Mathematical Programming 78 (1997) 109–129

A polynomial time primal network simplex algorithm for minimum cost flows

James B. Orlin

Sloan School of Management, MIT, Cambridge, MA 02139, USA

Received 11 July 1995; revised manuscript received 11 May 1996

Abstract

Developing a polynomial time primal network simplex algorithm for the minimum cost flow problem has been a long standing open problem. In this paper, we develop one such algorithm that runs in $O(\min(n^2 m \log nC, n^2 m^2 \log n))$ time, where n is the number of nodes in the network, m is the number of arcs, and C denotes the maximum absolute arc costs if arc costs are integer and ∞ otherwise. We first introduce a pseudopolynomial variant of the network simplex algorithm called the "premultiplier algorithm". We then develop a cost-scaling version of the premultiplier algorithm that solves the minimum cost flow problem in $O(\min(nm \log nC, nm^2 \log n))$ pivots. With certain simple data structures, the average time per pivot can be shown to be $O(n)$. We also show that the diameter of the network polytope is $O(nm \log n)$. © 1997 The Mathematical Programming Society, Inc. Published by Elsevier Science B.V.

Keywords: Minimum cost flows; Network simplex; Polynomial time; Simplex algorithm; Premultipliers

1. Introduction

A fundamental problem within the area of network optimization is the minimum cost flow problem. The problem has applications to a remarkably wide range of fields, including chemistry and physics, computer networking, most branches of engineering, manufacturing, public policy and social systems, scheduling and routing, telecommunications, and transportation (see, for example, [1]). There are a number of different polynomial-time algorithms for the minimum cost flow problem. Currently, the fastest algorithms for the minimum cost flow problem are due to Ahuja et al. [2], Goldberg and Tarjan [3], and Orlin [4]. However, the algorithm of choice in practice is the network simplex algorithm due to its simplicity and speed. Although the network simplex algorithm seems to be excellent in practice, several natural pivot rules take an exponential number of pivots in the worst case, as proved by Zadeh [5]. To this date, there are no

0025-5610/97/$17.00
Published by Elsevier Science B.V.
PII S0025-5610(97)00011-7

specializations of the primal network simplex algorithm that run in polynomial time for the minimum cost flow problem. For graphs with n nodes and m arcs, the current best bound on the number of pivots for the primal network simplex algorithm is $O(n^{\log n/2+O(1)})$, due to Tarjan [6]. In this paper, we present the first polynomial time primal network simplex algorithm for the minimum cost flow problem. The number of pivots performed by this algorithm is $O(nm \log nC)$ or $O(nm^2 \log n)$, whichever is smaller, where C is the largest integral cost coefficient. This resolves a long-standing open research question.

We note that there are other closely related algorithms. There are polynomial time primal network simplex algorithms for (i) the assignment problem (see, for example, [7–10,11]); (ii) the shortest path problem (see, for example, [7,12–14,10,11]); and (iii) the maximum flow problem (see, for example, [15–17])). There are also polynomial time dual network simplex algorithms for the minimum cost flow problem (see, for example, [18–21]). Finally, [22] and [6] established that there are primal network simplex algorithms that have a polynomial number of pivots if one permits pivots that increase the objective function value. The number of pivots for Tarjan's rule is $O(nm\min((\log nC), m \log n))$, which is the same as the number of pivots of the rule given in this paper. These algorithms are not primal network simplex rules in the usual sense that the objective function value is monotonically non-increasing; however, they are useful from a theoretical perspective.

We present a cost-scaling variant of a novel network simplex pivot rule, called the *premultiplier algorithm*. Normally for the network simplex algorithm, one maintains a vector of simplex multipliers so that the reduced cost is 0 for each arc (i,j) of the spanning tree. Here we relax this condition and maintain a vector of "premultipliers" so that the reduced cost is non-positive for each arc that is directed towards the root in the spanning tree and the reduced cost is non-negative for each arc directed away from the root. An unusual feature of this implementation is that the root of the spanning tree is permitted to change at each pivot; indeed, it is often required to change. This feature is also shared by one of Tarjan's algorithms [6].

The amortized running time per pivot of the algorithm presented here is $O(n)$. [23] develops a variant of dynamic trees that reduces the amortized running time per pivot to $O(\log n)$ over a sequence of m pivots. This reduces the running time of the algorithm to $O((nm \log n)(\min(\log nC, m \log n))$. This running time is slower than running time of the Goldberg–Tarjan [3] cost scaling algorithm by a factor of $(\log n)/(\log n^2/m)$. Thus, for sparse graphs, the two running times are comparable.

1.1. Notations and definitions

Let $G = (N, A)$ be a directed network with n nodes and m arcs. Each arc $(i,j) \in A$ has a *cost* c_{ij}, a *capacity* u_{ij}, and a lower bound l_{ij} on the flow in the arc. If costs are integral, we let C denote $\max(|c_{ij}|: (i,j) \in A)$. If costs are not integral, then $C = \infty$. We associate with each node $i \in N$ a number $b(i)$ which indicates its supply or demand depending upon whether $b(i) > 0$ or $b(i) < 0$, respectively.

The minimum cost flow problem can be stated as follows:

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

subject to

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

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

For convenience here, we modify the network flow problem by adding artificial arcs $(1,j)$ and $(j,1)$ for each node $j \neq 1$. The capacity of each of these artificial arcs is ∞, the lower bound on flow is 0, and the cost is $1 + \max(n|c_{ij}|\colon (i,j) \in A)$. These arcs may be used in the initial basic feasible solution, but no artificial arc will have a strictly positive flow in an optimal solution unless the original minimum cost flow problem is infeasible.

For each feasible flow x, we associate residual capacities of the arcs as follows: If $(i,j) \in A$, then the *residual capacity* of (i,j) is $r_{ij} = u_{ij} - x_{ij}$, and the cost of (i,j) is c_{ij}. If $(j,i) \in A$, then the residual capacity of (i,j) is $r_{ij} = x_{ji} - l_{ji}$ and the cost of (i,j) is $c_{ij} = -c_{ji}$. The residual network, denoted by $G(x)$, consists of all arcs whose residual capacity is strictly greater than 0. Thus, for any arc $(i,j) \in A$, the residual network may contain arc (i,j) or (j,i) or both.

We will use the notation "(i,j)" as though there is at most one arc directed from i to j in the residual network. The algorithm readily accommodates multiple arcs from i to j, but representing them as different arcs can be cumbersome. In general, this should cause no confusion in describing and analyzing the algorithm. (The reader may prefer to assume that there is at most one arc from i to j in the residual network.) In this paper, we define walks, directed walks, paths, directed paths, cycles, directed cycles, and cuts as in [1].

A vector of *node potentials* for the network G is a vector π with n components. For each vector π of node potentials, the *reduced costs* c^π are defined as follows: $c^\pi_{ij} = c_{ij} - \pi_i + \pi_j$. It is well known that minimizing the objective function $c^\pi x$ for a network flow problem is equivalent to minimizing the objective function cx. Let W be any directed cycle, and let $c(W)$ denote the sum of the costs of arcs of W. It is also well known that $c(W) = c^\pi(W)$. A flow x is said to be *ε-optimal* with respect to the node potentials π if $c^\pi_{ij} \ge -\varepsilon$ for all arcs (i,j) in the residual network $G(x)$. A flow x that is 0-optimal is also optimal.

1.2. Overview of the paper

In Section 2, we describe the network simplex algorithm in a general form. We describe the pseudopolynomial version of the premultiplier algorithm in Section 3 and show that this algorithm is a special case of the network simplex algorithm. In Section 4 of this paper, we give the details of the cost scaling variant of the premultiplier algorithm and show that the number of pivots per scaling phase is $O(nm)$. We also show

that the number of scaling phases is O(min(log nC,m log n)), and the amortized time per pivot is O(n). In Section 5, we show that the diameter of the network polytope is O(nm log n).

The primary focus of this paper is on the worst-case analysis and on developing polynomial time primal network simplex pivot rules. Although we conjecture that the premultiplier algorithms introduced here can be implemented in a manner that is efficient in practice, we will not investigate practical implementations in this paper.

2. The network simplex algorithm

The network simplex algorithm maintains a basic feasible solution at each stage. A basic solution of the minimum cost flow problem is denoted by a triple (T,L,U); T, L, and U partition the arc set A. The set T denotes the set of basic arcs, that is, arcs of the spanning tree. L and U respectively denote the sets of non-basic arcs at their lower and upper bounds. We refer to the triple (T,L,U) as a *basis structure*. The flow x associated with the basis structure is the flow obtained as follows:

for each arc $(i,j) \in U$, set $x_{ij} = u_{ij}$;
for each arc $(i,j) \in L$, set $x_{ij} = l_{ij}$;
obtain x_{ij} for each arc $(i,j) \in T$ so that constraints in (1b) are satisfied.

We say that the basis structure (T,L,U) is *feasible* if $l_{ij} \le x_{ij} \le u_{ij}$ for each arc $(i,j) \in T$.

Non-degeneracy assumption. We will assume that every basic feasible solution is non-degenerate; that is, no tree arc is either at its lower bound or at its upper bound.

The non-degeneracy assumption may sound like a severe restriction, but it is easily satisfied without loss of generality using a perturbation technique (see, for example, [10] and [24]). One can increase $b(1)$ by ε for some strictly positive but small value of ε, and decrease $b(i)$ for every other node i by $\varepsilon/(n-1)$. Under the assumption that ε is sufficiently small, it can be shown that any optimal basic feasible solution for the perturbed problem is also optimal for the original problem, and that every basic feasible flow is non-degenerate. Under the assumption that all supplies, demands and capacities are integral, one can choose $\varepsilon = (n-1)/n$. In the case that the data are non-integral, one can choose Cunningham's [28] combinatorial rule to carry out the pivots. This rule maintains a class of basic feasible solutions, called *strongly* basic feasible solutions. Cunningham developed this combinatorial rule and showed that it is equivalent to the perturbation technique described above.

The non-degeneracy assumption implies that if x is the solution associated with the basis structure (T,L,U), then $G(x)$ contains all arcs of T as well as reversals of all arcs of T; that is, if $(i,j) \in T$, then both (i,j) and (j,i) will be in $G(x)$. Our algorithms use two additional concepts.

Definition 1. We denote by $G^*(x)$ the subgraph of $G(x)$ in which all arcs of T and their reversals have been deleted. (By the non-degeneracy assumption, $G^*(x)$ consists of those arcs $(i,j) \in G(x)$ such that $(j,i) \notin G(x)$. Therefore, $G^*(x)$ is fully determined by the flow x.)

Definition 2. For any tree T and a root node v, we denote by $T(v)$ a subgraph of $G(x)$ which is a directed spanning in-tree T and in which all arcs are directed towards node v. Costs and capacities of arcs in $T(v)$ are defined as in $G(x)$.

We illustrate $T(v)$ in Fig. 1. Fig. 1(a) is a tree rooted at node 1 whereas Fig. 1(b) is the same tree rerooted at node 4. The arcs of $T(1)$ have the same orientation as the arcs of $T(4)$ except for the arcs on the path from 4 to 1 in $T(1)$. The arcs on this path are reoriented in $T(4)$.

Suppose that x is a basic feasible flow, and let T be the associated spanning tree. If $(k,l) \in G^*(x)$, then the *basic cycle* W created by adding (k,l) to T is (k,l) together with the directed path from node l to node k in $T(k)$. To send flow around W is to decrease the residual capacity of each arc of W by $\delta = \min(r_{ij}: (i,j) \in W)$, and correspondingly increase the residual capacity of all the reverse arcs of W by δ. By sending δ units of flow around W, one of the arcs of W has its residual capacity reduced to 0. By the non-degeneracy assumption, $\delta > 0$, and prior to sending flow around W there is a unique arc of W with residual capacity of δ. A *simplex pivot* consists of adding arc (k,l) to the tree, sending δ units of flow around W, and pivoting out the arc whose residual capacity is reduced to 0. In the case that the cost of the cycle is negative and $\delta > 0$, the solution obtained by the simplex pivot has a strictly lower objective function than the solution prior to the pivot. The network simplex algorithm uses the following well-known fact:

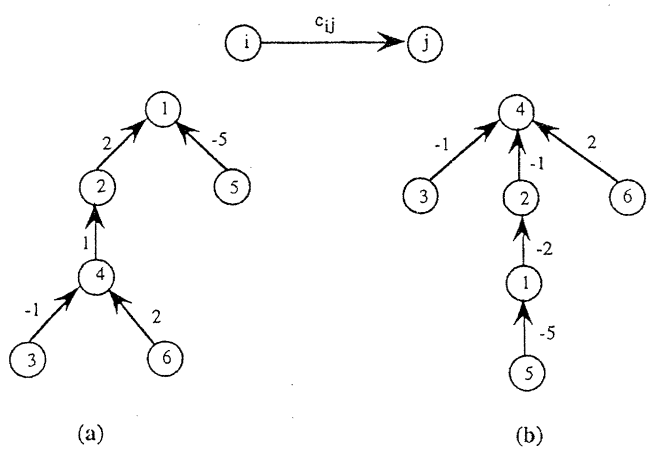

Fig. 1. (a) An in-tree rooted at node 1. The arc values are costs. (b) The same in-tree as in (a) but rooted at node 4.

Optimality conditions. *A basis structure (T,L,U) is optimal if the cost of each basic cycle is non-negative.*

We now describe the network simplex algorithm in a very general form.

algorithm *network-simplex*;
begin
  find a feasible basis structure (T,L,U);
  let x be the basic feasible flow;
  while x is not optimum **do**
  begin
    find an arc $(k,l) \in G^*(x)$ for which the corresponding basic cycle W has a negative cost;
    perform a simplex pivot by sending $\delta = \min(r_{ij}: (i,j) \in W)$ units of flow around W;
    update x and (T,L,U);
  end
end;

Under the non-degeneracy assumption, the network simplex algorithm is finite since there are a finite number of basis structures, and the sequence of basis structures obtained by the algorithm have strictly decreasing objective function values.

A vector π of node potentials is called a *vector of simplex multipliers* for tree T if $c_{ij}^{\pi} = 0$ for all $(i,j) \in T$. In standard implementations of the network simplex algorithm, one maintains a vector of simplex multipliers for each basic feasible solution. Simplex multipliers have the following computational advantage: if simplex multipliers are maintained, then the cost of the basic cycle induced by the arc $(k,l) \in G^*(x)$ is c_{kl}^{π}, therefore, one can test whether a basic cycle has negative cost by simply evaluating $c_{kl}^{\pi} = c_{kl} - \pi_k + \pi_l$ and checking whether $c_{kl}^{\pi} < 0$. This computational advantage comes at the cost of maintaining the simplex multipliers at each stage. The time to maintain the simplex multipliers is $O(n)$ steps per pivot in the worst case, although computational experience suggests that it is far less in practice.

3. The premultiplier algorithm

In our implementation of the simplex algorithm, we will be maintaining a set of node potentials that we refer to as "premultipliers". In this section, we define premultipliers and also describe a simple way of implementing the simplex algorithm using premultipliers, which we call the premultiplier algorithm.

Definition 3. A vector π of node potentials is a *set of premultipliers* with respect to the rooted in-tree $T(v)$ if $c_{ij}^{\pi} \leq 0$ for every arc $(i,j) \in T(v)$. A vector π of node potentials is a *vector of premultipliers* for tree T if it is a vector of premultipliers with respect to

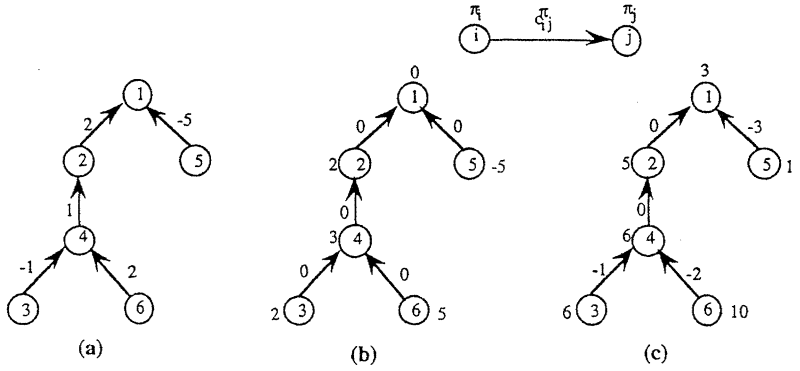

Fig. 2. (a) A spanning in-tree $T(1)$ with arc costs. (b) The simplex multipliers. The reduced cost of each tree arc is 0. (c) Simplex premultipliers. Each arc of $T(1)$ has a non-positive reduced cost.

$T(v)$ for some node v. (Notice that simplex multipliers are a special case of premultipliers in which $c_{ij}^{\pi} = 0$ for every arc (i,j) in $T(v)$.)

Fig. 2(a) illustrates a spanning tree $T(1)$, and Fig. 2(b) gives a set of simplex multipliers for the in-tree. Fig. 2(c) illustrates a set of premultipliers with respect to $T(1)$. Each arc of the rooted in-tree has a non-positive reduced cost. If $(i,j) \in T(1)$ and $(i,j) \notin A$, then $(j,i) \in A$, and its reduced cost is non-negative.

Lemma 1. *Suppose that T is a tree, and π is a set of premultipliers with respect to $T(v)$. Then π is also a set of premultipliers with respect to $T(i)$ if and only if each arc of T on the path from node v to node i has a reduced cost of 0.*

Proof. Note that $T(i)$ may be obtained from $T(v)$ by reversing the orientation of each arc on the path P from i to v. Suppose first that each arc on the path P in $T(v)$ has a reduced cost of 0. Then the reduced costs of the reversals of these arcs are also 0, and every other arc of $T(i)$ has a non-positive reduced cost. Suppose now that some arc (j,k) on the path P in $T(v)$ from i to v has a reduced cost that is not 0. Then its reduced cost is negative, and in $T(i)$ the reduced cost of (k,j) is positive. Thus in this case, π is not a vector of premultipliers with respect to $T(i)$, leading to a contradiction. □

In the premultiplier algorithm, we define eligible arcs differently than in the standard network simplex algorithm. We also need the concept of eligible nodes.

Definition 4. Let T denote a tree and let π denote a vector of premultipliers with respect to $T(v)$ for some node v. We say that node i is *eligible* if π is a vector of premultipliers with respect to $T(i)$. We call an arc $(i,j) \in G^*(x)$ *eligible* if node i is eligible and $c_{ij}^{\pi} < 0$.

We will show in Lemma 2 that the basic cycles induced by eligible arcs have a negative cost. And in the premultiplier method that follows, each arc that is pivoted into

a spanning tree will be an eligible arc. In general, it is not the case that a negative reduced cost arc in $G^*(x)$ induces a negative cost basic cycle. For instance, consider the example in Fig. 2(c) and suppose that $c^{\pi}_{52} = -1$. Then the cost of the basic cycle 5–2–1–5 containing (5,2) is 2.

Lemma 2. *Let T be a tree, and suppose that π is a set of premultipliers with respect to the rooted in-tree $T(v)$. Then the basic cycle induced by each eligible arc has a negative cost.*

Proof. Let W be a basic cycle in $T(v)$ induced by the arc (k,l) and let w be the apex of cycle W. Then, $W = \{(k,l)\} \cup P \cup P'$ where P is a directed path from node l to node w in $T(v)$, and P' is a path from node w to node k in $T(v)$. Hence, $c(W) = c^{\pi}(W) = c^{\pi}_{kl} + \sum_{(i,j)\in P} c^{\pi}_{ij} + \sum_{(i,j)\in P'} c^{\pi}_{ij}$. Since (i) $c^{\pi}_{kl} < 0$ (by definition); (ii) $c^{\pi}_{ij} \le 0$ for each arc $(i,j) \in P$ (because π is a set of premultipliers for $T(v)$); and (iii) $c^{\pi}_{ij} = 0$ for each arc $(i,j) \in P'$ (by Lemma 1); it follows that $c(W) < 0$. □

In the premultiplier algorithm described below, the entering arc is always an eligible arc.

```
algorithm premultiplier;
begin
  choose an initial basic feasible solution x and a vector π of premultipliers with
  respect to T(v);
  while x is not optimal do
    if there is an eligible arc then
    begin
      select an eligible arc (k,l);
      simplex-pivot(k,l);
    end
    else modify-premultipliers;
end;

procedure simplex-pivot(k,l);
begin
  let W denote the basic cycle containing (k,l);   {W is (k,l) plus the path from l to
  k in T(k)};
  let δ = min(r_ij: (i,j) ∈ W);
  send δ units of flow around W;
  let (p,q) denote the arc of W that is pivoted out;
  reset the root of the tree to be p;
end;
```

procedure *modify-premultipliers*;
begin
let S denote the set of eligible nodes;
$Q := \{(i,j) \in T(v)\colon i \notin S,\ j \in S\}$;
$\Delta := \min(-c_{ij}^{\pi}\colon (i,j) \in Q)$; {observe that $\Delta > 0$ whenever $S \neq N$}
for each node $j \in S$, increase π_j by Δ;
end;

The subroutine *simplex-pivot* pivots in the arc (k,l) and pivots out the arc (p,q) so as to maintain primal feasibility. It also adjusts the root node of the tree.

We will now prove that the premultiplier algorithm solves the minimum cost flow problem correctly in a finite number of iterations. Our proof consists of showing that at each iteration the algorithm either increases the number of eligible nodes or performs a non-degenerate pivot. As there can be at most n eligible nodes, the algorithm will perform a non-degenerate pivot within n iterations. Each non-degenerate pivot obtains a new basis structure with lesser objective function value. Since the minimum cost flow problem has a finite number of basis structures, the premultiplier algorithm terminates finitely.

Lemma 3. *The premultiplier algorithm maintains a vector of premultipliers at every step.*

Proof. We will prove the lemma by performing induction on the number of steps performed by the algorithm. By assumption, the result is true for the initial basis structure. Let us first study the effect of an execution of the procedure modify-premultipliers. Let π be the premultipliers with respect to the rooted tree $T(v)$, and let π' be the vector of premultipliers subsequently. We need to show that $c_{ij}^{\pi'} \leq 0$ for each arc (i,j) in $T(v)$. (Recall that the procedure modify-premultipliers does not change the tree $T(v)$, it only changes the premultipliers.)

Let S denote the set of eligible nodes with respect to the vector π. The procedure modify-premultipliers increases the premultiplier of each node in S by Δ units. Clearly, this change does not affect the reduced costs of those arcs in $T(v)$ which have either both of their endpoints in S or neither of their endpoints in S. By definition, $T(v)$ does not contain any arc (i,j) with $i \in S$ and $j \notin S$. So we have only to consider those arcs in $T(v)$ for which $i \notin S$ and $j \in S$. Notice that the algorithm denotes this set of arcs by Q. Increasing the potentials of nodes in S by Δ increases the reduced cost of every arc in Q by Δ units, but it is easy to see that each one of these reduced costs remains non-positive and the reduced cost of at least one of these arcs becomes zero.

We now consider the execution of the procedure *simplex-pivot*. Recall that arc (k,l) is the entering arc. Since (k,l) is an eligible arc, node k is an eligible node. Therefore, π is a set of premultipliers with respect to $T(k)$, and $c_{ij}^{\pi} \leq 0$ for each arc in $T(k)$. The basic cycle W induced by arc (k,l) consists of arc (k,l) and the tree path P in $T(k)$ from node l to node k (see Fig. 3). The subsequent flow augmentation reduces the

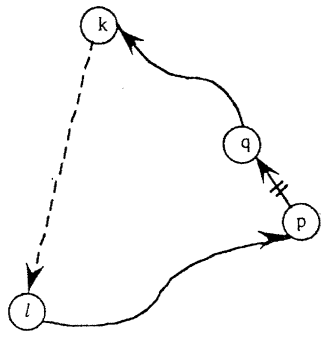

Fig. 3. Illustrating the proof of Lemma 3.

capacity of exactly one arc (p,q) in W to zero. Let $T' = T - (p,q) + (k,l)$. Then the orientation of each arc in $T'(p) - \{(k,l)\}$ is the same as the orientation of the arc in $T(k)$, and so $c_{ij}^{\pi} \leq 0$ for every arc in $T'(p) - \{(k,l)\}$. Further, notice that $c_{kl}^{\pi} < 0$ completing the proof. □

In the preceding proof, we observed that in an execution of the procedure modify-premultipliers the reduced cost of some tree arc, say (α,β) satisfying $\alpha \notin S$ and $\beta \in S$, becomes zero. Consequently, node α becomes an eligible node after the procedure has been executed, establishing the following lemma.

Lemma 4. *Each call of modify-premultipliers strictly increases the number of eligible nodes.*

We are now ready to prove the main result of this section.

Theorem 1. *The premultiplier algorithm is a special case of the network simplex algorithm. As such, it solves the minimum cost flow problem in a finite number of iterations.*

Proof. The premultiplier algorithm maintains a vector of premultipliers at every step and, therefore, the basic cycle induced by an eligible arc always has a negative cost. Therefore, the premultiplier method is a special case of the network simplex algorithm. The finiteness follows directly from the non-degeneracy assumption, which guarantees that basis structures are not repeated. □

4. The scaling premultiplier algorithm

In this section, we apply the Goldberg–Tarjan [3] cost scaling algorithm to the premultiplier algorithm of Section 3. The resulting specialization of the primal network simplex algorithm runs in $O(\min(\log nC, m \log n))$ scaling phases, each of which performs $O(nm)$ pivots. So the total number of pivots is $O(\min(nm \log nC, nm^2 \log n))$.

We also show how to implement the algorithm so that the average time per pivot is $O(n)$ and the total running time is $O(\min(n^2 m \log nC, n^2 m^2 \log n))$. Recently [23] has improved the amortized running time to $O(\log n)$ per pivot using a variant of dynamic trees. The cost scaling version of the premultiplier algorithm uses four additional notations which we define next.

Definition 5. The set N^* denotes a subset of nodes whose multipliers have yet to change during the ε-scaling phase. For example, if π^0 denotes the multipliers at the beginning of the current scaling phase, and if π denotes the current multipliers, then $N^* = \{i: \pi_i = \pi_i^0\}$.

Definition 6. Let π be a vector of premultipliers with respect to a basic feasible flow x. Then, π is a vector of *ε-premultipliers* if $c_{ij}^{\pi} \geq -\varepsilon$ for all arcs (i,j) in $G(x)$.

Definition 7. We call a node *awake* if $i \in N^*$ or if π_i is an integral multiple of $\varepsilon/4$. Nodes that are not awake are called *asleep*.

Definition 8. An arc (k,l) in $G^*(x)$ is called *admissible* for the ε-scaling phase if node k is an eligible and awake node, and $c_{kl}^{\pi} \leq -\varepsilon/4$.

The cost scaling version of the premultiplier algorithm is similar to Goldberg and Tarjan's [3] cost scaling algorithm and performs a number of scaling phases. The algorithm maintains a set of ε-premultipliers. A scaling phase, called an *ε-scaling phase*, consists of an execution of the procedure improve-approximation which takes in a vector π of ε-premultipliers with respect to a basic feasible solution x and transforms it into a vector π' of $\varepsilon/2$-premultipliers with respect to a basic feasible solution x'. The phase terminates when $N^* = \emptyset$. The formal description of the cost scaling algorithm is as follows.

algorithm *scaling-premultiplier*;
begin
  choose an initial basic feasible solution x and a vector π of premultipliers with respect to $T(v)$;
  $\varepsilon := \max(|c_{ij}^{\pi}|: c_{ij}^{\pi} \leq 0, (i,j) \in G(x))$;
  while x is not optimal **do**
  begin
    improve-approximation(x, ε, π);
    $\varepsilon := \max(|c_{ij}^{\pi}|: c_{ij}^{\pi} < 0$ and $(i,j) \in G(x))$;
    {In general, ε is decreased by at least a factor of 2 at each stage.}
  end;
end;

```
procedure improve-approximation(x, ε, π);
begin
  N* := N;
  while N* ≠ ∅ do
  begin
    if there is an admissible arc do
    begin
      select an admissible arc (k,l); {recall that c^π_kl ⩽ −ε/4}
      simplex-pivot(k,l);
    end
    else modify-ε-premultipliers;
  end;
end;
procedure modify-ε-premultipliers;
begin
  let S be the set of eligible nodes;
  N* := N* − S;
  if N* = ∅ then terminate improve-approximation(x, ε, π);
  Δ1 := min(−c^π_ij: (i,j) ∈ T(v), i ∉ S, j ∈ S);
  Δ2 := min(ε/4 − π_i (mod ε/4): i ∈ S);
  increase π_i by Δ = min(Δ1, Δ2) for each i ∈ S;   {observe that Δ > 0}
end;
```

Observe that $\pi_i \pmod{\varepsilon/4} \in [0, \varepsilon/4)$, and so $\Delta_2 > 0$. In the procedure *modify-ε-premultipliers*, we define Δ_2 to be the least positive real number such that for some eligible node i, $\pi_i + \Delta_2$ is an integral multiple of $\varepsilon/4$. The subroutine *simplex-pivot* is the same subroutine as in the premultiplier algorithm described earlier. It pivots in the arc (k,l) and pivots out arc (p,q) so as to maintain primal feasibility. If arc (p,q) is pivoted out, then node p is set to be the root of the new spanning tree. By the non-degeneracy assumption, there is a unique choice for the leaving arc.

There is a simpler version of this algorithm which guarantees the same number of pivots, but for which the running time is $O(m)$ per pivot. The changes needed are as follows: First, relax the definition of an admissible arc (i,j) so that node i is eligible, but not necessarily awake. In addition, we still require that (i,j) have residual capacity, and that the reduced cost of (i,j) is at most $-\varepsilon/4$. Second, eliminate Δ_2 in the modification of the premultipliers, and let $\Delta = \min(\Delta_1, \varepsilon/4)$. These alterations in the algorithm do not affect the bound on the number of pivots nor do they affect the bound on the number of modifications of the premultipliers. With these modifications, the amortized time per pivot increases to $O(m)$.

In order to guarantee an amortized average time of $O(n)$ per pivot, we need to be careful in how we identify admissible arcs. For this purpose, we maintain a widely used data structure, called the *current arc data structure* (see, for example, [1]). Each node i has a current arc, which is an arc in $G(x)$ and is the next candidate for admissibility testing. Initially, the current arc of node i is the first arc emanating from node i. We

assume that arcs emanating from each node are ordered in some fashion and this ordering once defined remains unchanged throughout the algorithm. Whenever the algorithm attempts to find an admissible arc emanating from node i, it tests whether the node's current arc is admissible. If not, it designates the next arc in the arc list as the current arc. The algorithm repeats this process until it either finds an admissible arc or reaches the end of the arc list. When the arc list is exhausted, the current arc of i is set to be $\emptyset$, representing the fact that there are no arcs to scan. The current arc of node i is reset to $FirstArc(i)$ when i is reawakened, that is, when the premultiplier of node i has increased to the next integral multiple of $\varepsilon/4$.

In a search for an admissible arc, the algorithm will first perform a depth first search starting at the root node to identify nodes that are eligible. For each node that is both eligible and awake, the algorithm will scan the node's arc list starting with its current arcs. The time to identify nodes that are both eligible and awake and for which the current arc is not null is $O(n)$ following a pivot or update of premultipliers. We conclude that the number of times that arc (i,j) is scanned per scaling phase is at most the number of times in which node i is awakened.

In principle, one could once again scan arcs emanating from node i as soon as π_i increases; however, if one were to do so, then scanning for admissible arcs would become the bottleneck operation of the algorithm. In order to eliminate this bottleneck, we say that node i goes to sleep subsequent to scanning its arc list, and does not wake up until π_i has increased by a total of $\varepsilon/4$ units. Since each node is awakened $O(n)$ times per scaling phase (see Lemma 9), each arc is scanned $O(n)$ times per scaling phase, for a total running time of $O(nm)$ per scaling phase for scanning arc lists for admissible arcs. Moreover, suppose that the premultipliers are π' and the flow is x' when node i goes to sleep and the multipliers are π and the flow is x when node i awakens. Then for each arc $(i,j) \in G(x')$, $c_{ij}^{\pi'} > -\varepsilon/4$. Also $\pi_i = \pi_i' + \varepsilon/4$, and one can show that for each arc $(i,j) \in G(x)$, $c_{ij}^{\pi} > -\varepsilon/2$. So $\varepsilon/2$-optimality is maintained for all arcs emanating from a node not in N^* (see Lemma 6).

We now outline some important features of the cost scaling premultiplier algorithm.

1. As in the premultiplier algorithm of Section 3, during the ε-scaling phase, each multiplier will be monotonically non-decreasing from iteration to iteration. Moreover, there will be at least one multiplier that does not change during the entire scaling phase, and each other multiplier will increase by at most $3n\varepsilon/2$.
2. Each arc (k,l) that is pivoted in is an admissible arc with reduced cost $c_{kl}^{\pi} \leq -\varepsilon/4$, and so the cost of the basic cycle is less than or equal to $-\varepsilon/4$.
3. Each arc (k,l) will be pivoted in at most $6n$ times during a scaling phase, and the total number of pivots per scaling phase is $O(nm)$.
4. The number of scaling phases is $O(\min(n \log n, \log nC))$.

We will now establish a polynomial bound on the number of pivots performed by the algorithm. We will prove this result after establishing a series of lemmas.

Lemma 5. *Suppose that π is a vector of ε-premultipliers obtained during the ε-scaling phase, and let i be any node not in N^*. Let π' be the vector of premultipliers*

immediately prior to the most recent execution of modify-ε-premultipliers at which time i was both eligible and awake. Then $0 < \pi_i - \pi'_i \leq \varepsilon/4$.

Proof. Let π^0 denote the premultipliers at the beginning of the scaling phase. We first note that node i is both eligible and awake at the first time that the potential of node i is increased in the ε-scaling phase, and so π' is well-defined. Consider first the case that $\pi_i - \pi_i^0 \leq \varepsilon/4$. In this case the lemma is easily seen to be true. We next consider the case that $\pi_i - \pi_i^0 > \varepsilon/4$. In this case, let α be the largest integral multiple of $\varepsilon/4$ that is strictly less than π_i. It follows that $\pi_i - \alpha \leq \varepsilon/4$, and $\pi_i^0 < \alpha < \pi_i$. We will show that $\pi'_i = \alpha$, and this will complete the proof.

Consider the first iteration at which the premultiplier of node i is increased to a value that is at least α. By construction of Δ_2 in the subroutine modify-ε-premultipliers, the premultiplier of node i is increased to exactly α. Now consider the first iteration at which the premultiplier of node i is increased to a value that is strictly larger than α. At this iteration, node i was both eligible and awake. We conclude that node i will not again become awake until its node potential becomes $\alpha + \varepsilon/4 \geq \pi_i$, and so $\pi'_i = \alpha$, completing the proof. □

Lemma 6. *Suppose that x is a basic feasible flow and that π is a vector of ε-premultipliers obtained during the ε-scaling phase. For all* $(i,j) \in G(x)$, *if* $i \notin N^*$, *then* $c_{ij}^{\pi} \geq -\varepsilon/2$. *In particular, if* $N^* = \emptyset$, *then π is a vector of ε/2-premultipliers with respect to x.*

Proof. Consider arc $(i,j) \in G(x)$. Let π' be the vector of premultipliers immediately prior to the most recent iteration of *modify-ε-premultipliers* at which time node i was both awake and eligible. At this point, arc (i,j) was a tree arc or was an inadmissable non-tree arc, and so $c_{ij}^{\pi'} > -\varepsilon/4$ or else (i,j) had no residual capacity. Let us first consider the case that $c_{ij}^{\pi'} > -\varepsilon/4$. In this case, $c_{ij}^{\pi} = c_{ij}^{\pi'} - (\pi_i - \pi'_i) + (\pi_j - \pi'_j)$. By Lemma 5, $(\pi_i - \pi'_i) \leq \varepsilon/4$, and $\pi_j \geq \pi'_j$. It follows that $c_{ij}^{\pi} \geq -\varepsilon/4 - \varepsilon/4 + 0 = -\varepsilon/2$.

We now consider the case that (i,j) had no residual capacity at the iteration in which the premultipliers were π'. Suppose at some later iteration, when the pre-multipliers are π', (i,j) receives some residual capacity. Arc (i,j) can receive residual capacity only when arc (j,i) enters the basis, and so $c_{ji}^{\pi'} \leq -\varepsilon/4$ implying $c_{ij}^{\pi'} = -c_{ji}^{\pi'} \geq \varepsilon/4$. Moreover,

$$c_{ij}^{\pi} = c_{ij}^{\pi'} - (\pi_i - \pi'_i) + (\pi_j - \pi'_j) \geq c_{ij}^{\pi'} - (\pi_i - \pi'_i) + (\pi_j - \pi'_j)$$
$$\geq \varepsilon/4 - \varepsilon/4 + 0 = 0$$

completing the proof. □

Lemma 7. *Either some node becomes awake subsequent to the execution of modify-ε-premultipliers or the number of eligible nodes strictly increases.*

Proof. Let π denote the vector of premultipliers prior to the execution of *modify-ε-premultipliers*, and let π' denote the premultipliers subsequently. Let S denote the nodes

that are eligible with respect to π. Note that all of the nodes in S are eligible with respect to π'. In the case that $\Delta = \Delta_2$, at least one node becomes awake. We now consider the case that $\Delta = \Delta_1$. This is the same case as in the proof of Lemma 4, and so there is at least one node that is eligible with respect to π' that is not eligible with respect to π. □

We now proceed to bound the total increase in the multipliers during the scaling phase. We will show in Lemmas 8 and 9 that each π_i increases by $O(n\varepsilon)$ during the ε-scaling phase. The counterpart of Lemmas 8 and 9 for pseudoflow algorithms for the minimum cost flow problem was proved in [3].

Lemma 8. *Let x and x' denote any two distinct non-degenerate basic feasible flows. Then for any pair of nodes i and j, there is a path P in $G(x)$ from node i to node j such that the reversal of P is a path from node j to node i in $G(x')$.*

Proof. Let T denote the tree corresponding to the basic flow x. Let $d(i,j)$ denote the number of arcs in the unique path from node i to node j in tree T. We prove the result inductively on $d(i,j)$. Consider first the case that $d(i,j) = 1$, and thus either $(i,j) \in T$ or $(j,i) \in T$. Suppose that $(i,j) \in T$. Let r'_{ji} denote the residual capacity of (j,i) with respect to x'. If $r'_{ji} > 0$, then let $P = (i,j)$ and the result is true. So suppose that $r'_{ji} = 0$. It follows from the non-degeneracy assumption that $r_{ji} > 0$. By flow decomposition, $x' - x$ can be decomposed into flows around cycles each of which is a directed cycle in $G(x)$. Equivalently, x may be transformed into x' by sequentially sending flow around a number of directed cycles, each of which is in $G(x)$ (see, for example, [1] for more details). Since $r'_{ji} = 0 < r_{ji}$, it follows that one of these cycles, say W, contains arc (j,i). Let P denote the path in W from node i to node j. Then P is in $G(x)$, and the reversal of P is in $G(x')$, completing the proof in the case that $(i,j) \in T$.

Now suppose that $d(i,j) = K > 1$, and assume that the lemma is true for all pairs of nodes u and w such that $d(u,w) \leq K - 1$. Let h be the node that precedes j on the path from i to j in T. Thus $d(i,h) = K - 1$, and $d(h,j) = 1$. By the inductive hypothesis, there are paths P_1 and P_2 in $G(x)$ such that P_1 is a path from i to h, P_2 is a path from h to j, and the reversals of P_1 and P_2 are in $G(x')$. Let P_3 be a path obtained by concatenating P_1 and P_2 into a directed walk from i to j and then removing any directed cycles contained in the walk. Then P_3 is in $G(x)$ and its reversal is in $G(x')$, completing the proof. □

Lemma 9. *During the ε-scaling phase, π_i is increased by at most $3/2 \cdot n\varepsilon$ units.*

Proof. Let x denote the basic flow and π denote the premultipliers at the beginning of the ε-scaling phase. Let x' denote basic flow and π' the premultipliers at some iteration during the ε-scaling phase. Select a node $j \in N^*$. (The ε-scaling phase ends immedi-

ately when $N^* = \emptyset$, and so throughout the scaling phase $N^* \neq \emptyset$.) By the definition of N^*, $\pi'_j = \pi_j$.

Let P be a path from node j to node i in $G(x)$ such that the reversal of P, denoted as P^r, is in $G(x')$. Without loss of generality, we assume that node j is the only node of N^* in P. (Otherwise, we could replace j by the last node j^* of N^* in P, and replace P by the subpath P^* of P from j^* to i.) Then, it follows from the ε-optimality of the flow that

$$c^{\pi}(P) = c(P) - \pi_j + \pi_i \geq -(n-1)\varepsilon.$$

Moreover, since every arc of P^r emanates from a node in $N - N^*$, it follows from Lemma 6 that

$$c^{\pi'}(P^r) = c(P^r) - \pi'_i + \pi'_j \geq -(n-1)\varepsilon/2.$$

Adding the negative of the above two inequalities and noting that (i) $\pi'_j = \pi_j$, and (ii) $c(P) + c(P^r) = 0$ yields the following inequality: $\pi'_i \leq \pi_i + 3(n-1)\varepsilon/2$. □

Lemma 10. *The scaling premultiplier algorithm performs at most $6nm$ pivots per scaling phase.*

Proof. In the ε-scaling phase, suppose that π is the vector of premultipliers at some iteration at which (i,j) or (j,i) is pivoted in, and that π' is the vector of premultipliers at the next iteration at which (i,j) or (j,i) is pivoted in again. We claim that $\pi'_i - \pi_i + \pi'_j - \pi_j \geq \varepsilon/2$. By Lemma 9, π_i and π_j may each increase by at most $3n\varepsilon/2$ during the scaling phase. If our claim is true, then the number of iterations at which (i,j) or (j,i) is pivoted in is at most $6n$, and the lemma would follow.

We now justify our claim that $\pi'_i - \pi_i + \pi'_j - \pi_j \geq \varepsilon/2$. Suppose that π is the vector of premultipliers when arc (i,j) is pivoted in. (The proof in the case when arc (j,i) is pivoted in at that iteration is essentially the same.) Thus, $c^{\pi}_{ij} = c_{ij} - \pi_i + \pi_j \leq -\varepsilon/4$. Consider first the case that (j,i) is pivoted in when π' are the multipliers. Then, $c^{\pi'}_{ji} \leq -\varepsilon/4$, and so $c^{\pi'}_{ij} = -c^{\pi'}_{ji} \geq \varepsilon/4$. It follows that

$$\varepsilon/2 \leq c^{\pi'}_{ji} - c^{\pi}_{ji} = -(\pi'_i - \pi_i) + (\pi'_j - \pi_j) \leq (\pi'_j - \pi_j)$$

and thus the claim is true in this case.

We now consider the case that arc (i,j) in $G(x)$ is pivoted in when π' is the vector of premultipliers. It is easy to see that in between the two iterations when (i,j) is pivoted in, arc (j,i) must be pivoted out of the rooted in-tree (because when arc (i,j) is pivoted in, positive flow is sent from node i to node j and it cannot pivot in again until flow is sent back from node j to node i). Let π'' be the vector of premultipliers when (j,i) is pivoted out of the rooted in-tree $T(v)$. Since π'' is a vector of premultipliers and $(j,i) \in T(v)$, it follows that $c^{\pi''}_{ji} \leq 0$, or equivalently $c^{\pi''}_{ij} \geq 0$. Since premultipliers are non-decreasing during the scaling phase, $\pi \leq \pi'' \leq \pi'$. Now observe that $c_{ij} - \pi_i + \pi_j \leq -\varepsilon/4$, $c_{ij} - \pi''_i + \pi''_j \geq 0$, and $c_{ij} - \pi'_i + \pi'_j \leq -\varepsilon/4$. It follows from these obser-

vations that $\pi_j' - \pi_j \geq \pi_j'' - \pi_j \geq \varepsilon/4$, and $\pi_i' - \pi_i \geq \pi_i' - \pi_i'' \geq \varepsilon/4$, and so our claim that $\pi_i' - \pi_i + \pi_j' - \pi_j \geq \varepsilon/2$ is true, completing the proof. □

Lemma 11. *The scaling premultiplier algorithm terminates in* $O(\min(m \log n, \log nC))$ *scaling phases with an optimal flow.*

Proof. We first bound the number of scaling phases to $\log nC$ in the case when the data are integral. The initial value of ε is $O(nC)$. (Recall that we have added artificial arcs with cost $nC + 1$ to create the initial basis.) By Lemma 6, ε decreases by at least a factor of 2 at each scaling phase. Thus, within $O(\log nC)$ scaling phases, ε is reduced to a number that is strictly less than $1/n$. We now claim that x is an optimal flow if $\varepsilon < 1/n$. By Lemma 6, π is a vector of ε-premultipliers, and so $c_{ij}^{\pi} \geq -\varepsilon > -1/n$ for each arc (i,j) in $G(x)$. Suppose W is any directed cycle in $G(x)$. Then $c(W) = c^{\pi}(W) > -1$. Since $c(W)$ is integral, it also follows that $c(W) \geq 0$, and thus the flow is optimal. (Here we make use of a well-known result that if the residual network $G(x)$ does not contain any negative cycle, then x is an optimal flow.) This completes the proof that there are $O(\log nC)$ scaling phases.

We now show that the number of scaling phases at which a pivot takes place is $O(m \log n)$. We say that an arc is *permanently non-basic* at the ε-scaling phase if it is not in any feasible basis in the ε-scaling phase or in any subsequent phase. (The idea of permanently non-basic arcs is based on the work of Tardos [25,26].) For every scaling phase with a pivot, we will show that some arc in the spanning tree becomes permanently non-basic within $(4 + \lceil 2 \log n \rceil)$ additional scaling phases. Since each arc may become permanently non-basic at most once, this will bound the number of scaling phases with a pivot at $(4m + m\lceil 2 \log n \rceil)$.

Since (i) the total increase in π_i in the ε-scaling phase is $3n\varepsilon/2$, and (ii) ε decreases by a factor of at least 2 in two consecutive scaling phases, it follows that the total increase in π_i in the ε-scaling phase and all subsequent scaling phases is bounded by $3n\varepsilon/2 + 3n\varepsilon/4 + 3n\varepsilon/8 + \cdots = 3n\varepsilon$. So if $c_{ij}^{\pi} > 3n\varepsilon$ during the ε-scaling phase, (i,j) is permanently non-basic because its reduced cost will never become negative.

Suppose that arc (i,j) is pivoted in during the ε-scaling phase, and let W denote the basic cycle containing (i,j). Then $c(W) \leq -\varepsilon/4$ because arc (i,j) is $\varepsilon/4$-eligible. Let ε' denote the scale factor $(4 + \lceil 2 \log n \rceil)$ scaling phases later, and let π' be the vector of premultipliers at the beginning of ε' scaling phase. Then $\varepsilon' < \varepsilon/(16n^2)$. Thus $c^{\pi'}(W) = c(W) < -4n^2\varepsilon'$. Since W has at most n arcs, there is an arc (i,j) of W for which $c_{ij}^{\pi'} < -3n\varepsilon'$. Since π' is a vector of ε'-premultipliers (and thus satisfies $c_{ij}^{\pi'} \geq -\varepsilon'$ for each arc (i,j) in $G(x)$), it follows that arc (i,j) is not present in $G(x)$. But then arc (j,i) is present in $G(x)$, and $c_{ji}^{\pi'} > 3n\varepsilon'$. Therefore (j,i) is permanently non-basic, which is what we wanted to prove. We conclude that the number of scaling phases in which a pivot takes place is $O(m \log n)$.

Finally, we bound the number of scaling phases at which no pivot takes place. Actually, we will show that there are never two consecutive scaling phases in which no pivot takes place. Suppose that no pivot takes place at the ε-scaling phase. Since no arc

is pivoted in, the subroutine *modify-ε-premultipliers* is called consecutively until each node is eligible. So, at the end of the scaling phase, the vector π of premultipliers is a vector of simplex multipliers, and each arc of the tree has a reduced cost of 0. At the next scaling phase, the scale factor $\varepsilon = \max(-c_{ij}^{\pi}: c_{ij}^{\pi} < 0)$. Thus there will be some arc (k,l) with $c_{kl}^{\pi} = -\varepsilon$. Since all nodes including node k are both eligible and awake, it follows that (k,l) is an admissible arc, and thus a pivot takes place during this scaling phase. Therefore, the number of scaling phases is at most twice the number of scaling phases at which a pivot takes place, and this is $O(m \log n)$. □

Lemma 12. *The subroutine modify-ε-premultipliers is executed* $O(nm)$ *times per scaling phase.*

Proof. By Lemma 7, each execution of *modify-ε-premultipliers* either awakens a new node or it leads to an eligible node being added. By Lemma 9, the former case can happen at most $6n$ times per node, and thus $O(n^2)$ times in total. We now bound the number of times that a new node is made eligible.

Whenever a new node is made eligible in the subroutine *modify-ε-premultipliers*, there is an arc $(i,j) \in T$ such that the reduced cost of (i,j) was non-zero before the change in premultipliers, and the reduced cost was zero subsequently. We refer to this as *cancelling* a tree arc. We now claim that the number of times that an arc (i,j) can be cancelled is at most the number of times that (i,j) is pivoted in. To see that, notice that the reduced cost of (i,j) is negative when (i,j) is pivoted in, and it remains negative until either (i,j) is cancelled or else (i,j) is pivoted out. Once (i,j) is cancelled, then the reduced cost of (i,j) stays at zero until (i,j) is pivoted out. Thus each arc (i,j) is cancelled at most once in between successive times that (i,j) is pivoted in, proving that (i,j) is cancelled $O(n)$ times per scaling phase. □

We are now ready to prove our main result:

Theorem 2. *The scaling premultiplier algorithm solves a minimum cost flow problem in a sequence of* $O(\min(m \log n, \log nC))$ *scaling phases, each of which has* $O(nm)$ *pivots. Moreover, the running time per scaling phase is* $O(n^2 m)$.

Proof. By Lemma 11, the number of scaling phases is $O(\min(m \log n, \log nC))$, and the algorithm ends with an optimal flow. By Lemma 10, each scaling phase has $O(nm)$ pivots. The time to send flow around the cycle and update the spanning tree is $O(n)$ per pivot. If there is an admissible arc, then the time to find it is $O(n)$ per pivot plus the time to update current arcs. We have noted earlier in this section that updating of current arcs requires $O(nm)$ time per scaling phase. By Lemma 12, there are $O(nm)$ updates of the vector of premultipliers in the subroutine *modify-ε-premultipliers*. Each call of *modify-ε-premultipliers* potentially involves scanning all of the nodes in N, once to compute Δ_i

and once to compute Δ_2, and thus takes $O(n)$ time. We conclude that the running time per scaling phase is $O(n^2 m)$. □

5. Diameter of the network polytope

In this section, we show that the diameter of the network polytope is $O(nm \log n)$. That is, for any basic feasible solutions x' and x^*, it is possible to obtain x^* from x' via a sequence of $O(nm \log n)$ primal simplex pivots. Our proof uses the same construction as does [22], who showed that the diameter of the network polytope is $O(n^2 m \log n)$.

Theorem 3. *The diameter of the network polytope is* $O(nm \log n)$.

Proof. Let x' be a basic feasible solution corresponding to the basis structure (T, L, U). We first consider the case in which every basis is non-degenerate. We consider the degenerate case subsequently.

Let $c^*_{ij} = 1$ if $x^*_{ij} = 0$, let $c^*_{ij} = -1$ if $x^*_{ij} = u_{ij}$, and let $c^*_{ij} = 0$ if $l_{ij} < x^*_{ij} < u_{ij}$. Then x^* is the unique optimum flow that minimizes $c^* x$. Using the premultiplier algorithm, one can obtain x^* as a sequence of $O(nm \log nC) = O(nm \log n)$ pivots (because $C = O(1)$).

We now consider the case that bases may be degenerate. We observe that by perturbing the right-hand side to make the problem non-degenerate, the diameter can not decrease. So the diameter of the degenerate problem is bounded as well by $O(nm \log n)$. □

Network flow problems are a special case of linear programs defined on totally unimodular matrices. Dyer and Frieze [27] showed that the diameter of these linear programs is polynomially bounded, but their ingenious proof technique did not result in a bound of low degree. Their bound on the diameter of totally unimodular linear programming polytopes is a polynomial of high degree, one that exceeds the $O(n^2 m \log n)$ bound of Goldfarb and Hao as well as the result presented here.

We note that the results of Section 4 show that there is a sequence of $O(nm \log nC)$ pivots moving from any basic feasible solution to another basic feasible solution so that the costs are monotonically non-decreasing (or non-increasing).

Acknowledgements

This paper was partially supported by ONR Grant N00014-94-1-0099 and a grant from the UPS foundation. I want to thank Charu Aggarwal and Izumi Sakuta for helpful comments on an earlier draft of this manuscript. I want to thank both Don Goldfarb and

Ravi Ahuja for encouraging my work on the problem of finding a polynomial-time primal network simplex algorithm. I also wish to thank Ravi Ahuja for extensive editorial suggestions that led to simplifications at several points, improvements in notation at several points, and that filled in a number of small gaps in arguments. His suggestions led to numerous improvements in the exposition.

References

[1] R.K. Ahuja, T.L. Magnanti and J.B. Orlin, *Network Flows: Theory, Algorithms and Applications* (Prentice Hall, Englewood Cliffs, NJ, 1993).
[2] R.K. Ahuja, A.V. Goldberg, J.B. Orlin and R.E. Tarjan, Finding minimum-cost flows by double scaling, *Mathematical Programming* 53 (1992) 243–266.
[3] A.V. Goldberg and R.E. Tarjan, Solving minimum cost flow problem by successive approximation, *Mathematics of Operations Research* 15 (1990) 430–466.
[4] J.B. Orlin, A faster strongly polynomial minimum cost flow algorithm, *Operations Research* 41 (1993) 338–350.
[5] N. Zadeh, A bad network problem for the simplex method and other minimum cost flow algorithms, *Mathematical Programming* 5 (1973) 255–266.
[6] R.E. Tarjan, Efficiency of the primal network simplex algorithm for the minimum-cost circulation problem, *Mathematics of Operations Research* 16 (1991) 272–291.
[7] R.K. Ahuja and J.B. Orlin, The scaling network simplex algorithm, *Operations Research* 40, Supplement 1 (1992) S5–S13.
[8] M. Akgul, A genuinely polynomial primal simplex algorithm for the assignment problem, *Discrete Applied Mathematics* 45 (1993) 93–115.
[9] M.S. Hung, A polynomial simplex method for the assignment problem, *Operations Research* 31 (1983) 595–600.
[10] J.B. Orlin, On the simplex algorithm for networks and generalized networks, *Mathematical Programming Study* 24 (1985) 166–178.
[11] P.T. Sokkalingam, P. Sharma and R.K. Ahuja, A new primal simplex algorithm for network flow problem, Unpublished manuscript, 1993; Presented at NETFLOW'93 at San Miniato, Italy.
[12] M. Akgul, Shortest path and simplex method, Research Report, Department of Computer Science and Operations Research, North Carolina State University, Raleigh, NC, 1985.
[13] R. Dial, F. Glover, D. Karney and D. Klingman, A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees, *Networks* 9 (1979) 215–248.
[14] D. Goldfarb, J. Hao and S. Kai, Efficient shortest path simplex algorithms, *Operations Research* 38 (1990) 624–628.
[15] A.V. Goldberg, M.D. Grigoriadis and R.E. Tarjan, Use of dynamic trees in a network simplex algorithm for the maximum flow problem, *Mathematical Programming* 50 (1991) 277–290.
[16] D. Goldfarb and J. Hao, A primal simplex algorithm that solves the maximum flow problem in at most nm pivots and $O(n^2m)$ time, *Mathematical Programming* 47 (1990) 353–365.
[17] D. Goldfarb and J. Hao, On strongly polynomial variants of the network simplex algorithm for the maximum flow problem, *Operations Research Letters* 10 (1991) 383–387.
[18] J.B. Orlin, Genuinely polynomial simplex and nonsimplex algorithms for the minimum cost flow problem, Technical Report No. 1615-84, Sloan School of Management, MIT, Cambridge, MA, 1984.
[19] S.A. Plotkin and E. Tardos, Improved dual network simplex, in: *Proceedings of the 1st ACM–SIAM Symposium on Discrete Algorithms* (1990) 367–376.
[20] J.B. Orlin, S.A. Plotkin and E. Tardos, Polynomial dual network simplex algorithms, *Mathematical Programming* 60 (1993) 255–276.
[21] R. Armstrong and Z. Jin, A strongly polynomial dual network simplex algorithm, *Mathematical Programming* 78 (1997) 131–148.
[22] D. Goldfarb and J. Hao, Polynomial simplex algorithms for the minimum cost network flow problem, *Algorithmica* 8 (1992) 145–160.

[23] R.E. Tarjan, Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm, *Mathematical Programming* 78 (1997) 169–177.
[24] R.K. Ahuja, T.L. Magnanti and J.B. Orlin, Network flows, in: G.L. Nemhauser, A.H.G. Rinnooy Kan and M.J. Todd, eds., *Handbooks in Operations Research and Management Science. Vol. 1: Optimization* (North-Holland, Amsterdam, 1989) 211–369.
[25] É. Tardos, A strongly polynomial minimum cost circulation algorithm, *Combinatorica* 5 (1985) 247–255.
[26] É. Tardos, A strongly polynomial algorithm to solve combinatorial linear programs, *Operations Research* 34 (1986) 250–256.
[27] M. Dyer and A. Frieze, Random walks, totally unimodular matrices and a randomised dual simplex algorithm, *Mathematical Programming* 64 (1994) 1–16.
[28] W.H. Cunningham, A network simplex method, *Mathematical Programming* 11 (1976) 105–116.