THE RELAX CODES FOR LINEAR MINIMUM COST
NETWORK FLOW PROBLEMS

by

Dimitri P. Bertsekas
and
Paul Tseng*

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 speedup 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.

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

**The authors are with the Laboratory for Information and Decision Systems and the Operations Research Center, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.
DISCLAIMER OF QUALITY

Due to the condition of the original material, there are unavoidable flaws in this reproduction. We have made every effort possible to provide you with the best copy available. If you are dissatisfied with this product and find it unusable, please contact Document Services as soon as possible.

Thank you.

Due to the poor quality of the original document, there is some spotting or background shading in this document.
1. Introduction

Consider a directed graph with set of nodes $N$ and set of arcs $A$. 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

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

subject to

$$\sum_{(m,i) \in A} f_{mi} - \sum_{(i,m) \in A} f_{im} = 0, \forall i \in N \quad \text{(Conservation of Flow)} \quad (1)$$

$$l_{ij} \leq f_{ij} \leq c_{ij}, \forall (i,j) \in A \quad \text{(Capacity constraint)} \quad (2)$$

where $l_{ij}$ 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 $f_{ij}$, $(i,j) \in A$ and $p_i$, $i \in N$ respectively, we can write the corresponding Lagrangian function

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

The dual problem is
maximize \( q(p) \) \hspace{2cm} (3)

subject to no constraints on \( p \),

where the dual functional \( q \) is given by

\[
q(p) = \min_{l_{ij} < f_{ij} < c_{ij}} L(f, p) \hspace{2cm} (4)
\]

\[
= \sum_{(i,j) \in A} \min_{l_{ij} < f_{ij} < c_{ij}} \{ (a_{ij} + p_j - p_i) f_{ij} \} \Delta \sum_{(i,j) \in A} q_{ij}(p_i - p_j)
\]

The form of the dual arc cost functions \( q_{ij} \) is shown in Figure 1.

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

\[
t_{ij} = p_i - p_j, \quad \forall (i,j) \in A \hspace{2cm} (5)
\]

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:

- Inactive if \( t_{ij} < a_{ij} \) \hspace{2cm} (6)
- Balanced if \( t_{ij} = a_{ij} \) \hspace{2cm} (7)
- Active if \( t_{ij} > a_{ij} \) \hspace{2cm} (8)

For any flow vector \( f \) the scalar
Primal cost for arc \((i,j)\)

Dual cost for arc \((i,j)\)

Figure 1: Primal and dual costs for arc \((i,j)\)
\[ d_i = \sum_{m} f_{im} - \sum_{m} f_{mi} \quad \forall i \in N \] (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), (4) state that \((f,p)\) is a primal and dual optimal solution pair if and only if

\[ f_{ij} = \ell_{ij} \quad \text{for all inactive arcs } (i,j) \] (10)

\[ \ell_{ij} \leq f_{ij} \leq c_{ij} \quad \text{for all balanced arcs } (i,j) \] (11)

\[ f_{ij} = c_{ij} \quad \text{for all active arcs } (i,j) \] (12)

\[ d_i = 0 \quad \text{for all nodes } i. \] (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 \[ \sum_{m} \left| d_m \right| \] 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 correspond-
ing 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 Section 3 we describe our relaxation methods. In Section 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 $N$, and has the form $v = \{v_{ij} | (i,j) \in A\}$, where

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

(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

$$C(v,t) = \lim_{\alpha \to 0^+} \frac{q_{ij}(t_{ij} + \alpha v_{ij}) - q_{ij}(t_{ij})}{\alpha}$$

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

(15)

and

$$e_{ij}(v_{ij}, t_{ij}) = \begin{cases} 
-v_{ij}^{i,j} & \text{if } (i,j) \text{ is inactive or if } (i,j) \\
         & \text{is balanced and } v_{ij} < 0 \\
-v_{ij}^{c,i,j} & \text{if } (i,j) \text{ is active or if } (i,j) \\
         & \text{is balanced and } v_{ij} > 0.
\end{cases}$$

(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 nonempty strict subset $S$ of $N$ and every tension vector $t$ there holds

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

(17)

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

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

(18)

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

$$\delta = \inf\{ -a_{im} | i \in S, m \notin S, (i,m): \text{active}],$$

$$\{a_{mi} - t_{mi} | i \in S, m \notin S, (m,i): \text{inactive}] \}. $$

(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 stepsize $\gamma$ 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 $l_{km} < f_{km}$. If $v$ is the vector corresponding to $S$ as in (14) and $C(v,t) > 0$ (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 $l_{kn} < f_{kn}$ for all $(k,n) \in P^+$ and $f_{kn} < c_{kn}$ for all $(k,n) \in P^-$ where

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

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

Let

$$\varepsilon = \min\{d_s - d_m, \{f_{kn} - l_{kn} | (k,n) \in P^+\}, \{c_{kn} - f_{kn} | (k,n) \in P^-\}\}. \quad (23)$$

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

Step 5 (Price Adjustment): Let

$$\delta = \min\{t_{km} - a_{kn} | k \in S, m \notin S, (k,m): \text{ active} \}, \quad \{a_{mk} - t_{mk} | k \in S, m \notin S, (m,k): \text{ inactive} \}. \quad (24)$$
where $S$ is the set of scanned nodes constructed in Step 2. Set

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

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

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

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

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 node labels 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 $f_{km} > l_{km}$. 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 circumstances above, 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 $a_{ij}$, $b_{ij}$, and $c_{ij}$ are integer for all $(i,j) \in A$ 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
\[ C(v,t) = \sum_{i \in S} d_i - \sum_{(i,j) \text{ balanced}} (f_{ij} - \ell_{ij}) - \sum_{(i,j) \text{ balanced}} (c_{ij} - f_{ij}). \]

We note that a similar iteration can be constructed starting from a node with negative deficit. Here the set S consists of nodes with non-positive 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 use instead 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 breakpoints 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.

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

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. 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 slope at $p_s$ is

$$-C(v_s,t)$$

while the right slope is

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

$$+ \sum_{(m,s) \in A} q_{ms} + \sum_{(m,s) \in A} \ell_{ms} (m,s) : \text{active or balanced}$$

We have

$$-C(v_s,t) \leq -d_s \leq -C(v_s,t)$$

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 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 sth coordinate, it may happen that (see Figure 2)

\[
-C(v_s,t) < -d_s < 0 < -C(v_s,t)
\]

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

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 with the case \( d_s > 0 \) is that 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.
Figure 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.
Figure 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).
Figure 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.
Degenerate Ascent Iterations

If, for a given $t$, we can find a connected subset $S$ of $N$ 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.

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

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 $C(v, t) > 0$ 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 non-negative 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 Descriptions and Computational Results

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

\[
\begin{align*}
\text{minimize} & \quad \sum_{(i,j) \in A} a_{ij} f_{ij} \\
\text{subject to} & \quad \sum_{(m,i) \in A} f_{mi} - \sum_{(i,m) \in A} f_{im} = b_i, \quad \forall i \in N \\
& \quad 0 \leq f_{ij} \leq c_{ij} \quad \forall (i,j) \in A.
\end{align*}
\]

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 Section 1 can be reduced to the form above 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 \(|A|\). 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 assistance from Jon Eckstein, is faster than RELAX-II but requires 9.5 \(|A|\) 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 Section 3. Line search and degenerate ascent steps are implemented as discussed in Section 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 (Ashtiani 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 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 Figure 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
Figure 5: Speedup factor of RELAX-II and RELAXT-II over RNET for the transportation problems of Table 6. The normalized dimension D gives the number of nodes N and arcs A as follows:

\[ N = 1000 \times D, \quad A = 4000 \times D, \quad \text{for Problems 6 - 15} \]

\[ N = 500 \times D, \quad A = 5000 \times D, \quad \text{for Problems 16 - 20}. \]
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 in groups and each group has the same average degree per node (8 for Problems 6-15, and 20 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 Figure 6. More experimentation and/or analysis is needed to establish conclusively the computational complexity implications of these experiments.
Figure 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:

$$N = 200 \cdot D, \quad A = 3000 \cdot D.$$ 

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.
<table>
<thead>
<tr>
<th>Problem Type</th>
<th>Problem #</th>
<th># of Nodes</th>
<th># of Arcs</th>
<th>RELAX-II (VMS 3.7/ VMS 4.1)</th>
<th>RELAXT-II (VMS 3.7/ VMS 4.1)</th>
<th>KILTER VMS 3.7</th>
<th>RNET VMS 3.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transportation</td>
<td>1</td>
<td>200</td>
<td>1300</td>
<td>2.07/1.75</td>
<td>1.47/1.22</td>
<td>8.81</td>
<td>3.15</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>200</td>
<td>1500</td>
<td>2.12/1.76</td>
<td>1.61/1.31</td>
<td>9.04</td>
<td>3.72</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>200</td>
<td>2000</td>
<td>1.92/1.61</td>
<td>1.80/1.50</td>
<td>9.22</td>
<td>4.42</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>200</td>
<td>2200</td>
<td>2.52/2.12</td>
<td>2.38/1.98</td>
<td>10.45</td>
<td>4.98</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>200</td>
<td>2900</td>
<td>2.97/2.43</td>
<td>2.53/2.05</td>
<td>16.48</td>
<td>7.18</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>300</td>
<td>3150</td>
<td>4.37/3.66</td>
<td>3.57/3.00</td>
<td>25.08</td>
<td>9.43</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>300</td>
<td>4500</td>
<td>5.46/4.53</td>
<td>3.83/3.17</td>
<td>35.55</td>
<td>12.60</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>300</td>
<td>5155</td>
<td>5.39/4.46</td>
<td>4.30/3.57</td>
<td>46.30</td>
<td>15.31</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>300</td>
<td>6075</td>
<td>6.38/5.29</td>
<td>5.15/4.30</td>
<td>43.12</td>
<td>18.99</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>300</td>
<td>6300</td>
<td>4.12/3.50</td>
<td>3.78/3.07</td>
<td>47.80</td>
<td>16.44</td>
</tr>
<tr>
<td>Total (Problems 1-10)</td>
<td></td>
<td></td>
<td></td>
<td>37.32/31.11</td>
<td>30.42/25.17</td>
<td>251.85</td>
<td>96.22</td>
</tr>
<tr>
<td>Assignment</td>
<td>11</td>
<td>400</td>
<td>1500</td>
<td>1.23/1.03</td>
<td>1.35/1.08</td>
<td>8.09</td>
<td>4.92</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>400</td>
<td>2250</td>
<td>1.38/1.16</td>
<td>1.54/1.25</td>
<td>10.76</td>
<td>6.43</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>400</td>
<td>3000</td>
<td>1.68/1.42</td>
<td>1.87/1.54</td>
<td>8.99</td>
<td>8.92</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>400</td>
<td>3750</td>
<td>2.43/2.07</td>
<td>2.67/2.16</td>
<td>14.52</td>
<td>9.90</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>400</td>
<td>4500</td>
<td>2.79/2.34</td>
<td>3.04/2.46</td>
<td>14.53</td>
<td>10.20</td>
</tr>
<tr>
<td>Total (Problems 11-15)</td>
<td></td>
<td></td>
<td></td>
<td>9.51/8.02</td>
<td>10.47/8.49</td>
<td>56.89</td>
<td>40.37</td>
</tr>
<tr>
<td>Capacitated &amp; Lightly Capacitated Problems</td>
<td>16</td>
<td>400</td>
<td>1306</td>
<td>2.79/2.40</td>
<td>2.60/2.57</td>
<td>13.57</td>
<td>2.76</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>400</td>
<td>2443</td>
<td>2.67/2.29</td>
<td>2.80/2.42</td>
<td>16.89</td>
<td>3.42</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>400</td>
<td>1306</td>
<td>2.56/2.20</td>
<td>2.74/2.39</td>
<td>13.05</td>
<td>2.56</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>400</td>
<td>2443</td>
<td>2.73/2.52</td>
<td>2.83/2.41</td>
<td>17.21</td>
<td>3.61</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>400</td>
<td>1416</td>
<td>2.85/2.40</td>
<td>2.66/2.29</td>
<td>11.88</td>
<td>3.00</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>400</td>
<td>2836</td>
<td>3.80/3.23</td>
<td>3.77/3.23</td>
<td>19.06</td>
<td>4.48</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>400</td>
<td>1416</td>
<td>2.56/2.18</td>
<td>2.82/2.44</td>
<td>12.14</td>
<td>2.36</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>400</td>
<td>2836</td>
<td>4.91/4.24</td>
<td>3.83/3.33</td>
<td>19.65</td>
<td>4.58</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>400</td>
<td>1382</td>
<td>1.27/1.07</td>
<td>1.47/1.27</td>
<td>13.07</td>
<td>2.63</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>400</td>
<td>2676</td>
<td>2.01/1.68</td>
<td>2.13/1.87</td>
<td>26.17</td>
<td>5.34</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>400</td>
<td>1382</td>
<td>1.79/1.57</td>
<td>1.60/1.41</td>
<td>11.31</td>
<td>2.48</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>400</td>
<td>2676</td>
<td>2.15/1.84</td>
<td>1.97/1.75</td>
<td>18.88</td>
<td>3.62</td>
</tr>
<tr>
<td>Total (Problems 16-27)</td>
<td></td>
<td></td>
<td></td>
<td>32.09/27.42</td>
<td>31.22/27.38</td>
<td>192.88</td>
<td>41.94</td>
</tr>
</tbody>
</table>

TABLE 1 (continued on next page)
<table>
<thead>
<tr>
<th>Problem Type</th>
<th>Problem #</th>
<th># of Nodes</th>
<th># of Arcs</th>
<th>RELAX-II (VMS 3.7/ VMS 4.41)</th>
<th>RELAXT-II (VMS 3.7/ VMS 4.1)</th>
<th>KILTER VMS 3.7</th>
<th>RNET VMS 3.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncapacitated and Lightly Capacitated Problems</td>
<td>28</td>
<td>1000</td>
<td>2900</td>
<td>4.90/4.10</td>
<td>5.67/5.02</td>
<td>29.77</td>
<td>8.60</td>
</tr>
<tr>
<td></td>
<td>29</td>
<td>1000</td>
<td>3400</td>
<td>5.57/4.76</td>
<td>5.13/4.43</td>
<td>32.36</td>
<td>12.01</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1000</td>
<td>4400</td>
<td>7.31/6.47</td>
<td>7.18/6.26</td>
<td>42.21</td>
<td>11.12</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>1000</td>
<td>4800</td>
<td>5.76/4.95</td>
<td>7.14/6.30</td>
<td>39.11</td>
<td>10.45</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>1500</td>
<td>4342</td>
<td>8.20/7.07</td>
<td>8.25/7.29</td>
<td>69.28</td>
<td>18.04</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>1500</td>
<td>4302</td>
<td>10.39/8.96</td>
<td>8.94/7.43</td>
<td>63.59</td>
<td>17.29</td>
</tr>
<tr>
<td></td>
<td>34</td>
<td>1500</td>
<td>5107</td>
<td>9.49/8.11</td>
<td>8.88/7.81</td>
<td>72.51</td>
<td>20.50</td>
</tr>
<tr>
<td></td>
<td>35</td>
<td>1500</td>
<td>5730</td>
<td>10.95/9.74</td>
<td>10.52/9.26</td>
<td>67.49</td>
<td>17.81</td>
</tr>
<tr>
<td>Total (Problems 28-35)</td>
<td></td>
<td></td>
<td></td>
<td>62.57/54.16</td>
<td>61.71/53.80</td>
<td>356.32</td>
<td>115.82</td>
</tr>
<tr>
<td>Large Uncapacitated Problems</td>
<td>37</td>
<td>5000</td>
<td>23000</td>
<td>87.05/73.64</td>
<td>74.67/66.66</td>
<td>681.94</td>
<td>281.87</td>
</tr>
<tr>
<td></td>
<td>38</td>
<td>3000</td>
<td>35000</td>
<td>68.25/57.84</td>
<td>55.84/47.33</td>
<td>607.89</td>
<td>274.46</td>
</tr>
<tr>
<td></td>
<td>39</td>
<td>5000</td>
<td>15000</td>
<td>89.83/75.17</td>
<td>66.23/58.74</td>
<td>558.60</td>
<td>151.00</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>3000</td>
<td>23000</td>
<td>50.42/42.73</td>
<td>35.91/30.56</td>
<td>369.40</td>
<td>174.74</td>
</tr>
<tr>
<td>Total (Problems 37-40)</td>
<td></td>
<td></td>
<td></td>
<td>295.55/249.38</td>
<td>252.65/203.29</td>
<td>2,217.83</td>
<td>882.07</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>#</th>
<th>Problem Type</th>
<th># of Sources</th>
<th># of Sinks</th>
<th># of Arcs</th>
<th>Cost Range</th>
<th>Total Supply</th>
<th>RELAX-II</th>
<th>RELAXT-II</th>
<th>RNET</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Assignment</td>
<td>1,000</td>
<td>1,000</td>
<td>8,000</td>
<td>1-10</td>
<td>1,000</td>
<td>4.68</td>
<td>4.60</td>
<td>79.11</td>
</tr>
<tr>
<td>2</td>
<td>Assignment</td>
<td>1,500</td>
<td>1,500</td>
<td>12,000</td>
<td>1-10</td>
<td>1,500</td>
<td>7.23</td>
<td>7.03</td>
<td>199.44</td>
</tr>
<tr>
<td>3</td>
<td>Assignment</td>
<td>2,000</td>
<td>2,000</td>
<td>16,000</td>
<td>1-10</td>
<td>2,000</td>
<td>12.65</td>
<td>9.95</td>
<td>313.64</td>
</tr>
<tr>
<td>4</td>
<td>Assignment</td>
<td>1,000</td>
<td>1,000</td>
<td>8,000</td>
<td>1-1,000</td>
<td>1,000</td>
<td>9.91</td>
<td>10.68</td>
<td>118.60</td>
</tr>
<tr>
<td>5</td>
<td>Assignment</td>
<td>1,500</td>
<td>1,500</td>
<td>12,000</td>
<td>1-1,000</td>
<td>1,500</td>
<td>17.82</td>
<td>14.58</td>
<td>227.57</td>
</tr>
<tr>
<td>6</td>
<td>Transportation</td>
<td>1,000</td>
<td>1,000</td>
<td>8,000</td>
<td>1-10</td>
<td>100,000</td>
<td>31.43</td>
<td>27.83</td>
<td>129.95</td>
</tr>
<tr>
<td>7*</td>
<td>Transportation</td>
<td>1,500</td>
<td>1,500</td>
<td>12,000</td>
<td>1-10</td>
<td>153,000</td>
<td>60.86</td>
<td>56.20</td>
<td>300.79</td>
</tr>
<tr>
<td>8*</td>
<td>Transportation</td>
<td>2,000</td>
<td>2,000</td>
<td>16,000</td>
<td>1-10</td>
<td>220,000</td>
<td>127.73</td>
<td>99.69</td>
<td>531.14</td>
</tr>
<tr>
<td>9*</td>
<td>Transportation</td>
<td>2,500</td>
<td>2,500</td>
<td>20,000</td>
<td>1-10</td>
<td>275,000</td>
<td>144.66</td>
<td>115.65</td>
<td>790.57</td>
</tr>
<tr>
<td>10*</td>
<td>Transportation</td>
<td>3,000</td>
<td>3,000</td>
<td>24,000</td>
<td>1-10</td>
<td>330,000</td>
<td>221.81</td>
<td>167.49</td>
<td>1,246.45</td>
</tr>
<tr>
<td>11</td>
<td>Transportation</td>
<td>1,000</td>
<td>1,000</td>
<td>8,000</td>
<td>1-1,000</td>
<td>100,000</td>
<td>32.60</td>
<td>31.99</td>
<td>152.17</td>
</tr>
<tr>
<td>12*</td>
<td>Transportation</td>
<td>1,500</td>
<td>1,500</td>
<td>12,000</td>
<td>1-1,000</td>
<td>153,000</td>
<td>53.84</td>
<td>54.32</td>
<td>394.12</td>
</tr>
<tr>
<td>13*</td>
<td>Transportation</td>
<td>2,000</td>
<td>2,000</td>
<td>16,000</td>
<td>1-1,000</td>
<td>220,000</td>
<td>101.97</td>
<td>71.85</td>
<td>694.32</td>
</tr>
<tr>
<td>14*</td>
<td>Transportation</td>
<td>2,500</td>
<td>2,500</td>
<td>20,000</td>
<td>1-1,000</td>
<td>275,000</td>
<td>107.93</td>
<td>96.71</td>
<td>1,030.35</td>
</tr>
<tr>
<td>15*</td>
<td>Transportation</td>
<td>3,000</td>
<td>3,000</td>
<td>24,000</td>
<td>1-1,000</td>
<td>330,000</td>
<td>133.85</td>
<td>102.93</td>
<td>1,533.50</td>
</tr>
<tr>
<td>16+</td>
<td>Transportation</td>
<td>500</td>
<td>500</td>
<td>10,000</td>
<td>1-100</td>
<td>15,000</td>
<td>16.44</td>
<td>11.43</td>
<td>84.04</td>
</tr>
<tr>
<td>17+</td>
<td>Transportation</td>
<td>750</td>
<td>750</td>
<td>15,000</td>
<td>1-100</td>
<td>22,500</td>
<td>28.30</td>
<td>18.12</td>
<td>176.55</td>
</tr>
<tr>
<td>18+</td>
<td>Transportation</td>
<td>1,000</td>
<td>1,000</td>
<td>20,000</td>
<td>1-100</td>
<td>30,000</td>
<td>51.01</td>
<td>31.31</td>
<td>306.97</td>
</tr>
<tr>
<td>19+</td>
<td>Transportation</td>
<td>1,250</td>
<td>1,250</td>
<td>25,000</td>
<td>1-100</td>
<td>37,500</td>
<td>71.61</td>
<td>38.96</td>
<td>476.57</td>
</tr>
<tr>
<td>20+</td>
<td>Transportation</td>
<td>1,500</td>
<td>1,500</td>
<td>30,000</td>
<td>1-100</td>
<td>45,000</td>
<td>68.09</td>
<td>41.03</td>
<td>727.38</td>
</tr>
</tbody>
</table>

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 the 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.
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 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], [11], [12], [13]) including general linear programming problems. It also lends itself to distributed or parallel computation (see [14], [10], [15], [13]).

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


