

## A PRIMAL SIMPLEX ALGORITHM THAT SOLVES THE MAXIMUM FLOW PROBLEM IN AT MOST $nm$ PIVOTS AND $O(n^2m)$ TIME

Donald GOLDFARB

*Department of Industrial Engineering and Operations Research, Columbia University,  
New York, NY 10027, USA*

Jianxiu HAO

*GTE Laboratories, Waltham, MA 02254, USA*

Received 29 June 1988

Revised manuscript received 10 January 1989

We propose a primal network simplex algorithm for solving the maximum flow problem which chooses as the arc to enter the basis one that is *closest* to the source node from amongst all possible candidates. We prove that this algorithm requires at most  $nm$  pivots to solve a problem with  $n$  nodes and  $m$  arcs, and give implementations of it which run in  $O(n^2m)$  time. Our algorithm is, as far as we know, the first strongly polynomial primal simplex algorithm for solving the maximum flow problem.

*Key words:* Simplex method, maximum flow, strongly polynomial, network flow.

### 1. Introduction

The maximum flow problem is an important problem with a wide range of applications. It has been extensively studied and numerous algorithms have been developed to solve it. Many of these are based upon the classical *augmenting path* method of Ford and Fulkerson [7]. Although this method may fail to terminate if some of the data for the problem are irrational [8], if all flow augmentations are made along shortest augmenting paths, then at most  $nm$  augmentations are required to solve a problem with  $n$  nodes and  $m$  arcs.

This result was obtained independently by Edmonds and Karp [6] and Dinic [5], both of whom gave strongly polynomial variants of the augmenting path method. The Edmonds–Karp algorithm computes each shortest path in  $O(m)$  time and, hence, runs in  $O(nm^2)$  time. The Dinic algorithm performs flow augmentations in stages. In each stage a so-called *layered network* of shortest augmenting paths is

constructed. Flow is then pushed along these paths, all of which are of the same length, until no further flow augmentation is possible using such paths. Dinic's algorithm runs in  $O(n^2m)$  time since there are at most  $n - 1$  stages, each requiring  $O(nm)$  time to process.

Currently, the algorithms with the best worst-case time bounds are all based upon a recently proposed augmenting path algorithm due to Goldberg [9]. This algorithm uses *preflow*—a concept introduced by Karzanov [17] which allows the flow into a node to exceed the flow out of that node and pushes *excess* flow along paths that are only *estimated* to be shortest. Variants of this  $O(n^3)$  algorithm which have improved running time bounds have been proposed by Goldberg and Tarjan [11],  $O(nm + n^2 \log(n^2/m))$ , Ahuja and Orlin [1],  $O(nm + n^2 \log U)$ , and Ahuja, Orlin and Tarjan [2],  $O(nm \log((n \log U)/(m \log \log U) + 2))$ , where  $U$  is an upper bound on the integral edge capacities.

It is well-known that the primal simplex algorithm [4] can be used to solve the maximum flow problem. However, until now no one has shown that any variant of this algorithm exhibits a strongly polynomial worst-case time bound for the maximum flow problem. It is the main purpose of this paper to do this. In fact, we propose here a primal simplex algorithm which, like the augmenting path algorithms of Edmonds and Karp and Dinic, solves the maximum flow problem in at most  $nm$  flow augmentations. Actually, our algorithm performs at most  $nm$  simplex pivots, some of which may be degenerate, i.e., zero-flow augmentations. Also, like Dinic we are able to obtain an algorithm which runs in  $O(n^2m)$  time. However, unlike Dinic, we can obtain this bound without resorting to *layered networks*. We also note that if the most negative cost pivot rule is used with the primal simplex algorithm, then the number of pivots may be exponentially large as can be shown by modifying Zadeh's "bad example" [19] in an obvious way to convert it into a maximum flow problem.

Our paper is organized as follows. The maximum flow problem and its linear programming formulation are given in the next section. Notation and the key concept of an *augmenting path* relative to a flow and basis tree, which may *not* be an augmenting path in the usual sense since it can include arcs in the basis tree that are *blocking*, are also introduced there. In Section 3 we consider several properties of pivot rules that are crucial to our development of a strongly polynomial simplex algorithm. In particular, we categorize pivots by characteristics of the arc that leaves the basis, define what we call *admissible* pivot rules, and prove some important properties of such rules. In Section 4, we propose a primal network simplex algorithm, which chooses as the arc to enter the basis one that is *closest* to the source node amongst all nonbasic arcs with positive reduced cost. Here a closest arc is one for which the *length* of a shortest *augmenting path* to it is minimal. Our main result that this algorithm requires at most  $nm$  pivots is proved in Section 4. Implementations of our algorithm that run in  $O(n^2m)$  time are presented in Section 5, and a relaxed version of it that requires at most  $2nm$  pivots is developed in Section 6. Finally, we offer some observations about our algorithm in Section 7.

## 2. Preliminaries

Let  $G = (N, A)$  be a connected finite directed graph with node set  $N$  of size  $n$  and arc set  $A$  of size  $m$  and let each arc  $(u, v) \in A$  have assigned to it a positive real number  $c_{u,v}$ , called the capacity of  $(u, v)$ . For each node  $v$ , let  $F(v)$  denote the set  $\{w \mid (v, w) \in A\}$  and  $B(v)$  denote the set  $\{u \mid (u, v) \in A\}$ . Given two distinguished nodes  $s$  and  $t$  in  $N$  called the source and the sink respectively, an  $(s, t)$ -flow is any real-valued function  $x$  defined on  $A$  such that

$$\begin{aligned} \sum_{v \in F(u)} x_{u,v} &= \sum_{w \in B(u)} x_{w,u} \quad \forall u \neq s, t, \\ 0 \leq x_{u,v} &\leq c_{u,v} \quad \forall (u, v) \in A. \end{aligned} \quad (1)$$

The maximum flow problem is that of finding an  $(s, t)$ -flow  $x^*$  that maximizes

$$\sum_{v \in F(s)} x_{s,v} = \sum_{w \in B(t)} x_{w,t}.$$

Adding an arc from  $t$  to  $s$ , we can formulate the maximum flow problem as the following linear programming problem:

$$\begin{aligned} \text{maximize} \quad & x_{t,s} \\ \text{subject to} \quad & \sum_{v \in F(u)} x_{u,v} = \sum_{w \in B(u)} x_{w,u} \quad \forall u \in N, \\ & 0 \leq x_{u,v} \leq c_{u,v} \quad \forall (u, v) \in A. \end{aligned} \quad (2)$$

In solving this linear program, the simplex method proceeds from basic feasible solution to *adjacent* basic feasible solution while never decreasing the objective function value. In the special case of (2), a basic solution corresponds to an arc-set  $T \subseteq A \cup \{(t, s)\}$ , where  $T$  is a spanning tree of  $(N, A \cup \{(t, s)\})$ . A basic solution  $x$  is uniquely determined by the tree  $T$ , the flow conservation equations (2), and the requirements that  $x_a = 0$  or  $x_a = c_a$  for all arcs  $a \notin T$ . If  $0 \leq x_a \leq c_a$  for all arcs  $a \in T$  then  $x$  is a basic feasible solution.

Let  $x^i$  and  $T_i$  denote the basic feasible solution and basis tree, respectively, just prior to the  $i$ th iteration (i.e., pivot) of the simplex method, and let  $(S_i, Z_i)$  be the partition of  $T_i$  that results from the removal of  $(t, s)$  from  $T_i$ , where  $s \in S_i$ . Clearly, if the *candidate set*

$$C_i = \{(u, v) \in A \mid x_{u,v}^i = 0, u \in S_i, v \in Z_i; \text{ or } x_{u,v}^i = c_{u,v}, u \in Z_i, v \in S_i\}$$

is empty, then the current flow  $x^i$  is a maximum flow and  $(S_i, Z_i)$  is a minimum cut. Otherwise (i.e.,  $C_i \neq \emptyset$ ), the simplex method selects an *in-arc*  $p_i \in C_i$  to add to  $T_i$ , creating a unique cycle. Flow in this cycle in the direction of arc  $(t, s)$  is then increased as much as possible without violating feasibility and an *out-arc*  $q_i$  that limits this increase is deleted from the cycle. This yields a new basis tree,  $T_{i+1} = T_i \cup \{p_i\} \setminus \{q_i\}$ , with flow  $x_{q_i}^{i+1}$  in nonbasic arc  $q_i$  equal either to zero or  $c_{q_i}$ . Notice that pivots with  $p_i = q_i$ , and hence  $T_{i+1} = T_i$ , are possible. In such cases the flow on arc  $p_i$  goes from its lower to its upper bound, or vice versa.

Given a basic feasible flow  $x^i$  and a basis tree  $T_i$ , we define an *augmenting path* relative to  $x^i$  and  $T_i$  as a path  $P_i(u, v)$  from  $u$  to  $v$  in  $G$  such that if  $a$  is in  $P_i(u, v)$  and  $a$  is not in  $T_i$ , then  $x_a^i = 0$  if  $a$  is a forward arc in  $P_i(u, v)$  and  $x_a^i = c_a$  if  $a$  is a backward arc in  $P_i(u, v)$ . Notice that such a path is *not* an augmenting path in the usual sense since it may not be possible to send flow along it because forward (backward) arc  $a$  is allowed to have flow  $x_a^i$  equal to  $c_a(0)$  if  $a$  is in  $T_i$ . We shall use  $D_i(u, v)$  to denote a shortest augmenting path from node  $u$  to node  $v$  and  $d_i(u, v)$  to denote its length, i.e., the number of arcs in it. Since we shall primarily be concerned with shortest paths from the source node  $s$ , we shall also use the notation  $D_i(v)$  for  $D_i(s, v)$  and  $d_i(v)$  for  $d_i(s, v)$ , and shall call  $d_i(v)$  the *label* of node  $v$ . If there are no augmenting paths from  $s$  to  $v$ ,  $d_i(v) = \infty$ . We shall also use  $\delta_{u,v}^i = \min\{d_i(u), d_i(v)\}$ , which we shall call the *label* of arc  $(u, v)$ .

The following property is obvious.

**Property 1.** *If the current basic feasible flow is not optimal (i.e.,  $C_i \neq \emptyset$ ), then*

$$d_i(w) \geq \min\{\delta_e^i \mid e \in C_i\} + 1 \quad \text{for all } w \in Z_i. \quad \square$$

### 3. Simplex pivoting rules for the maximum flow problem

In this section we consider several properties of pivot rules that are crucial to our development of a strongly polynomial simplex algorithm for the maximum flow problem. The following definition categorizes the three possible types of pivots based upon what happens to the flow in the out-arc.

**Definition 1.** Given a basic feasible solution  $x^i$  and basis tree  $T_i$ , the out-arc  $q_i = (u, v)$  on the  $i$ th pivot will be called

- (i) *saturated*, if  $d_i(v) = d_i(u) + 1$  and  $x_{u,v}^{i+1} = c_{u,v}$  or  $d_i(v) = d_i(u) - 1$  and  $x_{u,v}^{i+1} = 0$ ;
- (ii) *nonsaturated*, if  $d_i(v) = d_i(u) + 1$  and  $x_{u,v}^{i+1} = 0$  or  $d_i(v) = d_i(u) - 1$  and  $x_{u,v}^{i+1} = c_{u,v}$ ;
- (iii) *pseudo-saturated*, if  $d_i(v) = d_i(u)$ .

Our first lemma gives criteria for choosing the arc to enter the basis on simplex pivot  $i$  so that the labels  $d_i(w)$  of all nodes  $w \in N$  do not decrease.

**Lemma 1.** *Let  $p_i = (u', v')$  be the in-arc on the  $i$ th simplex pivot. If*

- (i)  $d_i(v') \geq d_i(u') - 1$  and  $x_{u',v'}^i = 0$ , or
- (ii)  $d_i(u') \geq d_i(v') - 1$  and  $x_{u',v'}^i = c_{u',v'}$ ,

then:

(a)  $d_{i+1}(w) \geq d_i(w) \quad \forall w \in N.$

(b) *Furthermore if the out-arc  $q_i = (u, v)$  in pivot  $i$  is either nonsaturated or pseudo-saturated, then  $d_{i+1}(w) = d_i(w) \quad \forall w \in N.$*

**Proof.** We shall consider here only case (i), i.e.,  $d_i(v') \geq d_i(u') - 1$  and  $x_{u',v'}^i = 0$ . Case (ii) can be proved in a similar fashion. Because all arcs in  $T_i$  can be part of an augmenting path regardless of the flow in those arcs, the only way that  $d_{i+1}(w)$  can be less than  $d_i(w)$  is that in-arc  $(u', v')$  is a backward arc in some  $D_{i+1}(w)$ . However, in this case,  $d_{i+1}(w) = d_{i+1}(v') + 1 + d_{i+1}(u', w) \geq d_i(v') + 1 + d_i(u', w) \geq d_i(u') + d_i(u', w) \geq d_i(w)$ . The first inequality follows from the facts that  $d_{i+1}(v') \geq d_i(v')$  and  $d_{i+1}(u', w) \geq d_i(u', w)$ , since no path of the type  $D_{i+1}(s, v')$  or  $D_{i+1}(u', w)$  can contain arc  $(u', v')$  as a backward arc. This proves part (a). To prove part (b), we note that out-arc  $q_i = (u, v)$  is nonsaturated or pseudo-saturated in  $T_i$  if and only if

$$(\alpha) \quad d_i(v) \geq d_i(u) \text{ and } x_{u,v}^{i+1} = 0, \text{ or}$$

$$(\beta) \quad d_i(u) \geq d_i(v) \text{ and } x_{u,v}^{i+1} = c_{u,v}.$$

Let us consider case  $(\alpha)$ ; i.e., arc  $(u, v)$  leaves the basis with  $x_{u,v}^{i+1} = 0$  and  $d_i(v) \geq d_i(u)$ . Case  $(\beta)$  can be proved the same way. The only way that  $d_{i+1}(w)$  can be greater than  $d_i(w)$  is for some  $D_i(w)$  to contain arc  $(u, v)$  as a backward arc. But this is impossible since  $d_i(v) \geq d_i(u)$ . So  $d_{i+1}(w)$  will neither increase after pivoting out arc  $(u, v)$  nor decrease, by part (a) of the lemma; hence  $d_{i+1}(w) = d_i(w)$ .  $\square$

If the in-arc on pivot  $i$  satisfies the conditions of Lemma 1, then if the out-arc  $q_i = (u, v)$  is saturated, it is the last forward (backward) arc on a shortest augmenting path to  $v$  ( $u$ ) immediately before the pivot but not after it. If  $(u, v)$  is nonsaturated, it is the last forward (backward) arc on a shortest augmenting path to  $v$  ( $u$ ) both immediately before and after the pivot. In the pseudo-saturated case, arc  $(u, v)$  is not on any shortest augmenting path to either  $u$  or  $v$  both before and after the pivot.

**Definition 2.** Given a basis  $T_i$  and basic feasible flow  $x^i$ , a network simplex pivot rule is said to be *admissible*, if the arc  $(u, v)$  chosen to enter the basis always satisfies

$$d_i(v) = d_i(u) + 1 \quad \text{if } x_{u,v}^i = 0 \quad (\text{i.e., } (u, v) \text{ is forward}),$$

and

$$d_i(u) = d_i(v) + 1 \quad \text{if } x_{u,v}^i = c_{u,v} \quad (\text{i.e., } (u, v) \text{ is backward}).$$

Notice that in an admissible pivot, the in-arc  $(u, v)$  is the last forward (backward) arc on a shortest augmenting path to  $v$  ( $u$ ). Moreover, since Lemma 1 applies to such pivots, the labels  $d_i(w)$  do not decrease. The next lemma gives conditions under which the labels of certain nodes must strictly increase when an admissible pivot rule is used.

**Lemma 2.** Suppose that an admissible pivot rule is used and that arc  $(u, v)$  is the in-arc in pivots  $i$  and  $k$ , and the out-arc in pivot  $j$ , where  $i \leq j < k$ .

(a) If  $(u, v)$  is saturated on pivot  $j$ , then  $d_k(v) \geq d_j(v) + 2$  if the  $(u, v)$  is forward on pivot  $k$ , and  $d_k(u) \geq d_j(u) + 2$  if it is backward.

(b) If  $(u, v)$  is pseudo-saturated on pivot  $j$ , then  $d_j(u) \geq d_i(u) + 1$  if  $(u, v)$  is forward on pivot  $i$ , and  $d_j(v) \geq d_i(v) + 1$  if it is backward.

(c) If  $(u, v)$  is saturated or pseudo-saturated on pivot  $j$ , then  $d_k(u) + d_k(v) \geq d_i(u) + d_i(v) + 2$ .

**Proof.** (a) Without loss of generality we shall assume that after leaving the basis on pivot  $j$  arc  $(u, v)$  first re-enters the basis on pivot  $k$ ; consequently,  $x_{u,v}^{j+1} = 0$  if  $x_{u,v}^k = 0$  and  $x_{u,v}^{j+1} = c_{u,v}$  if  $x_{u,v}^k = c_{u,v}$ . It then follows from Definitions 1 and 2 and Lemma 1 that  $d_k(v) = d_k(u) + 1 \geq d_j(u) + 1 = d_j(v) + 2$  in the former case (i.e., in-arc  $(u, v)$  is forward on pivot  $k$ ), and  $d_k(u) = d_k(v) + 1 \geq d_j(v) + 1 = d_j(u) + 2$  in the latter case (i.e.,  $(u, v)$  is backward).

(b) Since  $(u, v)$  is pseudo-saturated on pivot  $j$ , it immediately follows from Definitions 1 and 2 and Lemma 1 that  $d_j(u) = d_j(v) \geq d_i(v) = d_i(u) + 1$  if  $(u, v)$  is forward on pivot  $i$  and  $d_j(v) = d_j(u) \geq d_i(u) = d_i(v) + 1$  if it is backward. Note that  $i < j$ , since an arc can be both the in-arc and out-arc on a pivot only if that pivot is a saturated pivot.

(c) First suppose that  $(u, v)$  is saturated on pivot  $j$ . Then from part (a) and Lemma 1,  $d_k(u) + d_k(v) \geq d_j(u) + d_j(v) + 2 \geq d_i(u) + d_i(v) + 2$ , whichever type of pivot  $k$  is. Now suppose that  $(u, v)$  is pseudo-saturated on pivot  $j$ . Then from part (b) and Lemma 1,  $2d_j(v) = 2d_j(u) = d_j(u) + d_j(v) \geq d_i(u) + d_i(v) + 1$ . Clearly, whichever type of pivot  $k$  is,  $d_k(u) + d_k(v) = 2 \min\{d_k(u), d_k(v)\} + 1 \geq 2d_j(u) + 1$ . Hence,  $d_k(u) + d_k(v) \geq d_i(u) + d_i(v) + 2$ .  $\square$

#### 4. A strongly polynomial maximum flow simplex algorithm

We now propose a primal simplex algorithm for the maximum flow problem, which chooses as the arc to enter the basis on a pivot step, an arc in the candidate set  $C_i$  which is *closest* to the source node  $s$  in the sense that its label is smallest.

##### Algorithm 1.

*Step 0.* Find an initial basis tree  $T_1$  (e.g., use breadth first search), set  $x^1 = 0$  and  $i = 1$ .

*Step 1.* If  $C_i = \emptyset$ , stop;  $x^i$  is a maximum flow.

*Step 2.* Otherwise choose an arc  $p = (u, v)$  such that  $\delta_p^i = \min\{\delta_e^i \mid e \in C_i\}$ , and perform a simplex pivot with  $p = (u, v)$  as the in-arc. Set  $i := i + 1$  and go to Step 1.

**Lemma 3.** *The pivot rule for choosing the in-arc  $p = (u, v)$  in Algorithm 1 is admissible.*

**Proof.** From the definition of  $C_i$ ,  $u \in S_i$  and  $v \in Z_i$  if  $x_{u,v}^i = 0$  and  $u \in Z_i$  and  $v \in S_i$  if  $x_{u,v}^i = c_{u,v}$ . Therefore, it follows from Property 1 that  $d_i(v) \geq \min\{\delta_e^i \mid e \in C_i\} + 1 = \delta_p^i + 1 = d_i(u) + 1$  if  $x_{u,v}^i = 0$  and similarly,  $d_i(u) \geq d_i(v) + 1$  if  $x_{u,v}^i = c_{u,v}$ . But clearly in the former case  $d_i(v) \leq d_i(u) + 1$  and in the latter case  $d_i(u) \leq d_i(v) + 1$ ; hence

$$d_i(v) = d_i(u) + 1 \quad \text{if } x_{u,v}^i = 0,$$

and

$$d_i(u) = d_i(v) + 1 \quad \text{if } x_{u,v}^i = c_{u,v},$$

i.e., the pivot rule in Algorithm 1 is admissible.  $\square$

Lemma 3 shows that Lemma 2 applies to Algorithm 1. Hence, as we shall show in the proof of Theorem 1 below, it follows from part (c) of Lemma 2 that there can be a total of at most  $nm$  pivots performed by Algorithm 1 which are either saturated or pseudo-saturated. To show that Algorithm 1 is strongly polynomial we now shall prove that a result analogous to Lemma 2 applies to nonsaturated pivots. Lemma 2 only required that the pivot rule be admissible. In contrast, the lemma below is only true if the pivot rule chooses the in-arc  $(u, v)$  on each pivot to be an arc that is closest to  $s$ —i.e., has minimum label  $\delta_{u,v}$ .

**Lemma 4.** *In Algorithm 1 if arc  $(u, v)$  is the in-arc on pivot  $i$  and it is the out-arc and is nonsaturated on pivot  $k$ , where  $i < k$ , then  $d_k(u) + d_k(v) \geq d_i(u) + d_i(v) + 2$ .*

**Proof.** Let us consider the case where in-arc  $(u, v)$  is forward on pivot  $i$ ; i.e.,  $d_i(v) = d_i(u) + 1$  and  $x_{u,v}^i = 0$ . (The proof for the case where in-arc  $(u, v)$  is backward is similar and is left to the reader.) There are two sub-cases to consider.

*Case 1.*  $(u, v)$  is nonsaturated on pivot  $k$  and  $d_k(v) = d_k(u) - 1$  and  $x_{u,v}^{k+1} = c_{u,v}$ . In this case by Lemma 1 we have that  $d_k(u) = d_k(v) + 1 \geq d_i(v) + 1 = d_i(u) + 2$ .

*Case 2.*  $(u, v)$  is nonsaturated on pivot  $k$  and  $d_k(v) = d_k(u) + 1$  and  $x_{u,v}^{k+1} = 0$ . We now claim that on some pivot  $p$ ,  $i + 1 \leq p \leq k$ , both  $u$  and  $v$  lie in  $Z_p$  and  $d_p(v') \geq d_i(u) + 1$ , where  $v'$  is the end of the in-arc on pivot  $p$  that lies in  $Z_p$ . Since our choice of in-arc ensures that

$$d_p(w) \geq d_p(v') \quad \text{for all } w \in Z_p \tag{3}$$

(this is another way of stating Property 1), it will then follow that  $d_p(u) \geq d_i(u) + 1$ , and hence that  $d_k(u) + d_k(v) = 2d_k(u) + 1 \geq 2d_p(u) + 1 \geq 2d_i(u) + 3 = d_i(u) + d_i(v) + 2$ .

To prove this claim, let us define  $\hat{Z}_j$  to be the set of nodes  $w$  in  $Z_j$  that are connected to  $v$  by a path in  $T_j$  that does not contain either nodes  $u$  or  $t$ , and  $\hat{S}_j$  to be the set of nodes  $w$  in  $S_j$  to which the path in  $T_j$  from  $s$  contains  $(u, v)$  as a forward arc. Clearly  $\hat{Z}_j = \emptyset$  if  $u, v \in S_j$  and  $\hat{S}_j = \emptyset$  if  $u, v \in Z_j$ . Let us define  $W_j = \hat{S}_j \cup \hat{Z}_j$  and let  $v'$  denote the end of the in-arc  $((u', v')$  or  $(v', u'))$  on pivot  $j$  that is in  $Z_j$ . We now show that for  $j = i + 1, \dots$ ,

$$W_j \neq \emptyset \quad \text{and} \quad d_j(w) \geq d_i(u) + 1 \quad \text{for all } w \in W_j \tag{4}$$

until  $v' \in W_j$ , in which case  $j = p$ . Also  $p \leq k$ , since on pivot  $k$  when  $(u, v)$  leaves the basis (as a backward arc) either  $v' \in W_k$  or  $W_k = \emptyset$ , and (4) implies that the latter can only happen if  $p < k$ .

We first note that (4) is true for  $j = i + 1$ . Now suppose that is true for some  $q$ ,  $i + 1 \leq q \leq p$ . If  $W_q = \hat{Z}_q$  and  $v' \in W_q$ , then  $q = p$  and our proof is complete. If  $W_q = \hat{Z}_q$  and  $v' \notin W_q$ , then  $W_{q+1} \subseteq W_q$  and  $W_{q+1} \neq \emptyset$ ; hence (4) holds for  $j = q + 1$  because of Lemma 1. Note that in this case  $W_{q+1} \subset W_q$  if and only if both ends of the out-arc lie in  $W_q$ .

If  $W_q = \hat{S}_q$  and  $u' \notin W_q$ , then  $W_{q+1} = W_q$  since no node in  $W_q$  lies on the *basic augmenting* path determined by the in-arc and  $T_q \setminus \{(t, s)\}$ . If  $W_q = \hat{S}_q$  and  $u' \in W_q$ , then  $W_{q+1} \subset W_q$  if and only if both ends of the out-arc lie in  $W_q$ ; otherwise  $W_{q+1} \supseteq W_q$ . In the former case  $W_{q+1} \neq \emptyset$  and (4) holds for  $j = q + 1$  because of Lemma 1. In the latter case we can have  $W_{q+1} \supset W_q$ . If this occurs,

$$d_q(w) \geq d_q(v') = d_q(u') + 1 \quad \text{for all } w \in W_{q+1} \setminus W_q \tag{5}$$

because of our choice of  $(u', v')$  as the in-arc and (3). Since  $u' \in W_q$ ,

$$d_q(u') \geq d_i(u) + 1. \tag{6}$$

Combining (5) and (6) then implies that

$$d_q(w) \geq d_i(u) + 2 \quad \text{for all } w \in W_{q+1} \setminus W_q. \tag{7}$$

Therefore from Lemma 1, it follows that  $d_{q+1}(w) \geq d_i(u) + 1$  for all  $w \in W_{q+1}$ .  $\square$

**Theorem 1.** (a) *In Algorithm 1, each arc  $(u, v) \in A$  can leave the basis at most  $n$  times.*  
 (b) *The total number of pivots required by Algorithm 1 is at most  $nm$ .*

**Proof.** (a) By Lemma 3, Lemma 2 part (c) and Lemma 4,

$$d_i(u) + d_i(v) \geq d_1(u) + d_1(v) + 2(k - 2) \tag{8}$$

after  $k \geq 2$  pivots involving  $(u, v)$  as the out-arc. Since  $d_i(w) - d_1(w) \leq n - 2$  for all  $w \neq s$  and  $d_i(s) = 0$  for all  $i$ , it follows that  $k \leq n$ . Part (b) follows trivially from part (a).  $\square$

The bounds in Theorem 1 can be tightened slightly by observing that (i)  $2(k - 2)$  in (8) can be replaced by  $2k - 1$  if all pivots in which  $(u, v)$  leaves the basis are either saturated or nonsaturated and by  $2k - 3$  if all such pivots are pseudo-saturated and (ii) there can be at most  $p$  nodes  $w$  that satisfy  $d_i(w) - d_1(w) \geq n - 1 - p$  for some iteration  $i$ . Since these details would only have complicated our proofs without reducing the order of magnitude of the worst-case bounds, we chose to omit them from our analysis.

### 5. $O(n^2m)$ implementations of Algorithm 1

A straight-forward implementation of Algorithm 1 runs in  $O(nm^2)$  time, since computing the node labels  $d_i(w)$  for all  $w \in N$  and computing the in-arc  $p$  or

determining that  $C_i = \emptyset$  on each of the at most  $nm$  pivots takes  $O(m)$  time and this dominates the  $O(n)$  time it takes to perform a pivot operation, i.e., updating a representation of the basis tree  $T_i$  and flow  $x^i$ .

To obtain an  $O(n^2m)$  bound for the overall algorithm we must implement it so as to avoid computing node labels and scanning all arcs in  $A$  on every iteration. With this in mind we note that according to Lemma 1, node labels can only change on the  $i$ th pivot if that pivot is saturated. We also note that when such a change occurs it can be detected by scanning the labels of the nodes in the forward and backward adjacency lists of  $v'$ , where  $v'$  is that end (i.e., head or tail) of the arc  $q$  that leaves the basis that is in  $Z_{i+1}$ , to see if there are alternative shortest augmenting paths to  $v'$  of length  $d_i(v')$  that do not include arc  $q$ . Moreover, if for each node  $w \in N$  we know the arc that precedes  $w$  on some shortest augmenting path  $D_i(w)$  from  $s$  to  $w$ , which we denote by  $f(w)$ , we can determine the in-arc  $p$  in  $O(n)$  time by scanning the set  $Z_i$  for the node  $v$  with the smallest label  $d(v) \equiv d_i(v)$ . With these refinements Algorithm 1 can be implemented as:

**Algorithm 2.**

*Step 0.* Find an initial basis tree  $T_i$  and set  $x^i = 0$  and  $i = 1$ .

*Step 1.* Use breadth-first search to compute  $d(w)$  and  $f(w)$  for all  $x \in N$ . If  $d(t) = \infty$ , stop;  $x^i$  is maximum flow.

*Step 2.* Otherwise, choose a node  $v \in Z$  such that  $d(v) = \min\{d(w) \mid w \in Z_i\}$ , set  $p = f(v)$ , and perform a simplex pivot with  $p$  as the in-arc. Set  $i := i + 1$ .

*Step 3.* If the pivot was not a saturated pivot, go to Step 2. Otherwise, let  $q$  be the out-arc and  $v'$  be the node in  $Z_{i+1}$  incident to  $q$ .

If  $q \neq f(v')$  go to Step 2.

Otherwise, if  $W \cup \bar{W} \neq \emptyset$ , where

$$W = \{u \in F(v') \mid x_{v',u}^{i+1} = c_{v',u} \text{ and } d(u) < d(v')\}$$

and

$$\bar{W} = \{u \in B(v') \mid x_{u,v'}^{i+1} = 0 \text{ and } d(u) < d(v')\},$$

choose some node  $u' \in W \cup \bar{W}$ , set

$$f(v') = \begin{cases} (v', u'), & \text{if } u' \in W, \\ (u', v'), & \text{if } u' \in \bar{W}, \end{cases}$$

and go to Step 2.

Otherwise, go to Step 1.

**Theorem 2.** *The computational complexity of Algorithm 2 is  $O(n^2m)$ .*

**Proof.** Each execution of Step 1 takes  $O(m)$  time, while each execution of Steps 2 and 3 takes  $O(n)$  time. Since Algorithm 2 is just a specific implementation of Algorithm 1 it follows from Theorem 1 that Steps 2 and 3 can occur at most  $nm$

times (once for each pivot). Hence the total time required by Steps 2 and 3 is  $O(n^2m)$ . Since node labels can increase only on saturated pivots and Step 1 is executed only when  $d(v)$  increases and

$$\max \left\{ \sum_{w \in N} d(w) \right\} - \min \left\{ \sum_{w \in N} d(w) \right\} = \frac{1}{2}n(n-1) - (n-1) = \frac{1}{2}(n-1)(n-2), \tag{9}$$

Algorithm 2 can perform Step 1 at most  $\frac{1}{2}(n-1)(n-2) + 1$  times. Hence the total time required by Step 1 and by the algorithm as well is  $O(n^2m)$ .  $\square$

The scanning of the forward and backward adjacency lists in Algorithm 2 can be avoided by constructing a *layered network* of all shortest augmenting paths  $D_i(w)$  to all nodes  $w$  rather than simply a tree of such paths. Specifically, if in Step 1 we compute for each  $w \in N$  a list  $L(w)$  of the arcs that precede  $w$  on all shortest augmenting paths  $D_i(w)$ , rather than just one of these arcs  $f(w)$ , and we replace set  $p = f(v)$  in Step 2 by *choose any arc  $p \in L(v)$* , then Step 3 can be replaced by:

*Step 3'*. If the pivot was not saturated, go to Step 2. Otherwise, set  $L(v) := L(v) \setminus \{p\}$ .

If  $L(v) = \emptyset$ , go to Step 1; otherwise go to Step 2.

Clearly this variant of Algorithm 2 also has a running time of  $O(n^2m)$ .

### 6. Relaxed versions of Algorithms 1 and 2

The worst-case bounds for Algorithm 1 proved in Theorem 1 are a consequence of Lemmas 2 and 4, which guarantee an increase in the node labels of the nodes adjacent to an arc after certain pivots occur. Lemma 2 does not require that the in-arc on pivot  $i$  be an arc in  $C_i$  that is closest to the source node  $s$ . It only requires that the pivot rule used be admissible. We now show that we can take advantage of this flexibility and relax somewhat the conditions that must be satisfied by the in-arc and still obtain an algorithm that terminates in at most  $O(nm)$  pivots and  $O(n^2m)$  time.

With this in mind, let us consider a *relaxed* variant of Algorithm 1, which we shall refer to as Algorithm 3, obtained by replacing the rule for choosing the in-arc  $p$  in Step 2 in Algorithm 1 by *choose an arc  $p \in C_i$  such that  $p$  is admissible and for  $i \geq 2$ ,  $\delta_p^i \leq d_i(v') - 1$ , where  $v'$  is the node in  $Z_i$  adjacent to the out-arc of the previous iteration*. It is easy to show that the in-arc chosen by Algorithm 1 is an acceptable choice for the in-arc in Algorithm 3.

Now let us define  $Q_i = \sum_{e \in R_i} \delta_e^i$ , where  $R_i \equiv T_i \cup \{p_i\}$ . Observe that

$$Q_{i+1} - Q_i = \delta_{p_{i+1}}^{i+1} - \delta_{q_i}^i + \sum_{e \in T_{i+1}} (\delta_e^{i+1} - \delta_e^i), \tag{10}$$

since  $R_{i+1} = T_{i+1} \cup \{p_{i+1}\}$  and  $R_i = T_{i+1} \cup \{q_i\}$ .

By Lemma 1, part (b), node labels can only change on a saturated pivot. If we let  $h_i = d_{i+1}(v') - d_i(v')$ , where  $v'$  is the node in  $Z_{i+1}$  that is adjacent to the out-arc  $q_i$  in pivot  $i$ , we have by our choice of pivot rule that

$$\delta_{p_{i+1}}^{i+1} \leq d_{i+1}(v') - 1 = d_i(v') + h_i - 1. \tag{11}$$

Since  $\delta_{q_i}^i = d_i(v')$  and  $h_i = 0$  if the  $i$ th pivot is either nonsaturated or pseudo-saturated, an immediate consequence of (10) and (11) is the following:

**Lemma 5.** *In Algorithm 3, if the  $i$ th pivot is nonsaturated or pseudo-saturated, then  $Q_{i+1} \leq Q_i - 1$ .  $\square$*

On a saturated pivot  $\delta_{q_i}^i = d_i(v') - 1$  and  $h_i \geq 0$ ; hence from (11) it follows that

$$\delta_{p_{i+1}}^{i+1} - \delta_{q_i}^i \leq h_i \tag{12}$$

on such a pivot. If we let  $S$  denote the set of saturated pivots and  $f$  denote the last pivot, then from (10) and (12) and Lemma 5 it follows that the total number of nonsaturated and pseudo-saturated pivots performed by Algorithm 3 is bounded above by

$$\begin{aligned} \Delta Q &= Q_1 - Q_f + \sum_{i \in S} (Q_{i+1} - Q_i) \leq Q_1 - Q_f + \sum_{i=1}^{f-1} h_i + \sum_{i=1}^{f-1} \sum_{e \in T_{i+1}} (\delta_e^{i+1} - \delta_e^i) \\ &\leq Q_1 - Q_f + \sum_{v \in N} d_f(v) - \sum_{v \in N} d_1(v) + m(n-1) - \frac{1}{2}n(n-1), \end{aligned} \tag{13}$$

since  $\sum_{i=1}^{f-1} h_i \leq \sum_{v \in N} (d_f(v) - d_1(v))$ ,  $\delta_e^f - \delta_e^1 \leq n-1$  for all arcs  $e \in A$ , and for  $k = 0, 1, \dots, n-1$  there is at least one arc  $e$  such that  $\delta_e^f - \delta_e^1 \leq k$ . Given a basis tree  $T_i$ , there is a unique arc  $e$  that precedes each node  $v \in N \setminus \{s\}$  on the path from  $s$  to  $v$  in  $T_i$  and  $d_i(v) - 1 \leq \delta_e^i \leq d_i(v)$ . Since  $d_i(s) = 0$ , it follows that

$$\sum_{e \in T_i} \delta_e^i \leq \sum_{v \in N} d_i(v) \leq \sum_{e \in T_i} \delta_e^i + n - 1.$$

Using this and the fact that  $0 \leq \delta_{p_i}^i \leq n-2$ , implies that

$$Q_1 - \sum_{v \in N} d_1(v) \leq Q_1 - \sum_{e \in T_1} \delta_e^1 = \delta_{p_1}^1 \leq n-2 \tag{14}$$

and

$$Q_f - \sum_{v \in N} d_f(v) \geq Q_f - \sum_{e \in T_f} d_e^f - (n-1) = \delta_{p_f}^f - (n-1) \geq -(n-1). \tag{15}$$

Combining (13)-(15), we obtain  $\Delta Q \leq 2n - 3 + m(n-1) - \frac{1}{2}n(n-1) \leq mn$  for  $n \geq 2$ . The following theorem then follows from this bound and Lemma 2.

**Theorem 3.** *The total number of pivots required by Algorithm 3 is at most  $2nm$ .  $\square$*

Thus the bound on the number of pivots required by this relaxed version of Algorithm 1 is just double the bound for the original algorithm. Clearly, the relaxed

algorithm can be implemented using the refinements introduced in the previous section to give an algorithm with a running time which is also  $O(n^2m)$ .

## 7. Some observations

The analysis in the previous sections rests upon our definition of *shortest augmenting paths* from the source node  $s$  to all other nodes  $w$  in  $N$ . Such paths are *not* augmenting in the usual sense. However, we could have developed a strongly polynomial network simplex algorithm based upon truly augmenting paths by considering a perturbed version of the linear programming problem (2). Specifically, let us define the linear program (2') as problem (2) with a positive constant  $\varepsilon$  added to the left-hand side of the flow conservation constraints for all nodes  $u \in N$ ,  $u \neq s$  and  $(n-1)\varepsilon$  added to the right-hand side of the flow conservation constraint for the source node  $s$ . It can be shown that (2') is nondegenerate for  $\varepsilon$  chosen sufficiently small [3]. Moreover, given a basis tree  $T$  and basic solution  $x$  for (2), and the corresponding basic solution  $x'$  for (2'), i.e.,  $x'_a = x_a$  for all  $a \notin T$ , it can be shown that  $x'$  is feasible to (2') if and only if  $x$  is *strongly feasible* to (2) [3]. By modifying Algorithms 1, 2 and 3 so that they start with an initial tree  $T_1$  that has all arcs directed away from node  $s$ , and providing rules for choosing the leaving arc when there is more than one candidate, strong feasibility of the basis tree can be maintained. Consequently, *augmenting paths* defined in terms of  $x$  and  $T$  in these modified algorithms correspond to augmenting paths in the usual sense relative to  $x'$ .

The steepest-edge simplex algorithm [16], when applied to the maximum flow problem [13, 15], chooses as the in-arc  $p_i$  on iteration  $i$ , an arc in  $C_i$  that yields an  $s$ - $t$  path that is shortest with respect to  $T_i$ , i.e., the cycle determined by  $p_i$  and  $T_i$  is shortest. Because of the close connection to the Edmonds–Karp algorithm, it is natural to ask, as was done in [14], whether the steepest-edge algorithm can solve the maximum flow problem in strongly polynomial time. The results reported here came out of an attempt to answer this question. The question, however, remains unanswered.

Another interesting open problem is whether there is a primal simplex algorithm for the general minimum cost network flow (or circulation) problem that is strongly polynomial. Recently, Goldberg and Tarjan [12] devised a *minimum-mean cycle canceling* algorithm for solving this problem that is strongly polynomial. As this algorithm when applied to (2) corresponds exactly to the Edmonds–Karp algorithm, it is plausible that some generalization of Algorithm 1 may provide a solution to this open problem.

Finally, we note that very recently Goldberg, Grigoriadis and Tarjan [10] have shown how to implement Algorithm 1 to run in  $O(nm \log n)$  time by using an extension of the *dynamic tree* data structure of Sleator and Tarjan [18]. As these authors point out, this bound is within a logarithmic factor of the worst-case bound for any other known maximum flow algorithm.

## References

- [1] R.K. Ahuja and J.B. Orlin, "A fast and simple algorithm for the maximum flow problem," Working Paper, Sloan School of Management, M.I.T. (Cambridge, MA, 1988).
- [2] R.K. Ahuja, J.B. Orlin and R.E. Tarjan, "Improved time bounds for the maximum flow problem," *SIAM Journal on Computing* (1989), to appear.
- [3] W.H. Cunningham, "A network simplex method," *Mathematical Programming* 11 (1976) 105–116.
- [4] G.B. Dantzig, "Maximization of a linear function of variables subject to linear inequalities," in: T.C. Koopmans, ed., *Activity Analysis of Production and Allocation* (Wiley, New York, 1951) pp. 339–347.
- [5] E.A. Dinic, "Algorithm for solution of a problem of maximum flow in networks with power estimation," *Soviet Mathematics Doklady* 11 (1970) 1277–1280.
- [6] J. Edmonds and R.M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the Association of Computing Machinery* 19 (1972) 248–264.
- [7] L.R. Ford Jr. and D.R. Fulkerson, "A simple algorithm for finding maximal network flows and an application to the Hitchcock problem," *Canadian Journal of Mathematics* 9 (1957) 210–218.
- [8] L.R. Ford Jr. and D.R. Fulkerson, *Flows in Networks* (Princeton University Press, Princeton, NJ, 1962).
- [9] A.V. Goldberg, "A new max-flow algorithm," Technical report MIT/LCS/TM-291, Laboratory for Computer Science, M.I.T. (Cambridge, MA, 1985).
- [10] A.V. Goldberg, M.D. Grigoriadis and R.E. Tarjan, "Efficiency of the network simplex algorithm for the maximum flow problem," Technical Report LCSR-TR-117, Laboratory for Computer Science Research, Rutgers University (New Brunswick, NJ, 1988).
- [11] A.V. Goldberg and R.E. Tarjan, "A new approach to the maximum flow problem," *18th Annual ACM Symposium on Theory of Computing* (1986) 136–146.
- [12] A.V. Goldberg and R.E. Tarjan, "Finding minimum-cost circulations by canceling negative cycles," Technical Report MIT/LCS/TM-334, Laboratory for Computer Science, M.I.T. (Cambridge, MA, 1987).
- [13] D. Goldfarb, "Steepest-edge simplex algorithms for network flow problems," Technical Report RC 6649, T.J. Watson Research Laboratory, IBM Corporation (Yorktown Heights, NY, 1977).
- [14] D. Goldfarb and M.D. Grigoriadis, "An efficient steepest-edge algorithm for the maximum flow problem," presented at *Xth International Symposium on Mathematical Programming* (Montreal, Que., 1979).
- [15] D. Goldfarb and M.D. Grigoriadis, "A computational comparison of the Dinic and network simplex methods for maximum flow," in: B. Simeone, P. Toth, G. Gallo, F. Maffioli and S. Pallottino, eds., *Fortran Codes for Network Optimization, Annals of Operations Research* 13 (Baltzer, Basel, 1988).
- [16] D. Goldfarb and J.K. Reid, "A practicable steepest-edge simplex algorithm," *Mathematical Programming* 12 (1977) 361–371.
- [17] A.V. Karzanov, "Determining the maximal flow in a network by the method of pre-flows," *Soviet Mathematics Doklady* 15 (1974) 434–437.
- [18] D.D. Sleator and R.E. Tarjan, "A data structure for dynamic trees," *Journal of Computer and System Sciences* 26 (1983) 362–391.
- [19] N. Zadeh, "A bad network problem for the simplex method and other minimum cost flow algorithms," *Mathematical Programming* 5 (1973) 255–266.