
C
 560     NSCAN=NSCAN+1
C
C        ****** check to see if SWITCH needs to be set ******
C        SWITCH indicates it may now be best to change over to a more
C        conventional primal-dual algorithm (one which can reuse old
C        labels to some extent).
C
         SWITCH = SWITCH .OR. ( (NSCAN .GT. TS) .AND. (NDFCT .LT. TS) )
C
C    **** scan next node on the list of labeled nodes ****
C    scanning will continue until either an OVERESTIMATE of the residual
C    capacity across the cut corresponding to the scanned set of nodes
C    (called DELX) exceeds the absolute value of the total deficit of the
C    scanned nodes (called DM), or else an augmenting path is found. Arcs
C    that are in the tree but are not balanced are purged as part of the
C    scanning process.
C
      I=LABEL(NSCAN)
      SCAN(I)=.TRUE.
      IF (POSIT) THEN
C
C     ******* scanning node I for case of positive deficit ******
C
      NAUGND=0
      PRVARC=0
```

```
      ARC = TFSTOU(I)
  570 IF (ARC.GT.0) THEN
C
C        ***** ARC is an outgoing arc from NODE *****
C
         IF (RC(ARC) .EQ. 0) THEN
            IF (X(ARC) .GT. 0) THEN
               NODE2=ENDN(ARC)
               IF (.NOT. MARK(NODE2)) THEN
C
C                 ***** NODE2 is not in the labeled set. Add NODE2 to the
C                 labeled set. *****
C
                  PRDCSR(NODE2)=ARC
                  IF (DFCT(NODE2).LT.0) THEN
                     NAUGND=NAUGND+1
                     SAVE(NAUGND)=NODE2
                  END IF
                  NLABEL=NLABEL+1
                  LABEL(NLABEL)=NODE2
                  MARK(NODE2)=.TRUE.
                  DELX=DELX+X(ARC)
               END IF
            END IF
            PRVARC = ARC
            ARC = TNXTOU(ARC)
         ELSE
            TMPARC = ARC
            ARC = TNXTOU(ARC)
            TNXTOU(TMPARC) = -1
            IF (PRVARC .EQ. 0) THEN
               TFSTOU(I) = ARC
            ELSE
               TNXTOU(PRVARC) = ARC
            END IF
         END IF
         GOTO 570
      END IF
C
C
C
      PRVARC = 0
      ARC=TFSTIN(I)
  580 IF (ARC.GT.0) THEN
C
C        ***** ARC is an incoming arc into NODE *****
C
         IF (RC(ARC) .EQ. 0) THEN
            IF (U(ARC) .GT. 0) THEN
               NODE2=STARTN(ARC)
               IF (.NOT. MARK(NODE2)) THEN
C
C                 ***** NODE2 is not in the labeled set. Add NODE2 to the
C                 labeled set. *****
C
                  PRDCSR(NODE2)=-ARC
                  IF (DFCT(NODE2).LT.0) THEN
                     NAUGND=NAUGND+1
                     SAVE(NAUGND)=NODE2
                  END IF
```

```
                NLABEL=NLABEL+1
                LABEL(NLABEL)=NODE2
                MARK(NODE2)=.TRUE.
                DELX=DELX+U(ARC)
             END IF
          END IF
          PRVARC = ARC
          ARC = TNXTIN(ARC)
       ELSE
          TMPARC = ARC
          ARC = TNXTIN(ARC)
          TNXTIN(TMPARC) = -1
          IF (PRVARC .EQ. 0) THEN
             TFSTIN(I) = ARC
          ELSE
             TNXTIN(PRVARC) = ARC
          END IF
       END IF
       GOTO 580
     END IF
C
C   * correct the residual capacity of the scanned nodes cut *
C
     ARC=PRDCSR(I)
     IF (ARC.GT.0) THEN
        DELX=DELX-X(ARC)
     ELSE
        DELX=DELX-U(-ARC)
     END IF
C
C  ********** end of scanning of I for positive deficit case ****
C
             ELSE
C
C     ******* scanning node I for case of negative deficit ****
C
     NAUGND=0
     PRVARC = 0
     ARC=TFSTIN(I)
 590 IF (ARC.GT.0) THEN
       IF (RC(ARC) .EQ. 0) THEN
          IF (X(ARC) .GT. 0) THEN
             NODE2=STARTN(ARC)
             IF (.NOT. MARK(NODE2)) THEN
                PRDCSR(NODE2)=ARC
                IF (DFCT(NODE2).GT.0) THEN
                   NAUGND=NAUGND+1
                   SAVE(NAUGND)=NODE2
                END IF
                NLABEL=NLABEL+1
                LABEL(NLABEL)=NODE2
                MARK(NODE2)=.TRUE.
                DELX=DELX+X(ARC)
             END IF
          END IF
          PRVARC = ARC
          ARC = TNXTIN(ARC)
       ELSE
          TMPARC = ARC
          ARC = TNXTIN(ARC)
```

```
            TNXTIN(TMPARC) = -1
            IF (PRVARC .EQ. 0) THEN
               TFSTIN(I) = ARC
            ELSE
               TNXTIN(PRVARC) = ARC
            END IF
         END IF
         GOTO 590
       END IF
C
C
C
       PRVARC = 0
       ARC = TFSTOU(I)
  600  IF (ARC.GT.0) THEN
         IF (RC(ARC) .EQ. 0) THEN
            IF (U(ARC) .GT. 0) THEN
               NODE2=ENDN(ARC)
               IF (.NOT. MARK(NODE2)) THEN
                  PRDCSR(NODE2)=-ARC
                  IF (DFCT(NODE2).GT.0) THEN
                     NAUGND=NAUGND+1
                     SAVE(NAUGND)=NODE2
                  END IF
                  NLABEL=NLABEL+1
                  LABEL(NLABEL)=NODE2
                  MARK(NODE2)=.TRUE.
                  DELX=DELX+U(ARC)
               END IF
            END IF
            PRVARC = ARC
            ARC = TNXTOU(ARC)
         ELSE
            TMPARC = ARC
            ARC = TNXTOU(ARC)
            TNXTOU(TMPARC) = -1
            IF (PRVARC .EQ. 0) THEN
               TFSTOU(I) = ARC
            ELSE
               TNXTOU(PRVARC) = ARC
            END IF
         END IF
         GOTO 600
       END IF
C
       ARC=PRDCSR(I)
       IF (ARC.GT.0) THEN
          DELX=DELX-X(ARC)
       ELSE
          DELX=DELX-U(-ARC)
       END IF
       END IF
C
C    ****** ADD DEFICIT OF NODE SCANNED TO DM ******
C
        DM=DM+DFCT(I)
C
C       *** check if the set of scanned nodes correspond
C       to a dual ascent direction; if yes, perform a
C       price adjustment step, otherwise continue labeling *
```

```
C
      IF (NSCAN.LT.NLABEL) THEN
        IF (SWITCH) GO TO 610
        IF ((DELX.GE.DM).AND.(DELX.GE.-DM)) GO TO 610
      END IF
C
C   ************* TRY A PRICE CHANGE ***********
C   Note that since DELX-ABS(DM) is an OVERESTIMATE of ascent slope, we
C   may occasionally try a direction that is not really an ascent.  In
C   this case the ANCNTx routines return with QUIT set to .FALSE. .  The
C   main code, it turn, then tries to label some more node.
C
          IF (POSIT) THEN
            CALL ASCNT1(DM,DELX,NLABEL,AUGNOD,FEASBL,
     *      SWITCH,NSCAN)
          ELSE
            CALL ASCNT2(DM,DELX,NLABEL,AUGNOD,FEASBL,
     *      SWITCH,NSCAN)
          END IF
          IF (.NOT.FEASBL) GO TO 640
          IF (.NOT.SWITCH)  GO TO 630
          IF ((SWITCH).AND.(AUGNOD.GT.0)) THEN
            NAUGND=1
            SAVE(1)=AUGNOD
          END IF
C
C         *** CHECK IF AUGMENTATION IS POSSIBLE.
C         IF NOT RETURN TO SCAN ANOTHER NODE. ***
C
 610      CONTINUE
C
          IF (NAUGND.EQ.0) GO TO 560
C
C         Do the augmentation.
C
          DO 620 J=1,NAUGND
          AUGNOD=SAVE(J)
          IF (POSIT) THEN
               CALL AUGFL1(AUGNOD)
          ELSE
               CALL AUGFL2(AUGNOD)
          END IF
 620      CONTINUE
C
C        ** RETURN TO TAKE UP ANOTHER NODE W/ NONZERO DEFICIT **
C
 630    CONTINUE

C        ********** TEST FOR TERMINATION ***********
C
C        We have just done a sweep throught all the nodes.  If they all
C        had zero defecit, we must be done.
C
         NDFCT=NNONZ
         NNONZ=0

         IF (NDFCT.EQ.0) THEN
            RETURN
         ELSE
           GO TO 230
```

```
         END IF
C
C     ******* problem is found to be infeasible ********************
  640 WRITE(6,*)' PROBLEM IS FOUND TO BE INFEASIBLE.'
      FEASBL = .FALSE.
      RETURN
      END



      SUBROUTINE PRFLOW(NODE)
C
C     ***** This subroutine prints the deficit and the flows of
C     arcs incident to NODE. It is used for diagnostic purposes
C     in case of an infeasible problem here. It can be used also
C     for more general diagnostic purposes. *****
C
C
      IMPLICIT INTEGER (A-Z)
C
      COMMON/ARRAYS/STARTN/ARRAYE/ENDN/ARRAYU/U/ARRAYX/X
     */ARRAYB/DFCT/BLK3/FOU/BLK4/NXTOU/BLK5/FIN/BLK6/NXTIN
C
      DIMENSION STARTN(1),ENDN(1),U(1),X(1),DFCT(1)
      DIMENSION FOU(1),NXTOU(1)
      DIMENSION FIN(1),NXTIN(1)
C
C   *************************************************************
C
      WRITE(6,*)'DEFICIT (I.E., NET FLOW OUT) OF NODE =',DFCT(NODE)
      WRITE(6,*)'FLOWS AND CAPACITIES OF INCIDENT ARCS OF NODE',NODE
      IF (FOU(NODE).EQ.0) THEN
        WRITE(6,*)'NO OUTGOING ARCS'
      ELSE
        ARC=FOU(NODE)
   10   IF (ARC.GT.0) THEN
          WRITE(6,*)'ARC',ARC,'  BETWEEN NODES',NODE,ENDN(ARC)
          WRITE(6,*)'FLOW =',X(ARC)
          WRITE(6,*)'RESIDUAL CAPACITY =',U(ARC)
          ARC=NXTOU(ARC)
          GO TO 10
        END IF
      END IF
C
C   *************************************************************
C
      IF (FIN(NODE).EQ.0) THEN
        WRITE(6,*)'NO INCOMING ARCS'
      ELSE
        ARC=FIN(NODE)
   20    IF (ARC.GT.0) THEN
          WRITE(6,*)'ARC',ARC,'  BETWEEN NODES',STARTN(ARC),NODE
          WRITE(6,*)'FLOW =',X(ARC)
          WRITE(6,*)'RESIDUAL CAPACITY =',U(ARC)
          ARC=NXTIN(ARC)
          GO TO 20
        END IF
      END IF
C
C   *************************************************************
```

```
C
      RETURN
      END


      SUBROUTINE AUGFL1(AUGNOD)
C
C     ***** This subroutine performs the flow augmentation step.
C     A flow augmenting path has been identified in the scanning
C     step and here the flow of all arcs positively (negatively)
C     oriented in the flow augmenting path is decreased (increased)
C     to decrease the total deficit. *****
C
      IMPLICIT INTEGER (A-Z)
      COMMON/ARRAYS/STARTN/ARRAYE/ENDN/ARRAYU/U/ARRAYX/X
     */ARRAYB/DFCT/BLK2/PRDCSR
      DIMENSION STARTN(1),ENDN(1),U(1),X(1),DFCT(1),PRDCSR(1)
C
C     ***** A flow augmenting path ending at AUGNOD is found.
C     Determine DX, the amount of flow change. *****
C
      DX=-DFCT(AUGNOD)
      IB=AUGNOD
   10 IF (PRDCSR(IB).NE.0) THEN
        ARC=PRDCSR(IB)
        IF (ARC.GT.0) THEN
          DX=MIN0(DX,X(ARC))
          IB=STARTN(ARC)
        ELSE
          DX=MIN0(DX,U(-ARC))
          IB=ENDN(-ARC)
        END IF
        GOTO 10
      END IF
      ROOT=IB
      DX=MIN0(DX,DFCT(ROOT))
      IF (DX .LE. 0) RETURN
C
C     ***** Update the flow by decreasing (increasing) the flow of
C     all arcs positively (negatively) oriented in the flow
C     augmenting path.  Adjust the deficits accordingly. *****
C
      DFCT(AUGNOD)=DFCT(AUGNOD)+DX
      DFCT(ROOT)=DFCT(ROOT)-DX
      IB=AUGNOD
   20 IF (IB.NE.ROOT) THEN
        ARC=PRDCSR(IB)
        IF (ARC.GT.0) THEN
          X(ARC)=X(ARC)-DX
          U(ARC)=U(ARC)+DX
          IB=STARTN(ARC)
        ELSE
          NARC=-ARC
          X(NARC)=X(NARC)+DX
          U(NARC)=U(NARC)-DX
          IB=ENDN(NARC)
        END IF
        GOTO 20
      END IF
```

```
      RETURN
      END


      SUBROUTINE ASCNT1(DM,DELX,NLABEL,AUGNOD,FEASBL,SWITCH,
     *NSCAN)
C
C     This subroutine essentially performs the multi-node
C     price adjustment step.  It first checks if the set
C     of scanned nodes correspond to a dual ascent direction.
C     If yes, then decrease the price of all scanned nodes.
C     There are two possibilities for price adjustment:
C     If SWITCH=.TRUE. then the set of scanned nodes
C     corresponds to an elementary direction of maximal
C     rate of ascent, in which case the price of all scanned
C     nodes are decreased until the next breakpoint in the
C     dual cost is encountered.  At this point some arc
C     becomes balanced and more node(s) are added to the
C     labeled set.
C     If SWITCH=.FALSE. then the prices of all scanned nodes
C     are decreased until the rate of ascent becomes
C     negative (this corresponds to the price adjustment
C     step in which both the line search and the degenerate
C     ascent iteration are implemented).
C
      IMPLICIT INTEGER (A-Z)
C
C     The two "tree"-based ascent routines have a common temporary
C     storage area whose dimension is set below. The maximum conceivable
C     amount needed equals the number of arcs, but this should never
C     actually occur.
C
C
      LOGICAL SCAN,MARK,SWITCH,FEASBL,QUIT
      COMMON/ARRAYS/STARTN/ARRAYE/ENDN/ARRAYU/U/ARRAYX/X/ARRAY9/RC
     */ARRAYB/DFCT/BLK1/LABEL/BLK2/PRDCSR/BLK3/FOU/BLK4/
     *NXTOU/BLK5/FIN/BLK6/NXTIN/BLK7/SAVE/BLK8/SCAN/BLK9/MARK
     */L/N,NA,LARGE
      COMMON /BLK10/TFSTOU/BLK11/TNXTOU/BLK12/TFSTIN/BLK13/TNXTIN
      COMMON /ASCBLK/B
      DIMENSION TFSTOU(1),TNXTOU(1),TFSTIN(1),TNXTIN(1)
      DIMENSION STARTN(1),ENDN(1),U(1),X(1),RC(1),DFCT(1),LABEL(1)
      DIMENSION PRDCSR(1),FOU(1),NXTOU(1),FIN(1),NXTIN(1)
      DIMENSION SAVE(1),SCAN(1),MARK(1)
C
C     ***** Store the arcs between the set of scanned nodes and
C     its complement in SAVE and compute DELPRC, the stepsize
C     to the next breakpoint in the dual cost in the direction
C     of decreasing prices of the scanned nodes. *****
C
      DELPRC=LARGE
      DLX=0
      NSAVE=0
C
C   **** calculate the array SAVE of arcs across the cut of scanned
C   nodes in a different way depending on whether NSCAN>N/2 or not.
C   This is done for efficiency. ****
C
      IF (NSCAN.LE.N/2) THEN
```

```
      DO 30 I=1,NSCAN
        NODE=LABEL(I)
          ARC=FOU(NODE)
   10     IF (ARC.GT.0) THEN
C
C           ***** ARC is an arc pointing from the set of scanned
C           nodes to its complement. *****
C
            NODE2=ENDN(ARC)
            IF (.NOT.SCAN(NODE2)) THEN
              NSAVE=NSAVE+1
              SAVE(NSAVE)=ARC
              RDCOST=RC(ARC)
      IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE2).NE.ARC)) DLX=DLX+X(ARC)
            IF ((RDCOST.LT.0).AND.(-RDCOST.LT.DELPRC)) DELPRC=-RDCOST
            END IF
            ARC=NXTOU(ARC)
            GOTO 10
          END IF
          ARC=FIN(NODE)

   20     IF (ARC.GT.0) THEN
C
C           ***** ARC is an arc pointing to the set of scanned
C           nodes from its complement. *****
C
            NODE2=STARTN(ARC)
            IF (.NOT.SCAN(NODE2)) THEN
              NSAVE=NSAVE+1
              SAVE(NSAVE)=-ARC
              RDCOST=RC(ARC)
      IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE2).NE.-ARC)) DLX=DLX+U(ARC)
              IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
            END IF
            ARC=NXTIN(ARC)
            GOTO 20
          END IF
   30 CONTINUE
C
      ELSE
C
      DO 60 NODE=1,N
        IF (SCAN(NODE)) GO TO 60
          ARC=FIN(NODE)
   40     IF (ARC.GT.0) THEN
            NODE2=STARTN(ARC)
            IF (SCAN(NODE2)) THEN
              NSAVE=NSAVE+1
              SAVE(NSAVE)=ARC
              RDCOST=RC(ARC)
      IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE).NE.ARC)) DLX=DLX+X(ARC)
            IF ((RDCOST.LT.0).AND.(-RDCOST.LT.DELPRC)) DELPRC=-RDCOST
            END IF
            ARC=NXTIN(ARC)
            GOTO 40
          END IF
          ARC=FOU(NODE)
   50     IF (ARC.GT.0) THEN
            NODE2=ENDN(ARC)
            IF (SCAN(NODE2)) THEN
```

```
              NSAVE=NSAVE+1
              SAVE(NSAVE)=-ARC
              RDCOST=RC(ARC)
      IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE).NE.-ARC)) DLX=DLX+U(ARC)
              IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
            END IF
            ARC=NXTOU(ARC)
            GOTO 50
          END IF
   60 CONTINUE
      END IF
C
C     ***** Check if the set of scanned nodes truly corresponds
C     to a dual ascent direction.  Here DELX+DLX is the exact
C     sum of the flow on arcs from the scanned set to the
C     unscanned set plus the ( capacity - flow ) on arcs from
C     the unscanned set to the scanned set. *****
C
      IF (DELX+DLX.GE.DM) THEN
        SWITCH=.TRUE.
        AUGNOD=0
        DO 70 I=NSCAN+1,NLABEL
          NODE=LABEL(I)
          IF (DFCT(NODE).LT.0) AUGNOD=NODE
   70   CONTINUE
        RETURN
      END IF
      DELX=DELX+DLX
C
C     ******* check that the problem is feasible *******************
C
   80 IF (DELPRC.EQ:LARGE) THEN
C
C       ***** We can decrease the dual cost without bound.
C       Therefore the primal problem is infeasible. *****
C
        FEASBL=.FALSE.
        RETURN
      END IF
C
C     ******* Decrease prices of the scanned nodes, add more
C     nodes to the labeled set & check if a newly labeled node
C     has negative deficit. *****
C
      IF (SWITCH) THEN
        AUGNOD=0
        DO 90 I=1,NSAVE
          ARC=SAVE(I)
          IF (ARC.GT.0) THEN
            RC(ARC)=RC(ARC)+DELPRC
            IF (RC(ARC).EQ.0) THEN
              NODE2=ENDN(ARC)
              IF (TNXTOU(ARC) .LT. 0) THEN
                TNXTOU(ARC) = TFSTOU(STARTN(ARC))
                TFSTOU(STARTN(ARC)) = ARC
              END IF
              IF (TNXTIN(ARC) .LT. 0) THEN
                TNXTIN(ARC) = TFSTIN(NODE2)
                TFSTIN(NODE2) = ARC
              END IF
```

```
          PRDCSR(NODE2)=ARC
          IF (DFCT(NODE2).LT.0) THEN
            AUGNOD=NODE2
          ELSE
            IF (.NOT.MARK(NODE2)) THEN
             MARK(NODE2)=.TRUE.
             NLABEL=NLABEL+1
             LABEL(NLABEL)=NODE2
            END IF
          END IF
        END IF
      ELSE
        ARC=-ARC
        RC(ARC)=RC(ARC)-DELPRC
        IF (RC(ARC).EQ.0) THEN
          NODE2=STARTN(ARC)
          IF (TNXTOU(ARC) .LT. 0) THEN
            TNXTOU(ARC) = TFSTOU(NODE2)
            TFSTOU(NODE2) = ARC
          END IF
          IF (TNXTIN(ARC) .LT. 0) THEN
            TNXTIN(ARC) = TFSTIN(ENDN(ARC))
            TFSTIN(ENDN(ARC)) = ARC
          END IF
          PRDCSR(NODE2)=-ARC
          IF (DFCT(NODE2).LT.0) THEN
            AUGNOD=NODE2
          ELSE
           IF (.NOT.MARK(NODE2)) THEN
            MARK(NODE2)=.TRUE.
            NLABEL=NLABEL+1
            LABEL(NLABEL)=NODE2
           END IF
          END IF
        END IF
      END IF
90    CONTINUE
      RETURN
C
    ELSE
C
C   ***** Decrease the prices of the scanned nodes by DELPRC.
C   Adjust arc flow to maintain complementary slackness with
C   the prices. *****
C
    NB = 0
    DO 100 I=1,NSAVE
      ARC=SAVE(I)
      IF (ARC.GT.0) THEN
          T1=RC(ARC)
          IF (T1.EQ.0) THEN
           T2=X(ARC)
           T3=STARTN(ARC)
           DFCT(T3)=DFCT(T3)-T2
           T3=ENDN(ARC)
           DFCT(T3)=DFCT(T3)+T2
           U(ARC)=U(ARC)+T2
           X(ARC)=0
          END IF
        RC(ARC)=T1+DELPRC
```

```
          IF (RC(ARC).EQ.0) THEN
             DELX=DELX+X(ARC)
             NB = NB + 1
             PRDCSR(NB) = ARC
          ENDIF
        ELSE
          ARC=-ARC
          T1=RC(ARC)
          IF (T1.EQ.0) THEN
            T2=U(ARC)
            T3=STARTN(ARC)
            DFCT(T3)=DFCT(T3)+T2
            T3=ENDN(ARC)
            DFCT(T3)=DFCT(T3)-T2
            X(ARC)=X(ARC)+T2
            U(ARC)=0
          END IF
          RC(ARC)=T1-DELPRC
          IF (RC(ARC).EQ.0) THEN
             DELX=DELX+U(ARC)
             NB = NB + 1
             PRDCSR(NB) = ARC
          END IF
        END IF
  100 CONTINUE
      END IF
C
      IF (DELX.LE.DM) THEN
C
C       ***** The set of scanned nodes still corresponds to a
C       dual (possibly degenerate) ascent direction.  Compute
C       the stepsize DELPRC to the next breakpoint in the
C       dual cost. *****
C
        DELPRC=LARGE
        DO 110 I=1,NSAVE
          ARC=SAVE(I)
          IF (ARC.GT.0) THEN
            RDCOST=RC(ARC)
          IF ((RDCOST.LT.0).AND.(-RDCOST.LT.DELPRC)) DELPRC=-RDCOST
         ELSE
            ARC=-ARC
            RDCOST=RC(ARC)
           IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
         END IF
  110    CONTINUE
         IF ((DELPRC.NE.LARGE).OR.(DELX.LT.DM)) GO TO 80
       END IF
C
C       *** Add new balanced arcs to the superset of balanced arcs.  ***
C
      DO 120 I=1,NB
        ARC=PRDCSR(I)
          IF (TNXTIN(ARC).EQ.-1) THEN
            J=ENDN(ARC)
            TNXTIN(ARC)=TFSTIN(J)
            TFSTIN(J)=ARC
          END IF
          IF (TNXTOU(ARC).EQ.-1) THEN
            J=STARTN(ARC)
```

```
            TNXTOU(ARC)=TFSTOU(J)
            TFSTOU(J)=ARC
          END IF
  120 CONTINUE
      RETURN
      END



      SUBROUTINE AUGFL2(AUGNOD)
      IMPLICIT INTEGER (A-Z)
      COMMON/ARRAYS/STARTN/ARRAYE/ENDN/ARRAYU/U/ARRAYX/X
     */ARRAYB/DFCT/BLK2/PRDCSR
      DIMENSION STARTN(1),ENDN(1),U(1),X(1),DFCT(1),PRDCSR(1)
C
C     ******* an augmenting path is found.  determine flow change ***
C
      DX=DFCT(AUGNOD)
      IB=AUGNOD
   10 IF (PRDCSR(IB).NE.0) THEN
        ARC=PRDCSR(IB)
        IF (ARC.GT.0) THEN
          DX=MIN0(DX,X(ARC))
          IB=ENDN(ARC)
        ELSE
          DX=MIN0(DX,U(-ARC))
          IB=STARTN(-ARC)
        END IF
        GOTO 10
       END IF
      ROOT=IB
      DX=MIN0(DX,-DFCT(ROOT))
      IF (DX .LE. 0) RETURN
C
C     ******* update the flow and deficits ************************
C
      DFCT(AUGNOD)=DFCT(AUGNOD)-DX
      DFCT(ROOT)=DFCT(ROOT)+DX
      IB=AUGNOD
   20 IF (IB.NE.ROOT) THEN
        ARC=PRDCSR(IB)
        IF (ARC.GT.0) THEN
          X(ARC)=X(ARC)-DX
          U(ARC)=U(ARC)+DX
          IB=ENDN(ARC)
        ELSE
          NARC=-ARC
          X(NARC)=X(NARC)+DX
          U(NARC)=U(NARC)-DX
          IB=STARTN(NARC)
        END IF
        GOTO 20
      END IF
      RETURN
      END



      SUBROUTINE ASCNT2(DM,DELX,NLABEL,AUGNOD,FEASBL,SWITCH,
     *NSCAN)
```

```
      IMPLICIT INTEGER (A-Z)
C
C     The two "tree"-based ascent routines have a common temporary
C     storage area whose dimension is set below. The maximum conceivable
C     amount needed equals the number of arcs, but this should never
C     actually occur.
C
      LOGICAL SCAN,MARK,SWITCH,FEASBL,QUIT
      COMMON/ARRAYS/STARTN/ARRAYE/ENDN/ARRAYU/U/ARRAYX/X/ARRAY9/RC
     */ARRAYB/DFCT/BLK1/LABEL/BLK2/PRDCSR/BLK3/FOU/BLK4/
     *NXTOU/BLK5/FIN/BLK6/NXTIN/BLK7/SAVE/BLK8/SCAN/BLK9/MARK
     */L/N,NA,LARGE
      COMMON /BLK10/TFSTOU/BLK11/TNXTOU/BLK12/TFSTIN/BLK13/TNXTIN
      COMMON /ASCBLK/B
      DIMENSION TFSTOU(1),TNXTOU(1),TFSTIN(1),TNXTIN(1)
      DIMENSION STARTN(1),ENDN(1),U(1),X(1),RC(1),DFCT(1),LABEL(1)
      DIMENSION PRDCSR(1),FOU(1),NXTOU(1),FIN(1),NXTIN(1)
      DIMENSION SAVE(1),SCAN(1),MARK(1)
C
C     ******* augment flows across the cut & compute price rise *****
C
      DELPRC=LARGE
      DLX=0
      NSAVE=0
      IF (NSCAN.LE.N/2) THEN
      DO 30 I=1,NSCAN
        NODE=LABEL(I)
          ARC=FIN(NODE)
 10       IF (ARC.GT.0) THEN
            NODE2=STARTN(ARC)
            IF (.NOT.SCAN(NODE2)) THEN
              NSAVE=NSAVE+1
              SAVE(NSAVE)=ARC
              RDCOST=RC(ARC)
      IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE2).NE.ARC)) DLX=DLX+X(ARC)
            IF ((RDCOST.LT.0).AND.(-RDCOST.LT.DELPRC)) DELPRC=-RDCOST
            END IF
            ARC=NXTIN(ARC)
            GOTO 10
          END IF
          ARC=FOU(NODE)
 20       IF (ARC.GT.0) THEN
            NODE2=ENDN(ARC)
            IF (.NOT.SCAN(NODE2)) THEN
              NSAVE=NSAVE+1
              SAVE(NSAVE)=-ARC
              RDCOST=RC(ARC)
        IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE2).NE.-ARC)) DLX=DLX+U(ARC)
              IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
            END IF
            ARC=NXTOU(ARC)
            GOTO 20
          END IF
 30   CONTINUE
      ELSE
      DO 60 NODE=1,N
        IF (SCAN(NODE)) GO TO 60
          ARC=FOU(NODE)
 40       IF (ARC.GT.0) THEN
            NODE2=ENDN(ARC)
```

```
          IF (SCAN(NODE2)) THEN
            NSAVE=NSAVE+1
            SAVE(NSAVE)=ARC
            RDCOST=RC(ARC)
    IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE).NE.ARC)) DLX=DLX+X(ARC)
          IF ((RDCOST.LT.0).AND.(-RDCOST.LT.DELPRC)) DELPRC=-RDCOST
          END IF
          ARC=NXTOU(ARC)
          GOTO 40
        END IF
        ARC=FIN(NODE)
 50     IF (ARC.GT.0) THEN
          NODE2=STARTN(ARC)
          IF (SCAN(NODE2)) THEN
            NSAVE=NSAVE+1
            SAVE(NSAVE)=-ARC
            RDCOST=RC(ARC)
      IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE).NE.-ARC)) DLX=DLX+U(ARC)
            IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
          END IF
          ARC=NXTIN(ARC)
          GOTO 50
        END IF
 60 CONTINUE
    END IF
    IF (DELX+DLX.GE.-DM) THEN
      SWITCH=.TRUE.
      AUGNOD=0
      DO 70 I=NSCAN+1,NLABEL
        NODE=LABEL(I)
        IF (DFCT(NODE).GT.0) AUGNOD=NODE
 70   CONTINUE
      RETURN
    END IF
    DELX=DELX+DLX
C
C   ******* check that the problem is feasible *******************
C
 80 IF (DELPRC.EQ.LARGE) THEN
      FEASBL=.FALSE.
      RETURN
    END IF
C
C    ***** INCREASE PRICES *****
C
    IF (SWITCH) THEN
      AUGNOD=0
      DO 90 I=1,NSAVE
        ARC=SAVE(I)
        IF (ARC.GT.0) THEN
          RC(ARC)=RC(ARC)+DELPRC
          IF (RC(ARC).EQ.0) THEN
            NODE2=STARTN(ARC)
            IF (TNXTOU(ARC) .LT. 0) THEN
              TNXTOU(ARC) = TFSTOU(NODE2)
              TFSTOU(NODE2) = ARC
            END IF
            IF (TNXTIN(ARC) .LT. 0) THEN
              TNXTIN(ARC) = TFSTIN(ENDN(ARC))
              TFSTIN(ENDN(ARC)) = ARC
```

```
              END IF
              PRDCSR(NODE2)=ARC
              IF (DFCT(NODE2).GT.0) THEN
                AUGNOD=NODE2
              ELSE
               IF (.NOT.MARK(NODE2)) THEN
                 MARK(NODE2)=.TRUE.
                NLABEL=NLABEL+1
                LABEL(NLABEL)=NODE2
               END IF
              END IF
            END IF
          ELSE
            ARC=-ARC
            RC(ARC)=RC(ARC)-DELPRC
            IF (RC(ARC).EQ.0) THEN
              NODE2=ENDN(ARC)
              IF (TNXTOU(ARC) .LT. 0) THEN
                TNXTOU(ARC) = TFSTOU(STARTN(ARC))
                TFSTOU(STARTN(ARC)) = ARC
              END IF
              IF (TNXTIN(ARC) .LT. 0) THEN
                TNXTIN(ARC) = TFSTIN(NODE2)
                TFSTIN(NODE2) = ARC
              END IF
              PRDCSR(NODE2)=-ARC
              IF (DFCT(NODE2).GT.0) THEN
                AUGNOD=NODE2
              ELSE
               IF (.NOT.MARK(NODE2)) THEN
                MARK(NODE2)=.TRUE.
                NLABEL=NLABEL+1
                LABEL(NLABEL)=NODE2
               END IF
              END IF
            END IF
          END IF
   90   CONTINUE
        RETURN
C
      ELSE
C
      NB = 0
      DO 100 I=1,NSAVE
        ARC=SAVE(I)
        IF (ARC.GT.0) THEN
            T1=RC(ARC)
            IF (T1.EQ.0) THEN
             T2=X(ARC)
             T3=STARTN(ARC)
             DFCT(T3)=DFCT(T3)-T2
             T3=ENDN(ARC)
             DFCT(T3)=DFCT(T3)+T2
             U(ARC)=U(ARC)+T2
             X(ARC)=0
            END IF
          RC(ARC)=T1+DELPRC
          IF (RC(ARC).EQ.0) THEN
             DELX=DELX+X(ARC)
             NB = NB + 1
```

```
              PRDCSR(NB) = ARC
            END IF
          ELSE
            ARC=-ARC
            T1=RC(ARC)
              IF (T1.EQ.0) THEN
                T2=U(ARC)
                T3=STARTN(ARC)
                DFCT(T3)=DFCT(T3)+T2
                T3=ENDN(ARC)
                DFCT(T3)=DFCT(T3)-T2
                X(ARC)=X(ARC)+T2
                U(ARC)=0
              END IF
            RC(ARC)=T1-DELPRC
            IF (RC(ARC).EQ.0) THEN
               DELX=DELX+U(ARC)
               NB = NB + 1
               PRDCSR(NB) = ARC
          END IF
        END IF
  100 CONTINUE
C
      END IF
      IF (DELX.LE.-DM) THEN
        DELPRC=LARGE
        DO 110 I=1,NSAVE
          ARC=SAVE(I)
          IF (ARC.GT.0) THEN
            RDCOST=RC(ARC)
          IF ((RDCOST.LT.0).AND.(-RDCOST.LT.DELPRC)) DELPRC=-RDCOST
         ELSE
            ARC=-ARC
            RDCOST=RC(ARC)
           IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
         END IF
  110    CONTINUE
         IF ((DELPRC.NE.LARGE).OR.(DELX.LT.-DM)) GO TO 80
       END IF
C
C       *** Add new balance arcs to the superset of balanced arcs.   ***
C
      DO 120 I=1,NB
        ARC=PRDCSR(I)
          IF (TNXTIN(ARC).EQ.-1) THEN
            J=ENDN(ARC)
            TNXTIN(ARC)=TFSTIN(J)
            TFSTIN(J)=ARC
          END IF
          IF (TNXTOU(ARC).EQ.-1) THEN
            J=STARTN(ARC)
            TNXTOU(ARC)=TFSTOU(J)
            TFSTOU(J)=ARC
          END IF
  120 CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE SENSTV
C
C     SENSITIVITY ANALYSIS FOR THE MINIMUM COST NETWORK FLOW PROBLEM.
C
C     ***************************************************************
C     ***      THE SUBROUTINE IS BASED ON THE PAPER                ***
C     ***      D.P. BERTSEKAS, P.TSENG "THE RELAX CODES FOR        ***
C     ***        LINEAR MINIMUM COST NETWORK FLOW PROBLEMS",       ***
C     ***        ANNALS OF OPERATIONS RESEARCH, THIS VOLUME        ***
C     ***                                                          ***
C     ***      THE SUBROUTINE IS WRITTEN IN STANDARD FORTRAN77     ***
C     ***                                                          ***
C     ***      QUESTIONS AND COMMENTS SHOULD BE DIRECTED TO        ***
C     ***      DIMITRI BERTSEKAS AND PAUL TSENG                    ***
C     ***      DEPARTMENT OF ELECTRICAL ENGINEERING &              ***
C     ***        COMPUTER SCIENCE                                  ***
C     ***      LABORATORY FOR INFORMATION AND DECISION SYSTEMS     ***
C     ***      M.I.T.  CAMBRIDGE, MASSACHUSETTS, 02139. U.S.A.     ***
C     ***************************************************************
C
C     ***** This subroutine allows the user to interactively
C     either change nodal supply, or change flow upper bound
C     of an existing arc, or change cost of an existing arc,
C     or delete an existing arc, or add an arc.  *****
C
C     NOTE : If in the system on which this subroutine is ran, the
C     variable local to a subroutine is re-initialized (to some default
C     value) each time the subroutine is called, then the user must make
C     the following currently local variables DELARC, DARC, DU, ADDARC,
C     AARC global (by either putting them in a common block or passing
C     them through the calling parameter).
      IMPLICIT INTEGER (A-Z)
      COMMON/ARRAYS/STARTN/ARRAYE/ENDN/ARRAYU/U/ARRAYX/X/ARRAY9/RC
     */ARRAYB/DFCT/BLK1/LABEL/BLK2/PRICE/BLK3/FOU/BLK4/NXTOU
     */BLK5/FIN/BLK6/NXTIN/BLK9/MARK/L/N,NA,LARGE
      COMMON/ARRAYC/C/BLKCAP/CAP/BLKR/REPEAT
      INTEGER CAP(1),U(1),X(1),C(1),RC(1),DFCT(1)
      INTEGER STARTN(1),ENDN(1),LABEL(1),PRICE(1),FOU(1),NXTOU(1),
     *FIN(1),NXTIN(1)
      LOGICAL ADDARC,DELARC,REPEAT,MARK(1)
      IF (.NOT.REPEAT) THEN
C
C     ***** Restore the arc capacity to that of the original problem
C     (recall that when solving the original problem, RELAX in the
C     problem preprocessing phase may decrease the arc capacity) and
C     update flow and deficit to agree with this "new" capacity. *****
        DO 10 I=1,NA
          IF (RC(I).LT.0) THEN
            DFCT(STARTN(I))=DFCT(STARTN(I))+CAP(I)-X(I)
            DFCT(ENDN(I))=DFCT(ENDN(I))-CAP(I)+X(I)
            X(I)=CAP(I)
          ELSE
            U(I)=CAP(I)-X(I)
          END IF
```

```
 10   CONTINUE
      REPEAT=.TRUE.
     END IF
 20   WRITE(6,30)
      WRITE(6,40)
      WRITE(6,50)
      WRITE(6,60)
      WRITE(6,70)
      WRITE(6,80)
      IF (ADDARC) WRITE(6,90) AARC
      IF (DELARC) WRITE(6,100) DARC
 30   FORMAT(' ','INPUT 0 TO SOLVE THE MODIFIED PROBLEM')
 40   FORMAT(' ','       1 TO CHANGE NODAL FLOW SUPPLY')
 50   FORMAT(' ','       2 TO CHANGE ARC FLOW UPPER BOUND')
 60   FORMAT(' ','       3 TO CHANGE ARC COST')
 70   FORMAT(' ','       4 TO DELETE AN ARC')
 80   FORMAT(' ','       5 TO ADD AN ARC')
 90   FORMAT(' ','       6 TO DELETE LAST ARC',I8,' ADDED')
100   FORMAT(' ','       7 TO RESTORE LAST ARC',I8,' DELETED')
      READ(5,*)SEL
      IF (SEL.EQ.0) THEN
        RETURN
      ELSE IF (SEL.EQ.1) THEN
C
C     ***** Change nodal flow supply *****
110     WRITE(6,120)
120     FORMAT(' ','INPUT NODE # WHERE FLOW SUPPLY IS INCREASED')
        READ(5,*)NODE
        IF ((NODE.LE.0).OR.(NODE.GT.N)) GO TO 110
        WRITE(6,130)
130     FORMAT(' ','INPUT AMOUNT OF INCREASE (<0 VALUE ALLOWED)')
        READ(5,*)DELSUP
        DFCT(NODE)=DFCT(NODE)-DELSUP
140     WRITE(6,150)
150     FORMAT(' ','INPUT NODE NO. WHERE FLOW SUPPLY IS DECREASED')
        READ(5,*)NODE
        IF ((NODE.LE.0).OR.(NODE.GT.N)) GO TO 140
        DFCT(NODE)=DFCT(NODE)+DELSUP
      ELSE IF (SEL.EQ.2) THEN
C
C     ***** Change arc flow capacity *****
C     *** Note that U is not the arc capacity but rather the flow margin
C     (i.e.  U = capacity - flow). ***
160     WRITE(6,170)
170     FORMAT(' ','INPUT ARC NO. AND THE INCREASE IN UPPER BOUND')
        READ(5,*)ARC,DELUB
        IF ((ARC.LE.0).OR.(ARC.GT.NA)) GO TO 160
        IF (RC(ARC).LT.0) THEN
C
C     ***** ARC is active, therefore maintain flow at (new) capacity. **
          DFCT(STARTN(ARC))=DFCT(STARTN(ARC))+DELUB
          DFCT(ENDN(ARC))=DFCT(ENDN(ARC))-DELUB
          X(ARC)=X(ARC)+DELUB
          IF (X(ARC).LT.0) WRITE(6,180)
        ELSE IF (RC(ARC).EQ.0) THEN
          IF (U(ARC).GE.-DELUB) THEN
            U(ARC)=U(ARC)+DELUB
          ELSE
C
C     ***** New capacity is less than current flow, therefore decrease
```

```
C     flow to new capacity. *****
                DEL=-DELUB-U(ARC)
                DFCT(STARTN(ARC))=DFCT(STARTN(ARC))-DEL
                DFCT(ENDN(ARC))=DFCT(ENDN(ARC))+DEL
                X(ARC)=X(ARC)-DEL
                IF (X(ARC).LT.0) WRITE(6,180)
                U(ARC)=0
              END IF
            ELSE
              U(ARC)=U(ARC)+DELUB
              IF (U(ARC).LT.0) WRITE(6,180)
  180         FORMAT(' ','FLOW UPPER BOUND IS NOW < 0')
            END IF
          ELSE IF (SEL.EQ.3) THEN
C
C     ***** Change arc cost *****
  190       WRITE(6,200)
  200       FORMAT(' ','INPUT ARC NO. & INCREASE IN COST')
            READ(5,*)ARC,DELC
            IF ((ARC.LE.0).OR.(ARC.GT.NA)) GO TO 190
            IF ((RC(ARC).GE.0).AND.(RC(ARC)+DELC.LT.0)) THEN
C
C     ***** ARC becomes active, therefore increase flow to capacity. ***
              DFCT(STARTN(ARC))=DFCT(STARTN(ARC))+U(ARC)
              DFCT(ENDN(ARC))=DFCT(ENDN(ARC))-U(ARC)
              X(ARC)=U(ARC)+X(ARC)
              U(ARC)=0
            ELSE IF ((RC(ARC).LE.0).AND.(RC(ARC)+DELC.GT.0))THEN
C
C     ***** ARC becomes inactive, therefore decrease flow to zero. *****
              DFCT(STARTN(ARC))=DFCT(STARTN(ARC))-X(ARC)
              DFCT(ENDN(ARC))=DFCT(ENDN(ARC))+X(ARC)
              U(ARC)=U(ARC)+X(ARC)
              X(ARC)=0
            END IF
            RC(ARC)=RC(ARC)+DELC
            C(ARC)=C(ARC)+DELC
          ELSE IF ((SEL.EQ.4).OR.(SEL.EQ.6)) THEN
C
C     ***** Delete an arc *****
            IF (SEL.EQ.6) THEN
              IF (.NOT.ADDARC) GO TO 20
              ADDARC=.FALSE.
              ARC=AARC
            ELSE
  210         WRITE(6,220)
  220         FORMAT(' ','INPUT ARC NO. FOR DELETION')
              READ(5,*)ARC
              IF ((ARC.LE.0).OR.(ARC.GT.NA)) GO TO 210
              DELARC=.TRUE.
              DARC=ARC
              DU=U(ARC)+X(ARC)
            END IF
C
C     ***** Remove ARC from the data array FIN, FOU, NXTIN, NXTOU. *****
            ARC1=FOU(STARTN(ARC))
            IF (ARC1.EQ.ARC) THEN
              FOU(STARTN(ARC))=NXTOU(ARC1)
            ELSE
  230         ARC2=NXTOU(ARC1)
```

```
            IF (ARC2.EQ.ARC) THEN
              NXTOU(ARC1)=NXTOU(ARC2)
              GO TO 240
            END IF
            ARC1=ARC2
            GO TO 230
          END IF
  240     ARC1=FIN(ENDN(ARC))
          IF (ARC1.EQ.ARC) THEN
            FIN(ENDN(ARC))=NXTIN(ARC1)
          ELSE
  250       ARC2=NXTIN(ARC1)
            IF (ARC2.EQ.ARC) THEN
              NXTIN(ARC1)=NXTIN(ARC2)
              GO TO 260
            END IF
            ARC1=ARC2
            GO TO 250
          END IF
C
C     *** Remove flow of ARC from network by setting its flow and
C     capacity to 0.
  260     DFCT(STARTN(ARC))=DFCT(STARTN(ARC))-X(ARC)
          DFCT(ENDN(ARC))=DFCT(ENDN(ARC))+X(ARC)
          X(ARC)=0
          U(ARC)=0
        ELSE IF ((SEL.EQ.5).OR.(SEL.EQ.7)) THEN
          IF (SEL.EQ.7) THEN
            IF (.NOT.DELARC) GO TO 20
            IARC=DARC
            IH=STARTN(IARC)
            IT=ENDN(IARC)
            DELARC=.FALSE.
            IU=DU
          ELSE
  270     WRITE(6,280)NA+1
  280     FORMAT(' ','INPUT HEAD & TAIL NODES OF NEW ARC',I8)
          READ(5,*)IH,IT
          IF ((IH.LE.0).OR.(IH.GT.N).OR.(IT.LE.0).OR.(IT.GT.N))GO TO 270
  290     WRITE(6,300)
  300     FORMAT(' ','INPUT COST & FLOW UPPER BD')
          READ(5,*)IC,IU
          IF (IU.LT.0) GO TO 290
          ADDARC=.TRUE.
          AARC=NA+1
          NA=NA+1
          C(NA)=IC
          STARTN(NA)=IH
          ENDN(NA)=IT
          IARC=NA
          END IF
C     ***** Determine the dual prices at IH and IT. *****
C     ***** We first set the price at node IH to zero and then construct
C     the price at the remaining nodes using the arc cost array C and
C     the reduced cost array RC (using the fact that RC(ARC) = C(ARC) -
C     PRICE(STARTN(ARC)) + PRICE(ENDN(ARC)) ).  This is done by breadth
C     first search. *****
        NSCAN=0
        NLABEL=1
        LABEL(1)=IH
```

```
      PRICE(IH)=0
      DO 310 I=1,N
 310  MARK(I)=.FALSE.
      MARK(IH)=.TRUE.
 320  IF (NLABEL.LE.NSCAN) GO TO 370
        NSCAN=NSCAN+1
        NODE=LABEL(NSCAN)
        ARC=FOU(NODE)
 330    IF (ARC.LE.0) GO TO 340
          NODE2=ENDN(ARC)
          IF (.NOT.MARK(NODE2)) THEN
            MARK(NODE2)=.TRUE.
            PRICE(NODE2)=RC(ARC)-C(ARC)+PRICE(NODE)
            IF (NODE2.EQ.IT) GO TO 370
            NLABEL=NLABEL+1
            LABEL(NLABEL)=NODE2
          END IF
          ARC=NXTOU(ARC)
        GO TO 330
 340    ARC=FIN(NODE)
 350    IF (ARC.LE.0) GO TO 360
          NODE2=STARTN(ARC)
          IF (.NOT.MARK(NODE2)) THEN
            MARK(NODE2)=.TRUE.
            PRICE(NODE2)=C(ARC)-RC(ARC)+PRICE(NODE)
            IF (NODE2.EQ.IT) GO TO 370
            NLABEL=NLABEL+1
            LABEL(NLABEL)=NODE2
          END IF
          ARC=NXTIN(ARC)
        GO TO 350
 360  GO TO 320
C
C    ***** Compute reduced cost of the new arc and update flow and
C    deficit accordingly. *****
 370  RC(IARC)=C(IARC)+PRICE(IT)
      IF (RC(IARC).LT.0) THEN
        DFCT(IH)=DFCT(IH)+IU
        DFCT(IT)=DFCT(IT)-IU
        X(IARC)=IU
        U(IARC)=0
      ELSE
        X(IARC)=0
        U(IARC)=IU
      END IF
      NXTOU(IARC)=FOU(IH)
      FOU(IH)=IARC
      NXTIN(IARC)=FIN(IT)
      FIN(IT)=IARC
    END IF
    GO TO 20
    END
```