RELAXATION METHODS FOR MINIMUM COST ORDINARY AND GENERALIZED NETWORK FLOW PROBLEMS

DIMITRI P. BERTSEKAS and PAUL TSENG

Massachusetts Institute of Technology, Cambridge, Massachusetts

(Received May 1985; revision received September 1986; accepted June 1987)

We propose a new class of algorithms for linear cost network flow problems with and without gains. These algorithms are based on iterative improvement of a dual cost and operate in a manner that is reminiscent of coordinate ascent and Gauss-Seidel relaxation methods. We compare our coded implementations of these methods with mature state-of-the-art primal simplex and primal-dual codes, and find them to be several times faster on standard benchmark problems, and faster by an order of magnitude on large, randomly generated problems. Our experiments indicate that the speedup factor increases with problem dimension.

In this paper, we are concerned with solving algorithmically both the classical linear minimum cost flow problem (also known as the transshipment problem) and its generalized version, which involves a network with gains. This problem, together with its various special cases (assignment, transportation, shortest-path, max-flow), arises very often in practice. It is probably the most frequently solved problem in optimization, and is discussed in numerous texts on linear and network programming (see, for example, Ford and Fulkerson 1962; Minieka 1978; Kennington and Helgason 1980; Papadimitriou and Steiglitz 1982; and Rockafellar 1984). In this introductory section, we formulate the problem and its dual, and provide a conceptual overview of the methods for its solution.

Consider a directed graph with set of nodes $\mathcal{N}$ and set of arcs $\mathcal{A}$. Each arc (i, j) has associated with it a scalar 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 following problem.

Problem MCF

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

subject to

$$\sum_{\substack{m \\ (m,i)\in\mathcal{A}}} K_{mi}f_{mi} - \sum_{\substack{m \\ (i,m)\in\mathcal{A}}} f_{im} = 0, \qquad (1)$$

for all $i \in \mathcal{N}$ (conservation of flow)

$$l_{ij} \leq f_{ij} \leq c_{ij}, \qquad (2)$$

for all $(i, j) \in \mathcal{A}$, (capacity constraint)

where K_{ij}, l_{ij} and c_{ij} are given scalars. We refer to the scalar K_{ij} as the *gain* of arc (i, j). Our main focus is in the *ordinary network case* in which $K_{ij} = 1$ for all $(i, j) \in \mathcal{A}$. We will also treat in parallel the *gain network case*, where K_{ij} can differ from unity. We assume throughout that there exists at least one feasible solution to (MCF). For simplicity, we also assume that at most one arc connects any pair of nodes, so that the arc (i, j) has unambiguous meaning. This restriction can be easily removed, and indeed, our computer codes allow for multiple arcs between nodes. Finally, we assume for simplicity that $K_{ij} > 0$ for all $(i, j) \in \mathcal{A}$. Our algorithm can be extended for the case where $K_{ij} \leq 0$ for some $(i, j) \in \mathcal{A}$, but the corresponding algorithmic description becomes complicated.

We formulate a dual problem to (MCF). We associate a Lagrange multiplier p_i (referred to as the *price* of node i) with the ith conservation of flow constraint (1). 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,

$$L(f, p) = \sum_{(i,j)\in\mathcal{A}} a_{ij}f_{ij} + \sum_{i\in\mathcal{N}} p_i\left(\sum_{\substack{m \\ (m,i)\in\mathcal{A}}} K_{mi}f_{mi} - \sum_{\substack{m \\ (i,m)\in\mathcal{A}}} f_{im}\right)$$

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

The dual problem is

$$\text{maximize} \quad q(p) \qquad (3)$$

subject to no constraints on p,

Subject classification: 484 optimization of single commodity network flows, 643 algorithms for linear network optimization.

0030-364X/88/3601-0093 $01.25

with the dual functional q given by

$$q(p) = \min_{l_{ij} \le f_{ij} \le c_{ij}} L(f, p)$$

$$= \sum_{(i,j) \in \mathcal{A}} q_{ij}(p_i - K_{ij}p_j), \tag{4}$$

where

$$q_{ij}(p_i - K_{ij}p_j) = \min_{l_{ij} \le f_{ij} \le c_{ij}} \{(a_{ij} + K_{ij}p_j - p_i)f_{ij}\}$$

$$= \begin{cases} (a_{ij} - t_{ij})c_{ij} & \text{if } t_{ij} \ge a_{ij} \\ (a_{ij} - t_{ij})l_{ij} & \text{if } t_{ij} \le a_{ij} \end{cases} \tag{5}$$

$$t_{ij} = p_i - K_{ij}p_j, \quad \text{for all } (i, j) \in \mathcal{A}. \tag{6}$$

Figure 1 shows the function $q_{ij}(p_i - K_{ij}p_j)$. This is a classical duality framework, treated extensively in Rockafellar (1970, 1984).

The vector t having elements t_{ij}, $(i, j) \in \mathcal{A}$ given by (6) is called the *tension vector* that corresponds to p. Since the dual functional depends on the price vector p only through the corresponding tension vector t, we will often make no distinction between p and t in the following discussion.

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

inactive if $t_{ij} < a_{ij}$ (7)

balanced if $t_{ij} = a_{ij}$ (8)

active if $t_{ij} > a_{ij}$. (9)

For any flow vector f, we will refer to the scalar

$$d_i = \sum_{\substack{m \\ (i,m) \in \mathcal{A}}} f_{im} - \sum_{\substack{m \\ (m,i) \in \mathcal{A}}} K_{mi} f_{mi} \quad \text{for all } i \in \mathcal{N} \tag{10}$$

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

$$f_{ij} = l_{ij} \quad \text{for all inactive arcs } (i, j) \tag{11}$$

$$l_{ij} \le f_{ij} \le c_{ij} \quad \text{for all balanced arcs } (i, j) \tag{12}$$

$$f_{ij} = c_{ij} \quad \text{for all active arcs } (i, j) \tag{13}$$

$$d_i = 0 \quad \text{for all nodes } i. \tag{14}$$

Conditions (11)–(13) are the *complementary slackness conditions*.

The approach of the present paper 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 we propose terminate when f satisfies primal feasibility (deficit of each node equals zero). The main feature of the algorithms that 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 we cannot effect a reduction of the total absolute deficit $\sum_m |d_m|$ through flow augmentation, we choose an adjacent node of i, say i_1, and attempt an ascent 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, we choose an adjacent node of either i or i_1 and continue the process. In practice,

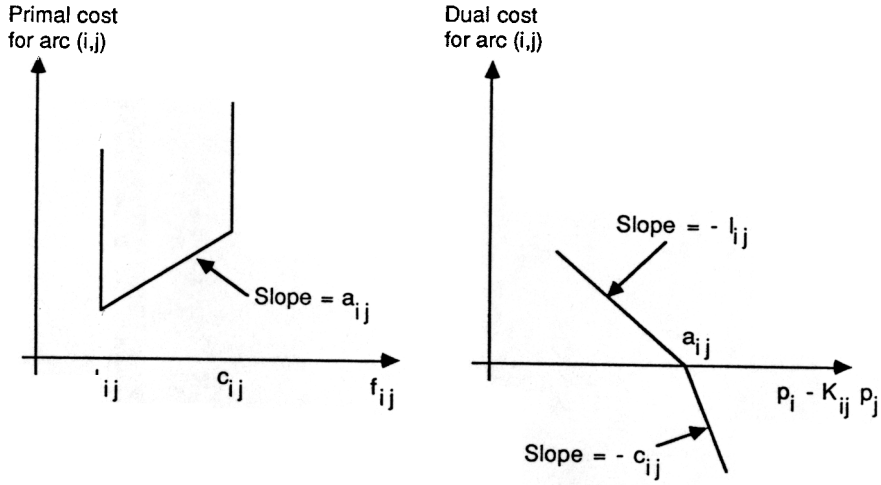


Figure 1. Primal and dual costs for arc (i, j).

most of the ascent directions are single coordinate directions, leading to the interpretation of the algorithms as *coordinate ascent* or *relaxation* methods. This characteristic is an important one, and is 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 (which is the one used by the classical primal-dual method (Bertsekas and Tseng 1983), but can be computed with considerably less overhead).

In the next section we characterize the ascent directions used in the algorithms. In Section 2 we describe our relaxation methods, and in Section 3 we prove their convergence using the novel notion of ϵ-complementary slackness. This notion is also important in other contexts (see Bertsekas 1986, 1988; Bertsekas, Hosein and Tseng 1987; and Bertsekas and Eckstein 1987). Sections 4 through 6 are devoted to computational comparisons of our experimental relaxation codes with mature state-of-the-art codes based on the primal simplex and primal-dual methods. A clear conclusion is that relaxation outperforms by an overwhelming margin the primal-dual method. Comparisons with primal simplex show that on standard benchmark problems relaxation is much faster (by as much as four times) than primal simplex. For large, randomly generated problems, the factor of superiority increases to an order of magnitude, indicating a superior average computational complexity for the relaxation method, and even larger speedup for larger problems.

The algorithm of this paper for the ordinary network case was first given in Bertsekas (1985) where the conceptual similarity to relaxation methods was also pointed out. The present paper considers in addition gain networks, emphasizes the dual ascent viewpoint and provides computational results. Bertsekas (1981) considers a special case for the assignment problem. Bertsekas and Tseng in an early version of the present paper, and Tseng (1986), present additional computational results and analysis. Bertsekas, Hosein and Tseng, and Bertsekas and El Baz (1987), consider the relaxation algorithm for strictly convex arc cost problems. In this case the algorithm is equivalent to the classical coordinate ascent method for unconstrained maximization, but there are some noteworthy convergence aspects of the algorithm, including convergence in a distributed, totally asynchronous computational environment. Tseng and Bertsekas (1987a,b) extend the relaxation algorithms of this paper and apply them to the general linear programming problem and to general convex programming problems with separable cost and linear constraints. Bertsekas (1986) and Bertsekas and Eckstein give a massively parallelizable relaxation algorithm for the linear cost problem (MCF) that has a strong conceptual relation with the one given in this paper. Goldberg and Tarjan (1986) give a related parallel max-flow algorithm (see also Ahuja and Orlin 1986; Goldberg and Tarjan 1987; and Bertsekas and Eckstein).

1. Characterization of Dual Ascent Directions

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

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

Changing any tension vector t in the direction v of (15) corresponds to decreasing the prices of all nodes in S by an equal amount while leaving the prices of all other nodes unchanged. From (5), we see that the directional derivative at t of the dual cost in the direction v is $C(v, t)$ where

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

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

and

$$e_{ij}(v_{ij}, t_{ij}) = \begin{cases} -v_{ij}l_{ij} & \text{if } (i, j) \text{ is inactive or if } (i, j) \\ & \quad \text{is balanced and } v_{ij} \leq 0 \\ -v_{ij}c_{ij} & \text{if } (i, j) \text{ is active or if } (i, j) \\ & \quad \text{is balanced and } v_{ij} \geq 0. \end{cases}$$

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 ordinary networks). *For every nonempty strict subset S of $\mathcal{N}$ and every tension vector t,*

$$w(t + \gamma v) = w(t) + \gamma C(v, t), \quad \textit{for all } \gamma \in [0, \delta),$$

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

$$w(t) = \sum_{(i,j)} q_{ij}(t_{ij}). \tag{19}$$

Here v is given by (15), *and δ is given by*

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

$$\{a_{mi} - t_{mi} \mid i \in S,\ m \notin S,\ (m, i)\text{: inactive}\}\}. \tag{20}$$

(We use the convention $\delta = +\infty$ if the set over which the infimum in (20) is taken is empty.)

Proof. We saw in Equation 16 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 γ up to the point where γ 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 γ is the one for which a new arc incident to S becomes balanced, and it equals the scalar δ of (20).

In the same manner, we can compute, for a gain network, directional derivatives for dual directions used by the algorithm. For purposes of future reference, we note here that the gain of a directed cycle Y with one direction arbitrarily chosen as positive is defined as

$$\left(\prod_{(i,j)\in Y^+} K_{ij}\right) \Big/ \left(\prod_{(i,j)\in Y^-} K_{ij}\right),$$

where Y^+ and Y^- are the portions of the cycle oriented along the positive and negative directions, respectively.

Given any connected subset of nodes S, and a set of arcs T forming a spanning tree for S, let $\{u_i \mid i \in S\}$ be a set of positive numbers such that

$$u_i - K_{ij}u_j = 0, \quad \text{for all } (i, j) \in T. \tag{21}$$

(Such a set of numbers is unique modulo multiplication with a positive scalar and can be obtained by assigning a positive number u_s to an arbitrary node $s \in S$, and determining the numbers u_i of the remaining nodes $i \in S$ from (21).) Each dual ascent direction used by the algorithm is associated with a pair (S, T) as just defined and is given by $v = \{v_{ij} \mid (i, j) \in \mathcal{A}\}$, where

$$v_{ij} = \begin{cases} K_{ij}u_j & \text{if } i \notin S, j \in S \\ -u_i & \text{if } i \in S, j \notin S \\ K_{ij}u_j - u_i & \text{if } i \in S, j \in S \\ 0 & \text{otherwise.} \end{cases} \tag{22}$$

Changing any tension vector t in the direction v corresponds to decreasing the price of each node i in S by an amount proportional to u_i while keeping the prices of all other nodes unchanged. From (5), (16) and (17) we see that the directional derivative of the dual cost at t along a vector v is again given by

$$C(v, t) = \sum_{(i,j)\in\mathcal{A}} e_{ij}(v_{ij}, t_{ij}). \tag{23}$$

We state the corresponding result as a proposition, but omit the proof since it is entirely analogous to that of Proposition 1.

Proposition 2 (for gain networks). *For every vector v defined by* (21), (22), *and every tension vector t,*

$$w(t + \gamma v) = w(t) + \gamma C(v, t), \quad \text{for all } \gamma \in [0, \delta), \tag{24}$$

where $w(\cdot)$ is the dual cost function given by (19), *and δ is given by*

$$\delta = \inf\left\{\left\{\frac{t_{im} - a_{im}}{v_{im}} \;\middle|\; v_{im} > 0,\ (i, m)\text{: active}\right\},\right.$$

$$\left.\left\{\frac{t_{mi} - a_{mi}}{v_{mi}} \quad v_{mi} < 0,\ (m, i)\text{: inactive}\right\}\right\}.$$

2. Relaxation Methods

In this section we provide an algorithm that implements the idea of dual ascent. The main difference from the classical primal-dual method is that, instead of trying to find the direction of maximal rate of ascent through a labeling process, 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) that satisfy complementary slackness by means of the following process.

Step 1. Choose a node s with $d_s > 0$. (The iteration can also be started 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, $(k \notin S)$, set $S = S \cup \{k\}$, and go to Step 3.

Step 3. Scan the label of 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 (15) and

$$C(v, t) > 0, \tag{26}$$

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. We construct the path by tracing labels backwards starting from m; it 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

$$P^+ = \{(k, n) \in P \mid (k, n) \text{ is oriented in the direction from } s \text{ to } m\} \tag{27}$$

$$P^- = \{(k, n) \in P \mid (k, n) \text{ is oriented in the direction from } m \text{ to } s\}. \tag{28}$$

Let

$$\epsilon = \min\{d_s, -d_m, \{f_{kn} - l_{kn} \mid (k, n) \in P^+\}, \{c_{kn} - f_{kn} \mid (k, n) \in P^-\}\} \tag{29}$$

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

Step 5 (Price Adjustment). Let

$$\delta = \min\{\{t_{km} - a_{km} \mid k \in S, m \notin S, (k, m) \text{: active}\}, \{a_{mk} - t_{mk} \mid k \in S, m \notin S, (m, k) \text{: inactive}\}\}, \tag{30}$$

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

$$f_{km} := l_{km},$$

for all balanced arcs (k, m) with $k \in S$, $m \in L$, $m \notin S$

$$f_{mk} := c_{mk},$$

for all 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 either with 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 that 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 definition (16), (17) (see also Equation 31), that

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

Under the circumstances just described, all nodes in S have a nonnegative deficit and at least one node in S (the starting node s) has a 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}, l_{ij}, and c_{ij} are integers for all $(i, j) \in \mathscr{A}$ and the starting t is an integer, then δ as given by (30) 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 an integer, all successive f's will be integers, so the amount of flow augmentation ϵ 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 an integer. If the problem data are not integers, it is necessary to introduce modifications in the algorithm in order to guarantee termination in a finite number of iterations to an ϵ-optimal solution—see Section 3.

It can be seen that the relaxation iteration involves an amount of computation per node scanned that is comparable to the usual primal-dual method (Ford and Fulkerson 1962). The only additional computation involves maintaining the quantity $C(v, t)$, but, with a little thought, we can see that this quantity 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

$$C(v, t) = \sum_{i \in S} d_i - \sum_{\substack{(i,j) \text{: balanced} \\ i \in S, j \notin S}} (f_{ij} - l_{ij}) \sum_{\substack{(i,j) \text{: balanced} \\ i \notin S, j \in S}} (c_{ij} - f_{ij}). \tag{31}$$

We note that a similar iteration can be constructed starting from a node with negative deficit. Here the set S consists of nodes with a nonpositive deficit, and in Step 5 the prices of the nodes in S are increased rather than decreased. We leave the details, which are straightforward, 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.

Typical Relaxation Iteration for a Gain Network

The relaxation iteration for gain networks is more complex because the starting node deficit may be reduced not just by augmenting flow along a path (see Step 4 earlier), but also by augmenting flow along a cycle of nonunity gain (Step 4b in the following algorithm). Furthermore, in order to identify the existence of such a cycle, it is necessary to occasionally restructure the tree of labels (Steps 2a and 2b in the following algorithm). These devices are also used in the classical primal-dual algorithm for gain networks—see Jewell (1962).

The main idea of the iteration can be motivated by considering the generalized version of (31). Consider a pair (S, T) where S is a connected subset of nodes and T is a spanning tree for S. Let $\{u_i \mid i \in S\}$ be a set of positive numbers such that

$$u_i - K_{ij}u_j = 0, \quad \text{for all } (i, j) \in T, \tag{32}$$

and let v be the corresponding vector given by (22). Let (f, t) be any pair satisfying complementary slackness. Then a straightforward calculation using the definitions (10) and (21)–(23) shows that

$$\begin{aligned} C(v, t) = {} & \sum_{i \in S} u_i d_i - \sum_{\substack{(i,j)\text{: balanced} \\ i \in S, j \notin S}} (f_{ij} - l_{ij})u_i \\ & - \sum_{\substack{(i,j)\text{: balanced} \\ i \notin S, j \in S}} (c_{ij} - f_{ij})K_{ij}u_j \\ & \sum_{\substack{(i,j)\text{: balanced} \\ i \in S, j \in S \\ u_i > K_{ij}u_j}} (f_{ij} - l_{ij})(u_i - K_{ij}u_j) \\ & \sum_{\substack{(i,j)\text{: balanced} \\ i \in S, j \in S \\ u_i < K_{ij}u_j}} (c_{ij} - f_{ij})(K_{ij}u_j - u_i). \end{aligned} \tag{33}$$

This equation generalizes (31) since for an ordinary network we have $K_{ij} = 1$ for all (i, j), so from (32) we can take $u_i = 1$ for all $i \in S$, and (33) reduces to (31). Note here that, in contrast with ordinary networks, different spanning trees of the node subset S can be associated with different vectors u and v having different values of $C(v, t)$. Similarly, as for ordinary networks, the relaxation iteration starts from a node with positive deficit and gradually builds a set of nodes S until either a flow augmentation occurs that reduces the deficit of the starting node, or the ascent condition $C(v, t) > 0$ is obtained. The main complication is that when a new node is added to the set S, the corresponding tree T is modified until either an augmentation occurs or the last two terms in (33) become zero (Steps 2a and 2b in the following algorithm). In the process of reducing the last two terms in (33) to zero, we see that the corresponding value of $\sum_{i \in S} u_i$ increases monotonically, which is important for proving termination in a finite number of operations. Finally, because the tree corresponding to each successive subset S is constructed so that the last two terms in (33) become zero, it again follows (see Equation 31) that the algorithm will always find either a flow augmentation or an ascent direction v with $C(v, t) > 0$.

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 $d_s \neq 0$. We assume in the following steps that $d_s > 0$. The case in which $d_s < 0$ is entirely similar. If no such node can be found, terminate the algorithm. Else set $S = \{s\}$, $T = \{\varnothing\}$, and go to Step 2.

Step 2 (Tree Restructuring). Construct the unique vector u satisfying

$$u_s = 1, \quad u_i - K_{ij}u_j = 0, \quad \text{for all } (i, j) \in T, \; u_i = 0, \quad \forall i \notin S.$$

Choose a balanced arc (i, j) such that $i \in S$, $j \in S$, $(i, j) \notin T$, and either

(a) $u_i - K_{ij}u_j > 0, \quad f_{ij} > l_{ij}$

or

(b) $u_i - K_{ij}u_j < 0, \quad f_{ij} < c_{ij}$.

If such an arc cannot be found, go to Step 3. Else go to Step 2a or to Step 2b depending on whether case (a) or (b) holds.

Step 2a. Let Y be the cycle formed by T and the arc (i, j) identified in Step 2. The cycle Y is connected with s by a path P consisting of arcs belonging to T (see Figure 2). Let w be the node that is common to P and Y. (Note that P may be empty, in which case $s = w$.) Let Y_{jw} be the set of arcs of Y on the undirected

path from j to w that does not contain node i (see Figure 2). There are two possibilities:

(1) We have $l_{km} < f_{km} < c_{km}$ for all $(k, m) \in Y_{jw}$. Then flow can be pushed around the cycle Y in the direction opposite to that of the arc (i, j) (Y is a flow generating cycle)—go to Step 4b; or

(2) There exists $(k, m) \in Y_{jw}$ such that $f_{km} = l_{km}$ or $f_{km} = c_{km}$. Then let $(\bar{k}, \bar{m})$ be the closest such arc to (i, j), remove $(\bar{k}, \bar{m})$ from T, add (i, j) to T, and go to Step 2.

Step 2b. Same as Step 2a except that in place of Y_{jw} we use Y_{iw}—the portion of Y from i to w along the direction opposite to the arc (i, j).

Step 3. If v is the vector corresponding to u, S, and T as in (22) and

$$C(v, t) > 0, \tag{34}$$

go to Step 5. Otherwise, choose nodes $i \in S$, $m \notin S$ such that either (a) (i, m) is a balanced arc with $f_{im} > l_{im}$, or (b) (m, i) is a balanced arc with $f_{mi} < c_{mi}$. If $d_m < 0$, go to Step 4a. Else, add to S node m, and add to T arc (i, m) or arc (m, i) depending on whether (a) or (b) above holds. Go to Step 2.

Step 4a (Flow Augmentation Involving a Simple Path). 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. Let P^+ and P^- be given by (27), (28). Let

$$\epsilon = \min\{d_s, -u_m d_m, \{(f_{kn} - l_{kn})u_k \mid (k, n) \in P^+\}, \{(c_{kn} - f_{kn})u_k \mid (k, n) \in P^-\}\}. \tag{35}$$

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

Step 4b (Flow Augmentation Involving a Cycle). From Step 2a or 2b, an arc (i, j) and a cycle Y connected to s by a simple path P are identified (Figure 2). Let P^+ and P^- be defined as in (27), (28).

If case (a) holds in Step 2, then set

$$q := \frac{K_{ij}u_j}{u_i}, \quad \bar{q} = 1 - q,$$

$u_k := u_k/q$, for all nodes k on Y_{jw} except for w,

Y^+: Set of arcs of Y oriented in the direction opposite to (i, j),

Y^-: The complement of Y^+ relative to Y;

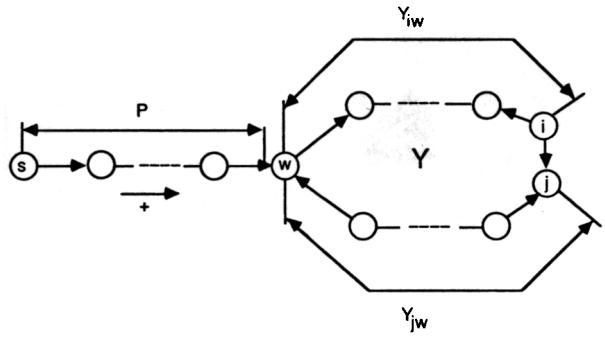


Figure 2. Cycle Y formed by tree T and arc (i, j) identified in Step 2.

else set

$$q := \frac{u_i}{K_{ij}u_j}, \quad \bar{q} := 1 - q,$$

$u_k := u_k/q$, for all nodes k on Y_{iw} except for w,

Y^+: Set of arcs of Y oriented in the direction of (i, j),

Y^-: The complement of Y^+ relative to Y.

Let

$$\epsilon_1 = \min\{\{(f_{kn} - l_{kn})u_k \mid (k, n) \in P^+\}, \{(c_{kn} - f_{kn})u_k \mid (k, n) \in P^-\}\},$$

$$\epsilon_2 = \min\{\{(f_{kn} - l_{kn})u_k \mid (k, n) \in Y^-\}, \{(c_{kn} - f_{kn})u_k \mid (k, n) \in Y^+\}\},$$

$$\epsilon = \min\{\epsilon_1, \bar{q}\epsilon_2, d_s\}.$$

Decrease by ϵ/u_k the flow of all arcs $(k, n) \in P^+$, increase by ϵ/u_k the flow of all arcs $(k, n) \in P^-$, decrease by $\epsilon/(\bar{q}u_k)$ the flow of all arcs $(k, n) \in Y^-$, increase by $\epsilon/(\bar{q}u_k)$ the flow of all arcs $(k, n) \in Y^+$, and go to the next iteration.

Step 5 (Price Adjustment). If v is the vector corresponding to u, S, and T as in (22), let

$$\delta = \min\left\{\left\{\frac{t_{im} - a_{im}}{v_{im}} \quad v_{im} > 0, (i, m)\text{: active}\right\}, \left\{\frac{t_{mi} - a_{mi}}{v_{mi}} \quad v_{mi} < 0, (m, i)\text{: inactive}\right\}\right\}. \tag{36}$$

Set

$f_{mk} := l_{km}$,

for all balanced arcs (k, m) with $k \in S$, $m \notin S$, or $k \in S$, $m \in S$ and $v_{km} > 0$

$f_{mk} := c_{km}$,

for all balanced arcs (m, k) with $k \in S$, $m \notin S$, or $k \in S$, $m \in S$ and $v_{mk} < 0$.

$t_{ij} := t_{ij} - \delta v_{ij}$, for any $(i, j) \in \mathscr{A}$.

Go to the next iteration.

The description of the iteration is quite complex, and thus we have avoided introducing features and data structures that would improve efficiency of implementation at the expense of further complicating the description. For example, the tree T and the vector u in Step 2 can be maintained and updated efficiently by means of a labeling scheme. Furthermore, the value of $C(v, t)$ can be efficiently updated using a labeling scheme and (33).

The iteration can terminate in two ways; either via a flow augmentation (Steps 4a and 4b), in which case the total absolute deficit is reduced, or else via a price adjustment (Step 5), in which case (by Proposition 2) the dual functional is increased. In order to guarantee that the iteration will terminate, however, it is necessary to show that we will not have indefinite cycling within Step 2 and that Step 3 can be properly carried out. What is happening in Step 2 is that the tree T corresponding to the set S is successively restructured so that all balanced arcs $(i, j) \notin T$ with $i \in S$, $j \in S$ and either (a) $u_i - K_{ij}u_j > 0$, $f_{ij} > l_{ij}$, or (b) $u_i - K_{ij}u_j < 0$, $f_{ij} < c_{ij}$, are eliminated (in which case the last two terms in (33) will be zero). Each time we enter Step 2a or 2b, either (1) an augmentation occurs (in which case Step 2 is exited), or (2) the offending arc (i, j) satisfying (a) or (b) enters the tree while another arc exits it, and the vector u is suitably updated in Step 2. It is clear that in the latter case, no scalar u_k, $k \in S$ will be decreased, while at least one scalar u_k will be strictly increased (u_j in case (a), or u_i in case (b)). Therefore the sum $\sum_{k \in S} u_k$ will be strictly increased each time we return from Step 2a or 2b to Step 2. In view of the fact that u_s remains fixed at unity, this implies that a tree cannot be reencountered within Step 2 and shows that, within a finite number of operations, we will exit Step 2 in order to either perform an augmentation in Step 4b, or to check the condition $C(v, t) > 0$ in Step 3. In the latter case, the two terms in (33) will be zero. With this in mind, we see, using (33), that if the condition $C(v, t) > 0$ fails, then there must exist nodes i and m with the property described in Step 3. It follows that the relaxation iteration is well defined and will terminate via Step 4a, 4b, or 5 in a finite number of arithmetic operations.

3. Convergence Analysis and Algorithmic Variations

The relaxation algorithm, consisting of successive iterations of the type described in the previous section, is not guaranteed to generate an optimal dual solution when applied to a gain network or to an ordinary network with irrational data. There are two potential difficulties:

(a) Only a finite number of dual ascent steps take place because all iterations after a finite number end up with a flow augmentation.
(b) When an infinite number of dual ascent steps are performed, the generated sequence of dual function values converges short of the optimal.

We can bypass difficulty (a) in the case of an ordinary network with irrational problem data by scanning nodes in Step 3 in a first labeled-first scanned mode (breadth-first). Tseng gives a proof of this fact and also gives an example showing that this device is inadequate for gain networks. The alternative is to employ an arc discrimination device in selecting the balanced arc (i, j) in Step 2 and the nodes i and m in Step 3, whereby arcs with flow strictly between the upper and the lower bound are given priority over other arcs (see Johnson 1966; Minieka; and Rockafellar 1984, pp. 36 and 66). With this device one can show (see Tseng) that an infinite number of successive augmentations cannot occur. In the subsequent discussion of convergence we will assume that this device is employed.

Difficulty (b) can occur, as Tseng shows in an example. It can be bypassed by employing an idea from the ϵ-subgradient method (Bertsekas and Mitter 1971, 1973). For any positive number ϵ and any tension vector t, define each arc (i, j) to be

ϵ-inactive if $t_{ij} < a_{ij} - \epsilon$ (37)

ϵ-balanced if $a_{ij} - \epsilon \leq t_{ij} \leq a_{ij} + \epsilon$ (38)

ϵ-active if $a_{ij} + \epsilon < t_{ij}$. (39)

We will show that *if in the relaxation iteration the usual notions of active, inactive, and balanced arcs are replaced by the corresponding notions* (37), (38) *and* (39) *just defined, the algorithm will terminate in a finite number of iterations with a solution that is within*

$\epsilon \sum_{(i,j)} (c_{ij} - l_{ij})$ *of being optimal.* Furthermore, *the final primal solution will be optimal if* ϵ *is chosen sufficiently small.*

We say that a pair (f, t) satisfies *ϵ-complementary slackness* if (11)–(14) hold, but with the usual definitions of active, inactive and balanced arcs replaced by those of (37)–(39). Suppose that we have such a pair which also satisfies primal and dual feasibility. Since f is primal feasible, we see by multiplying (1) with p_i and adding over i that $\sum_{(i,j)} t_{ij} f_{ij} = 0$, so the primal cost associated with f satisfies (see (4) and (5))

$$\begin{aligned}\sum_{(i,j)} a_{ij} f_{ij} &= \sum_{(i,j)} (a_{ij} - t_{ij}) f_{ij} \\ &\geq \sum_{\substack{(i,j) \\ t_{ij} > a_{ij}}} (a_{ij} - t_{ij}) c_{ij} \\ &\quad + \sum_{\substack{(i,j) \\ t_{ij} < a_{ij}}} (a_{ij} - t_{ij}) l_{ij} = q(p).\end{aligned}$$

Since f and t satisfy ϵ-complementary slackness, we also have

$$\begin{aligned}\sum_{(i,j)} a_{ij} f_{ij} &= \sum_{(i,j)} (a_{ij} - t_{ij}) f_{ij} \\ &= \sum_{\substack{(i,j) \\ t_{ij} > a_{ij} + \epsilon}} (a_{ij} - t_{ij}) c_{ij} + \sum_{\substack{(i,j) \\ t_{ij} < a_{ij} - \epsilon}} (a_{ij} - t_{ij}) l_{ij} \\ &\quad + \sum_{\substack{(i,j) \\ |a_{ij} - t_{ij}| \leq \epsilon}} (a_{ij} - t_{ij}) f_{ij} \\ &\leq \sum_{\substack{(i,j) \\ t_{ij} > a_{ij}}} (a_{ij} - t_{ij}) c_{ij} \\ &\quad + \sum_{\substack{(i,j) \\ t_{ij} < a_{ij}}} (a_{ij} - t_{ij}) l_{ij} + \epsilon \sum_{(i,j)} (c_{ij} - l_{ij}) \\ &= w(t) + \epsilon \sum_{(i,j)} (c_{ij} - l_{ij}).\end{aligned}$$

Combining the last two relations, we see that the primal cost corresponding to f and the dual cost corresponding to t are within $\epsilon \sum_{(i,j)} (c_{ij} - f_{ij})$ of each other. Since these two costs bracket the optimal cost, it follows that both f and t are within $\epsilon \sum_{(i,j)} (c_{ij} - l_{ij})$ of being primal and dual optimal, respectively.

Suppose next that we execute the relaxation iteration and replace the definitions (12)–(14) for active, inactive and balanced arcs by the corresponding "ϵ" notions of (37)–(39). Then we can see that the stepsize δ of (20) or (25) is bounded below by ϵL, where L is a positive lower bound for $1/\max\{|v_{ij}| \mid (i,j) \in \mathscr{A}\}$ as v ranges over the finite number of vectors v that can arise in the algorithm. Since the rate of dual cost increase along these vectors is also bounded below by a positive number, we see that the cost improvement associated with a price adjustment (Step 5) is bounded below by a positive number. It follows that the algorithm cannot generate an infinite number of price adjustment steps and therefore must terminate in a finite number of iterations with a solution that is within $\epsilon \sum_{(i,j)} (c_{ij} - l_{ij})$ of being optimal. This solution is really an optimal solution for a perturbed problem where each arc cost coefficient a_{ij} has been changed by an amount not exceeding ϵ. Since we are dealing with linear programs, it can be seen, after some thought, that, if ϵ is sufficiently small, then every solution of the perturbed primal problem is also a solution of the original primal problem. Therefore, *for sufficiently small* ϵ, *the modified algorithm based on the definitions* (37)–(39) *terminates in a finite number of iterations with an optimal primal solution.* However, the required size of ϵ cannot be easily estimated a priori.

Line Search

The stepsize δ of (30) or (36) 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 has shown that this type of line search is beneficial; we implemented the technique in the relaxation codes described in Section 5.

Single Node Iterations

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

$$C(v_s, t) = d_s - \sum_{(s,m)\text{: balanced}} (f_{sm} - l_{sm}) - \sum_{(m,s)\text{: balanced}} (c_{ms} - f_{ms}) K_{ms} > 0, \qquad (40)$$

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. Then only the price of node s is changed, and 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 situation is reminiscent of *relaxation methods* in which a change of a single variable is effected with the

purpose of satisfying a single constraint at the expense of violating others.

Alternately, we may view a single node iteration as a *coordinate ascent iteration*, whereby we choose a single (the sth) coordinate direction and perform a line search along this direction. Figure 3 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

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

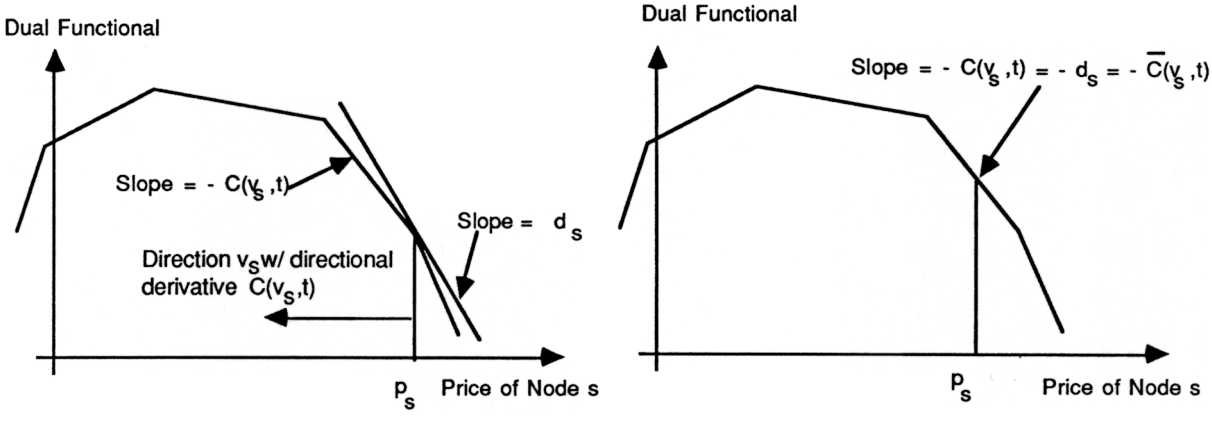
We have

$$-\bar{C}(v_s, t) \leq -d_s \leq -C(v_s, t), \tag{41}$$

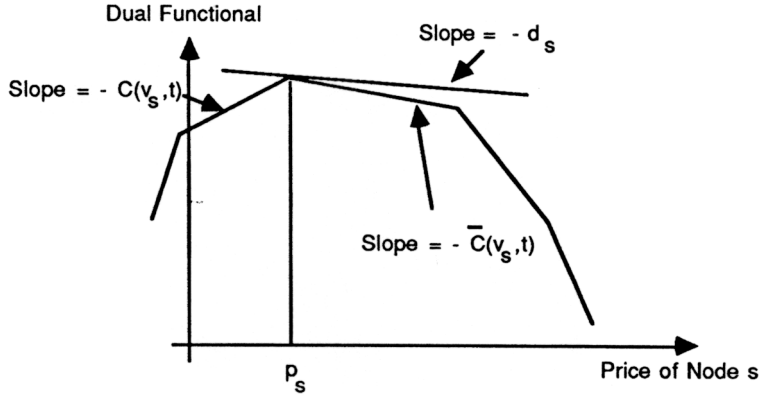
so $-d_s$ is a subgradient of the dual functional at p_s in the sth coordinate direction. A single node iteration will be possible if and only if the left slope is negative, or equivalently, $C(v_s, t) > 0$. This condition will always be true if we are not at a corner, and hence equality holds throughout in (41). However, if the dual cost is nondifferentiable at p_s, it may happen (Figure 3) that

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

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



CASES WHERE A SINGLE NODE ITERATION IS POSSIBLE



CASE WHERE A SINGLE NODE ITERATION IS NOT POSSIBLE

Figure 3. Illustration of the 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.

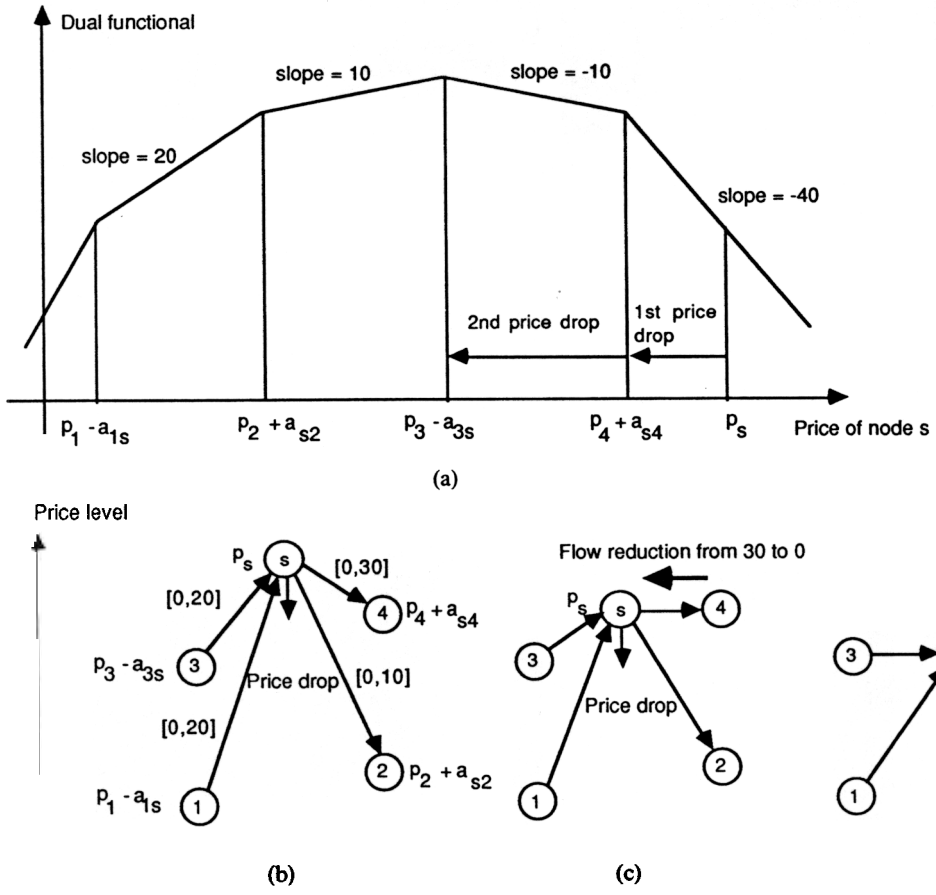


Figure 4. Illustration of an iteration involving a single node s with four adjacent arcs $(1, s)$, $(3, s)$, $(s, 2)$, $(s, 4)$ with feasible arc flow ranges $[1, 20]$, $[0, 20]$, $[0, 10]$, $[0, 30]$, respectively. (a) Form of the dual functional along p_s for given values of p_1, p_2, p_3, and p_4. 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 $(s, 4)$ changes from balanced to inactive 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).

Figures 4 and 5 illustrate a single node iteration for the cases in which $d_s > 0$ and $d_s < 0$, respectively.

Degenerate Ascent Iterations

Consider the case of an ordinary network. 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.,

$$w(t + \gamma v) = w(t), \quad \text{for all } \gamma \in [0, \delta),$$

where w, v, and δ are given by (15), (19), (20). We refer to such incremental changes in t as *degenerate* ascent iterations. If the ascent condition $C(v, t) > 0$ (see Equation 26) is replaced by $C(v, t) \geq 0$, 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

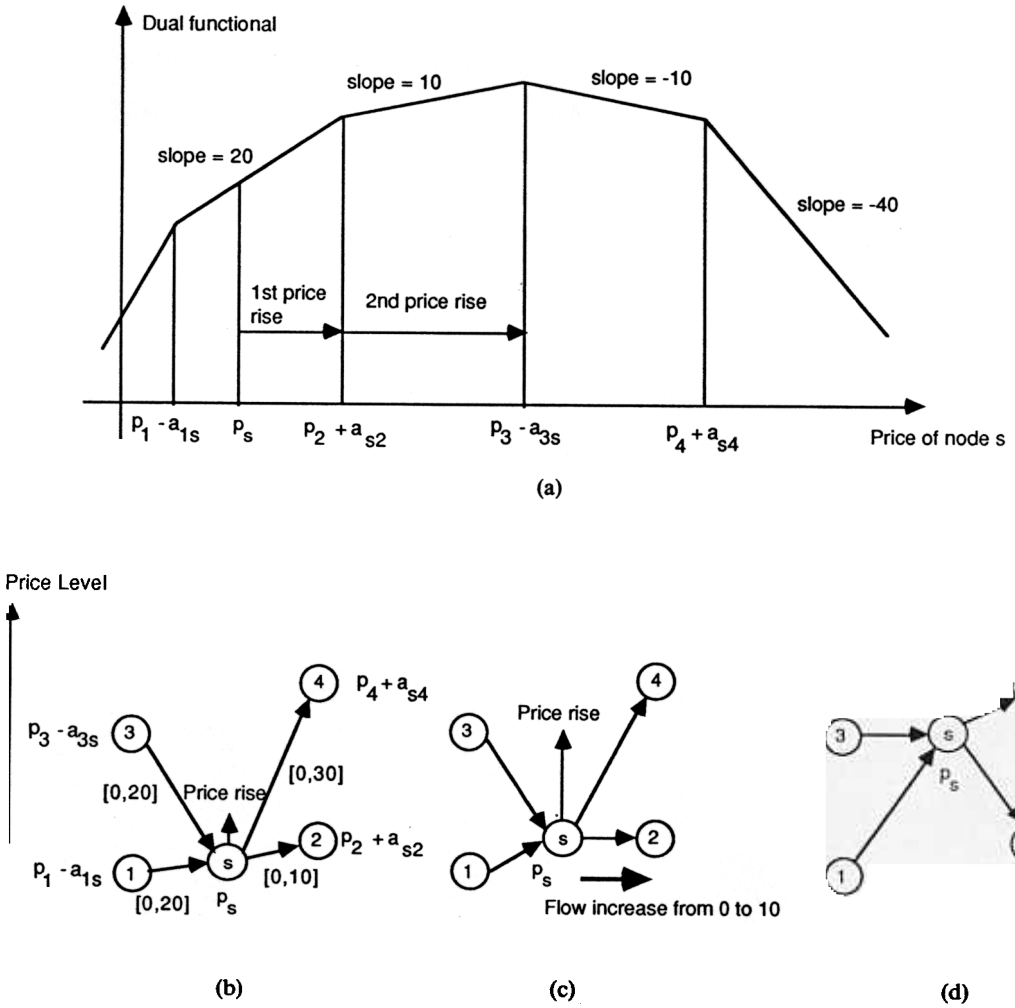


Figure 5. Illustration of a price rise involving the single node s for the example in Figure 4. Here the initial price p_s lies between the two leftmost break points corresponding to the arcs $(1, s)$ and $(s, 2)$. Initially, arcs $(1, s)$, $(s, 2)$, and $(s, 4)$ are inactive, and arc $(3, s)$ is active.

without degenerate steps under the following condition (see Bertsekas 1985):

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.

This condition holds when the set S consists of just the starting node s. Thus if we modify the ascent iteration 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. We implemented this modification in the relaxation codes (see Section 5); it 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, but feel it is probably due to the fact that degenerate ascent iterations help bring the prices of positive and negative deficit nodes "close" to each other more quickly. A computational complexity analysis, not given here, indicates that this factor is important in the speed of convergence of the algorithm.

4. Basis for Computational Experimentation

Historically, computational experimentation has been the primary method for comparative evaluation of network flow algorithms. During the 1960s it was generally believed that primal-dual methods held an advantage over simplex methods. However, during

the 1970s, major improvements in implementation (Srinivasan and Thompson 1973; Glover et al. 1974; Glover, Karney and Klingman 1974; Bradley, Brown and Graves 1977; Johnson; Grigoriadis 1978; and Kennington and Helgason) using sophisticated data structures propelled simplex algorithms to a position of prominence for solving general minimum cost flow problems. The situation is less clear for special classes of problems, such as assignment, where some computational comparisons (Hatch 1975; McGinnis 1983) suggest that primal-dual methods perform at least as well as simplex methods. Primal-dual methods are also generally better suited for sensitivity analysis and reoptimization.

Analytical results that substantively aid the comparison of different methods are in scarce supply. An interesting observation was made by Zadeh (1979), who showed that, for problems with nonnegative arc costs, primal-dual, dual simplex, primal simplex (with "big-M" starting method and most negative pivot rule), and the parametric method implement an essentially identical process—a sequence of augmentations along shortest paths between a supersource and a supersink node. The essential similarity between parametric and primal-dual methods actually extends to general linear programs with positive cost coefficients, as shown by Gallager (1983). This finding is significant in view of recent average complexity results for the parametric method (Haimovich 1983). The "big-M" method is known to be more effective for network problems than the Phase I-Phase II method (Mulvey 1978a). However, there are pivot rules that are empirically more effective than the most negative rule, and much research has been directed to this area (Gavish, Schweitzer and Shlifer 1977; Mulvey 1978a, b). Zadeh concludes that the "big M" method with empirically best pivot rule should be a better method than primal-dual for general minimum cost flow problems with nonnegative arc costs. This conclusion agrees with empirical observations of others (e.g., Glover, Karney and Klingman) as well as with our own (see Section 6).

We have compared our two relaxation codes, RELAX-II and RELAXT-II, with two state-of-the-art codes: KILTER (a primal-dual code due to Aashtiani and Magnanti 1976) and RNET (a primal simplex code due to Grigoriadis and Hsu 1980). The next section gives a description of each of these codes. We now describe our experimental approach.

Test Conditions

All methods were tested under identical conditions: same computer (a VAX 11/750 which ran VMS version 4.1), language (FORTRAN IV), compiler (standard FORTRAN of the VMS system version 3.7 in the OPTIMIZE mode), timing routine, and system conditions (empty system at night). RELAX-II and RELAXT-II were also compiled under VMS version 4.1; they ran about 15%–20% faster than when compiled under VMS version 3.7. We obtained the CPU times using the system routines LIB$INIT_TIMER and LIB$SHOW_TIMER. These times do not include problem input and output, but include algorithm initialization and testing for problem infeasibility. The VAX 11/750 is a relatively small machine on which problems of large size can produce an excessive number of page faults, thereby severely distorting the amount of computation time necessary. However, the size of problems used in our experiments and the system configuration were such that page faults never significantly affected the reported times.

The methods tested include parameters that must be set by the user. A single default set of parameters was chosen for each method and was kept unchanged throughout the experimentation. For RNET, these parameters are in the range suggested by its authors, with the parameter FRQ set at 7.0.

Efficiency of Implementation

RNET is a mature primal simplex code developed at Rutgers University. Indirect comparisons reported in Grigoriadis and in Bertsekas and Tseng suggest that RNET is faster on standard NETGEN benchmark problems given by Klingman, Napier and Stutz (1974) (see Table 1) than PNET (by the same authors) and GNET (Bradley, Brown and Graves), both of which are sophisticated simplex codes that represented an advance in the state of the art at the time they were introduced. Kennington and Helgason have compared RNET with their own primal simplex code NETFLO on the first 35 NETGEN benchmark problems and conclude (p. 255) that "RNET . . . produced the shortest times that we have seen on these 35 test problems." Our own experiments generally support these findings and suggest that for general minimum-cost flow problems RNET is at least as fast and probably faster than any other noncommercial simplex code for which computation times on benchmark problems are available to us (Klingman, Napier and Stutz; Aashtiani and Magnanti; Bradley, Brown and Graves; Kennington and Helgason; McGinnis). See also the experiments in Glover and Klingman (1982) that find the commercial code ARCNET slightly superior to RNET.

KILTER is an implementation of the primal-dual method that uses a sophisticated labeling scheme

described by Aashtiani and Magnanti. The version we tested is the fastest of nine versions tested by those researchers, who call it KILTER9. On the basis of their limited computational results and indirect comparisons, KILTER outperforms by a wide margin several earlier primal-dual codes, and is comparable to the simplex code of Klingman, Napier and Stutz. KILTER is also generally faster than the faster primal-dual codes that we have been able to implement (see Bertsekas and Tseng). However, an extensive computational study by Dembo and Mulvey (1976) shows that KILTER is outperformed on assignment problems under identical test conditions by LPNET (a primal simplex code due to Mulvey). Our own experiments also show that KILTER is consistently outperformed by RNET, and agree with the generally held opinion that the most efficient primal-dual codes are slower than primal simplex codes on general minimum cost flow problems.

The preceding discussion was intended to show that the implementations of both RNET and KILTER seem very efficient. Therefore it appears valid to consider these codes as representative of the best that has been achieved through the enormous collective efforts of many people over many years with the primal simplex and primal-dual methods.

5. Code Descriptions

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

$$\text{Minimize} \sum_{(i,j)\in\mathcal{A}} a_{ij}f_{ij}$$

subject to

$$\sum_{(m,i)\in\mathcal{A}} f_{mi} - \sum_{(i,m)\in\mathcal{A}} f_{im} = b_i, \quad \text{for all } i \in \mathcal{N}$$

$$0 \leq f_{ij} \leq c_{ij}, \quad \text{for all } (i, j) \in \mathcal{A}.$$

This form has become standard in network codes as it does not require storage and use of the array of lower bounds $\{l_{ij}\}$. Instead, the smaller size array $\{b_i\}$ is stored and used. Problem MCF in our introductory section can be reduced to the form just described by making the transformation of variables $f_{ij} := f_{ij} - l_{ij}$. The method we employed to represent the problem is the linked list structure suggested by Aashtiani and Magnanti and used in their KILTER code (see also Magnanti 1976). Briefly, while solving the problem we store for each arc its start and end node, capacity, reduced cost $(a_{ij} - t_{ij})$, flow f_{ij}, the next arc with the same start node, and the next arc with the same end node. For internal calculations, we use an additional array of length equal to half the number of arcs. 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 $|\mathcal{A}|$ and 7 $|\mathcal{N}|$ for node length arrays. RELAXT-II is similar to RELAX-II but employs two additional arc length arrays that store the set of all balanced arcs. This code, written with assistance from Jon Eckstein, is generally faster than RELAX-II but requires 9.5 $|\mathcal{A}|$ + 9 $|\mathcal{N}|$ total storage. This storage requirement compares unfavorably with that of primal simplex codes which can be implemented with four arc-length arrays.

RELAX-II and RELAXT-II implement, with minor variations, the relaxation algorithm of Section 2. 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. There is a preprocessing phase (included in the CPU time reported) in which arc capacities are reduced to as small a value as possible without changing optimal solutions of the problem. Thus for transportation problems we set the capacity of each arc at the minimum of the supply and demand at the head and tail nodes of the arc. We found experimentally that this tactic can markedly improve performance, 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 (potentially fewer extreme points to search over).

Finally, we note that RELAX-II and RELAXT-II, finalized in September 1986, are much more efficient than earlier versions (Bertsekas 1985; Bertsekas and Tseng; and Tseng), particularly for sparse and uncapacitated problems. In this connection, we note that Grigoriadis (1986) undertook a computational comparison of RELAX and RNET, using a memory-limited machine. However, the code he used was obtained by modifying, in ways unknown to us, a prerelease version of RELAX. That version was considerably less efficient than the code actually tested by Bertsekas (1985) and Tseng, let alone the second generation version tested in this paper.

Table I
Standard Benchmark Problems 1–40 of Klingman, Napier and Stutz, Obtained Using NETGEN[a]

Problem Type and 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						
	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)						
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)						
Uncapacitated and 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)						
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)						
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)						

[a] All times are in seconds on a VAX 11/750. All codes were compiled by FORTRAN in OPTIMIZE mode under VMS versions 3.7 and 4.1, as indicated. All codes ran on the same machine, under identical conditions.

6. Computational Results

We have organized our computational results into six tables. All the problems shown were generated using the widely used, publicly available NETGEN program (Klingman, Napier and Stutz). We used the random number seed 13502460 (the same number used by those authors). The tables provide all additional information needed to replicate these problems. In addition, the doctoral thesis of Tseng includes computational experience with gain networks. His results are preliminary and show that relaxation is roughly competitive with a state-of-the-art primal simplex code of Currin (1983) (tests done under identical conditions on the same machine). More experimentation is required to corroborate these results.

Table I (Standard NETGEN Benchmarks). This table shows the results for the 40 problems described in detail by Klingman, Napier and Stutz, and generated by the NETGEN program. Problem 36 was not solved because for some unexplained reason our NETGEN code produced an infeasible problem for the data given in that reference. The results show the substantial superiority of RELAX-II and RELAXT-II over the other codes for assignment and transportation problems. We corroborated this finding on a large number of additional assignment and transportation problems of widely varying size. For small, lightly capacitated and uncapacitated problems, RELAX-II and RELAXT-II outperform the other codes, and the margin of superiority increases for the large problems 37–40.

Table II (Transportation Problems). These results are in general agreement with those of Table I. Note that, for dense problems, RELAXT-II is substantially faster than RELAX-II, owing to the scheme for storing and using the set of balanced arcs.

Table III (Transportation Problems–Large Cost Range). The problems in this table are identical to those in Table II except that the cost range is from 1 to 10,000 instead of 1 to 100. It can be seen that RELAX-II and RELAXT-II still substantially outperform RNET, but the factor of superiority is less than in the case of the smaller cost range of Table II.

Table II
Transportation Problems[a]

Problem No.	No. of Sources	No. of Sinks	No. of Arcs	Cost Range	RELAX-II	RELAXT-II	RNET
Total (Problems 1–10)							
11	100	300	7,000	1–100			
12	200	600	7,000	1–100			
13	300	900	7,000	1–100			
14	350	1,050	7,000	1–100			
15	400	1,200	7,000	1–100			
	100	300	6,000	1–100			
	100	300	8,000	1–100			
	100	300	10,000	1–100			
	100	300	12,000	1–100			
	100	300	15,000	1–100			
Total (Problems 11–20)							

[a] Times are given in seconds on VAX 11/750. All problems were obtained using NETGEN with total supply 200,000 and 0% high cost arcs. RELAX-II and RELAXT-II were compiled under VMS 4.1; RNET was compiled under VMS 3.7.

Table III
Transportation Problems[a]

Problem No.	No. of Sources	No. of Sinks	No. of Arcs	Cost Range	RELAX-II	RELAXT-II	RNET
11	100	300	7,000	$1-10^4$			
12	200	600	7,000	$1-10^4$			
13	300	900	7,000	$1-10^4$			
14	350	1050	7,000	$1-10^4$			
15	400	1200	7,000	$1-10^4$			
		300	6,000	$1-10^4$			
		300	8,000	$1-10^4$			
		300	10,000	$1-10^4$			
		300	12,000	$1-10^4$			
		300	15,000	$1-10^4$			

[a] Times are given in seconds on VAX 11/750. All problems were obtained using NETGEN with total supply 200,000 and 0% high cost arcs. RELAX-II and RELAXT-II were compiled under VMS 4.1; RNET was compiled under VMS 3.7.

Table IV (Heavily Capacitated Transshipment Problems). Our experience with problems of this type with positive arc costs is similar to that for transportation problems.

Table V (Transshipment Problems with Both Positive and Negative Arc Costs). The problems in this table are identical to those of Table IV except that the cost range is from −50 to 50 instead of 1 to 100. When there are both positive and negative arc costs, the performance of RNET (in contrast with RELAX-II and RELAXT-II) depends on how flow is initialized. If all arc flows are initially set to zero, the performance of RNET degrades substantially (see Bertsekas and Tseng). A more efficient scheme is to set the flow of negative cost arcs to the upper bound and the flow of all other arcs to zero. We followed this policy in the runs shown in Table V. The table shows that the factor of superiority of RELAX-II and RELAXT-II over RNET increases somewhat relative to the results of Table IV.

Table VI (Large Assignment and Transportation Problems). 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. We can see this by comparing the results for the small problems 1–35 of Table I with the results for the larger problems 37–40 of that table, and with the problems of Tables II through V. The comparison shows an improvement in speedup factor that is not spectacular, but is noticeable and consistent. Table VI shows that for even larger problems the speedup factor increases further with problem dimension, and reaches or exceeds an order of magnitude (Figure 6). 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 we experienced 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. We resolved this problem by adding the same number (usually 10) to all source supplies and all sink demands, as noted in Table VI. Note that the transportation problems of the table are divided into groups, and that each group

Table IV
Capacitated Transhipment Problems[a]

Problem No.	No. of Sources	No. of Sinks	No. of Arcs	Cost Range	Capacity Range	RELAX-II	RELAXT-II	RNET
1	200	200	7,000	1–100	100–500	13.50	7.39	44.47
2	400	400	7,000	1–100	100–500	16.99	12.10	73.10
3	600	600	7,000	1–100	100–500	21.62	22.25	97.11
4	800	800	7,000	1–100	100–500	26.86	22.35	108.09
5	1,000	1,000	7,000	1–100	100–500	29.26	26.54	102.74
6	200	200	6,000	1–100	100–1,000	9.25	6.26	39.18
7	200	200	8,000	1–100	100–1,000	10.71	8.53	48.87
8	200	200	10,000	1–100	100–1,000	14.58	8.57	51.98
9	200	200	12,000	1–100	100–1,000	17.78	10.62	68.17
10	200	200	15,000	1–100	100–1,000	21.42	10.89	73.74
Total (Problems 1–10)						181.97	135.50	707.45
11	100	300	7,000	1–100	100–500	10.38	7.18	43.98
12	200	600	7,000	1–100	100–500	20.11	13.21	68.29
13	300	900	7,000	1–100	100–500	25.85	22.59	90.36
14	400	1,200	7,000	1–100	100–500	35.23	27.43	109.75
15	500	1,500	7,000	1–100	100–500	40.92	31.60	107.32
16	100	300	6,000	1–100	100–1,000	9.02	6.97	33.45
17	100	300	8,000	1–100	100–1,000	12.44	9.56	39.59
18	100	300	10,000	1–100	100–1,000	16.26	8.87	48.61
19	100	300	12,000	1–100	100–1,000	20.46	11.06	59.36
20	100	300	15,000	1–100	100–1,000	22.47	11.36	72.41

[a] Times are given in seconds on VAX 11/750. All problems were obtained using NETGEN with total supply 200,000, 100% of sources and sinks being transhipment nodes, 0% high cost arcs, and 100% of arcs capacitated. Each node is either a source or a sink. RELAX-II and RELAXT-II were compiled under VMS 4.1; RNET was compiled under VMS 3.7.

Table V
Capacitated Transhipment Problems with Both Negative and Positive Arc Costs[a]

Problem No.	No. of Sources	No. of Sinks	No. of Arcs	Cost Range	Capacity Range	RELAX-II	RELAXT-II	RNET
1	200	200	7,000	−50–50	100–500	11.02	11.44	55.50
2	400	400	7,000	−50–50	100–500	17.99	12.88	92.11
3	600	600	7,000	−50–50	100–500	17.10	16.38	109.35
4	800	800	7,000	−50–50	100–500	27.82	24.62	124.42
5	1,000	1,000	7,000	−50–50	100–500	38.02	31.48	123.87
6	200	200	6,000	−50–50	100–1,000	10.09	5.83	49.15
7	200	200	8,000	−50–50	100–1,000	12.40	7.99	67.74
	200	200	10,000	−50–50	100–1,000	12.71	9.79	81.95
9	200	200	12,000	−50–50	100–1,000	16.72	10.28	89.71
10	200	200	15,000	−50–50	100–1,000	30.78	13.73	94.58
Total (Problems 1–10)								
11	100	100	7,000	−50–50	100–500			
12	200	600	7,000	−50–50	100–500			
13	300	900	7,000	−50–50	100–500			
14	400	1,200	7,000	−50–50	100–500			
15	500	1,500	7,000	−50–50	100–500			
	100	300	6,000	−50–50	100–1,000			
	100	300	8,000	−50–50	100–1,000			
	100	300	10,000	−50–50	100–1,000			
	100	300	12,000	−50–50	100–1,000			
	100	300	15,000	−50–50	100–1,000			
Total (Problems 11–20)								

[a] Same problems as in Table IV except that the cost range is [−50, 50]. RELAX-II and RELAXT-II were compiled under VMS 4.1; RNET was compiled under VMS 3.7.

Table VI
Large Assignment and Transportation Problems[a]

Problem Type and No.	No. of Sources	No. of Sinks	No. of Arcs	Cost Range	Total Supply	RELAX-II	RELAXT-II	RNET
Assignment								
1	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
Transportation								
6	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
Transportation								
11	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
Transportation								
16+	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

[a] Times are given in seconds on VAX 11/750. All problems were obtained using NETGEN, as described in the text. RELAX-II and RELAXT-II were compiled under VMS 4.1; RNET was 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.

has the same average degree per node (8 for Problems 6–15, and 20 for Problems 16–20).

To corroborate the results of Table VI, we changed the random seed number of NETGEN, and solved additional problems using some of the problem data of the table. The results were qualitatively similar to those of Table VI. We also solved a set of transshipment problems of increasing size generated by our random problem generator, called RANET. Figure 7 gives the comparison between RELAX-II, RELAXT-II and RNET. More experimentation and/or analysis is needed to establish conclusively the computational complexity implications of these experiments.

7. Conclusions

Relaxation methods adapt nonlinear programming ideas to solve linear network flow problems. They are much faster than classical methods on both standard benchmark problems and a broad range of randomly generated problems. They are also better suited than primal simplex for post-optimization analysis. For example, suppose we solve a problem and then modify it (by changing a few arc capacities and/or node supplies). To solve the modified problem using the relaxation method, we use as starting node prices the prices obtained from the earlier solution, and 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 must 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. Note also that an alternative implementation of RELAXT-II, currently in the final stages of development, has resulted in reduction of the storage needed for arc length arrays by one-third, without sacrificing speed of execution.

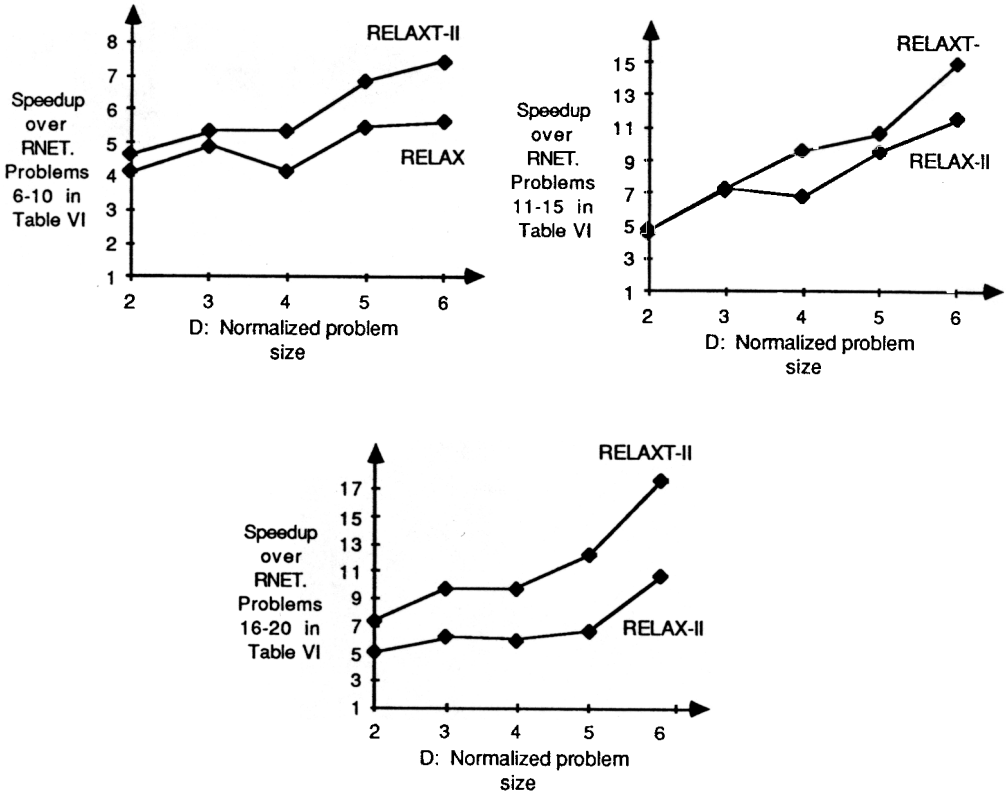


Figure 6. Speedup factor of RELAX-II and RELAXT-II over RNET for the transportation problems of Table VI. The normalized dimension D gives the number of nodes N and arcs A as follows:

$N = 1000*D$, $A = 4000*D$, for Problems 6–15
$N = 500*D$, $A = 5000*D$, for Problems 16–20.

Our computational results provide some indication that relaxation has a superior average computational complexity over primal simplex. Additional experimentation with large problems and/or analysis is needed to provide conclusive corroboration of these results.

The relaxation approach applies to a broad range of problems beyond the class considered in this paper (see Bertsekas, Hosein and Tseng; Tseng; and Tseng and Bertsekas 1987a,b), including general linear programming problems. It also lends itself well to massively parallel computation (see Bertsekas 1986, 1988; Bertsekas, Hosein and Tseng; Bertsekas and El Baz; Bertsekas and Eckstein; Tseng and Bertsekas; Goldberg and Tarjan 1986, 1987; Ahuja and Orlin; and Bertsekas and Tsitsiklis 1988).

The relaxation codes RELAX-II and RELAXT-II, together with other support programs, are in the public domain with no restrictions, and can be obtained free of charge from the authors on an IBM-PC or Mac-Intosh diskette.

Acknowledgment

This work was supported by the National Science Foundation under grant NSF-ECS-8217668. Thanks are due to Tom Magnanti, who supplied us with the primal-dual code KILTER, and who, together with John Mulvey and Bob Gallager, clarified several questions for us. Thanks are also due to the referees for their constructive comments, and to Alphatech, Inc., for its support.

The authors are with the Laboratory for Information and Decision Systems (LIDS) and the Operations Research Center, Massachusetts Institute of Technology. This article is based largely on their unpublished report which was written under the auspices of LIDS (see Bertsekas and Tseng 1983).

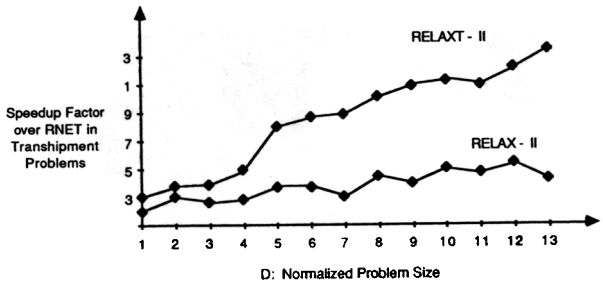


Figure 7. Speedup factor of RELAX-II and RELAXT-II over RNET in lightly capacitated transhipment problems generated by our own random problem generator RANET. Each node is a transhipment node, and it is either a source or a sink. The normalized problem size D gives the number of nodes and arcs as follows:

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

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.

References

AASHTIANI, H. A., AND T. L. MAGNANTI. 1976. Implementing Primal-Dual Network Flow Algorithm. MIT Operations Research Center Report 055-76 (June).

AJUJA, R. K., AND J. B. ORLIN. 1986. A Fast and Simple Algorithm for the Maximum Flow Problem. MIT Working Paper (November).

BERTSEKAS, D. P. 1981. A New Algorithm for the Assignment Problem. *Math. Program.* **21**, 152–171.

BERTSEKAS, D. P. 1985. A Unified Framework for Minimum Cost Network Flow Problems. *Math. Program.* **32**, 125–145.

BERTSEKAS, D. P. 1986. Distributed Relaxation Methods for Linear Network Flow Problems, pp. 2101–2106. In *Proceedings of the 25th IEEE Conference on Decision and Control*, Athens, Greece (December).

BERTSEKAS, D. P. 1988. The Auction Algorithm: A Distributed Relaxation Method for the Assignment Problem. To appear in *Ann. Opns. Res.*

BERTSEKAS, D. P., AND J. ECKSTEIN. 1987. Distributed Asynchronous Relaxation Methods for Linear Network Flow Problems. *Proc. IFAC-87*, Munich Germany (July). Pergamon Press, Oxford.

BERTSEKAS, D. P., AND D. EL BAZ. 1987. Distributed Asynchronous Relaxation Methods for Convex Network Flow Problems. *SIAM J. Control Optim.* **25**, 74–85.

BERTSEKAS, D. P., AND S. K. MITTER. 1971. Steepest Descent for Optimization Problems with Nondifferentiable Cost Functionals, pp. 374–351. In *Proceedings of the Fifth Annual Princeton Conference on Information Science Systems*, Princeton, N.J.

BERTSEKAS, D. P., AND S. K. MITTER. 1973. A Descent Numerical Method for Optimization Problems with Non-differentiable Cost Functionals. *SIAM J. Control* **11**, 637–652.

BERTSEKAS, D. P., AND P. TSENG. 1983. Relaxation Methods for Minimum Cost Network Flow Problems. MIT Laboratory for Information and Decision Sciences Report LIDS-P-1339 (October).

BERTSEKAS, D. P., AND J. N. TSITSIKLIS. 1988. *Parallel and Distributed Algorithms.* Prentice-Hall, Englewood Cliffs, N.J.

BERTSEKAS, D. P., P. HOSEIN AND P. TSENG. 1987. Relaxation Methods for Network Flow Problems with Convex Arc Costs. *SIAM J. Control Optim.* **25**, 1219–1243.

BRADLEY, G. H., G. G. BROWN AND G. W. GRAVES. 1977. Design and Implementation of Large Scale Transhipment Algorithms. *Mgmt. Sci.* **24**, 1–34.

CURRIN, D. C. 1983. A Comparative Evaluation of Algorithms for Generalized Network Problems. NRIMS Technical Report TWISK 289, Pretoria, South Africa.

DEMBO, R. S., AND J. M. MULVEY. 1976. On the Analysis and Comparison of Mathematical Programming Algorithms and Software. Harvard Business School Report 76-19.

FORD, L. R., JR., AND D. R. FULKERSON. 1962. *Flows in Networks.* Princeton University Press, Princeton, N.J.

GALLAGER, R. G. 1983. Parametric Optimization and the Primal-Dual Method. MIT Laboratory for Information and Decision Systems Report (June).

GAVISH, B., P. SCHWEITZER AND E. SHLIFER. 1977. The Zero Pivot Phenomenon in Transportation Problems and Its Computational Implications. *Math. Program.* **12**, 226–240.

GLOVER, F., AND D. KLINGMAN. 1982. Recent Developments in Computer Implementation Technology for Network Flow Algorithms. INFOR **20**, 433–452.

GLOVER, F., D. KARNEY AND D. KLINGMAN. 1974. Implementation and Computational Comparisons of Primal, Dual and Primal-Dual Computer Codes for Minimum Cost Network Flow Problems. *Networks* **4**, 191–212.

GLOVER, F., D. KARNEY, D. KLINGMAN AND A. NAPIER. 1974. A Computation Study on Start Procedures, Basis Change Criteria and Solution Algorithms for Transportation Problems. *Mgmt. Sci.* **20**, 793–819.

GOLDBERG, A. V., AND R. E. TARJAN. 1986. A New Approach to the Maximum Flow Problem, pp. 136–146. In *Proceedings of the 18th ACM STOC.*

GOLDBERG, A. V., AND R. E. TARJAN. 1987. Solving Minimum Cost Flow Problems by Successive Approximation. In *Proceedings of the 19th ACM STOC* (May).

GRIGORIADIS, M. D. 1978. Algorithms for the Minimum Cost Single and Multicommodity Network Flow Problems. Notes for Summer School in Combinatorial Optimization SOGESTA, Urbino, Italy (July).

GRIGORIADIS, M. D. 1986. An Efficient Implementation of the Network Simplex Method. *Math. Program. Studies* **26.**

GRIGORIADIS, M. D., AND T. HSU. 1980. The Rutgers Minimum Cost Network Flow Subroutines (RNET documentation). Department of Computer Science, Rutgers University (November).

HAIMOVICH, M. 1983. The Simplex Algorithm is Very Good!—On the Expected Number of Pivot Steps and Related Properties of Random Linear Programs. Columbia University Report (April).

HATCH, R. S. 1975. Bench Marks Comparing Transportation Codes Based on Primal Simplex and Primal-Dual Algorithms. *Opns. Res.* **23,** 1167–1172.

JEWELL, W. S. 1962. Optimal Flow Through Networks with Gains. *Opns. Res.* **10,** 476–499.

JOHNSON, E. 1966. Networks and Basic Solutions. *Opns. Res.* **14,** 619–623.

KENNINGTON, J., AND R. HELGASON. 1980. *Algorithms for Network Programming.* John Wiley & Sons, New York.

KLINGMAN, D., A. NAPIER AND J. STUTZ. 1974. NETGEN—A Program for Generating Large Scale (Un)capacitated Assignment, Transportation and Minimum Cost Flow Network Problems. *Mgmt. Sci.* **20,** 814–822.

MAGNANTI, T. 1976. Optimization for Sparse Systems. In *Sparse Matrix Computations*, pp. 147–176, J. R. Bunch and D. J. Rose (eds.). Academic Press, New York.

MCGINNIS, L. F. 1983. Implementation and Testing of a Primal-Dual Algorithm for the Assignment Problem. *Opns. Res.* **31,** 277–291.

MINIEKA, E. 1978. *Optimization Algorithms for Networks and Graphs.* Marcel Dekker, New York.

MULVEY, J. M. 1978a. Pivot Strategies for Primal-Simplex Network Codes. *J. Assoc. Comput. Mach.* **25,** 266–270.

MULVEY, J. M. 1978b. Testing of a Large-Scale Network Optimization Program. *Math. Program.* **15,** 291–314.

ORTEGA, J. M., AND W. C. RHEINBOLDT. 1970. *Iterative Solution of Nonlinear Equations in Several Variables.* Academic Press, New York.

PAPADIMITRIOU, C. H., AND K. STEIGLITZ. 1982. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, Englewood Cliffs, N.J.

ROCKAFELLAR, R. T. 1970. *Convex Analysis.* Princeton University Press, Princeton, N.J.

ROCKAFELLAR, R. T. 1984. *Network Flows and Monotropic Optimization.* John Wiley & Sons, New York.

SRINIVASAN, V., AND G. L. THOMPSON. 1973. Benefit-Cost Analysis of Coding Techniques for the Primal Transportation Algorithm. *J. Assoc. Comput. Mach.* **20,** 194–213.

TSENG, P. 1986. Relaxation Methods for Monotropic Programs. Ph.D. thesis, Massachusetts Institute of Technology (May).

TSENG, P., AND D. P. BERTSEKAS. 1987a. Relaxation Methods for Problems with Strictly Convex Separable Costs and Linear Constraints. *Math. Program.* **38,** 303–321.

TSENG, P., AND D. P. BERTSEKAS. 1987b. Relaxation Methods for Linear Programs. *Math. Opns. Res.* **12,** 569–596.

ZADEH, N. 1979. Near-Equivalence of Network Flow Algorithms. Technical Report 26, Department of Operations Research, Stanford University (December).