May 1985 LIDS-P-1462
Revised September 1986

RELAXATION METHODS FOR MINIMUM COST ORDINARY AND GENERALIZED NETWORK FLOW PROBLEMS[†]

by

Dimitri P. Bertsekas*

and

Paul Tseng*

ABSTRACT

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. Our coded implementations of these methods are compared with mature state-of-the-art primal simplex and primal-dual codes and are found 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.

[†]This work has been 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. He together with John Mulvey and Bob Gallager, clarified several questions for us. Thanks are also due to the referees for their constructive comments.

Support provided by Alphatech, Inc. is also gratefully acknowledged. This paper is largely based on the unpublished report [10].

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

1. Introduction

Consider a directed graph with set of nodes N and set of arcs 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 classical minimum cost flow problem with gains

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

subject to

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

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

where K_{ij}, ℓ_{ij} and c_{ij} are given scalars. The scalar K_{ij} is referred to as the gain of arc (i,j). The main focus of the paper is in the ordinary network case where $K_{ij} = 1$ for all $(i,j)\in A$. 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 of (MCF). For simplicity we also assume that there is at most one arc connecting 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 abl $(i,j)\in A$.

Our algorithm can be extended for the case where $K_{ij} \le 0$ for some $(i,j)\in A$, 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}, $(i,j)\varepsilon A$ and p_i, $i\varepsilon N$ respectively we can write the corresponding Lagrangian function

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

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

The dual problem is

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

$$\text{subject to no constraints on } p,$$

with the dual functional q given by

$$q(p) = \min_{\ell_{ij}\le f_{ij}\le c_{ij}} L(f,p) \tag{4}$$

$$= \sum_{(i,j)\varepsilon A} q_{ij}(p_i-K_{ij}p_j)$$

where

$$q_{ij}(p_i-K_{ij}p_j) = \min_{\ell_{ij}\le f_{ij}\le c_{ij}} \{(a_{ij}+K_{ij}p_j-p_i)f_{ij}\}. \tag{5}$$

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

and

$$t_{ij} = p_i - K_{ij}p_j, \qquad \forall\ (i,j)\varepsilon A \tag{6}$$

The function $q_{ij}(p_i - K_{ij}p_j)$ is shown in Figure 1. This is a classical duality framework treated extensively in [1], [2].

The vector t having elements t_{ij}, $(i,j)\varepsilon A$ given by (6) is called the <u>tension vector</u> corresponding 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 what follows.

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

$$\text{Inactive} \quad \text{if} \quad t_{ij} < a_{ij} \tag{7}$$

$$\text{Balanced} \quad \text{if} \quad t_{ij} = a_{ij} \tag{8}$$

$$\text{Active} \quad \text{if} \quad t_{ij} > a_{ij}. \tag{9}$$

For any flow vector f the scalar

$$d_i = \sum_{\substack{m \\ (i,m)\varepsilon A}} f_{im} - \sum_{\substack{m \\ (m,i)\varepsilon A}} K_{mi}f_{mi} \qquad \forall\ i\varepsilon N \tag{10}$$

will be referred to as the <u>deficit</u> 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

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

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

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

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

Conditions (11)-(13) are the <u>complementary slackness conditions</u>.

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 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 $\sum_m |d_m|$ cannot be effected through flow augmentation, an adjacent node of i, say i_1, is chosen and an ascent is attempted along the sum of the coordinate vectors corresponding to i and i_1. If such an ascent is not possible, and flow augmentation is not possible either, an adjacent node of either i or i_1 is chosen and the process is continued. In practice, most of the ascent directions are single coordinate directions, leading to the interpretation of the algorithms as <u>coordinate ascent</u> or <u>relaxation methods</u>. 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 [10]), 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, and in Section 4 we prove their convergence using the novel notion of ε-complementary slackness. This notion is also important in other contexts [4], [6], [45]. Sections 5 through 7 are devoted to computational comparisons of our experimental relaxation codes with mature state-of-the-art codes

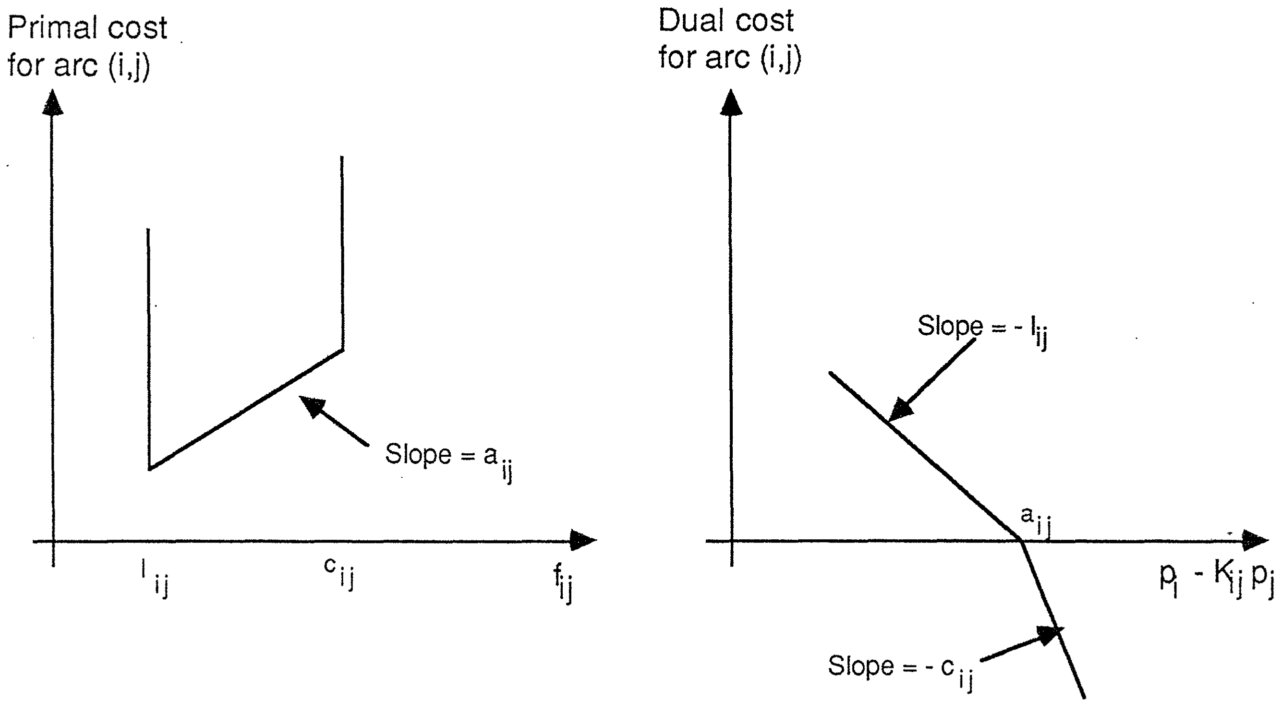


Figure 1: Primal and dual costs for arc (i,j)

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 [3] where the conceptual similarity with relaxation methods was also pointed out. The present paper considers in addition gain networks, emphasizes the dual ascent viewpoint and provides computational results. A special case for the assignment problem has been considered in Bertsekas [5]. Reference [10], which is an early version of the present paper, and the thesis [35] contain additional computational results and analysis. The relaxation algorithm for strictly convex arc cost problems is considered in [6], [11]. 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. The relaxation algorithms of this paper are extended in [35], [42], and are applied to the general linear programming problem. A massively parallelizable relaxation algorithm for the linear cost problem (MCF) that has a strong conceptual relation with the one of the present paper is given in [4] and [45].

2. Characterization of Dual Ascent Directions

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

$$v_{ij} = \begin{cases} 1 & \text{if } i\notin S,\ j\varepsilon S \\ -1 & \text{if } i\varepsilon S,\ j\notin S \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

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. It is seen from (5) that the directional derivative at t of the dual cost in the direction v is C(v,t) where

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

$$= \sum_{(i,j)\varepsilon A} e_{ij}(v_{ij},\ t_{ij}) \tag{16}$$

and

$$e_{ij}(v_{ij},t_{ij}) = \begin{cases} -v_{ij}\ell_{ij} & \text{if } (i,j) \text{ is inactive or if } (i,j) \text{ is balanced and } v_{ij} \le 0 \\ -v_{ij}c_{ij} & \text{if } (i,j) \text{ is active or if } (i,j) \text{ is balanced and } v_{ij} \ge 0. \end{cases} \tag{17}$$

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 N and every tension vector t there holds

$$w(t+\gamma v) = w(t) + \gamma C(v,t), \quad \forall\ \gamma \varepsilon [0,\delta) \tag{18}$$

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

$$\delta = \inf\{\{t_{im}-a_{im} | i\varepsilon S,\ m\notin S,\ (i,m)\text{: active}\}, \\ \{a_{mi}-t_{mi} | i\varepsilon S,\ m\notin S,\ (m,i)\text{: inactive}\}\}. \tag{20}$$

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

Proof: It was seen [cf. (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) Q.E.D.

Directional derivatives for dual directions used by the algorithm may be similarly computed for a gain network. We assume that there exists at least one cycle of nonunity gain so that the gain network cannot be reduced to an ordinary network ([2], p. 459). 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

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

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 $\{u_i | i\varepsilon S\}$ be a set of positive numbers such that

$$u_i - K_{ij}u_j = 0, \qquad \forall\ (i,j)\varepsilon T. \tag{21}$$

[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 $s\varepsilon S$, and determining the numbers u_i of the remaining nodes $i\varepsilon S$ from (21)]. Each dual ascent direction used by the algorithm is associated with a pair (S,T) as defined above and is given by $v = \{v_{ij} | (i,j)\varepsilon A\}$ where

$$v_{ij} = \begin{cases} K_{ij}u_j & \text{if } i\not\varepsilon S,\ j\varepsilon S \\ -u_i & \text{if } i\varepsilon S,\ j\not\varepsilon S \\ K_{ij}u_j - u_i & \text{if } i\varepsilon S,\ j\varepsilon S \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

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) it is seen that the directional derivative of the dual cost at t along a vector v is again given by

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

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

<u>Proposition 2</u>: (For gain networks) For every vector v defined by (21), (22), and every tension vector t there holds

$$w(t+\gamma v) = w(t) + \gamma C(v,t), \qquad \forall\ \gamma\varepsilon[0,\delta) \tag{24}$$

where w(•) is the dual cost function given by (19), and δ is given by

$$\delta = \inf\{\{\frac{t_{im}-a_{im}}{v_{im}} \mid v_{im} > 0,\ (i,m)\text{: active}\},$$

$$\{\frac{t_{mi}-a_{mi}}{v_{mi}} \mid v_{mi} < 0,\ (m,i)\text{:inactive}\}\}. \tag{25}$$

3. 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) 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, $(k \notin S)$, set $S = S \cup \{k\}$, and go to step 3.

Step 3: Scan the label of the node k as follows: Give the label "k" to all unlabeled nodes m such that (m,k) is balanced and $f_{mk} < c_{mk}$, and to all unlabeled m such that (k,m) is balanced and $\ell_{km} < f_{km}$. If v is the vector corresponding to S as in (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. The path is constructed by tracing labels backwards starting from m, and consists of balanced arcs such that we have $\ell_{kn} < f_{kn}$ for all $(k,n)\varepsilon P^+$ and $f_{kn} < c_{kn}$ for all $(k,n)\varepsilon P^-$ where

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

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

Let

$$\varepsilon = \min\{d_s, -d_m, \{f_{kn}-\ell_{kn} \mid (k,n)\varepsilon P^+\}, \{c_{kn}-f_{kn} \mid (k,n)\varepsilon P^-\}\}. \tag{29}$$

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

Step 5 (Price Adjustment): Let

$$\delta = \min\{\{t_{km}-a_{km} \mid k\varepsilon S,\ m\not\varepsilon S,\ (k,m)\text{: active}\}, \tag{30}$$

$$\{a_{mk}-t_{mk} \mid k\varepsilon S,\ m\not\varepsilon S,\ (m,k)\text{: inactive}\}\}.$$

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

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

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

where L is the set of labeled nodes. Set

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

Go to the next iteration.

The relaxation iteration terminates with either a flow augmentation (via step 4) or with a dual cost improvement (via step 5). In order for the procedure to be well defined, however, we must show that whenever we return to step 2 from step 3 there is still some labeled node which is unscanned. Indeed, when all 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 $f_{km} > \ell_{km}$. It follows from the definition (16), (17) [see also the following equation (31)] 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}, ℓ_{ij}, and c_{ij} are integer for all $(i,j)\varepsilon A$ and the starting t is 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 integer all sucessive f will be integer 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 integer. If the problem data is not integer 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 the next section.

It can be seen that the relaxation iteration involves a comparable amount of computation per node scanned as the usual primal-dual method [7]. The only additional computation involves maintaining the quantity $C(v,t)$, but it can be seen that this can be computed _incrementally_ in step 3 rather than recomputed each time the set S is enlarged in step 2. As a result this additional computation is insignificant.
To compute $C(v,t)$ incrementally in the context of the algorithm, it is helpful to use the identity

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

We note that a similar iteration can be constructed starting from a node with negative deficit. Here the set S consists of nodes with 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.

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 non-unity 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 [38].

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 $\{u_i | i\in S\}$ be a set of positive numbers such that

$$u_i - K_{ij}u_j = 0, \qquad \forall\ (i,j)\varepsilon T \tag{32}$$

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

$$\begin{aligned} C(v,t) = \sum_{i\varepsilon S} u_i d_i &- \sum_{\substack{(i,j):\text{balanced}\\ i\varepsilon S,\, j\not\in S}} (f_{ij}-\ell_{ij})u_i \\ &- \sum_{\substack{(i,j):\text{balanced}\\ i\not\in S,\, j\varepsilon S}} (c_{ij}-f_{ij})K_{ij}u_j \\ &- \sum_{\substack{(i,j):\text{balanced}\\ i\varepsilon S,\, j\varepsilon S\\ u_i>K_{ij}u_j}} (f_{ij}-\ell_{ij})(u_i-K_{ij}u_j) \\ &- \sum_{\substack{(i,j):\text{balanced}\\ i\varepsilon S,\, j\varepsilon S\\ u_i<K_{ij}u_j}} (c_{ij}-f_{ij})(K_{ij}u_j-u_i). \end{aligned} \tag{33}$$

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 $i\varepsilon S$ 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 below). In the process of reducing the last two terms in (33) to zero, the corresponding value of $\sum_{i\varepsilon S} u_i$ 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 [cf. (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 where $d_s < 0$ is entirely similar. If no such node can be found terminate the algorithm. Else set $S = \{s\}$, $T = \{\emptyset\}$, and go to step 2.

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

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

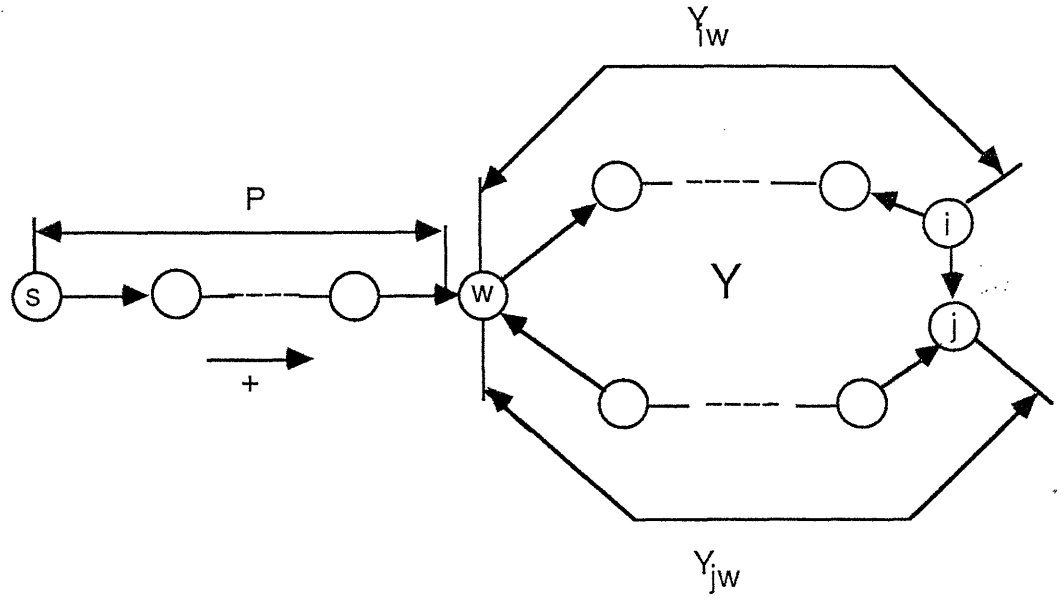


Figure 2: Cycle Y formed by tree T and arc (i,j) identified in step 2.

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

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

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) above 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 $\ell_{km} < f_{km} < c_{km}$ 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.

(2) There exists $(k,m) \in Y_{jw}$ such that $f_{km} = \ell_{km}$ 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 $i \epsilon S$, $m \notin S$ such that either (a) (i,m) is a balanced arc with $f_{im} > \ell_{im}$, 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.

<u>Step 4a (Flow Augmentation Involving a Simple Path)</u>: 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

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

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

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

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

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

$$u_k := u_k/q, \quad \forall \text{ nodes } k \text{ on } Y_{jw} \text{ except for } w$$

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

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

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

$u_k := u_k/q, \ \forall$ 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

$$\varepsilon_1 = \min\{\{(f_{kn}-\ell_{kn})u_k \mid (k,n)\varepsilon P^+\},\{(c_{kn}-f_{kn})u_k \mid (k,n)\varepsilon P^-\}\}$$

$$\varepsilon_2 = \min\{(f_{kn}-\ell_{kn})u_k \mid (k,n)\varepsilon Y^-\}, \ \{(c_{kn}-f_{kn})u_k \mid (k,n)\varepsilon Y^+\}\}$$

$$\varepsilon = \min\{\varepsilon_1, \bar{q}\varepsilon_2, d_s\}.$$

Decrease by ε/u_k the flow of all arcs $(k,n)\varepsilon P^+$, increase by ε/u_k the flow of all arcs $(k,n)\varepsilon P^-$, decrease by $\varepsilon/(\bar{q}u_k)$ the flow of all arcs $(k,n)\varepsilon Y^-$, increase by $\varepsilon/(\bar{q}u_k)$ the flow of all arcs $(k,n)\varepsilon Y^+$, 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

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

Set

$f_{km} := \ell_{km}$, $\forall$ balanced arcs (k,m) with $k \in S$, $m \notin S$, or $k \in S$, $m \in S$ and $v_{km} > 0$

$f_{mk} := c_{km}$, $\forall$ balanced arcs (m,k) with $k \in S$, $m \notin S$, or $k \in S$, $m \in S$ and $v_{mk} < 0$

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

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) $u_i - K_{ij} u_j > 0$, $f_{ij} > \ell_{ij}$, 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 step 2a or 2b is entered either (1) an augmentation occurs (in which case step 2 is exited), or (2) the offending arc (i,j) satisfying (a) or (b) above enters the tree while another arc exits the tree and the vector u is suitably updated in step 2. It is seen that in case (2) 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) above, or u_i in case (b) above]. 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 re-encountered within step 2 and shows that, within a finite number of operations, step 2 will be exited 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 it is seen 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.

4. 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 an ordinary network with irrational data. There are two potential difficulties here:

(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) While an infinite number of dual ascent steps are performed, the generated sequence of dual function values converges short of the optimal.

Difficulty (a) above can be bypassed 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). A proof of this fact is given in Tseng [35] which also provides 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 [18], Minieka [37], Rockafellar [2], pp. 36, 66). With this device it can be shown (see [35]) 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) above can occur as shown in an example given in [35]. It can be bypassed by employing an idea from the ε-subgradient method (Bertsekas and Mitter [39], [40]). For any positive number ε and any tension vector t define each arc (i,j) to be

$$\varepsilon\text{-Inactive} \quad \text{if} \quad t_{ij} < a_{ij} - \varepsilon \tag{37}$$

$$\varepsilon\text{-Balanced} \quad \text{if} \quad a_{ij} - \varepsilon \leq t_{ij} \leq a_{ij} + \varepsilon \tag{38}$$

$$\varepsilon\text{-Active} \quad \text{if} \quad a_{ij} + \varepsilon < t_{ij}. \tag{39}$$

We will show that <u>if in the relaxation iteration the usual notions of active, inactive, and balanced arcs are replaced by the corresponding notions defined above, the algorithm will terminate in a finite number of iterations with a solution that is within</u> $\varepsilon \sum_{(i,j)} (c_{ij}-\ell_{ij})$ <u>of being optimal</u>. Furthermore <u>the final primal solution will be optimal if ε is chosen sufficiently small</u>.

Suppose first that we have a pair (f,t) satisfying primal and dual feasibility, and "ε-complementary slackness" in the sense of (11)-(14) but with the usual definitions of active, inactive and balanced arcs replaced by those of (37)-(39). Since f is primal feasible, it is seen 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 [cf.(4),(5)]

$$\sum_{(i,j)} a_{ij}f_{ij} = \sum_{(i,j)} (a_{ij}-t_{ij})f_{ij}$$

$$\geq \sum_{\substack{(i,j)\\ t_{ij}>a_{ij}}} (a_{ij}-t_{ij})c_{ij} + \sum_{\substack{(i,j)\\ t_{ij}<a_{ij}}} (a_{ij}-t_{ij})\ell_{ij} = q(p)$$

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

$$\sum_{(i,j)} a_{ij}f_{ij} = \sum_{(i,j)} (a_{ij}-t_{ij})f_{ij} =$$

$$= \sum_{\substack{(i,j)\\ t_{ij}>a_{ij}+\varepsilon}} (a_{ij}-t_{ij})c_{ij} + \sum_{\substack{(i,j)\\ t_{ij}<a_{ij}-\varepsilon}} (a_{ij}-t_{ij})\ell_{ij}$$

$$+ \sum_{\substack{(i,j)\\ |a_{ij}-t_{ij}|\leq\varepsilon}} (a_{ij}-t_{ij})f_{ij}$$

$$\leq \sum_{\substack{(i,j)\\ t_{ij}>a_{ij}}} (a_{ij}-t_{ij})c_{ij} + \sum_{\substack{(i,j)\\ t_{ij}<a_{ij}}} (a_{ij}-t_{ij})\ell_{ij} + \varepsilon \sum_{(i,j)} (c_{ij}-\ell_{ij})$$

$$= w(t) + \varepsilon \sum_{(i,j)} (c_{ij}-\ell_{ij}).$$

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

Suppose next that the relaxation iteration is executed with the definitions (12)-(14) for active, inactive and balanced arcs replaced by the corresponding "ε" notions of (37)-(39). Then it can be seen that the stepsize δ of (20) or (25) is bounded below by εL where L is a positive lower bound for $1/\max\{|v_{ij}|\,|\,(i,j)\varepsilon A\}$ 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 $\varepsilon \sum_{(i,j)} (c_{ij}-\ell_{ij})$ 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 is seen that if ε is sufficiently small then every solution of the perturbed primal problem is also a solution of the original primal problem. Therefore, <u>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</u>. However the required size of ε cannot be easily estimated a priori.

<u>Line Search</u>

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 showed that this type of line search is beneficial, and was implemented in the relaxation codes described in Section 6.

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.

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

reduced the price p_s (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 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 sth) coordinate direction is chosen and a line search is performed 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

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

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

We have

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

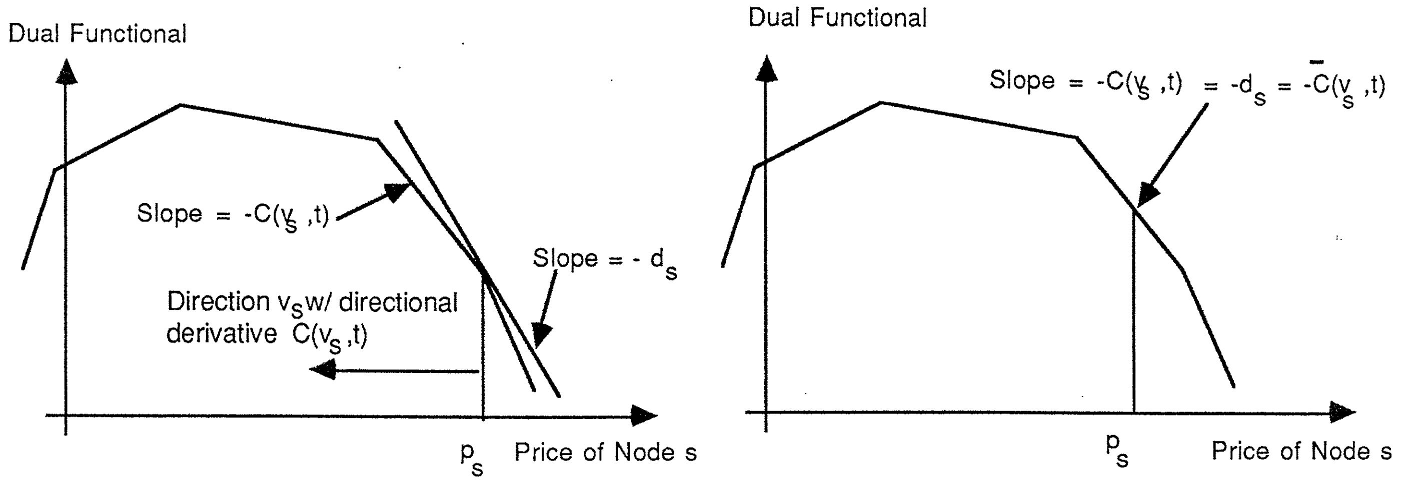
so $-d_s$ is a subgradient of the dual functional at p_s in the sth coordinate direction. A single node iteration will be possible if and only if the right slope is negative or equivalently $\overline{C}(v_s,t) > 0$.

This 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 that (see Figure 3)

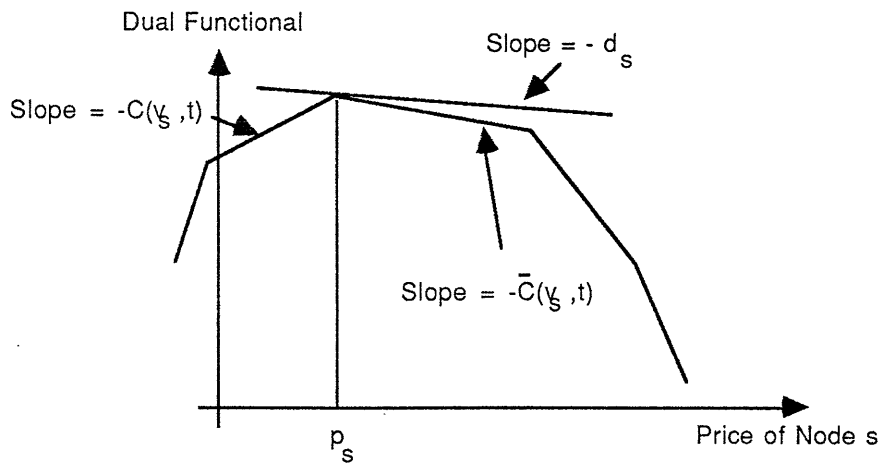
$$-\bar{C}(v_s,t) \leq -d_s < 0 \leq -C(v_s,t)$$

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

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



CASES WHERE A SINGLE NODE ITERATION IS POSSIBLE



CASE WHERE A SINGLE NODE ITERATION IS NOT POSSIBLE

Figure 3: 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.

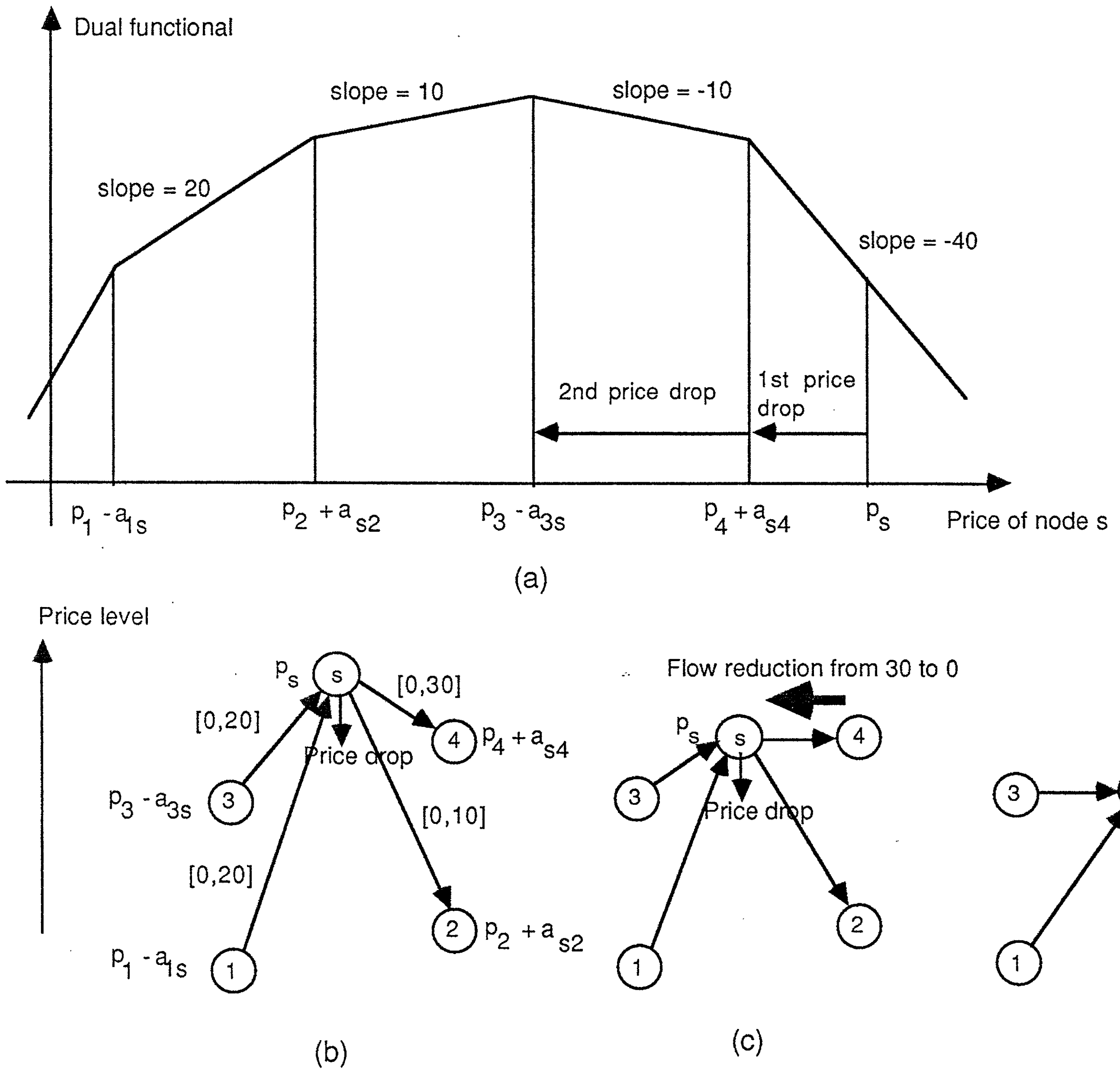
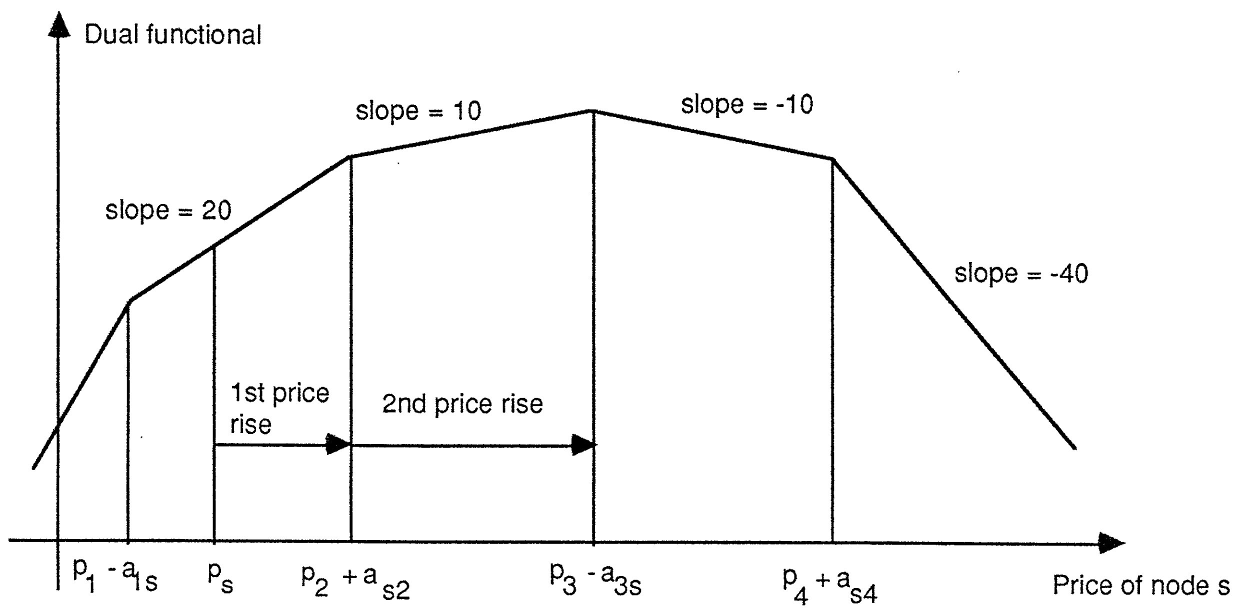
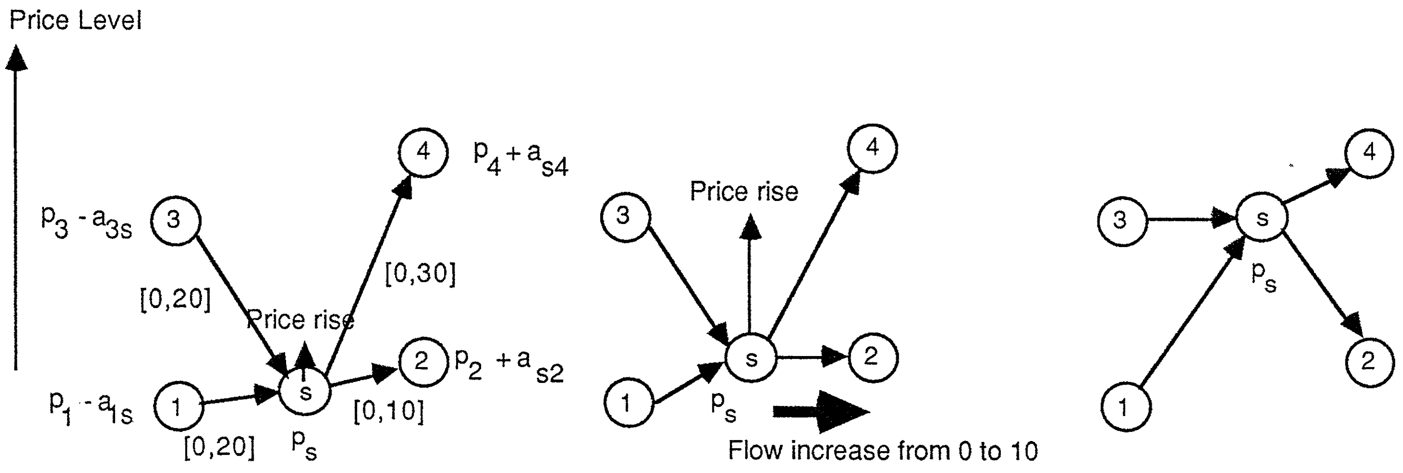


Figure 4: Illustration of an iteration involving a single node s with four adjacent arcs (1,s), (3,s), (s,2), (s,4) with feasible arc flow ranges [1,20], [0,20], [0,10], [0,30] respectively.
(a) Form of the dual functional along p_s for given values of p_1, p_2 p_3, and p_4. The break points correspond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



(a)



(b) (c) (d)

Figure 5: Illustration of a price rise involving the single node s for the example of Fig. 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.

Degenerate Ascent Iterations

Consider the case of an ordinary network. 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.

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

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 $C(v,t) > 0$ [cf. (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 without degenerate steps under the following condition (see [6]):

(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.

This condition holds when the set S consists of just the starting node s. Thus 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 (see Section 6) 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, but 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. The computational complexity analysis of [45] indicates that this is an important factor in the speed of convergence of the algorithm.

5. Basis for Computational Experimentation

Historically computational experimentation has been the primary method for comparative evaluation of network flow algorithms. During the sixties it was generally believed that primal-dual methods held an advantage over simplex methods. However during the seventies major improvements in implementation [14]-[20] using sophisticated data structures have propelled simplex algorithms to a position of prominence as far as general minimum cost flow problems are concerned. The situation is less clear for special classes of problems such as assignment where some computational comparisons [21], [22] 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 aiding substantively the comparison of different methods are in scarce supply. An interesting observation was made by Zadeh [23] 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 [24]. This is significant in view of recent average complexity results for the parametric method (Haimovich [25]). The "big-M" method is known to be more effective for network problems than the PhaseI-PhaseII method (Mulvey [26]). However there are pivot rules that are empirically more effective than the most negative rule and much research has been directed along this direction ([26]-[28]). Zadeh [23] 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. [16]) as well as our own (see Section 7).

We have compared our two relaxation codes called RELAX-II and RELAXT-II with two state-of-the-art codes; KILTER (a primal-dual code due to Aashtiani and Magnanti [29]) and RNET (a primal simplex code due to Grigoriadis and Hsu [30]). A description of each of these is given in the next section. We describe below our experimental approach:

Test Conditions

All methods were tested under identical conditions, same computer (VAX 11/750 running VMS version 4.1), same language (FORTRAN IV), same compiler (standard FORTRAN of the VMS system version 3.7 in the OPTIMIZE mode), same timing routine, and same system conditions (empty system at night). RELAX-II and RELAXT-II were also compiled under VMS version 4.1 and run about 15%-20% faster than when compiled under VMS version 3.7. The CPU times reported were obtained 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 computation time. The size of problems used in our experiments and the system configuration were such that page faults never affected significantly 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 [19] and [10] suggest that RNET is faster

on standard NETGEN benchmark problems [31] (see Table 1) than PNET [31] and GNET [17] 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 in their 1980 book ([20], p. 255) that "RNET... produced the shortest times that we have seen on these 35 test problems". Our own experimentation generally supports these findings and suggests that for general minimum cost flow problems RNET is at least as fast and probably faster than any other non-commercial simplex code for which computation times on benchmark problems are available to us ([17], [20], [22], [29], [31])). See also the experimentation in [41] which finds the commercial code ARCNET slightly superior to RNET.

KILTER is an implementation of the primal-dual method that uses a sophisticated labeling scheme described in [29]. The version we tested is the fastest of nine versions tested in [29] where it is called KILTER9. On the basis of the limited computational results and indirect comparisons of [29], KILTER outperforms by a wide margin earlier primal-dual codes such as SUPERK, SHARE and BOEING [31], and is comparable to the simplex code PNET [31]. KILTER is also generally faster than the faster primal-dual codes that we have been able to implement (see [10]). However an extensive computational study by Mulvey and Dembo [33] shows that KILTER is outperformed on assignment problems under identical test conditions by LPNET (a primal simplex code due to Mulvey [18]). Our own experimentation also shows that KILTER is consistently outperformed by RNET and agrees 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 to date with the primal simplex and primal-dual methods, respectively.

6. Code Descriptions

The relaxation codes RELAX-II and RELAXT-II that we implemented solve the problem

$$\text{mimimize} \quad \sum_{(i,j)\varepsilon A} a_{ij}\, f_{ij}$$

subject to

$$\sum_{(m,i)\varepsilon A} f_{mi} - \sum_{(i,m)\varepsilon A} f_{im} = b_i, \quad \forall\ i\varepsilon N$$

$$0 \leq f_{ij} \leq c_{ij} \qquad , \quad \forall\ (i,j)\varepsilon A.$$

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 [29] and used in their KILTER code (see also Magnanti [34]). 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 $|A|$ and 7$|N|$ for node length arrays. RELAXT-II is a code that 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 faster than RELAX-II but requires 9.5 $|A|$ + 9 $|N|$ total storage.

This compares unfavorably with 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 3. Line search and degenerate ascent steps are implemented as discussed in Section 4. 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) 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 head and tail nodes of the arc. We found experimentally that this 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 (fewer extreme points to potentially search over).

We finally note that RELAX-II and RELAXT-II, finalized in September 1986, are much more efficient than earlier versions [3], [10], [33], particularly for sparse and uncapacitated problems. In this connection we note that a computational comparision of RELAX and RNET was undertaken in [44] using a memory-limited machine. However the code used there 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 in [3], let alone the second generation version tested in this paper.

7. Computational Results

Our computational results are organized in six tables. All the problems shown were generated using the widely used, publicly available NETGEN program [31]. The random number seed used is 13502460 (the same as the one in [31]). All additional information needed to replicate these problems is given in the corresponding tables. The thesis of the second author [35] includes computational experience with gain networks. These results are preliminary and show that relaxation is roughly competitive with a state of the art primal simplex code of Currin [36] (tests done under identical conditions on the same machine). More experimentation is required to corroborate these results.

Table 1: (Standard NETGEN Benchmarks)

This table shows the results for the 40 problems described in detail in [31] and generated by the NETGEN program. Problem 36 was not solved because for some unexplained reason our NETGEN code was producing an infeasible problem for the problem data given in [31]. The results show the substantial superiority of RELAX-II and RELAXT-II over the other codes for assignment and transportation problems. This finding was corroborated on a large number of additional assignment and transportation problems of broadly 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 2: (Transportation Problems)

These results are in general agreement with those of Table 1. 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 3: (Transportation Problems-Large Cost Range)

The problems in this table are identical to those in Table 2 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 2.

Table 4: (Heavily Capacitated Transhipment Problems)

Our experience with heavily capacitated transhipment problems with positive arc costs is similar to that for transportation problems.

Table 5: (Transhipment Problems with Both Positive and Negative Arc Costs)

The problems in this table are identical to those of Table 4 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 [10]). 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. This policy was followed in the runs shown in Table 5. It can be seen that the factor of superiority of RELAX -II and RELAXT-II over RNET increases somewhat relative to the results of Table 4.

Table 6: (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. This can be seen by comparing the results for the small problems 1-35 of Table 1 with the results for the larger problems 37-40 of Table 1, and the problems of Tables 2 through 5. The comparison shows an improvement in speedup factor that is not spectacular, but is noticeable and consistent. Table 6 shows that for even larger problems the speedup factor increases further with problem dimension, and reaches or exceeds an order of magnitude (see 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 there was some difficulty in generating the transportation problems of this table with NETGEN. Many of the problems generated were infeasible because some node supplies and demands were coming out zero or negative. This was resolved by adding the same number (usually 10) to all source supplies and all sink demands as noted in Table 6. 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 6 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 6. 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 7. More experimentation and/or analysis is needed to establish conclusively the computational complexity implications of these experiments.

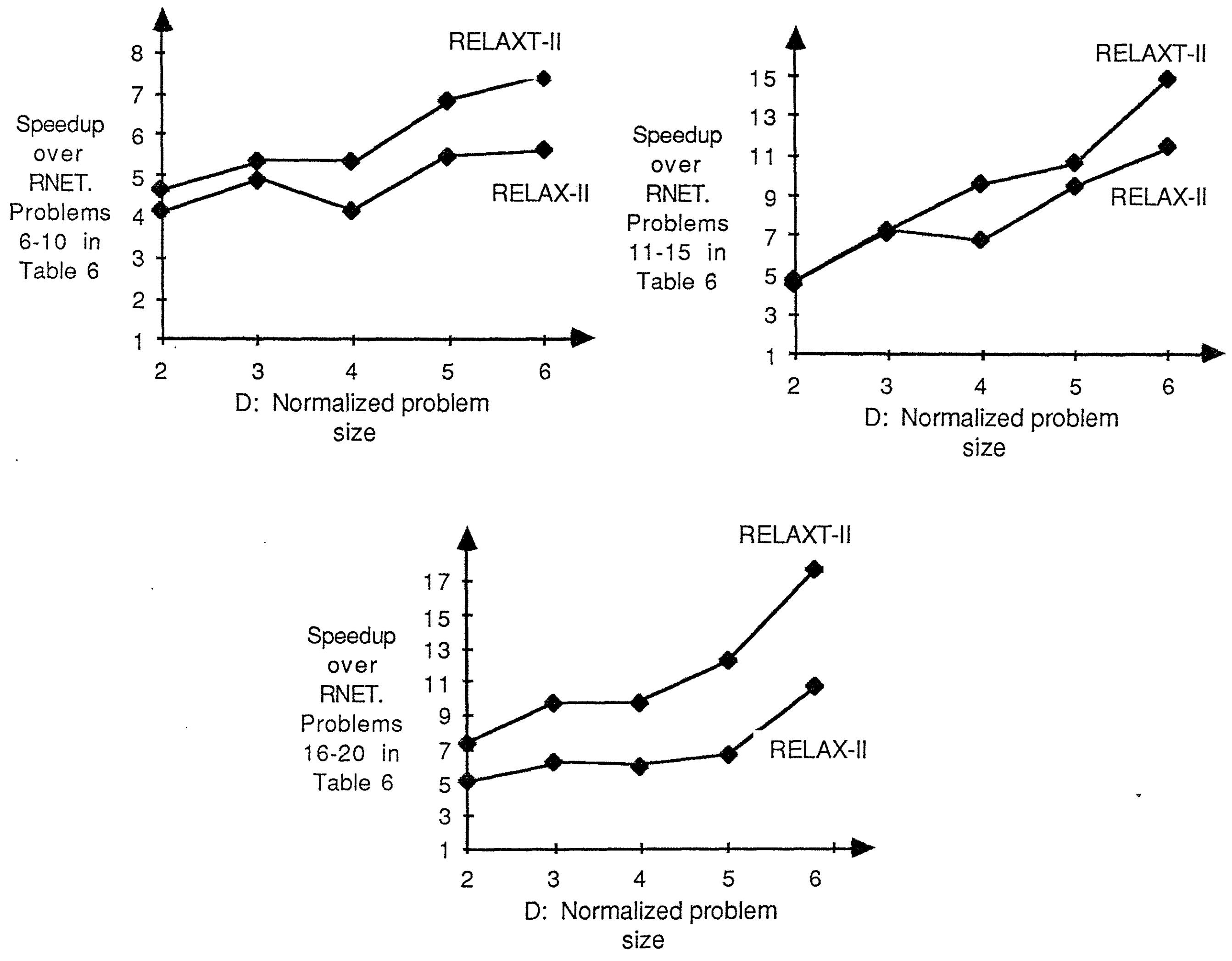


Figure 6: Speedup factor of RELAX-II and RELAXT-II over RNET for the transportation problems of Table 6. The normalized dimension D gives the number of nodes N and arcs A as follows:

$N = 1000^*D$, $A = 4000^*D$, for Problems 6 - 15

$N = 500^*D$, $A = 5000^*D$, for Problems 16 - 20.

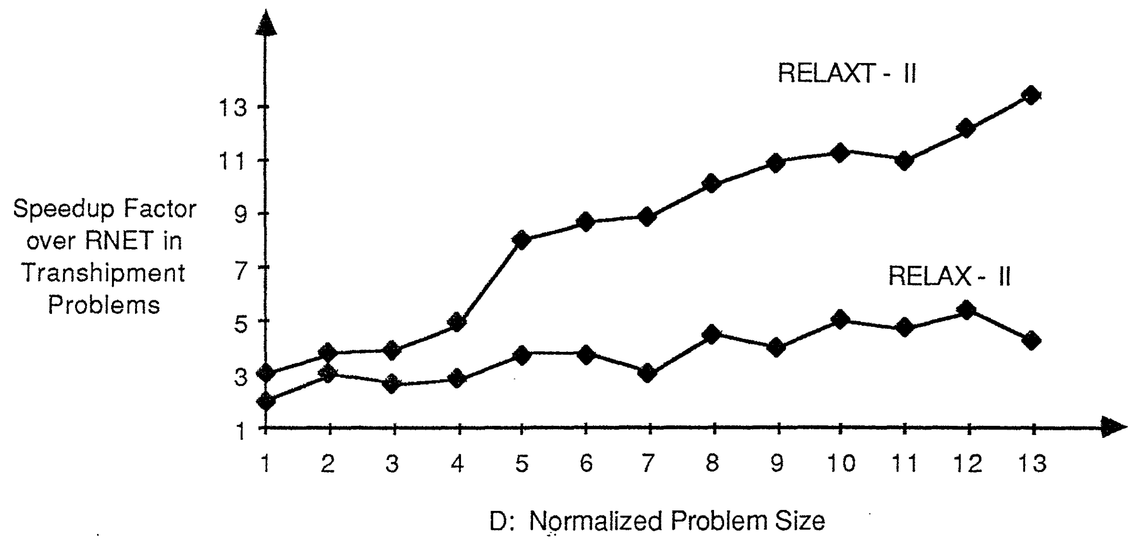


Figure 7: Speedup factor of RELAX-II and RELAXT-II over RNET in lightly capacitated transhipment problems generated by our own random problem generator RANET. Each node is a transhipment node, and it is either a source or a sink. The normalized problem size D gives the number of nodes and arcs as follows:

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

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.

Problem Type	Problem #	# of Nodes	# of Arcs	RELAX-II (VMS 3.7/ VMS 4.1)	RELAXT-II (VMS 3.7/ VMS 4.1)	KILTER VMS 3.7	RNET VMS 3.7
Transportation	1	200	1300	2.07/1.75	1.47/1.22	8.81	3.15
	2	200	1500	2.12/1.76	1.61/1.31	9.04	3.72
	3	200	2000	1.92/1.61	1.80/1.50	9.22	4.42
	4	200	2200	2.52/2.12	2.38/1.98	10.45	4.98
	5	200	2900	2.97/2.43	2.53/2.05	16.48	7.18
	6	300	3150	4.37/3.66	3.57/3.00	25.08	9.43
	7	300	4500	5.46/4.53	3.83/3.17	35.55	12.60
	8	300	5155	5.39/4.46	4.30/3.57	46.30	15.31
	9	300	6075	6.38/5.29	5.15/4.30	43.12	18.99
	10	300	6300	4.12/3.50	3.78/3.07	47.80	16.44
Total (Problems 1-10)				37.32/31.11	30.42/25.17	251.85	96.22
Assignment	11	400	1500	1.23/1.03	1.35/1.08	8.09	4.92
	12	400	2250	1.38/1.16	1.54/1.25	10.76	6.43
	13	400	3000	1.68/1.42	1.87/1.54	8.99	8.92
	14	400	3750	2.43/2.07	2.67/2.16	14.52	9.90
	15	400	4500	2.79/2.34	3.04/2.46	14.53	10.20
Total (Problems 11-15)				9.51/8.02	10.47/8.49	56.89	40.37
Uncapacitated & Lightly Capacitated Problems	16	400	1306	2.79/2.40	2.60/2.57	13.57	2.76
	17	400	2443	2.67/2.29	2.80/2.42	16.89	3.42
	18	400	1306	2.56/2.20	2.74/2.39	13.05	2.56
	19	400	2443	2.73/2.32	2.83/2.41	17.21	3.61
	20	400	1416	2.85/2.40	2.66/2.29	11.88	3.00
	21	400	2836	3.80/3.23	3.77/3.23	19.06	4.48
	22	400	1416	2.56/2.18	2.82/2.44	12.14	2.36
	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	5.34
	26	400	1382	1.79/1.57	1.60/1.41	11.31	2.48
	27	400	2676	2.15/1.84	1.97/1.75	18.88	3.62
Total (Problems 16-27)				32.09/27.42	31.22/27.38	192.88	41.94

TABLE 1 (continued on next page)

Problem Type	Problem #	# of Nodes	# of Arcs	RELAX-II (VMS 3.7/ VMS 4.41)	RELAXT-II (VMS 3.7/ VMS 4.1)	KILTER VMS 3.7	RNET VMS 3.7
Uncapacitated and Lightly Capacitated Problems	28	1000	2900	4.90/4.10	5.67/5.02	29.77	8.60
	29	1000	3400	5.57/4.76	5.13/4.43	32.36	12.01
	30	1000	4400	7.31/6.47	7.18/6.26	42.21	11.12
	31	1000	4800	5.76/4.95	7.14/6.30	39.11	10.45
	32	1500	4342	8.20/7.07	8.25/7.29	69.28	18.04
	33	1500	4385	10.39/8.96	8.94/7.43	63.59	17.29
	34	1500	5107	9.49/8.11	8.88/7.81	72.51	20.50
	35	1500	5730	10.95/9.74	10.52/9.26	67.49	17.81
Total (Problems 28-35)				62.57/54.16	61.71/53.80	356.32	115.82
Large Uncapacitated Problems	37	5000	23000	87.05/73.64	74.67/66.66	681.94	281.87
	38	3000	35000	68.25/57.84	55.84/47.33	607.89	274.46
	39	5000	15000	89.83/75.17	66.23/58.74	558.60	151.00
	40	3000	23000	50.42/42.73	35.91/30.56	369.40	174.74
Total (Problems 37-40)				295.55/249.38	232.65/203.29	2,217.83	882.07

TABLE 1: Standard Benchmark Problems 1-40 of [31] 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 #	# of Sources	# of Sinks	# of Arcs	Cost Range	RELAX-II	RELAXT-II	RNET
1	200	200	7,000	1-100	6.54	4.78	26.58
2	400	400	"	"	10.18	8.83	53.71
3	600	600	"	"	16.74	11.83	80.98
4	800	800	"	"	15.72	12.89	122.46
5	1,000	1,000	"	"	20.19	19.98	129.91
6	200	200	6,000	1-100	6.23	4.06	23.02
7	"	"	8,000	"	8.30	5.00	28.74
8	"	"	10,000	"	10.61	6.36	32.77
9	"	"	12,000	"	17.13	8.26	36.36
10	"	"	15,000	"	14.10	8.38	36.60
Total (Problems 1-10)					125.74	90.37	571.13
11	100	300	7,000	1-100	7.88	5.08	20.94
12	200	600	"	"	9.36	8.47	49.14
13	300	900	"	"	13.13	9.54	68.64
14	350	1,050	"	"	18.04	13.84	89.32
15	400	1,200	"	"	16.66	15.12	110.05
16	100	300	6,000	1-100	5.88	4.00	18.68
17	"	"	8,000	"	11.76	6.46	21.97
18	"	"	10,000	"	8.69	6.75	25.44
19	"	"	12,000	"	12.49	7.66	26.14
20	"	"	15,000	"	16.24	9.16	38.74
Total (Problems 11-20)					120.13	86.08	469.06

TABLE 2: Transportation Problems. Times in Secs on VAX 11/750. All Problems Obtained Using NETGEN with Total Supply 200,000 and 0% High Cost Arcs. RELAX-II and RELAXT-II compiled under VMS 4.1; RNET compiled under VMS 3.7.

Problem #	#of Sources	#of Sinks	# of Arcs	Cost Range	RELAX-II	RELAXT-II	RNET
1	200	200	7,000	$1\text{-}10^4$	14.65	12.63	28.52
2	400	400	7,000	$1\text{-}10^4$	17.64	18.37	57.77
3	600	600	7,000	$1\text{-}10^4$	30.21	40.41	80.12
4	800	800	7,000	$1\text{-}10^4$	26.17	40.91	117.89
5	1,000	1,000	7,000	$1\text{-}10^4$	61.16	34.75	145.31
6	200	200	6,000	$1\text{-}10^4$	16.76	11.35	23.01
7	200	200	8,000	$1\text{-}10^4$	19.07	15.02	28.88
8	200	200	10,000	$1\text{-}10^4$	16.44	12.70	35.32
9	200	200	12,000	$1\text{-}10^4$	19.32	18.63	41.55
10	200	200	15,000	$1\text{-}10^4$	34.21	24.23	41.87
Total (Problems 1-10)					255.63	229.00	600.24
11	100	300	7,000	$1\text{-}10^4$	11.89	11.22	22.78
12	200	600	7,000	$1\text{-}10^4$	27.57	13.38	46.69
13	300	900	7,000	$1\text{-}10^4$	19.33	22.79	72.95
14	350	1050	7,000	$1\text{-}10^4$	31.01	34.11	91.97
15	400	1200	7,000	$1\text{-}10^4$	32.70	27.90	108.97
16	100	300	6,000	$1\text{-}10^4$	9.35	13.14	20.21
17	100	300	8,000	$1\text{-}10^4$	21.42	18.12	25.95
18	100	300	10,000	$1\text{-}10^4$	18.75	17.59	30.07
19	100	300	12,000	$1\text{-}10^4$	21.37	22.14	30.66
20	100	300	15,000	$1\text{-}10^4$	40.29	29.31	42.67
Total Problems (11-20)					233.68	209.70	492.92

TABLE 3: Transportation Problems. Times in Secs on VAX 11/750. All Problems Obtained Using NETGEN with Total Supply 200,000 and 0% High Cost Arcs. RELAX-II and RELAXT-II compiled under VMS 4.1; RNET compiled under VMS 3.7.

Problem #	# of Sources	# of Sinks	# 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	"	"	"	16.99	12.10	73.10
3	600	600	"	"	"	21.62	22.25	97.11
4	800	800	"	"	"	26.86	22.35	108.09
5	1,000	1,000	"	"	"	29.26	26.54	102.74
6	200	200	6,000	1-100	100-1,000	9.25	6.26	39.18
7	"	"	8,000	"	"	10.71	8.53	48.87
8	"	"	10,000	"	"	14.58	8.57	51.98
9	"	"	12,000	"	"	17.78	10.62	68.17
10	"	"	15,000	"	"	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	"	"	"	20.11	13.21	68.29
13	300	900	"	"	"	25.85	22.59	90.36
14	400	1,200	"	"	"	35.23	27.43	109.75
15	500	1,500	"	"	"	40.92	31.60	107.32
16	100	300	6,000	1-100	100-1,000	9.02	6.97	33.45
17	"	"	8,000	"	"	12.44	9.56	39.59
18	"	"	10,000	"	"	16.26	8.87	48.61
19	"	"	12,000	"	"	20.46	11.06	59.36
20	"	"	15,000	"	"	22.47	11.36	72.41
Total (Problems 11-20)						213.14	149.83	673.12

TABLE 4: Capacitated Transhipment Problems. Times in Secs on VAX 11/750. All Problems 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 compiled under VMS 4.1; RNET compiled under VMS 3.7.

Problem #	# of Sources	# of Sinks	# 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	"	"	"	17.99	12.88	92.11
3	600	600	"	"	"	17.10	16.38	109.35
4	800	800	"	"	"	27.82	24.62	124.42
5	1,000	1,000	"	"	"	38.02	31.48	123.87
6	200	200	6,000	-50-50	100-1,000	10.09	5.83	49.15
7	"	"	8,000	"	"	12.40	7.99	67.74
8	"	"	10,000	"	"	12.71	9.79	81.95
9	"	"	12,000	"	"	16.72	10.28	89.71
10	"	"	15,000	"	"	30.78	13.73	94.58
Total (Problems 1-10)						194.65	144.42	888.38
11	100	100	7,000	-50-50	100-500	10.86	5.74	36.35
12	200	600	"	"	"	17.00	12.07	79.24
13	300	900	"	"	"	21.95	18.61	107.43
14	400	1,200	"	"	"	27.14	21.47	136.58
15	500	1,500	"	"	"	41.19	30.86	111.57
16	100	300	6,000	-50-50	100-1,000	10.15	7.17	48.22
17	"	"	8,000	"	"	11.52	8.10	64.62
18	"	"	10,000	"	"	15.68	13.02	86.84
19	"	"	12,000	"	"	19.87	11.04	97.36
20	"	"	15,000	"	"	32.17	16.53	128.33
Total (Problems 11-20)						207.53	144.61	896.54

TABLE 5: Capacitated Transhipment Problems with Both Negative and Positive Arc Costs. Same Problems as in Table 4 except that the Cost Range is [-50,50]. RELAX-II and RELAXT-II compiled under VMS 4.1; RNET compiled under VMS 3.7.

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

Table 6: 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. Note also that an alternative implementation of RELAXT-II, currently in the final stages of development has resulted in reduction of the arc length arrays by one third without sacrificing speed of execution.

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 [6], [35], [42], [43]) including general linear programming problems. It also lends itself well to massively parallel computation (see [4], [6], [11], [43], [45], [46]).

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 from the authors at no cost on IBM-PC or Macintosh diskette.

References

[1] Rockafellar, R. T., Convex Analysis, Princeton Univ. Press, 1970.

[2] Rockafellar, R. T., Network Flows and Monotropic Optimization, J. Wiley, N.Y., 1984.

[3] Bertsekas, D. P., "A Unified Framework for Minimum Cost Network Flow Problems", LIDS Report LIDS-P-1245-A, M.I.T., October 1982; also Math. Programming, Vol. 25, 1985.

[4] Bertsekas, D. P., "Distributed Relaxation Methods for Linear Network Flow Problems", Proc. 25th IEEE Conference on Decision and Control, Athens, Greece, Dec. 1986.

[5] Bertsekas, D. P., "A New Algorithm for the Assignment Problem", Math. Programming, Vol. 21, 1982, pp. 152-171.

[6] Bertsekas, D. P., Hosein, P. and Tseng, P., "Relaxation Methods for Network Flow Problems with Convex Arc Costs", SIAM J. Control and Opt., to appear.

[7] Ford, L. R., Jr., and Fulkerson, D. R., Flows in Networks, Princeton Univ. Press, Princeton, N.J., 1962.

[8] Lawler, E. L., Combinatorial Optimization: Networks and Matroids, Holt, Rinehart & Winston, New York, 1976.

[9] Papadimitriou, C. H., and Steiglitz, K., Combinatorial Optimization: Algorithms and Complexity, Prentice Hall, Englewood Cliffs, N. J., 1982.

[10] Bertsekas, D. P. and Tseng, P., "Relaxation Methods for Minimum Cost Network Flow Problems", LIDS Report P-1339, M.I.T., Oct. 1983.

[11] Bertsekas, D. P and El Baz, D., "Distributed Asynchronous Relaxation Methods for Convex Network Flow Problems", LIDS Report-P-1417, M.I.T., Oct. 1984, SIAM. J. Control and Optimization, Vo. 25, 1987, pp. 74-85.

[12] Lasdon, L. S., Optimization Theory for Large Systems, McMillan, N.Y., 1970.

[13] Ortega, J. M. and Rheinboldt, W. C., Iterative Solution of Nonlinear Equations in Several Variables, Academic Press, N.Y., 1970.

[14] Srinivasan, V. and Thompson, G. L., "Benefit-Cost Analysis of Coding Techniques for the Primal Transportation Algorithm", JACM, Vol. 20, pp. 194-213.

[15] Glover, F., Karney, D., Klingman, D., and Napier, A., "A Computation Study on Start Procedures, Basis Change Criteria and Solution Algorithms for Transportation Problems", Management Science, Vol. 20, 1974, pp. 793-819.

[16] Glover, F., Karney, D., and Klingman, D., "Implementation and Computational Comparisons of Primal, Dual and Primal-Dual Computer Codes for Minimum Cost Network Flow Problems", Networks, Vol. 4, 1974, pp. 191-212.

[17] Bradley, G. H., Brown, G. G., and Graves, G. W., "Design and Implementation of Large Scale Transhipment Algorithms", Management Science, Vol. 24, 1977, pp. 1-34.

[18] Johnson E., "Networks and Basic Solutions", Operations Research, Vol. 14, 1966, pp. 619-623.

[19] Grigoriadis, M. D., "Algorithms for the Minimum Cost Single and Multicommodity Network Flow Problems", Notes for Summer School in Combinatorial Optimization SOGESTA, Urbino, Italy, July 1978.

[20] Kennington, J., and Helgason, R., Algorithms for Network Programming, Wiley, N.Y., 1980.

[21] Hatch, R. S., "Bench Marks Comparing Transportation Codes Based on Primal Simplex and Primal-Dual Algorithms", Operations Research, Vol. 23, 1975, pp. 1167-1172.

[22] McGinnis, L. F., "Implementation and Testing of a Primal-Dual Algorithm for the Assignment Problem", Operations Research, Vol. 31, pp. 277-291, 1983.

[23] Zadeh, N., "Near-Equivalence of Network Flow Algorithms", Tech. Rep. No. 26, Department of Operations Research, Stanford University, Dec. 1979.

[24] Gallager, R. G., "Parametric Optimization and the Primal-Dual Method", Lab. for Information and Decision Systems Report, Mass. Institute of Technology, June 1983.

[25] Haimovich, M., "The Simplex Algorithm is Very Good! -- On the Expected Number of Pivot Steps and Related Properties of Random Linear Programs", Columbia Univ. Report, April 1983.

[26] Mulvey, J. M., "Pivot Strategies for Primal-Simplex Network Codes", JACM, Vol. 25, 1978, pp. 266-270.

[27] Mulvey, J. M., "Testing of a Large-Scale Network Optimization Program", Math. Programming, Vol. 15, 1978, pp. 291-314.

[28] Gavish, B., Schweitzer, P., and Shlifer, E., "The Zero Pivot Phenomenon in Transportation Problems and Its Computational Implications", Math. Programming, Vol. 12, 1977, pp. 226-240.

[29] Aashtiani, H. A., and Magnanti, T. L., "Implementing Primal-Dual Network Flow Algorithms", Operations Research Center Report 055-76, Mass. Institute of Technology, June 1976.

[30] Grigoriadis, M. D., and Hsu, T., "The Rutgers Minimum Cost Network Flow Subroutines", (RNET documentation), Dept. of Computer Science, Rutgers University, Nov. 1980.

[31] Klingman, D., Napier, A., and Stutz, J., "NETGEN--A Program for Generating Large Scale (Un)capacitated Assignment, Transportation and Minimum Cost Flow Network Problems", Management Science, Vol. 20, pp. 814-822., 1974.

[32] Barr, R. S., and Turner, J. S., "Microdata File Merging Through Large-Scale Network Technology", Math. Programming Study 15, 1981, pp. 1-22.

[33] Dembo, R. S., and Mulvey, J. M., "On the Analysis and Comparison of Mathematical Programming Algorithms and Software", Harvard Business School Report 76-19, 1976.

[34] Magnanti, T., "Optimization for Sparse Systems", in Sparse Matrix Computations (J. R. Bunch and D. J. Rose, eds.), Academic Press, N.Y., 1976, pp. 147-176.

[35] Tseng, Paul, "Relaxation Methods for Monotropic Programs", Ph.D. Thesis, M.I.T., May 1986.

[36] Currin, D. C., "A Comparative Evaluation of Algorithms for Generalized Network Problems", NRIMS Tech. Report TWISK 289, Pretoria, South Africa, 1983.

[37] Minieka, E., Optimization Algorithms for Networks and Graphs, Marcel Dekker, New York, 1978.

[38] Jewell, W. S., "Optimal Flow Through Networks With Gains", Operations Research, Vol. 10, No. 4, pp. 476-499, 1962.

[39] Bertsekas, D. P. and Mitter, S. K., "Steepest Descent for Optimization Problems with Non-differentiable Cost Functionals", Proc. Annual Princeton Conference Inform. Sci. Systems, 5th, Princeton, N.J., 1971, pp. 347-351.

[40] Bertsekas, D. P. and Mitter, S. K., "A Descent Numerical Method for Optimization Problems with Non-differentiable Cost Functionals", SIAM J. Control, Vol. 11, No. 4, Nov. 1973, pp. 637-652.

[41] Glover, F. and Klingman, D., "Recent Developments in Computer Implementation Technology for Network Flow Algorithms", INFOR, Vol. 20, No. 4, November 1982, pp. 433-452.

[42] Tseng, P., and Bertsekas, D. P., "Relaxation Methods for Linear Programs", LIDS Report LIDS-P1553, M.I.T., April 1986, Math. of O.R., (to appear 1987).

[43] Tseng, P., and Bertsekas, D. P., "Relaxation Methods for Problems with Strictly Convex Separable Costs and Linear Constraints", LIDS Report LIDS-P-1567, M.I.T., June 1986.

[44] Grigoriadis, M. D., "An Efficient Implementation of the Network Simplex Method", Math. Programming Studies, Vol. 26, 1986.

[45] Bertsekas, D. P., and Eckstein, J., "Distributed Asynchronous Relaxation Methods for Linear Network Flow Problems", Report LIDS-P-1606, M.I.T., Sept. 1986 (rev. Jan. 1987), to appear in Proc of IFAC-87, Munich, Germany, Pergamon Press, Oxford, England, July 1987.

[46] Bertsekas, D. P., "The Auction Algorithm: A Distributed Relaxation Method for the Assignment Problem", Report LIDS-P-1653, M.I.T., March 1987, to appear in Annals of Operations Research.