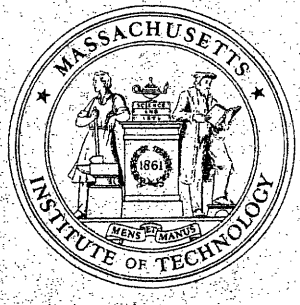
OPERATIONS RESEARCH CENTER

working paper



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

IMPROVED PRIMAL SIMPLEX ALGORITHMS FOR SHORTEST PATH, ASSIGNMENT AND MINIMUM COST FLOW PROBLEMS

Ravindra K. Ahuja
and
James B. Orlin

OR 189-88 November, 1988

Improved Primal Simplex Algorithms for Shortest Path, Assignment and Minimum Cost Flow Problems

Ravindra K. Ahuja* and James B. Orlin
Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA. 02139, U.S.A.

* On leave from Indian Institute of Technology, Kanpur - 208 016, INDIA

Improved Primal Simplex Algorithms for Shortest Path, Assignment and Minimum Cost Flow Problems

Abstract

In this paper, we present a new primal simplex pivot rule and analyse the worst-case complexity of the resulting simplex algorithm for the minium cost flow problem, the assignment problem and the shortest path problem. We consider networks with n nodes, m arcs, integral arc capacities bounded by an integer number U, and integral arc costs bounded by an integer number C. Let L and U denote the nonbasic arcs at their lower and upper bounds respectively, and $\bar{c}_{ij}$ denote the reduced cost of any arc (i, j). Further, let Δ be a parameter whose initial value is C. Then our pivot rule is as follows: Select as an entering arc any $(i, j) \in L$ with $\bar{c}_{ij} \leq -\Delta/2$ or any $(i, j) \in U$ with $\bar{c}_{ij} \geq \Delta/2$; select the leaving arc so that the strong feasibility of the basis is maintained. When there is no nonbasic arc satisfying this rule then replace Δ by $\Delta/2$. We show that the simplex algorithm using this rule performs O(nm U logC) pivots and can be implemented to run in $O(m^2 U \log C)$ time. Specializing these results for the assignment and shortest path problems we show that the simplex algorithm solves these problems in $O(n^2 \log C)$ pivots and O(nm logC) time. These algorithms use the same data structures that are typically used to implement the primal simplex algorithms for network problems and have enough flexibility for fine tuning the algorithms in practice. We also use these ideas to obtain an O(nm logC) label correcting algorithm for the shortest path problem with arbitrary arc lengths, and an improved implementation of Dantzig's pivot rule.

Subject Classification.

Networks/Graphs : Improved simplex algorithms for network flow problems

Keywords : Network flow algorithms, Minimum cost flow problem, Assignment problem, Shortest path problem, Simplex algorithm, Scaling, Matchings.

In this paper, we present a new primal simplex pivot rule and analyze the worst-case behavior of the resulting simplex algorithm for the minimum cost flow problem. These results are also specialized for the assignment and shortest path problems.

We consider a directed network $G = (N, A)$ with node set N and arc set A. Each arc $(i, j) \in A$ has a non-negative integer capacity u_{ij} and an integer cost c_{ij}. We denote by n, m, U, and C, the number of nodes, number of arcs, maximum arc capacity, and maximum absolute value of an arc cost, respectively. We represent by A(i) the set of arcs incident on node i, i.e., the set of incoming and outgoing arcs at node i. Let $deg(i) = |A(i)|$ denote the degree of node i for each $i \in N$.

The primal simplex algorithm for the minimum cost flow problem, subsequently referred to as the *network simplex algorithm*, has been extensively studied in the literature. Researchers have also been interested in developing polynomial time network simplex algorithms for the minimum cost flow problem and its special cases. Polynomial time primal simplex algorithms have been developed for the shortest path problem, the assignment problem and the maximum flow problem. The only polynomial time simplex algorithm for the minimum cost flow problem is a dual simplex algorithm due to Orlin [1984] which performs $O(n^3 \log n)$ pivots for the uncapacitated minimum cost flow problem. Developing a polynomial time primal simplex algorithm for the minimum cost flow problem is still an open problem.

We now briefly review the literature devoted to this area of research. Dial, Glover, Karney and Klingman [1979] and Zadeh [1979] showed that Dantzig's pivot rule (i.e., pivoting in the arc with minimum reduced cost) for the shortest path problem with non-negative arc lengths performs O(n) pivots when started from an all artificial basis. Roohy-Laleh [1980] in his unpublished Ph.D. thesis developed an alternative simplex pivot rule for the assignment problem for which the number of pivots is $O(n^3)$. Hung [1983] developed a simplex method for the assignment problem in which the number of pivots is $O(n^3 \log(nC))$.

Orlin [1985] showed that for integer data the primal simplex algorithm maintaining strongly feasible bases performs O(nm CU) pivots for any arbitrary pivot rule and O(nm U log (mCU)) for Dantzig's pivot rule. (This result was independently developed by Dantzig [1983].) When specialized to the shortest path problem with

arbitrary arc lengths and the assignment problem, the algorithm performs $O(n^2 \log(nC))$ pivots.

Akgul [1985a, 1985b] developed primal simplex algorithms for the shortest path and assignment problems that perform $O(n^2)$ pivots. Using simple data structures, Akgul's algorithms run in $O(n^3)$ time using simple data structure, and this time can be reduced to $O(nm + n^2 \log n)$ using Fibonacci heap data structure due to Fredman and Tarjan [1984]. Goldfarb, Hao and Kai [1986] describe another primal simplex algorithm for the shortest path problem whose number of pivots and running times are comparable to that of Akgul's algorithm.

Among the polynomial time dual simplex algorithms are the algorithms by Roohy-Laleh [1980] for the shortest path problem, by Balinski [1985] and Goldfarb [1985] for the assignment problem, and by Orlin [1984] for the minimum cost flow problem.

Recently, Goldfarb and Hao [1988] developed a polynomial time primal simplex algorithm for the maximum flow problem. This algorithm performs O(nm) pivots and can be implemented to run in $O(n^2m)$ time. Tarjan [1988] showed how to use dynamic trees to further improve the running time to O(nm logn)

This paper is based on the results contained in Orlin [1985] where the worst-case behavior of the primal simplex algorithm with Dantzig's pivot rule is analyzed. Our rule is essentially obtained by incorporating a scaling technique in Dantzig's pivot rule. Let L and U denote the nonbasic arcs at their lower and upper bounds respectively, and $\bar{c}_{ij}$ denote the reduced cost of any arc (i, j). Further, let Δ be a parameter whose initial value is C. Then our pivot rule is as follows: Select as an entering arc any $(i, j) \in L$ with $\bar{c}_{ij} \leq -\Delta/2$ or any $(i,j) \in U$ with $\bar{c}_{ij} \geq \Delta/2$; select the leaving arc so that the strong feasibility of the basis is maintained. (We discuss strong feasibility in detail in Section 1.1.) When there is no nonbasic arc satisfying this rule then replace Δ by $\Delta/2$. We call this pivot rule the *scaling pivot rule* and the simplex algorithm using the scaling pivot rule the *scaling network simplex algorithm.*

We show that the scaling network simplex algorithm solves the minimum cost flow problem in O(nm U logC) pivots and can be implemented to run in $O(m^2 U \log C)$ time. Specializing these results for the assignment and shortest path problems, we show that the scaling network simplex algorithm solves both of these problems in $O(n^2 \log C)$ pivots and O(nm logC) time. These results on the number of pivots are comparable to

that of Orlin [1985] for Dantzig's pivot rule, but the running times here are better by a factor of n.

Intuitively, the reason why the scaling network simplex algorithm is good is that the algorithm selects an entering arc with "sufficiently large" violation of the optimality conditions. This causes a "sufficiently large" improvement in the objective function and helps to show that the number of pivots are "sufficiently small." Further, the arc with "sufficiently large" violation can be picked up with little effort.

Among other results we obtain a scaling version of the label correcting algorithm that solves the shortest path problem with arbitrary arc lengths in O(nm logC) time. We also show that the primal simplex algorithm for the minimum cost flow problem with Dantzig's pivot rule can be implemented in $O(m^2 U \log C)$ heap operations where each heap operation takes O(min (log n, loglog(nC)) time.

The results in this paper rely on properties of the network simplex algorithm, strongly feasible bases and the "perturbation technique". Though we have tried to make this paper self-contained, we refer the reader to the papers of Orlin [1985] and Ahuja, Magnanti and Orlin [1988] for a more thorough discussion on these topics.

1. The Minimum Cost Flow Problem

In this section, we describe a network simplex algorithm that solves the minimum cost flow problem in O(nmU logC) pivots and can be implemented in $O(m^2 U \log C)$ time. The analysis of this algorithm, as specialized for the assignment problem and the shortest path problem, is presented in Sections 2 and 3.

1.1 Background

We consider the following node-arc formulation of the minimum cost flow problem.

$$\text{Minimize} \quad \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{1a}$$

subject to

$$\sum_{\{j:\,(i,j)\in A\}} x_{ij} - \sum_{\{j:\,(j,i)\in A\}} x_{ji} = b(i), \text{ for all } i \in N, \tag{1b}$$

$$0 \le x_{ij} \le u_{ij}, \text{ for each } (i,j) \in A. \tag{1c}$$

In this formulation, if $b(i) > 0$ then node i is a *supply node* and if $b(i) < 0$ then node i is a *demand node*. We assume that $\sum_{i \in N} b(i) = 0$. We also assume that the minimum cost flow problem has a feasible solution. The feasibility of the minimum cost flow problem can be ascertained by solving a maximum flow problem.

The simplex algorithm maintains a basic feasible solution at each stage. A basic solution of the minimum cost flow problem is denoted by a triple (B, L, U); B , L and U partition the arc set A. The set B denotes the set of *basic arcs*, i.e., arcs of a spanning tree, and L and U respectively denote the sets of *nonbasic arcs* at their lower and upper bounds. We refer to the triple (B, L, U) as a *basis structure*. A basis structure (B, L, U) is called *feasible* if by setting $x_{ij} = 0$ for each $(i, j) \in L$, and by setting $x_{ij} = u_{ij}$ for each $(i, j) \in U$, the problem has a feasible solution satisfying (1b) and (1c).

A dual solution to the minimum cost flow problem is a vector π of *node potentials* and a vector $\bar{c}$ of *reduced costs* defined as $\bar{c}_{ij} = c_{ij} - \pi(i) + \pi(j)$. Since one of the mass balance constraint in (1b) is redundant, we can set one node potential arbitrarily. We henceforth assume that $\pi(1) = 0$.

A feasible basis structure (B, L, U) is called an *optimum* basis structure if it is possible to obtain a set of node potentials π so that the reduced costs satisfy the following optimality conditions:

$$\bar{c}_{ij} = 0, \text{ for each } (i,j) \in B, \tag{2a}$$

$$\bar{c}_{ij} \ge 0, \text{ for each } (i,j) \in L, \tag{2b}$$

$$\bar{c}_{ij} \le 0, \text{ for each } (i,j) \in U, \tag{2c}$$

Given a basis structure, the node potentials can be uniquely determined by setting $\pi(1) = 0$ and then using the (n-1) equations of (2a). A nonbasic arc (i, j) not satisfying (2b) or (2c), whichever is applicable, violates its optimality condition. In such a case, $|\bar{c}_{ij}|$ is called the *violation* of the arc (i, j).

The network simplex algorithm maintains a feasible basis structure at each iteration and successively modifies the basis structure via pivots until it becomes an optimum basis structure. The special structure of the basis enables the simplex computations to be performed very efficiently. In the following discussion, we give a brief summary of the network simplex algorithm and the data structure required to implement the algorithm.

The basis B of the minimum cost flow problem is a spanning tree. We consider this tree as "hanging" from node 1. The tree arcs either are *upward pointing* (towards node 1) or are *downward pointing* (away from node 1). We associate three indices with each node i in the tree: a *predecessor* index *pred(i)*, a depth index *depth(i)*, and a *thread* index, *thread(i)*. Each node i has a unique path connecting it to node 1. The predecessor index stores the first node in that path (other than node i) and the depth index stores the number of arcs in the path . For node 1, these indices are zero. We say that *pred(i)* is the *predecessor* of node i, and i is the *successor* of node *pred(i)*. The *descendants* of a node i consist of node i itself, its successors, successors of its successors, and so on. The set of descendants of a node i induce a subtree rooted at node i. We denote the nodes of the subtree by the set D(i). Node i is called an *ancestor* of nodes in D(i). The thread indices define a traversal of the tree, a sequence of nodes that walks or threads its way through the nodes of the tree, starting at node 1 and visiting nodes in a "top to bottom" and "left to right" order, and then finally returning to node 1. The thread indices can be formed by performing a depth first search of the tree. The thread indices provide a means for visiting (or finding) all descendants of a node i in O(| D(i) |) time. For a detailed description of the tree indices see Kennington and Helgason [1980]. In the basis there is a unique path connecting any two nodes. We refer to this path as the *basis path*.

The network simplex algorithm works as follows. It starts with an initial basic feasible flow x and the corresponding basis structure (**B**, **L**, **U**). It selects a nonbasic arc (k, *l*) violating its optimality condition. Adding arc (k, *l*) to the basis forms a unique cycle. We refer to this cycle as the *basis cycle*. The algorithm sends a maximum possible amount of flow in the basis cycle without violating any of the lower and upper bound constraints on arcs. An arc which *blocks* further increase of flow in the basis cycle is called a *blocking arc*. The algorithm drops a blocking arc, say (p,q), from **B**. This gives a new basis structure. If (k, *l*) $\neq$ (p, q) then the node potentials also change. Let T_1 and T_2 be the two subtrees formed by deleting arc (p, q) from the previous basis **B** where T_1 contains node 1. In the new basis, potentials of all nodes in T_1 remain unchanged and potentials of all

potentials of all nodes in T_2 change by a constant amount. If (k, *l*) was a nonbasic arc at its lower bound in the previous basis structure, then they increase by an amount $|\bar{c}_{kl}|$, else they decrease by an amount $|\bar{c}_{kl}|$.

Our network simplex algorithm maintains a strongly feasible basis at every iteration. We say that a basis structure (B, L, U) is *strongly feasible* if we can send a positive amount of flow from any node in the spanning tree to node 1 along arcs in the tree without violating any of the flow bounds. An equivalent way of stating this property is that no upward pointing arc of the spanning tree can be at its upper bound and no downward pointing arc can be at its lower bound. In the subsequent discussion we shall be using both the definitions of the strongly feasible basis. An example of a strongly feasible basis is given in Figure 1.

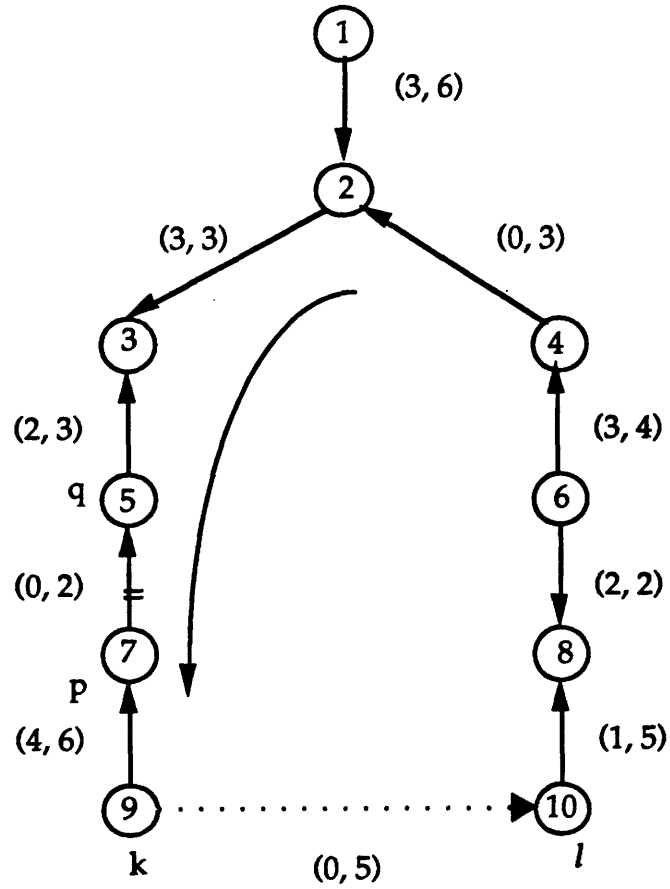


Figure 1. An example of a strongly feasible basis.

The network simplex algorithm selects the leaving arc appropriately so that the next basis is also strongly feasible. Suppose that the entering arc (k, *l*) is at its lower bound and node w is the first common ancestor of nodes k and *l*. Let W be the basis cycle formed by adding arc (k, *l*) to the basis tree. This cycle consists of the basis path from node w to node k, the arc (k, *l*) , and the basis path from node *l* to node w. After updating the flow, the algorithm identifies blocking arcs. If the blocking arc is unique, then it leaves the basis. If there are more than one blocking arcs, then the algorithm selects the leaving arc to be the last blocking arc encountered in traversing W along its orientation starting at node w. For example, in Figure 1, the entering arc is (9,10), the blocking arcs are (2,3) and (7,5), and the leaving arcs is (7,5). It can be shown (see, e.g., Cunningham [1976]) that the above rule guarantees that the next basis is strongly feasible. We use the following property of a strongly feasible basis.

Observation 1. In every degenerate pivot, the leaving arc lies in the basis path from node w to node k.

This observation follows from the facts that the entering arc (k, *l*) has a positive capacity and that a positive flow can be sent from node *l* to node w. Hence no arc on the basis path from node *l* to node w can be blocked in a degenerate pivot.

In case the entering arc (k, *l*) is at its upper bound then the criteria to select the leaving arc is the same except that the orientation of the cycle W is defined opposite to that of arc (k, *l*). The cycle consists of the basis path from node k to node w, the basis path from node w to node *l*, and the arc (k, *l*). In this case, the leaving arc lies in the basis path from node w to node *l* in a degenerate pivot.

1.2 Scaling Pivot Rule

We now describe the network simplex algorithm with the scaling pivot rule. The algorithm performs a number of *scaling phases*. The algorithm maintains a parameter Δ which remains unchanged throughout a scaling phase. Hence a phase of the algorithm is called a *Δ-scaling phase*. Initially, it is sufficient to select any $\Delta \geq C$, but to maintain integrality of Δ in the subsequent phases we let $\Delta = 2^{\lceil \log C \rceil}$. In the Δ-scaling phase, the algorithm selects *any* nonbasic arc with violation at least $\Delta/2$ and enters it into the basis. The leaving arc is selected so that the next basis is strongly feasible. When every nonbasic arc has violation less than $\Delta/2$ then Δ is replaced by $\Delta/2$ and a new scaling phase

phase begins. When $\Delta < 1$, the integrality of costs shows that the algorithm has obtained an optimum flow. Hence, the algorithm performs $1 + \lceil \log C \rceil$ scaling phases.

A formal description of the scaling network simplex algorithm is given below. In the algorithm description, we define a node set S such that any arc with violation at least $\Delta/2$ is incident to some node in S.

```
algorithm SCALING NETWORK SIMPLEX;
begin
    let (B, L, U) be a feasible basis structure,
        π(j) be the corresponding node potentials and
        c̄ij be the corresponding reduced costs;
    Δ: = 2^⌈logC⌉;
    while Δ ≥ 1 do
    begin
        S : = N;
        while S ≠ ø do
        begin
            select a node i in S and delete it from S;
            for each nonbasic arc (k, l) in A(i) with violation at least Δ/2 do
            begin
                add arc (k, l) to the basis creating a cycle W;
                augment maximum possible flow in W and
                    drop the appropriate blocking arc;
                update the node potentials and add to S each
                    node whose potential changes;
            end;
        end;
    end;
end;
```

The following observations can be made about the algorithm:

Observation 2. The algorithm maintains the invariant that the set S contains all nodes i for which some nonbasic arc in A(i) may have violation at least $\Delta/2$.

This property follows inductively from the facts that at the beginning of each scaling phase $S = N$, and whenever the potential of a node changes, it is added to S. Nodes are deleted from S only when each incident arc has a violation less than $\Delta/2$. Hence when S is empty, the Δ-scaling phase ends. This also shows that at the end of the last scaling phase, the algorithm terminates with a minimum cost flow.

Observation 3. The Δ is an upper bound on the maximum arc violation at the beginning of a scaling phase (except the first phase). This property need not be maintained during a phase. At the beginning of the first scaling phase, the maximum arc violation may be as high as $(n-1)\Delta$.

Observation 4. The algorithm scans $\deg(i) = |A(i)|$ arcs whenever node i is added to S. Let $P_\Delta(i)$ denote the number of updates of $\pi(i)$ during the Δ-scaling phase. Since a node is added to S at the beginning of a phase and whenever its potential changes, it follows that the total scanning time during the Δ-scaling phase is $m + \sum_{i \in N} \deg(i) \cdot P_\Delta(i)$.

Also observe that there are two bottleneck operations in the scaling network simplex algorithm: (i) augmenting flows in cycles; and (ii) arc scanning time. The time to update node potentials is dominated by the arc scanning time.

We next analyze the complexity of the scaling network simplex algorithm. The analysis depends on the *perturbation technique* that we discuss next.

1.3 Peturbation Technique

The *perturbation technique* is a well-known method for avoiding cycling in the simplex algorithm for linear programming. The technique slightly perturbs the right-hand-side vector so that every feasible basis is non-degenerate and an optimum solution of the perturbed problem can be easily converted to an optimum solution to the original problem. Cunningham [1976] showed that a specific variant of the perturbation technique for the network simplex method is equivalent to the strongly feasible basis technique. Orlin [1985] extended Cunningham's result to allow for more general perturbations.

The minimum cost flow problem can be perturbed by changing the supply/demand vector b to $b + \varepsilon$. We say that $\varepsilon = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n)$ is a *feasible perturbation* if it satisfies the following conditions:

(i) $\varepsilon_i > 0$ for all $i = 2, \ldots, n$;

(ii) $\sum_{i=2}^{n} \varepsilon_i < 1$; and

(iii) $\varepsilon_1 = -\sum_{i=2}^{n} \varepsilon_i$.

One possible choice for a feasible perturbation is $\varepsilon_i = 1/n$ for $i = 2, \ldots, n$ (and thus $\varepsilon_1 = -(n-1)/n$). Another choice is $\varepsilon_i = \alpha^i$ for $i = 2, \ldots, n$, with α chosen as a very small positive number. The perturbation changes the flow on basic arcs. It can be easily shown that perturbation of b by a feasible perturbation ε changes the flow on basic arcs in the following manner:

Observation 5. If (i, j) is a downward pointing arc of tree **B** and D(j) is the set of descendants of node j, then the perturbation decreases the flow in arc (i, j) by $\sum_{k \in D(j)} \varepsilon_k$.

Since $0 < \sum_{k \in D(j)} \varepsilon_k < 1$, the resulting flow is non-integral and thus non-zero.

Observation 6. If (i, j) is an upward pointing arc of tree **B** and D(i) is the set of descendants of node i, then the perturbation increases the flow in arc (i, j) by $\sum_{k \in D(i)} \varepsilon_k$.

Since $0 < \sum_{k \in D(i)} \varepsilon_k < 1$, the resulting flow is non-integral and thus non-zero.

In the perturbed problem flow on every arc is non-zero, hence every pivot is a non-degenerate pivot and the leaving arc is uniquely identified. The following result establishes a connection between a primal simplex algorithm in which strongly feasible bases are maintained for the original problem, and the ordinary primal simplex algorithm for the perturbed problem.

Lemma 1. *(Cunningham [1976], Orlin [1985]). For any basis structure (B, L, U) of the minimum cost flow problem, the following statements are equivalent:*

(i) *(B, L, U) is strongly feasible.*

(ii) *(B, L, U) is feasible if b is replaced by $b + \varepsilon$ for any feasible perturbation ε .*

Proof. (i) $\Rightarrow$ (ii). Since $(\mathbf{B}, \mathbf{L}, \mathbf{U})$ is strongly feasible, each upward pointing arc has flow less than its upper bound and each downward pointing arc has flow greater than zero. The perturbation increases the flow on upward pointing arcs by at most one unit and decreases the flow on downward pointing arcs by at most one unit. Hence $(\mathbf{B}, \mathbf{L}, \mathbf{U})$ is a feasible basis structure for the perturbed problem.

(ii) $\Rightarrow$ (i). Consider a feasible basis structure $(\mathbf{B}, \mathbf{L}, \mathbf{U})$ for the perturbed problem. Each arc in the basis has a positive non-integral flow. If we consider the same basis tree for the original problem, then flows on upward pointing arcs decrease, flows on downward pointing arcs increase and the resulting flows are integral. Consequently, $x_{ij} < u_{ij}$ for an upward pointing arc (i, j), $x_{ij} > 0$ for a downward pointing arc (i, j), and $\mathbf{B}$ is strongly feasible for the original problem. ■

We shall now analyze the worst-case complexity of the scaling network simplex algorithm. Our arguments rely on the scaling network simplex algorithm as applied to the perturbed problem. However, because of Lemma 1, the perturbation need not actually be done; maintaining strongly feasible bases is sufficient to realize these bounds since the pivot sequences are the same as for the perturbation technique.

1.4 Complexity Analysis

Let z denote the objective function value of the minimum cost flow problem at the beginning of the Δ-scaling phase. Let x denote the corresponding flow and let $(\mathbf{B}, \mathbf{L}, \mathbf{U})$ denote the basis structure. Further, let z^* denote the optimum objective function value. We first show that $z - z^*$ is bounded by $mU\Delta$. Observe that

$$\sum_{(i,j) \in A} \bar{c}_{ij} x_{ij} = \sum_{(i,j) \in A} c_{ij} x_{ij} + \sum_{(i,j) \in A} \pi(i) x_{ij} - \sum_{(i,j) \in A} \pi(j) x_{ij},$$

$$= \sum_{(i,j) \in A} c_{ij} x_{ij} + \sum_{i \in N} \pi(i) \left[\sum_{\{j:\,(i,j) \in A\}} x_{ij} - \sum_{\{j:\,(i,j) \in A\}} x_{ij} \right],$$

$$= \sum_{(i,j) \in A} c_{ij} x_{ij} + \sum_{i \in N} \pi(i)\, b(i).$$

Consequently for a given set of node potentials, the total improvement with respect to the objective function $\sum_{(i,j) \in A} c_{ij} x_{ij}$ is equal to the total improvement with

respect to the objective function $\sum_{(i,j) \in A} \bar{c}_{ij} x_{ij}$. Further, the total improvement in the objective function $\sum_{(i,j) \in A} \bar{c}_{ij} x_{ij}$ is bounded by the total improvement in the following relaxed problem:

$$\text{minimize} \quad \sum_{(i,j) \in A} \bar{c}_{ij} x_{ij} \tag{3a}$$

subject to

$$0 \le x_{ij} \le u_{ij} \text{ , for all } (i,j) \in A. \tag{3b}$$

For any basis structure (B,L,U) of (1) and a corresponding flow, we can obtain an optimum solution of problem (3) by setting $x_{ij} = u_{ij}$ for each arc $(i,j) \in L$ with $\bar{c}_{ij} < 0$, by setting $x_{ij} = 0$ for each arc $(i,j) \in U$ with $\bar{c}_{ij} > 0$, and by leaving the flow on other arcs in B unchanged. At the beginning of the Δ–scaling phase (except when it is the first phase), all arc violations are no more than Δ; hence this readjustment of flows decreases the objective function by at most $mU\Delta$. We have thus shown that at the beginning of a scaling phase

$$z - z^* \le m U \Delta . \tag{4}$$

Since in the primal simplex algorithm, the objective function value is non-increasing, (4) remains true during the Δ-scaling phase as well. In the first scaling phase $\Delta \ge C$ and $m U C$ (or $- m U C$) is an upper bound (or lower bound) on the objective function value of the minimum cost flow problem. We have thus established the following result.

Lemma 2. *If z is the objective function value of the minimum cost flow problem at the beginning of the Δ-scaling phase, then $z - z^* \le 2 m U \Delta$.* ■

We now consider the non-degenerate pivots and the time needed to perform these pivots. During the Δ-scaling phase each non-degenerate pivot augments at least one unit of flow, and since the entering arc has violation at least $\Delta/2$ units, the pivot decreases the objective function value by at least $\Delta/2$ units. It follows from Lemma 2 that there are at most $4mU$ non-degenerate pivots during a scaling phase. Each such pivot takes O(n) time to augment the flow and update node potentials. Hence the total augmentation time in O(nmU) time. Further, since each node potential is updated

O(mU) times, it follows from Observation 4 that the arc scanning time is $O(m^2 U)$ per scaling phase.

We next consider the number of degenerate pivots and the time needed to perform these pivots. We first use arguments based on analyzing the perturbed problem with perturbation $\varepsilon = (-(n-1)/n,\ 1/n,\ 1/n, \ldots, 1/n)$. For this perturbation, the flow on each arc is a positive integral multiple of $1/n$, and hence each pivot carries at least $1/n$ units of flow. Thus during the Δ-scaling phase, each degenerate pivot for the original problem decreases the objective function value of the perturbed problem by at least $\Delta/2n$ units. It follows from Lemma 2 that there are O(nmU) degenerate pivots during a scaling phase.

We now compute the time spent in the degenerate pivots. For this purpose, we use the perturbation defined by $\varepsilon = (-\sum_{i=2}^{n} \deg(i)/2m,\ \deg(2)/2m,\ \deg(3)/2m, \ldots,$ $\deg(n)/2n)$, where $\deg(i) = |A(i)|$. It can be easily verified that this perturbation is a feasible perturbation. We first determine the flow augmented in the perturbed problem corresponding to a degenerate pivot in the original problem. We assume that the entering arc (k, *l*) is at its lower bound. The case when arc (k, *l*) is at its upper bound can be handled similarly.

It follows from Observation 1 that the leaving arc, say (p, q), lies in the basis path from node w (the first common ancestor of nodes k and *l*) to node k. Since this pivot is a degenerate pivot, arc (p, q) is either a downward pointing arc at its upper bound or an upward pointing arc at its lower bound. It follows from Observations 5 and 6 that if (p, q) is a downward pointing arc, then in the perturbed problem the flow in (p, q) decreases by $H(q) = \sum_{k \in D(q)} \deg(k)/2m$, and if (p, q) is an upward pointing arc then the flow in (p, q) in the perturbed problem increases by $H(p) = \sum_{k \in D(p)} \deg(k)/2m$. In either case, the flow around the basis cycle in the perturbed problem is at least H(v), where v is either node p or node q. Since the entering arc has violation at least $\Delta/2$, this pivot decreases the objective function value of the perturbed problem by at least $H(v)\,\Delta/2$ units.

In this pivot operation, potentials of nodes in D(v) are changed and these nodes are added to the set S. From Observation 4, this pivot operation increases the scanning

time by at most 2mH(v) . We have thus shown that the pivot increases the scanning time by at most 2mH(v) while decreasing the objective function by at least H(v) $\Delta/2$. Consequently, the scanning time is at most $(4m/\Delta)$ times the improvement in the objective function. As the total possible improvement in the objective function is at most $2mU\Delta$, the scanning time is $O(m^2 U)$ for degenerate pivots per scaling phase.

Finally, we consider the time needed to identify a degenerate pivot. We incorporate an additional test in the method described in Barr, Glover and Klingman [1979] to identify the basis cycle for an entering arc (k, *l*). We trace the predecessor indices of nodes k and *l* backward until either we locate the first common ancestor of nodes k and *l*, or we identify a blocking arc. In the latter case, the time spent in this method can be charged to updating node potentials in D(v) which is dominated by the arc scanning time. In a degenerate pivot, no flow is augmented so we need not consider the augmentation time.

We have thus shown the following result:

Theorem 1. *The scaling network simplex algorithm solves the minimum cost flow problem in $O(nmU \log C)$ pivots and $O(m^2 U \log C)$ time.* ■

Remark. Although the analysis is carried out primarily on the perturbed problem, the running time applies to the original problem. The reason for this is that we require the time for a degenerate pivot to be proportional to the size of the subtree on which potentials change, and irrespective of how large is the basic cycle. In the perturbed problem, the running time per pivot is at least as large as the size of the basic cycle, and we have not bounded this value over all pivots.

2. The Assignment Problem

In this section, we specialize the scaling network simplex algorithm to the assignment problem. We show that the algorithm performs $O(n^2 \log C)$ pivots and runs in $O(nm \log C)$ time.

The assignment problem is a minimum cost flow problem on a bipartite network $G=(N_1 \cup N_2, A)$ such that $A \subseteq N_1 \times N_2$. This problem can be stated as the following minimum cost flow problem:

$$\text{Minimize} \quad \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{5a}$$

subject to

$$\sum_{\{j:\,(i,j) \in A\}} x_{ij} = 1, \quad \text{for all } i \in N_1, \tag{5b}$$

$$-\sum_{\{i:\,(i,j) \in A\}} x_{ij} = -1, \quad \text{for all } j \in N_2, \tag{5c}$$

$$x_{ij} \geq 0, \quad \text{for all } (i,j) \in A. \tag{5d}$$

A basis structure of the assignment problem is defined by B, the set of basic arcs. A basis of the assignment problem has 2n-1 arcs out of which exactly n arcs have non-zero flow equal to 1. For a discussion of the graph theoretic properties of strongly feasible bases of the assignment problem, see Barr, Glover and Klingman [1977].

We associate with any feasible flow x in (5), a set of indices X defined by $X=\{(i,j) \in A: x_{ij} = 1\}$. Observe that $\sum_{(i,j) \in A} c_{ij} x_{ij} = \sum_{(i,j) \in X} c_{ij} x_{ij}$. We denote by x^* an optimum solution of the assignment problem. The following result enables us to obtain a tighter bound on the number of pivots performed by the scaling network simplex algorithm.

Lemma 3. If x is the assignment solution at the beginning of the Δ–scaling phase, then $cx - cx^* \leq 2n\Delta$.

Proof. Let π be the node potentials corresponding to the assignment x. Since $\bar{c}_{ij} = 0$ for each $(i,j) \in X$, we note that

$$0 = \sum_{(i,j) \in X} \bar{c}_{ij} x_{ij} = cx - \sum_{i \in N_1} \pi(i) + \sum_{j \in N_2} \pi(j). \tag{6}$$

At the beginning of the Δ–scaling phase (except when it is the first phase), $\bar{c}_{ij} \geq -\Delta$ for each arc $(i,j) \in A$. Therefore

$$\sum_{(i,j) \in X^*} \bar{c}_{ij} x^*_{ij} = cx^* - \sum_{i \in N_1} \pi(i) + \sum_{j \in N_2} \pi(j) \geq -n\Delta . \qquad (7)$$

Combining (6) and (7) we get

$$cx - cx^* \leq n\Delta . \qquad (8)$$

For the first scaling phase, $\Delta \geq C$ and the lemma follows from the fact that nC is an upper bound on the cost of any assignment solution and -nC is a lower bound on the cost of any assignment solution. ■

We now count the number of non-degenerate and degenerate pivots in a scaling phase. Each non-degenerate pivot in the Δ-scaling phase carries one unit of flow and decreases the objective function value by at least $\Delta/2$ units. Hence there are at most 4n non-degenerate pivots in a scaling phase. To bound the number of degenerate pivots, we consider the perturbation $\varepsilon = (-(n-1)/n,\ 1/n, \ldots, 1/n)$. Corresponding to each degenerate pivot in the original problem in the Δ–scaling phase, the flow around the cycle in the perturbed problem is at least $1/n$ units and thus the objective function value decreases by at least $\Delta/2n$ units. Hence there are at most $4n^2$ degenerate pivots per scaling phase.

We next consider the time needed per scaling phase. Each non-degenerate pivot takes O(n) time to augment flow and increases the arc scanning time by at most O(m). Thus the non-degenerate pivots take a total of O(mn) time in a scaling phase. To bound the time needed for degenerate pivots, we consider the perturbation $\varepsilon = (-\sum_{i=2}^{n} \deg(i)/2m,\ \deg(2)/2m, \ldots, \deg(n)/2m\ )$. The same analysis, as done for the minimum cost flow problem together with inequality (8), shows that all degenerate pivots also take O(nm) time in a scaling phase. We have thus established the following result.

Theorem 2. *The scaling network simplex algorithm solves the assignment problem in O(nm logC) time.* ■

3. The Shortest Path Problem

In this section we analyze the complexity of the scaling network simplex algorithm for the shortest path problem. We show that the algorithm performs $O(n^2 \log C)$ pivots and takes $O(nm \log C)$ time.

We consider the problem of determining a shortest path from node 1 to all other nodes in the network. This problem can be stated as the following minimum cost flow problem:

$$\text{Minimize} \quad \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{9a}$$

subject to

$$\sum_{\{j:\,(i,j) \in A\}} x_{ij} - \sum_{\{j:\,(j,i) \in A\}} x_{ji} = \begin{cases} (n-1), & \text{if } i = 1 \\ -1, & \text{if } i \neq 1 \end{cases} \quad \text{for all } i \in N, \tag{9b}$$

$$x_{ij} \geq 0, \text{ for all } (i,j) \in A. \tag{9c}$$

A basis structure of the shortest path problem is defined by **B**, the set of basic arcs. The basis **B** is a *directed out-tree rooted at node 1*, i.e., a spanning tree in which every tree arc is downward pointing. As earlier, we store the basis tree using predecessor, depth and thread indices. The directed out-tree property of the basis and the fact that each node (except node 1) has demand equal to 1 imply that every node i has exactly one incoming arc (pred(i),i) and the flow on this arc is $|D(i)|$. (Recall that D(i) is the set of descendants of node i in the basis tree.) Hence every basis of the shortest path problem is non-degenerate.

In the basis **B** there is a unique directed path from node 1 to every other node. Let P(k) denote this path from node 1 to node k. The node potentials corresponding to the basis **B** can be obtained by setting $\pi(1)=0$ and then using the equation $c_{ij} - \pi(i) + \pi(j) = 0$ for each $(i, j) \in \mathbf{B}$. The directed out-tree property of the basis implies that $\pi(k) = -$

$\sum_{(i,j) \in P(k)} c_{ij}$. Hence $\pi(k)$ is the negative of the length of the unique path from node 1 to node k in **B**.

With this background we are ready to analyze the complexity of the scaling network simplex algorithm for the shortest path problem. For the shortest path problem every pivot is a non-degenerate pivot. To bound the number of non-degenerate pivots and the computational time, we need the following result. Here we assume that $\pi^*(k)$ denotes the optimum node potential of node k.

Lemma 4. If $\pi(k)$ represents the potential of node k at the beginning of the Δ-scaling phase, then $\pi^*(k) - \pi(k) \leq 2n\Delta$.

Proof. Let $P^*(k)$ denote the shortest path from node 1 to node k and $\bar{c}_{ij}$ represent the reduced costs with respect to the node potentials π. Then

$$\sum_{(i,j) \in P^*(k)} \bar{c}_{ij} = \sum_{(i,j) \in P^*(k)} c_{ij} - \pi(1) + \pi(k) = \pi(k) - \pi^*(k). \tag{10}$$

At the beginning of the Δ-scaling phase (except when it is the first scaling phase), $\bar{c}_{ij} \geq -\Delta$ for each arc $(i, j) \in P^*(k)$. Since $|P^*(k)| \leq n$, (10) yields

$$\pi(k) - \pi^*(k) \geq -n\Delta, \tag{11}$$

and the lemma follows. In the first scaling phase, $\Delta \geq C$ and the lemma follows from the fact that the largest node potential in the algorithm is at most nC and the smallest node potential is at least –nC. ■

In the Δ-scaling phase, the entering arc (k, *l*) has violation at least $\Delta/2$ and potentials of all nodes in D(*l*) increase by at least $\Delta/2$ units. Lemma 4 implies that the potential of any node can change at most 4n times during a scaling phase. Hence the number of pivots are $O(n^2)$ per scaling phase. Observation 4 implies that the arc scanning time $O(m + \sum_{i=2}^{n} \deg(i)\,(4n)) = O(nm)$ per scaling phase. We summarize our discussion below.

Theorem 3. The scaling network simplex algorithms solves the shortest path problem in $O(n^2 \log C)$ pivots and $O(nm \log C)$ time. ■

4. Scaling Label Correcting Algorithm

The label correcting algorithms for the shortest path problem have been extensively studied. In this section, we incorporate a scaling technique in the generic label correcting algorithm and show that this algorithm runs in O(nm logC) time. In this section, we assume that the set A(i) represents the set of outgoing arcs at node i.

For a set of distance labels d(i), the optimality conditions for the shortest path problem can be stated as follows (see, e.g., Ahuja, Magnanti and Orlin [1988]):

C1. d(i) is the length of some path from source to node i for all $i \in N$; and
C2. $d(j) \leq d(i) + c_{ij}$, for all $(i,j) \in A$.

The label correcting algorithms always maintain condition C1 and iteratively try to satisfy condition C2. The basic approach is to maintain a set S of all nodes i for which the condition C2 may be violated for some arc (i,j) and repeat the iterative step given below. Initially, S={1}.

Label Correcting Step. If $S = \emptyset$ then STOP . Otherwise select a node $i \in S$ and delete it from S. Scan each arc (i,j) in A(i), and if $d(j) > d(i) + c_{ij}$ then update $d(j)=d(i)+ c_{ij}$, and add j to S (if j is not already in S).

The correctness of the label correcting algorithm is easy to establish by performing induction on the set S. The label correcting algorithm in its most generic form is exponential time as shown by Edmonds [1970] and Kershenbaum [1981]. However, algorithms with improved running times can be obtained by specifying some rules for selecting and adding nodes to S. The algorithm which maintains the set S as a queue runs in O(nm) time. Several other polynomial time label correcting algorithms are described in Glover et al. [1985].

We now show that a scaling version of the generic label correcting algorithm runs in O(mn logC) time. We call this algorithm the *scaling label correcting algorithm.* The basic idea is to cause "sufficiently large" decrease during each distance update so that the number of updates are "sufficiently small." The scaling label correcting algorithm repeats the iterative step given below. The algorithm uses a parameter Δ. Initially, $\Delta = 2^{\lceil \log C \rceil}$ and when $\Delta < 1$, the algorithm terminates.

Scaling Label Correcting Step. If S is empty, then replace Δ by $\Delta/2$ and set S=N. Select a node $i \in S$ and delete it from S. Scan each arc (i,j) in A(i) and if $d(j) > d(i)+c_{ij} + \Delta/2$ then update $d(j)= d(i)+ c_{ij}$, and add j to S (if j is not already in S).

The accuracy of the algorithm can be shown using inductive arguments. The set S stores all nodes i for which some arc (i,j) violates $d(j) > d(i) + c_{ij} + \Delta/2$. This property is certainly true at the beginning of each scaling phase. Whenever a distance label d(i) decreases, then by adding i to S we keep this property satisfied. Observe that while scanning arcs incident to node i, we need to scan only outgoing arcs. This follows from the fact that whenever distance label of node i decreases, all arcs (k,i) keep satisfying condition C2. Whenever a node i is deleted from S, then $d(j) \leq d(i) + c_{ij} + \Delta/2$ for all $(i, j) \in A(i)$. In the last scaling phase, $\Delta = 1$ and the set S stores all nodes i for which some arc (i,j) violates $d(j) > d(i) + c_{ij}$. Thus the algorithm terminates with the shortest path distances.

We next come to the complexity of the scaling label correcting algorithm. If we treat $\pi(k) = -d(k)$ for each $k \in N$, then Lemma 4 implies that any distance label can decrease by at most $2n\Delta$ units during a scaling phase. As each update decreases a distance label by at least $\Delta/2$ units, the distance label of any node decreases at most 4n times during a scaling phase. Thus arc scanning time is O(nm) per scaling phase and O(nm logC) overall. We state this result as a theorem.

Theorem 4. *The scaling label correcting algorithm solves a shortest path problem with arbitrary arc lengths in O(nm logC) time.* ■

5. Implementing Dantzig's Pivot Rule.

Dantzig's rule is to select the entering arc whose violation is maximum. Thus it is a special implementation of the scaling pivot rule, and the bounds of Theorems 1, 2 and 3 concerning the number of pivots apply. However, the time to determine an arc with the maximum violation may be as much as O(m) per pivot, which would give a running time of $O(nm^2U \log C)$ for the minimum cost flow problem and $O(n^2m \log C)$ for the assignment and shortest path problems. Indeed, these are the times given by Orlin [1985]. Here we show how to reduce the running time by nearly a factor of n.

To obtain these improvements, we maintain a set Q of arcs with positive violations of the optimality conditions. The set Q is maintained as a heap with violation as the key of each arc. Changing the violation of an arc, adding or deleting an arc from the heap, or finding an arc with maximum violation counts as one heap operation. Each heap operation takes O(logn) time if Q is maintained as a binary heap (see, e.g., Aho, Hopcroft and Ullman [1974], or it takes O(log log (nC)) time if Q is maintained using the data structure of Johnson [1982].

The network simplex algorithm with Dantzig's pivot rule is implemented as follows. Using a heap operation, the algorithm selects an arc with maximum violation. During the pivot operation, whenever the potential of a node is updated then arcs incident on that node are examined. The violations of arcs are updated and the necessary heap operations are performed. The dominant term in the computations is performing heap operations corresponding to this arc scanning. The network simplex algorithm has the same order of arc scanning for Dantzig's pivot rule and the scaling pivot rule. Hence the network simplex algorithm with Dantzig's pivot rule solves the minimum cost flow problem in $O(m^2U \log C)$ heap operations and the assignment and shortest path problems in $O(nm \log C)$ heap operations, where each heap operation takes $O(\min \{\log n, \log \log (nC)\})$ time.

6. Conclusions

We have presented a variant of the network simplex algorithm that is very close in spirit to some of the best implementations in practice but whose worst-case bound is much better. We anticipate that our rule will give running times in practice that are comparable to the best other primal simplex pivot rule; however, our conjecture in this matter awaits thorough computational testing.

The algorithms presented here illustrate the power of scaling approaches. Under the *similarity assumption*, i.e., $C = O(n^k)$ and $U = O(n^k)$ for some k, the scaling based approaches are currently the best available in the worst-case complexity for all of the following problems among others : the shortest path problem with arbitrary arc lengths, the assignment problem, the minimum cycle mean problem, the maximum flow problem, the minimum cost flow problem (see Ahuja, Orlin and Magnanti[1988]). Our results in this paper demonstrate that the ideas of scaling are useful not only in purely combinatorial algorithms, but also in a more general perspective.

The time bounds of our algorithms for the assignment and shortest path problems are both O(nm logC). This time bound is not as attractive as the time bound of $O(\sqrt{n}\ m \log(nC))$ achieved by the algorithms of Gabow and Tarjan [1987] and Orlin and Ahuja [1988]; however, it is nearly as fast as the O(nm) label correcting algorithm for the shortest problem and the O(n(m + n log n)) successive shortest path algorithm for the assignment problem (see Engquist [1982] and Fredman and Tarjan [1984]). In addition, they dominate the running times of a number of algorithms that are used in practice for solving the shortest path and the assignment problems (see for example, Glover et al. [1985], Glover et al. [1986]).

Perhaps the greatest advantage with respect to the assignment and shortest path problems is that they do not require a special purpose algorithm in order to have the advantage of a good worst-case performance. The algorithms suggested here are general purpose network simplex algorithms. The rules are simple to state and simple to implement. In addition, they allow a reasonable amount of flexibility in the selection of the entering variable. In particular, one can search the node list S in several different ways without increasing the computational complexity. This flexibility contrasts with the inflexibility of the existing polynomial time primal simplex algorithm for these problems, which are very restrictive in the selection of the entering variables.

Another possible flexibility in our algorithms is the choice of the *scale factor*, the ratio by which Δ is reduced at the end of a scaling phase. The algorithms as stated use a scale factor of 2. However, the algorithms can use any arbitrary scale factor β. In this case, the network simplex algorithm can be shown to perform $O(\beta\ nm\ U \log_\beta C)$ pivots and require $O(\beta\ m^2\ U \log_\beta C)$ time. For the assignment and shortest path problems, the number of pivots would be $O(\beta\ n^2 \log_\beta C)$ and the time would be $O(\beta\ nm \log_\beta C)$. Thus from the worst-case point of view, any constant value of β is optimal; empirically the best value may be determined through computational testing.

We have also provided a robust rule for the label correcting algorithm for the shortest path problem whose worst-case running time is only a factor logC greater than the best other implementation. Again, the effectiveness of our approach in practice awaits computational testing. Simultaneously, we have improved the worst-case running time of implementing Dantzig's pivot rule by nearly a factor of n. While we anticipate that this result is primarily of theoretical interest, it may be practical if some implementation of the network simplex algorithm requires that the arc with maximum violation is selected.

In passing, we again mention the problem of finding a polynomial time primal network simplex algorithm. The algorithm presented here is pseudo-polynomial time, and is exponential time in the worst-case. The only known polynomial time simplex algorithm is due to Orlin [1984], but this algorithm is a dual simplex algorithm and is not a "natural" implementation.

Acknowledgement

This research was supported in part by the Presidential Young Investigator Grant 8451517-ECS of the National Science Foundation, by Grant AFORS-88-0088 from the Air Force Office of Scientific Research, and by grants from Analog Devices, Apple Computer, Inc., and Prime Computer.

References

Aho, A.V., J.E. Hopcroft, and J.D. Ullman. 1974. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA.

Ahuja, R.K., T. L. Magnanti, and J.B. Orlin. 1988. Network Flows. Working Paper No. 2059-88, Sloan School of Management, M.I.T., Cambridge, MA. (To appear in *Handbooks in Operations Research and Management Science, Vol. 1. Optimization.*)

Akgul, M. 1985a. Shortest Path and Simplex Method. Research Report, Department of Computer Science and Operations Research, North Carolina State University, Raleigh, N.C.

Akgul, M. 1985b. A Genuinely Polynomial Primal Simplex Algorithm for the Assignment Problem. Research Report, Department of Computer Science and Operations Research, North Carolina State University, Raleigh, N.C.

Balinski, M.L. 1985. Signature Methods for the Assignment Problem. *Oper. Res.* 33, 527-536.

Barr, R., F. Glover, and D. Klingman. 1977. The Alternating Path Basis Algorithm for the Assignment Problem. *Math. Prog.* 12, 1-13.

Barr, R., F. Glover, and D. Klingman. 1979. Enhancement of Spanning Tree Labeling Procedures for Network Optimization. *INFOR* 17, 16-34.

Cunningham, W.H. 1976. A Network Simplex Method. *Math. Prog.* 11, 105-116.

Dantzig, G.B. 1983. Personal Communication with J. B. Orlin.

Dial, R., F. Glover, D. Karney, and D. Klingman. 1979. A Computational Analysis of Alternative Algorithms and Labeling Techniques for Finding Shortest Path Trees. *Networks* 9, 215-248.

Edmonds, J. 1970. Exponential Growth of the Simplex Method for the Shortest Path Problem. Unpublished Paper, University of Waterloo, Ontario, Canada.

Engquist, M. 1982. A Successive Shortest Path Algorithm for the Assignment Problem. *INFOR* 20, 370-384.

Fredman, M.L., and R.E. Tarjan. 1984. Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *25th Annual IEEE Symp. on Found. of Comp. Sci.*, 338-346, also in *J. of ACM* 34(1987), 596-615.

Gabow, H.N., and R.E. Tarjan. 1987. Faster Scaling Algorithms for Network Problems. *SIAM J. Comput.* (submitted).

Glover, F., R. Glover, and D. Klingman. 1986. Threshold Assignment Algorithm. *Math. Prog. Study* 26, 12-37.

Glover, F., D. Klingman, N. Phillips, and R.F. Schneider. 1985. New Polynomial Shortest Path Algorithms and Their Computational Attributes. *Man. Sci.* 31, 1106-1128.

Goldfarb, D. 1985. Efficient Dual Simplex Algorithms for the Assignment Problem. *Math. Prog.* 33, 187-203.

Goldfarb, D., and J. Hao. 1988. A Primal Simplex Algorithm that Solves the Maximum Flow Problem in At Most nm Pivots and $O(n^2 m)$ Time. Technical Report, Dept. of Industrial Engineering and Operations Research, Columbia University, New York, N.Y.

Goldfarb, D., J. Hao, and S. Kai. 1986. Efficient Shortest Path Simplex Algorithms. Research Report, Dept. of Industrial Engineering and Operations Research, Columbia University, New York, N.Y.

Hung, M.S. 1983. A Polynomial Simplex Method for the Assignment Problem. *Oper. Res.* 31, 595-600.

Johnson, D. B. 1982. A Priority Queue in Which Initialization and Queue Operations Take O(log log D) Time. *Math. Sys. Theory* 15, 295-309.

Kennington, J.L., and R.V. Helgason. 1980. *Algorithms for Network Programming*, Wiley-Interscience, N.Y.

Kershenbaum, A. 1981. A Note on Finding Shortest Path Trees. *Networks* 11, 399-400.

Orlin, J.B. 1984. Genuinely Polynomial Simplex and Non-Simplex Algorithms for the Minimum Cost Flow Problem. Technical Report No. 1615-84, Sloan School of Management, M.I.T., Cambridge, MA.

Orlin, J.B. 1985. On the Simplex Algorithm for Networks and Generalized Networks. *Math. Prog. Study* 24, 166-178.

Orlin, J.B., and R.K. Ahuja. 1988. New Scaling Algorithms for the Assignment and Minimum Cycle Mean Problems. Working Paper No. OR 178-88, Operations Research Center, M.I.T., Cambridge, MA.

Roohy-Laleh, E. 1980. *Improvements in the Theoretical Efficiency of the Network Simplex Method*. Unpublished Ph.D. Dissertation, Carleton University, Ottawa, Canada.

Tarjan, R. E. 1988. Personal Communication.

Zadeh, N. 1979. Near Equivalence of Network Flow Algorithms. Technical Report No. 26, Dept. of Operations Research, Stanford University, CA.