New Polynomial-Time Cycle-Canceling
Algorithms for Minimum Cost Flows

by
P.T. Soddalingam
Ravindra K. Ahuja
James B. Orlin

WP #3914-96

July 1996

New Polynomial-Time Cycle-Canceling Algorithms for Minimum Cost Flows

P. T. Sokkalingam
Department of Mathematics
Indian Institute of Technology,
Kanpur-208 016, INDIA

Ravindra K. Ahuja
Department of Industrial & Management Engg.
Indian Institute of Technology,
Kanpur-208 016, INDIA

James B. Orlin
Sloan School of Management
Massachusetts Institute of Technology,
Cambridge, MA 02139, USA

New Polynomial-Time Cycle-Canceling Algorithms for Minimum Cost Flows

P. T. Sokkalingam^{*}, Ravindra K. Ahuja^{**}, and James B. Orlin^{***}

ABSTRACT

The cycle-canceling algorithm is one of the earliest algorithms to solve the minimum cost flow problem. This algorithm maintains a feasible solution x in the network G and proceeds by augmenting flows along negative cost directed cycles in the residual network $G(x)$ and thereby canceling them. For the minimum cost flow problem with integral data, the generic version of the cycle-canceling algorithm runs in pseudo-polynomial time, but several polynomial-time specific implementations can be obtained by specifying the choices of cycles to be canceled. In this paper, we describe a new cycle-canceling algorithm that solves the minimum cost flow problem in polynomial time. Our algorithm is a scaling algorithm and proceeds by augmenting flows along negative cycles with "sufficiently large" residual capacity. Further, it identifies such a cycle by solving a shortest path problem with nonnegative arc lengths. For a network with n nodes and m arcs, our cycle-canceling algorithm performs $O(m \log(nU))$ augmentations and runs in $O(m(m + n \log n) \log(nU))$ time, where U is an upper bound on the node supplies/demands and finite arc capacities. We next show that the cycle-canceling algorithm (i) can solve the uncapacitated minimum cost flow problem in $O(n(m + n \log n) \log(nU))$ time; (ii) can obtain an integer optimal solution of the convex cost flow problem in $O(m(m + n \log n) \log(nU))$ time; and (iii) can be modified so that it runs in $O(m(m + n \log n) \min\{\log(nU), m \log n\})$ time, which is a strongly-polynomial time bound.

---

^{*} Department of Mathematics, Indian Institute of Technology, Kanpur-208 016, INDIA

^{**} Department of Industrial & Management Engg., Indian Institute of Technology, Kanpur-208 016, INDIA

^{***} Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

1. INTRODUCTION

The minimum cost flow problem is one of the most fundamental of all network flow problems. The cycle-canceling algorithm was one of the first algorithms developed to solve the minimum cost flow problem. This algorithm maintains a feasible solution at every iteration. It proceeds by identifying negative cost directed cycles in the residual network and augmenting maximum possible flows in these cycles and thereby canceling them. The algorithm terminates when the residual network contains no negative cost directed cycle. The cycle canceling algorithm was suggested by Klein [1967], and its generic version is known to run in pseudo-polynomial time for networks with integer data. The generic version of this algorithm does not specify the method for selecting negative cycles in the residual network. Different rules for selecting negative cycles (that is, negative cost directed cycles) produce different versions of the algorithm, each with different worst-case and theoretical behavior. Several researchers have developed specific implementations of the cycle canceling algorithm that run in polynomial time. Figure 1 summarizes these contributions and compares them with our contribution made in this paper (the weakly polyomial time bounds hold for the minimum cost flow problem with integral data). In the table, we assume that n denotes the number of nodes in the network, m denotes the number of arcs, C denotes an upper bound on arc costs and U denotes an upper bound on node supplies/demands and finite arc capacities.

S. No	Algorithmic Description	Reference	Weakly Poly-nomial Bound	Strongly Poly-nomial Bound
1.	Augment flow along a negative cycle with a minimum mean cost.	Goldberg and Tarjan [1988]	$O(n^2 m^2 \log(nC))$	$O(n^2 m^3 \log n)$
2.	Augment flow along a cycle composed entirely of negative reduced cost arcs	Goldberg and Tarjan [1988]	$O(nm \log n \log(nC))$	$O(nm^2 \log^2 n)$
3.	Augment flows in a collection of node-disjoint negative cycles with maximum improvement.	Barahona and Tardos [1989]	$O(mn^3 \log(mCU))$	Not available
4.	Augment flow in a negative cycle with sufficiently large residual capacity	This paper	$O(m \log U (m + n \log n))$	$O(m^2 \log n (m + n \log n))$

Figure 1. Running times of several polynomial-time implementations of the cycle-canceling algorithm.

This paper describes a new polynomial-time implementation of the cycle-canceling algorithm. This implementation proceeds by identifying negative cycles with "sufficiently large" residual

capacity and augmenting flows along them. It uses a scaling technique to define "sufficiently large" residual capacity. Hence, we call our algorithm the *scaling cycle-canceling algorithm*. An important feature of the algorithm is that negative cycles are identified by solving a shortest path problem with nonnegative arc lengths, which in theory is more efficient than solving a shortest path problem with negative arc lengths, as done by most other cycle-canceling algorithm. Our algorithm is shown to run in $O(m(m + n \log n) \log(nU))$ time. We also describe a variant of this algorithm which runs in strongly polynomial time; the running time of this implementation is $O(m(m+n \log n) \min\{\log(nU), m \log n\})$.

The contributions made by this paper are two-fold:

1. We suggest new capacity scaling algorithms for the minimum cost flow and convex cost flow problems with attractive worst-case behavior, and potentially attractive empirical behavior.
2. All the existing capacity scaling algorithms for the minimum cost flow problem are dual algorithms (that is, which maintain complementary slackness conditions all the time but obtain a feasible solution only at the end). Our algorithm is the first capacity scaling algorithm for the minimum cost flow problem which is a primal algorithm (that is, which maintains a feasible solution all the time). The primal algorithms have the advantage that they give a feasible solution even when they are terminated prematurely. Our algorithm for the convex cost flow problem is also the first polynomial-time primal algorithm to obtain an integer solution of the convex cost flow problem.

2. PRELIMINARIES

This section presents network notation and reviews some well known results used in the subsequent sections.

Problem Statement

Let $G = (N, A)$ be a directed network defined by a set N of n nodes and a set A of m directed arcs. Each arc (i, j) has an associated cost c_{ij} , and an associated integral capacity u_{ij} . We associate with each node $i \in N$ an integer number $b(i)$ representing its supply/demand such that $\sum_{i \in N} b(i) = 0$. The minimum cost flow problem is to determine the flow x_{ij} on each arc $(i, j) \in A$ which optimizes the following decision problem:

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

subject to

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

$$0 \leq x_{ij} \leq u_{ij} \quad \text{for all } (i, j) \in A. \quad (1c)$$

We refer to a flow x satisfying the mass balance constraint (1b), and the flow bound constraint (1c) as a *feasible flow*. Our algorithms maintain a feasible flow at every step and proceed by improving its objective function value. We denote by U the largest magnitude among node supplies/demands and finite arc capacities, that is,

$$U = \max[\max\{b(i) : 1 \leq i \leq n\}, \max\{u_{ij} : (i, j) \in A \text{ and } u_{ij} < \infty\}].$$

We use the network notation, such as directed paths, directed cycles, cuts, given in Ahuja, Magnanti and Orlin [1993]. We denote the complement of a set S by placing a bar on it, that is, by $\bar{S}$.

For any set $S \subset N$ of nodes, we denote by $b(S) = \sum_{i \in S} b(i)$. The set S defines a cut $[S, \bar{S}]$ consisting of all arcs in A which have one end point in S and another in $\bar{S}$. We denote by $(S, \bar{S}) = \{(i, j) \in A : i \in S \text{ and } j \in \bar{S}\}$ and $(\bar{S}, S) = \{(i, j) \in A : i \in \bar{S} \text{ and } j \in S\}$, the sets of forward and backward arcs in the cut $[S, \bar{S}]$, respectively. In our subsequent discussion, we shall use the following identity which can be obtained, for any feasible flow x , by summing the mass balance constraints (1b) corresponding to nodes in S :

$$b(S) = \sum_{(i,j) \in (S, \bar{S})} x_{ij} - \sum_{(i,j) \in (\bar{S}, S)} x_{ij}. \quad (2)$$

Assumptions

We henceforth assume the following:

Assumption 1. For any pair i and j of nodes, the network does not contain both the arcs (i, j) and (j, i) .

Assumption 2. The minimum cost flow problem is feasible and its optimal objective function value is finite.

The first assumption is for notational convenience only as the results of this paper are true even if there are multiple arcs. As for the second assumption, it is without any loss of generality since the feasibility of a minimum cost flow problem can be determined by solving a maximum flow problem. Further, the minimum cost flow problem has a finite optimal solution value if and only if the network does not contain any uncapacitated negative cost directed cycle, the presence of which can be detected in $O(m)$ time.

It will be convenient in our exposition in the subsequent sections (except Sections 5 and 7) to assume that all arcs in the network have finite capacities. It can be shown using the flow decomposition theory that if the minimum cost flow problem satisfies Assumption 2, then there exists an optimal solution x in which each arc flow $x_{ij} \leq (n+m)U \leq 2mU$ (see, Ahuja, Magnanti and Orlin [1993]). Consequently, if we impose a capacity of $2mU$ on all uncapacitated arcs, then an optimal solution for the capacitated problem is also optimal for original problem satisfying Assumption 2. We, therefore, can make the following assumption without any loss of generality.

Assumption 3. *In the minimum cost flow problem (1), each arc has a finite capacity.*

Residual Network $G(x)$

The residual network $G(x)$ corresponding to a feasible flow x is defined as follows: We replace each arc $(i, j) \in A$ by two arcs (i, j) and (j, i) . The arc (i, j) has cost c_{ij} and a residual capacity $r_{ij} = u_{ij} - x_{ij}$ and the arc (j, i) has cost $c_{ji} = -c_{ij}$ and a residual capacity $r_{ji} = x_{ij}$. The residual network consists only of arcs with positive residual capacities.

Reduced Costs

A dual solution of the minimum cost flow problem is a vector π of *node potentials*. For a given set π of node potentials, the *reduced cost* of an arc (i, j) in the residual network is $c_{ij}^\pi = c_{ij} - \pi(i) + \pi(j)$. Our algorithm often works with reduced costs in place of arc costs and make use of the following well-known property:

Property 1. *For any directed cycle W in $G(x)$ and any set of node potentials π , $\sum_{(i,j) \in W} c_{ij}^\pi = \sum_{(i,j) \in W} c_{ij}$.*

Shortest Path Optimality Conditions

Consider the directed network $G = (N, A)$ with a_{ij} as the length of each arc $(i, j) \in A$ and a specially designated node q . Suppose we want to find a shortest (directed) path from node q to every other node in the network G . For each node i , let $d(i)$ denote the length of some directed path from node q to node i . It is well known (see, for example, Ahuja, Magnanti and Orlin [1993]) that the $d(i)$'s represent the shortest path distances if they satisfy the following conditions:

$$d(j) \leq d(i) + a_{ij} \text{ for each } (i, j) \in A. \quad (3)$$

Further, the condition (3) is satisfied with equality for every arc (i, j) present in a shortest path from node q to some node j .

Minimum Cost Flow Optimality Conditions

Optimal solutions of the minimum cost flow problem can be characterized in a variety of ways. The cycle-canceling algorithm uses the following well known characterization.

Theorem 1. Negative Cycle Optimality Condition. *A feasible solution x^* is an optimal solution of the minimum cost flow problem if and only if the residual network $G^*(x)$ contains no negative cost directed cycle.*

Cycle-Canceling Algorithm

The negative cycle optimality condition suggests one simple algorithmic approach for solving the minimum cost flow problem known as the *cycle-canceling algorithm*. The algorithm first establishes a feasible flow x in the network G , for example, by solving a maximum flow problem. Then, it iteratively finds negative cost directed cycles in the residual network and augments flows in these cycles. The algorithm terminates when the residual network contains no negative cost directed cycle. Theorem 1 implies that when the algorithm terminates, it has found a minimum cost flow. Figure 2 specifies the generic version of the cycle-canceling algorithm.

```

algorithm cycle-canceling;
begin
    establish a feasible flow  $x$  in the network;
    while  $G(x)$  contains a negative cycle do
        begin
            use some algorithm to identify a negative cycle  $W$ ;
             $\delta := \min\{r_{ij} : (i, j) \in W\}$ ;
            augment  $\delta$  units of flow in the cycle  $W$  and update  $G(x)$ ;
        end;
    end;

```

Figure 2. Description of the cycle-canceling algorithm.

3. IDENTIFYING VALID CYCLES

Our scaling cycle-canceling algorithm proceeds by sending flows along negative cycles with "sufficiently large" residual capacity and containing a specified arc. In this section, we describe how we can identify such a cycle by solving a shortest path problem with nonnegative arc lengths in $O(m + n \log n)$ time.

First we define some notation which are defined with respect to a flow x , a set of node potential π , and the scaling parameter Δ .

Δ -Residual Network. For a given parameter Δ , the Δ -residual network, denoted by $G(x, \Delta)$, is defined as a subgraph of $G(x)$ which contains all the arcs whose residual capacities are greater than or equal to Δ .

Eligible Arc. We call an arc (i, j) in the residual network an *eligible arc* if $c_{ij}^\pi < 0$. An arc in the residual network, which is not eligible, is called an *ineligible arc*.

Δ -Eligible Arc. An eligible arc (i, j) is said to be a Δ -eligible arc if $r_{ij} \geq \Delta$.

Valid Cycle. A negative cost directed cycle in $G(x, \Delta)$ is called a *valid cycle*.

The procedure *find-valid-cycle* $((p, q), found, W)$ lies at the heart of the scaling cycle-canceling algorithm. This procedure, whose algorithmic description is given in Figure 3, takes as an input a Δ -eligible arc (p, q) and attempts to find a valid cycle which contains the arc (p, q) as a forward arc. If it succeeds, then *found* is set to true and *W* is set to be one such cycle; otherwise, *found* is set to false and *W* is undefined. In the latter case, the procedure updates the node potentials so that arc (p, q) becomes an ineligible arc.

procedure *find-valid-cycle* $((p, q), found, W)$;
begin

found := false ;
determine the set *S* of nodes reachable from node *q* in $G(x, \Delta)$ through directed paths;
if $p \notin S$ **then** define

$$\pi'(j) = \begin{cases} \pi(j) + |c_{pq}^\pi| & \text{for all } j \in S \\ \pi(j) & \text{for all } j \in \bar{S} \end{cases} \quad (4)$$

else
begin

determine the shortest path distance $d(i)$ and a shortest path $P[i]$ from node *q* to each node $i \in S$ in $G(x, \Delta) \setminus \{(q, p)\}$ with $a_{ij} = \max\{0, c_{ij}^\pi\}$ as the length of each arc (i, j) in $G(x, \Delta)$;

set $d_{\max} := \max \{d(j) : j \in S\}$ and define

$$\pi'(j) = \begin{cases} \pi(j) + d_{\max} - d(j) & \text{for all } j \in S \\ \pi(j) & \text{for all } j \in \bar{S} \end{cases} \quad (5)$$

if $c_{pq}^{\pi'} < 0$ **then** *found* := true and $W := P[p] \cup \{(p, q)\}$

end ;
 $\pi := \pi'$;

end ;

Figure 3. Description of the procedure *find-valid-cycle*.

We will now prove some results about the procedure find-valid-cycle, which will be used in the next section.

Lemma 1. *The procedure find-valid-cycle does not create any new eligible arc.*

Proof. The procedure find-valid-cycle changes potentials of nodes only in S ; this operation may change the reduced costs of only those arcs which have one endpoint in S or both endpoints in S . We consider these two cases separately.

Case 1. Arc (i, j) in $G(x, \Delta)$ has only one endpoint in S . By the definition of the set S , no node in $\bar{S}$ is reachable from any node in S , therefore $i \in \bar{S}$ and $j \in S$. Since the reduced cost of arc (i, j) is defined by $c_{ij}^\pi = c_{ij} - \pi(i) + \pi(j)$, and $\pi(j)$ does not decrease whether it is modified using (4) or (5), it follows that $c_{ij}^{\pi'} \geq c_{ij}^\pi$, and no new eligible arc will be created.

Case 2. Arc (i, j) in $G(x, \Delta)$ has both endpoints in S . Notice that in this case, if node potentials are modified using (4), then $c_{ij}^{\pi'} = c_{ij}^\pi$, and the lemma clearly holds. Now consider the case when potential are modified using (5). It follows from (5) and the definition of reduced costs that

$$c_{ij}^{\pi'} + d(j) - d(i) = c_{ij}^\pi. \quad (6)$$

Since $d(j)$'s are shortest path distances with respect to $\max\{0, c_{ij}^\pi\}$ as arc lengths, it follows from (3) that $d(j) - d(i) \leq \max\{0, c_{ij}^\pi\}$. Substituting this in (6) yields

$$c_{ij}^{\pi'} + \max\{0, c_{ij}^\pi\} \geq c_{ij}^\pi. \quad (7)$$

Now observe from (7) that in the case when $\max\{0, c_{ij}^\pi\} = 0$, then $c_{ij}^{\pi'} \geq c_{ij}^\pi$, and in the case when $\max\{0, c_{ij}^\pi\} = c_{ij}^\pi$ then $c_{ij}^{\pi'} \geq 0$; and in either case no new eligible arc will be created. This completes the proof of the lemma. ♦

Lemma 2. *If the procedure find-valid-cycle indeed finds the cycle $W = P[p] \cup \{(p, q)\}$, then it is a valid cycle.*

Proof. Consider any arc (i, j) in the path $P[p]$. Since $P[p]$ is a shortest path from node q to node p in $G(x, \Delta)$, it follows from the shortest path optimality conditions (3) that $d(j) = d(i) + \max\{0, c_{ij}^\pi\}$, or $d(j) \geq d(i) + c_{ij}^\pi$. Substituting $c_{ij}^\pi = c_{ij} - \pi(i) + \pi(j)$ in the preceding inequality yields $c_{ij}^{\pi'} \leq 0$. Next notice that $c_{pq}^{\pi'} < 0$. Therefore, $\sum_{(i,j) \in W} c_{ij}^{\pi'} = \sum_{(i,j) \in P[p]} c_{ij}^{\pi'} + c_{pq}^{\pi'} < 0$. Finally, we use Property 1 to conclude that $\sum_{(i,j) \in W} c_{ij} = \sum_{(i,j) \in W} c_{ij}^{\pi'} < 0$, thereby proving that W is a valid cycle. ♦

Lemma 3. *An execution of the procedure find-valid-cycle either finds a valid cycle W in $G(x, \Delta)$ containing arc (p, q) as a forward arc or makes the arc (p, q) ineligible. Further, an execution of the procedure find-valid-cycle takes $O(m + n \log n)$ time.*

Proof. When the procedure is executed, then either $p \notin S$ or $p \in S$. In the former case, it follows from (4) that $c_{pq}^{\pi'} = c_{pq}^{\pi} - |c_{pq}^{\pi}| = 0$, and arc (p, q) becomes ineligible. In the latter case, when the procedure modifies node potentials using (5), then either (i) $c_{pq}^{\pi'} \geq 0$, in which case arc (p, q) becomes ineligible, or (ii) $c_{pq}^{\pi'} < 0$, in which case we have shown in Lemma 2 that $W = P[p] \cup \{(p, q)\}$ is a valid cycle containing arc (p, q) as a forward arc. This completes the proof of the first statement in the lemma. Next observe that the bottleneck operation in the procedure is solving a shortest path problem with nonnegative arc lengths, and Fredman and Tarjan's [1984] implementation of Dijkstra's algorithm can do it in $O(m + n \log n)$ time, completing the proof of the second statement in the lemma. ♦

4. SCALING CYCLE-CANCELING ALGORITHM

In this section, we describe the scaling cycle-canceling algorithm, and prove that it solves the minimum cost flow problem by canceling $O(m \log(nU))$ negative cycles and takes a total of $O(m(m + n \log n) \log(nU))$ time. Figure 4 gives a description of the scaling cycle-canceling algorithm.

```

algorithm scaling-cycle-canceling;
begin
    establish a feasible (integral) flow  $x$  in  $G$ ;
    set  $\pi := 0$ ;
     $\Delta := 2^{\lfloor \log(2mU) \rfloor}$ ;
    while  $\Delta \geq 1$  do
        begin {scaling phase}
            cancel-valid-cycles;
             $\Delta := \Delta/2$ ;
        end; {scaling phase}
    end;

```

(a)

```

procedure cancel-valid-cycles;
begin
  while  $G(x)$  contains a  $\Delta$ -eligible arc do
    begin
      select a  $\Delta$ -eligible arc  $(p, q)$ ;
      find-valid-cycle $((p, q), found, W)$ ;
      if  $found = \text{true}$  then
        begin
           $\delta := \min \{r_{ij} : (i, j) \in W\}$ ;
          augment  $\delta$  units of flow along  $W$  and update  $x$ ;
        end;
      end;
    end;
end;

```

(b)

Figure 4. Description of the scaling cycle-canceling algorithm.

The capacity cycle-canceling algorithm performs a number of scaling phases, called Δ -scaling phases. The Δ -scaling phase consists of calling the procedure *cancel-valid-cycles* and adjusting the value of Δ . We first study some properties of the procedure *cancel-valid-cycles*.

Lemma 4. *In the Δ -scaling phase, the procedure *cancel-valid-cycles* satisfies the following properties:*

- (a) *At the beginning of the procedure, $r_{kl} \leq 2\Delta$ for each eligible arc (k, l) .*
- (b) *In each iteration, the number of Δ -eligible arcs strictly decrease, and the residual capacities of eligible arcs do not increase.*
- (c) *At the termination of the procedure, $r_{kl} < \Delta$ for each eligible arc (k, l) .*

Proof. First we consider part (a) of the lemma. In the first scaling phase, $mU \leq \Delta \leq 2mU$. But since the largest arc capacity is at most $2mU$, part (a) holds. A scaling phase terminates when there is no Δ -eligible arc. At this point, replacing Δ by $\Delta/2$ preserves the condition in part (a).

To establish part (b), recall the proof of Lemma 2 where we showed that each arc (i, j) in the augmenting cycle W satisfies $c_{ij}^\pi \leq 0$. The augmentation decreases the residual capacities of arcs in W (which satisfy $c_{ij}^\pi \leq 0$) and increases the residual capacities of reversals of arcs in W (which satisfy $c_{ij}^\pi \geq 0$). Thus, the residual capacities of eligible arcs do not increase. Next consider an execution of the procedure *find-valid-cycle* which takes in a Δ -eligible arc (p, q) and either modifies the node potentials so that arc (p, q) becomes ineligible, or augments at least Δ units of flow on arc (p, q) so that $r_{pq} \leq \Delta$ and it again becomes ineligible. This completes the proof of part (b) of the lemma.

Finally, we consider part (c). This part follows directly from the fact that the procedure terminates when there is no Δ -eligible arcs, that is, $r_{kl} \leq \Delta$ for each eligible arc (k, l) . ♦

We now study the correctness of the scaling cycle-canceling algorithm. After the last scaling phase executed by the algorithm, $\Delta < 1$ and the integrality of residual capacities together with Lemma 4(c) implies that the residual network $G(x)$ contains no eligible arcs. In other words, $c_{ij}^\pi \geq 0$ for each arc (i, j) in the residual network. Consequently, for each directed cycle W in $G(x)$, $\sum_{(i,j) \in W} c_{ij}^\pi = \sum_{(i,j) \in W} c_{ij}^\pi \geq 0$. It follows from Theorem 1 that x is an optimal solution of the minimum cost flow problem.

We next study the time-complexity of the scaling cycle-canceling algorithm. A direct consequence of Lemma 4(b) is that during the Δ -scaling phase, at most m cycles are canceled and, therefore, each scaling phase can be implemented to run in $O(m(m+n \log n))$ time. We now count the number of scaling phases. Since (i) initially $\Delta \leq 2mU$, (ii) finally $\Delta < 1$, and (iii) each scaling phase reduces Δ by a factor of 2, it follows that there will be $O(\log(mU)) = O(\log(nU))$ scaling phases. We have thus established the following theorem.

Theorem 2. *The scaling cycle-canceling algorithm solves the minimum cost flow problem in $O(m(m+n \log n) \log(nU))$ time.*

5. IMPLEMENTATION FOR UNCAPACITATED NETWORKS

In this section, we show that a slight modification of the scaling cycle-canceling algorithm, when applied to uncapacitated networks, can be sped to obtain a time bound of $O(n(m + n \log n) \log(nU))$, where U is the largest supply/demand of any node. This is comparable to the running time of the well known capacity scaling algorithm for the uncapacitated minimum cost flow problem, of which the transportation problem is a special case. The modification we make in the scaling algorithm is that in the Δ -scaling phase, we send exactly Δ units of flow in the valid cycle instead of sending a maximum possible flow, which could be greater than Δ .

Let x be a basic feasible solution of the uncapacitated minimum cost flow problem. Associated with the solution x , there is a spanning tree T such that arcs in T carry a nonnegative flow and arcs not in T (denoted by the set $\bar{T}$) carry zero flow. The residual network $G(x)$ maintained by the algorithm will have arcs corresponding to T (that is, some arcs in T and some reversals of arcs in T) and will have arcs corresponding to $\bar{T}$. We denote by $T(x)$ the set of arcs in $G(x)$ associated with arcs in T , and denote by $\bar{T}(x)$ the set of arcs in $G(x)$ associated with arcs in $\bar{T}$. Our improvement relies on the following result.

Lemma 5. *In the entire execution of the algorithm, the residual capacity of each arc in $\bar{T}(x)$ is an integral multiple of Δ .*

Proof. We can prove this result by performing induction on the number of augmentations and scaling phases. At the beginning of the first scaling phase, $r_{ij} = \infty$ for each arc $(i, j) \in \bar{T}$ and no arc (j, i) with $(i, j) \in \bar{T}$ is present in $\bar{T}(x)$ because $x_{ij} = 0$. Hence the induction hypothesis is initially true. Subsequently, each augmentation carries exactly Δ units of flow and preserves the induction hypothesis. Replacing Δ by $\Delta/2$ at the beginning of the scaling phase also preserves the induction hypothesis. ♦

A byproduct of Lemma 5 is that at the beginning of the Δ -scaling phase, the residual capacity of each arc in $\bar{T}$ is an integral multiple of 2Δ . It follows from Lemma 4(a) that no arc in $\bar{T}(x)$ can be eligible. Consequently, each eligible arc must belong to $T(x)$. Next notice that if both the arcs (i, j) and (j, i) are present in $T(x)$ then at most one of them can be eligible. These observations imply that at the beginning of any scaling phase, there will be at most $(n-1)$ eligible arcs, and the scaling phase will perform at most $(n-1)$ augmentations along valid cycles. Hence the following theorem.

Theorem 3. *The scaling cycle-canceling algorithm solves the uncapacitated minimum cost flow problem in $O(n(m + n \log n) \log(nU))$ time.*

6. IMPLEMENTATION FOR CONVEX COST NETWORKS

In this section, we show that the scaling cycle-canceling algorithm described in Section 4 can be used to obtain an integer optimal solution of the convex cost flow problem without worsening its worst-case time bound. The convex cost flow problem we consider is to

$$\text{Minimize } \sum_{(i,j) \in A} C_{ij}(x_{ij})$$

subject to (1.b) and (1.c), where $C_{ij}(x_{ij})$ is a convex function of x_{ij} for each $(i, j) \in A$. Since we are interested in finding an integer optimal solution of the convex cost flow problem, we can assume without loss of generality that each $C_{ij}(x_{ij})$ is a piecewise linear convex function of x_{ij} whose breakpoints (that is, the points where the slopes of the linear segments change) are all integer. As earlier, we assume that the capacity of each arc is finite. Our cycle-canceling algorithm for the convex cost flow problem uses the same scaling technique as described in Chapter 14 of Ahuja, Magnanti and Orlin [1993].

As in Section 4, our scaling algorithm for the convex cost flow problem, performs a number of scaling phases. In the Δ -scaling phase, we approximate the cost function $C_{ij}(x_{ij})$ using linear segments of size Δ . In other words, we define the arc cost function $C_{ij}^\Delta(x_{ij}) = C_{ij}(x_{ij})$ whenever x_{ij} is an integer multiple of Δ and $C_{ij}^\Delta(x_{ij})$ is linear between multiple of Δ . We refer to this problem as the *Δ -scaled problem*. The Δ -scaled problem for the convex cost flow problem differs from the minimum cost flow problem (1) in the sense that the cost of flow on each arc is a piecewise linear convex function instead of a linear function. We now use a well known result that a mathematical programming problem with

piecewise linear convex cost functions and linear constraints can be transformed to a linear programming problem by introducing a separate variable for each linear segment. This result implies that the Δ -scaled problem for the convex cost flow problem can be transformed into a minimum cost flow problem by introducing a separate arc for each linear segment.

Since in the Δ -scaled problem, each arc capacity is an integral multiple of Δ , when we use the cycle-canceling algorithm to solve this problem, each arc flow is an integral multiple of Δ . For such a flow x , we define the residual network $G(x, \Delta)$ in the following manner. For any arc $(i, j) \in A$ with $x_{ij} + \Delta \leq u_{ij}$, the Δ -residual network contains the arc (i, j) with the residual capacity Δ and cost equal to $(C_{ij}(x_{ij} + \Delta) - C_{ij}(x_{ij}))/\Delta$. For any arc $(i, j) \in A$ with $x_{ij} \geq \Delta$, the Δ -residual network contains the arc (j, i) with a residual capacity of Δ and cost equal to $(C_{ij}(x_{ij} - \Delta) - C_{ij}(x_{ij}))/\Delta$.

In the first scaling phase, the Δ -residual network will contain at most $2m$ arcs of which at most m arcs will be eligible. Consequently, the procedure cancel-valid-cycles will cancel all valid cycles within m augmentations. In the next scaling phase, Δ is reduced by a factor of 2, which implies that we linearize $C_{ij}(x_{ij})$ by segments of size $\Delta/2$. When we move from the Δ -scaling phase to the $\Delta/2$ -scaling phase, then arc costs change. As a result, the reduced costs of arcs also change and some new eligible arcs might be created in the $\Delta/2$ -residual network. There will be at most m such eligible arcs and the procedure cancel-valid-cycle will make them ineligible within at most m augmentations. In the last scaling phase, $\Delta = 1$ and the Δ -scaled problem in this scaling phase is the same as the original integer convex cost flow problem. We summarize our discussion in this section by the following theorem.

Theorem 4. *The scaling cycle-canceling algorithm obtains an integer optimal flow of the convex cost flow problem in $O(m(m + n \log n) \log(nU))$ time.*

7. A STRONGLY-POLYNOMIAL IMPLEMENTATION

In this section, we describe a strongly polynomial implementation of the scaling cycle-canceling algorithm described in Section 4. We show that a modest variant of the scaling cycle-canceling algorithm can be implemented to run in $O(m(m+n \log n) \min(\log(nU), m \log n))$ time.

To simplify our presentation of the strongly-polynomial algorithm, we shall withdraw Assumption 3, and shall instead make the following two assumptions:

Assumption 4. *The capacity of each arc in the network $G = (N, A)$ is infinite.*

Assumption 5. *For any feasible flow x in the network G , the residual network $G(x)$ does not contain any zero-cost directed cycle except trivial cycles (a trivial cycle is an arc (i, j) plus its reversal (j, i)).*

Both the above assumptions can be made without any loss of generality. It is well known that a minimum cost flow problem in a capacitated network can be transformed into a minimum cost flow problem in an uncapacitated network (see, for example, Ahuja, Magnanti and Orlin [1993]). This transformation consists of dropping a node in the middle of each capacitated arc, and thus increases the size of the network. If n' and m' denote the number of nodes and arcs in the transformed network G' , then $n' \leq n+m$ and $m' \leq 2m$. If we first do this transformation and then apply the scaling cycle-canceling algorithm, then the running time of the algorithm will be $O(m'(m'+n' \log n') \log(n'U)) = O(m(m'+n' \log n') \log(nU))$. However, Orlin [1988] describes an implementation of Dijkstra's algorithm which allows the shortest path problem in G' to be solved in $O(m + n \log n)$ time. Using this implementation of Dijkstra's algorithm, the running time of the scaling cycle-canceling algorithm for G' remains $O(m(m+n \log n) \log(nU))$. We thus find that the transformation does not affect the worst-case complexity of the scaling cycle-canceling algorithm. We will henceforth assume that the transformation has been carried out before applying the cycle-canceling algorithm and n and m respectively denote the number of nodes and arcs in the transformed network.

Assumption 5 requires that the network G does not contain any zero cost directed cycles except trivial cycles. This assumption can be satisfied by a perturbation of arc costs, which is algorithmically equivalent to using lexicography.

We now need some additional notations, which we define next. These are defined with respect to a given flow x and a set of node potential π .

Admissible Arc. We call an arc (i, j) in the residual network an *admissible arc* if $c_{ij}^\pi \leq 0$.

Admissible Cycle. We call a directed cycle W in the residual network an *admissible cycle* if each arc in W is admissible.

We are now in a position to describe the strongly-polynomial variant of the capacity cycle-canceling algorithm, which we call the *improved capacity cycle-canceling algorithm*. As earlier, the improved algorithm performs a number of scaling phases. However, the Δ -scaling phase consists of two parts. In the first part, the algorithm calls the procedure cancel-valid-cycles which cancels all the valid cycles. The procedure cancel-valid-cycles is the same as in the capacity cycle-canceling algorithm. In the second part, the algorithm cancels all the admissible cycles by calling the procedure cancel-admissible-cycles. After the second part is over, the algorithm redefines the scaling parameter Δ and starts a new scaling phase. Instead of dividing Δ by a factor of 2, we redefine it as the largest arc capacity of eligible arcs divided by 2. This change allows us to occasionally reduce Δ by a factor larger than two. Figure 5 describes the improved capacity cycle-canceling algorithm.


```

algorithm improved-scaling-cycle-canceling;
begin
    establish a feasible (integral) flow  $x$  in  $G$ ;
    set  $\pi := 0$ ;
    set  $\Delta := mU$ ;
    while  $\Delta \geq 1$  do
        begin {scaling phase}
            cancel-valid-cycles;
            cancel-admissible-cycles;
             $r_{\max} := \max\{r_{ij} : (i, j) \text{ is eligible}\}$ ;
             $\Delta := r_{\max}/2$ ;
        end; {scaling phase}
    end;

```

(a)

```

procedure cancel-admissible-cycles ;
begin
    while  $G(x)$  contains an admissible cycle do
        begin
            select an admissible cycle  $W$ ;
             $\delta := \min \{r_{ij} : (i, j) \in W\}$ ;
            augment  $\delta$  units of flow along  $W$ ;
        end;
    end;

```

(b)

Figure 5. Description of the improved scaling cycle-canceling algorithm.

It is easy to see that the improved scaling cycle-canceling algorithm will continue to satisfy Lemma 4. We will use the following additional lemma to analyze the worst-case running time of the algorithm.

Lemma 6. *In the Δ -scaling phase, the procedure *cancel-admissible-cycles* satisfies the following properties:*

- (a) *Each admissible cycle identified by it is a negative cost cycle.*
- (b) *Each eligible arc (i, j) in the residual network satisfies $r_{ij} < \Delta$.*
- (c) *An augmentation along an admissible cycle carries at most Δ units of flow and strictly reduces the number of admissible arcs.*

Proof. Let W be an admissible cycle identified by the procedure. Since each arc in W is admissible, we have $\sum_{(i,j) \in W} c_{ij} = \sum_{(i,j) \in W} c_{ij}^{\pi} \leq 0$. Assumption 5 rules out zero cost cycles. Thus, $\sum_{(i,j) \in W} c_{ij} < 0$, establishing part (a) of the lemma.

Next consider part (b) of the lemma. The procedure cancel-admissible-cycles is applied when the procedure cancel-valid-cycles terminates, and it happens when $r_{ij} < \Delta$ for each eligible arc (i, j) . Since augmentation of flow on admissible arcs does not increase the residual capacity of any eligible arc, $r_{ij} < \Delta$ for each eligible arc (i, j) throughout the procedure. This establishes part (b) of the lemma.

Part (c) follows from the observation that sending flow around W reduces the residual capacity of one of the admissible arcs to zero. ♦

It follows from Lemma 6(a) that the procedure cancel-admissible-cycles also cancels negative cost cycles; therefore, the improved scaling cycle-canceling algorithm is a cycle-canceling algorithm. Identifying an admissible cycle consists of determining whether the set of admissible arcs contains a directed cycle or not; this can be done in $O(m)$ time using any topological sorting algorithm. It follows from Lemma 6(c) that the procedure cancel-admissible-cycles identifies at most m admissible cycles. Consequently, each execution of this procedure require $O(m^2)$ time, which is dominated by the time required to execute the procedure cancel-valid-cycles.

We next count the number of scaling phases performed by the improved scaling cycle-canceling algorithm. At the beginning of the procedure cancel-valid-cycles, $r_{kl} \leq 2\Delta$ for each eligible arc (k, l) , and at the termination of the procedure, $r_{kl} \leq \Delta$ for each eligible arc (k, l) . Thus the new value of Δ , say Δ' , which is half of the largest residual capacity among eligible arcs satisfies $\Delta' \leq \Delta/2$. Consequently, there will be $O(\log(nU))$ scaling phases, which is a weakly-polynomial bound. We will now obtain a strongly-polynomial bound on the number of scaling phases and show that there will be at most $O(m \log n)$ such phases. We need some more notations.

Abundant Arc. An arc $(i, j) \in A$ is said to be an *abundant arc* with respect to a flow x if $x_{ij} \geq 8m\Delta$.

Permanently Ineligible Arc. An arc (i, j) in the residual network is said to be a *permanently ineligible arc* if in all the subsequent steps of the algorithm arc (i, j) will be ineligible.

The following lemma establishes a connection between the above two types of arcs.

Lemma 7. *If $(i, j) \in A$ is an abundant arc, then both the arcs (i, j) and (j, i) are permanently ineligible.*

Proof. Suppose that the arc $(i, j) \in A$ is found to be an abundant arc in the Δ -scaling phase. Hence, $x_{ij} \geq 8m\Delta$. In the Δ -scaling phase, both the procedures cancel-valid-cycles and cancel-admissible-cycles perform at most m augmentations each, and each augmentation carries at most 2Δ units of flow. Therefore, at the end of the Δ -scaling phase, $x_{ij} \geq 4m\Delta$. In the next scaling phase, the new scaling parameter $\Delta' \leq \Delta/2$ and $x_{ij} \geq 8m\Delta'$, thus arc (i, j) is again an abundant arc. We have shown that for an arc (i, j) which is found to be abundant at some stage, $x_{ij} \geq 4m\Delta$ throughout the rest of the algorithm.

For such an arc $(i, j) \in A$, $r_{ij} = \infty$ and $r_{ji} \geq 4m\Delta$. Since the algorithm never allows eligible arcs to have residual capacities more than 2Δ (see Lemmas 4 and 6), it follows that both the arcs (i, j) and (j, i) will be permanently ineligible. ♦

The basic idea used in obtaining the strongly polynomial bound on the algorithm is to show that the algorithm will discover a new abundant arc (i, j) within $(3 + 2 \log n)$ scaling phases. For an abundant arc (i, j) , both (i, j) and (j, i) will be permanently ineligible in the residual network. Thus, after we have discovered at most m abundant arcs, there will be no eligible arc and the algorithm will terminate with an optimal solution of the minimum cost flow problem. Consequently, the algorithm will perform $O(m \log n)$ scaling phases.

Let us consider the situation at the termination of the procedure cancel-admissible-cycles. Let x denote the flow and π denote the node potentials at this stage. At this point, we reset $\Delta' := r_{\max}/2$, where $r_{\max} = \{r_{\max} : (i, j) \text{ in } G(x) \text{ and is eligible}\}$. Let arc (p, q) be the arc that attains the maximum in the computation of $r_{\max}$. Further, let S denote the set of all nodes reachable from node q in $G(x)$ through directed admissible paths (that is, through directed paths consisting solely of admissible arcs). Since $G(x)$ contains no admissible cycles, it follows that $p \in \bar{S}$. Thus $[S, \bar{S}]$ is a cut. Let $(S, \bar{S})$ and $(\bar{S}, S)$ denote the sets of forward and backward arcs in A in the cut $[S, \bar{S}]$.

Lemma 8. *The cut $[S, \bar{S}]$ satisfies the following properties:*

- (a) $x_{ij} = 0$ for each arc $(i, j) \in (\bar{S}, S)$.
- (b) $x_{ij} \leq \Delta$ for each arc $(i, j) \in (S, \bar{S})$.

Proof. We first prove part (a) of the lemma. Suppose that $x_{ij} > 0$ for some arc $(i, j) \in (\bar{S}, S)$. Notice that $r_{ij} = \infty$, and therefore $c_{ij}^\pi \geq 0$ (from Lemma 6(b)). Also notice that arc (j, i) is in $G(x)$ and $c_{ji}^\pi = -c_{ij}^\pi \leq 0$. Thus, (j, i) is an admissible arc in $G(x)$ and node i must have been in S , giving a contradiction. Now consider part (b) of the lemma. Suppose that $x_{ij} > \Delta$ for some arc $(i, j) \in (S, \bar{S})$. Then, both the arcs (i, j) and (j, i) are present in $G(x)$, and it follows from Lemma 6(b) that both these arcs must be ineligible. This is possible only if $c_{ij}^\pi = -c_{ji}^\pi = 0$. Thus (i, j) is an admissible arc in $G(x)$ and node j must be in S , again giving a contradiction. The proof of the lemma is complete. ♦

Lemma 8 shows that the set $(S, \bar{S})$ of arcs does not contain any abundant arc. We will show that after at most $(2 \log n + 3)$ scaling phases, $(S, \bar{S})$ will contain at least one abundant arc. We first observe that $(q, p) \in (S, \bar{S})$ and $x_{qp} = r_{pq} = \Delta'$ (because arc (p, q) is eligible and from Lemma 6(b) no arc $(i, j) \in A$ can become eligible because $r_{ij} = \infty$). We now consider the equation (2) and substitute (i) $x_{qp} = \Delta'$, (ii) $x_{ij} \geq 0$ for each arc $(i, j) \in (S, \bar{S})$, and (iii) $x_{ij} = 0$ for each arc $(i, j) \in (\bar{S}, S)$, which gives $b(S) \geq \Delta'$. Next

observe from (2) that in any feasible flow x' , there will be at least one arc $(i, j) \in (S, \bar{S})$ carrying flow $x'_{ij} \geq b(S)/m \geq \Delta'/m$. Within $(2 \log m + 3)$ scaling phases, the value of the scaling parameter $\Delta'' \leq \Delta'/2^{(2 \log m + 3)} = \Delta'/8m^2$, and at least one arc $(i, j) \in (S, \bar{S})$ will carry flow $x'_{ij} \geq \Delta'/m \geq 8m\Delta''$, implying that arc (i, j) will be an abundant arc. We have thus proved the following result.

Lemma 9. *Within $(2 \log n + 3)$ scaling phases, the improved scaling cycle-canceling algorithm either obtains an optimal solution or discovers a new abundant arc.*

In the previous section, we obtained a bound of $O(\log(nU))$ on the number of scaling phases, which also holds for the improved scaling cycle-canceling algorithm. Lemma 8 yields a bound of $O(m \log n)$ on the number of scaling phases. In a scaling phase, the procedure cancel-eligible-cycles requires $O(m(m+n \log n))$ time and the procedure cancel-admissible-cycles requires $O(m^2)$ time. Consequently, the total running time of the algorithm is $O(m(m+n \log n) \min\{\log(nU), m \log n\})$. We state this result as a theorem.

Theorem 5. *The improved scaling cycle-canceling algorithm solves the minimum cost flow problem in $O(m(m+n \log n) \min\{\log(nU), m \log n\})$ time.*

ACKNOWLEDGEMENTS

This research was supported by the Office of Naval Research under Contract No. N00014-96-1-0051, as well as a grant from the United Parcel Service.

REFERENCES

- Ahuja, R. K., T. L. Magnanti, and J. B. Orlin. 1993. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, Englewood Cliff, NJ.
- Barahona, F., and E. Tardos. 1989. Note on Weintraub's minimum cost circulation algorithm. *SIAM Journal on Computing* 18, 579-583.
- Fredman, M. L., and R. E. Tarjan. 1984. Fibonacci heaps and their uses in improved network optimization algorithms. *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science*, pp. 338-346. Full paper in *Journal of ACM* 34 (1987), 596-615.
- Goldberg, A. V., and R. E. Tarjan. 1988. Finding minimum-cost circulation by canceling negative cycles. *Proceedings of the 20th ACM Symposium on the Theory of Computing*, pp. 388-397. Full paper in *Journal of ACM* 36 (1989), 873-886.
- Klein, M. 1967. A primal method for minimum cost flows with application to the assignment and transportation problem. *Management Science* 14, 205-220.
- Orlin, J. B. 1988. A faster strongly polynomial minimum cost flow algorithm. *Proceedings of the 20th ACM Symposium on the Theory of Computing*, pp. 377-387. Full paper in *Operations Research* 41(1993) 338-350.