FINDING MINIMUM-COST CIRCULATIONS BY CANCELING NEGATIVE CYCLES

Andrew V. Goldberg
Robert E. Tarjan

CS-TR-107-87

July 1987

Finding Minimum-Cost Circulations by Canceling Negative Cycles

Andrew V. Goldberg[*†]
Laboratory for Computer Science
M.I.T.
Cambridge MA 02139

Robert E. Tarjan[‡]
Department of Computer Science
Princeton University
Princeton, New Jersey 08544
and
AT&T Bell Laboratories
Murray Hill, New Jersey 07974

July 1987

*Author's current address is Department of Computer Science, Stanford University, Stanford, CA 94305.

†Supported by the Advanced Research Projects Agency of the Department of Defense, Contract No. N00014-80-C-0622.

‡Partially supported by the National Science Foundation, Grant No. DCR-8605962, and by the office of Naval Research, Contract No. N00014-87-K-0467.

Abstract

A classical algorithm for finding a minimum-cost circulation consists of repeatedly finding a residual cycle of negative cost and *canceling* it by pushing enough flow around the cycle to saturate an arc. We show that a judicious choice of cycles for canceling leads to a polynomial bound on the number of iterations in this algorithm. This gives a very simple strongly polynomial algorithm that uses no scaling. A variant of the algorithm that uses dynamic trees runs in $O(nm(\log n)\min\{\log(nC), m\log n\})$ time on a network of n vertices, m arcs, and arc costs of maximum absolute value C. This bound is comparable to those of the fastest previously known algorithms.

1 Introduction

The *minimum-cost circulation problem* is that of finding a circulation of minimum cost in a network whose arcs have flow capacities and costs per unit of flow. This problem is equivalent to the minimum-cost flow problem and to the transshipment problem, and it has a variety of applications [7,18,20].

The minimum-cost circulation problem has a rich history; a series of faster and faster algorithms for it have been devised. A discussion of many of these algorithms can be found in our earlier paper [13]; we summarize some of the previous results here.

The important parameters by which to measure the running time of an algorithm are n, the number of vertices in the network; m, the number of arcs; U, the maximum absolute value of an arc capacity; and C, the maximum absolute value of an arc cost. For ease in starting time bounds, we assume $m \geq n \geq 2$. In bounds containing U or C, the capacities or costs, respectively, are assumed to be integers.

All known polynomial-time algorithms for the problem use the idea of *scaling* or *successive approximation.* These algorithms compute closer and closer approximations to an optimal solution. The idea of scaling is due to Edmonds and Karp [6], who used this idea to devise the first polynomial-time algorithm for the problem. Their algorithm uses capacity-scaling and has a running time of $O(m(\log U)(m + n\log n))$ if the fastest known single-source, nonnegative-cost shortest path algorithm [8] is used. Röck [21] exhibited a simpler capacity-scaling algorithm achieving the same time bound and also presented a cost-scaling algorithm with a running time of $O(n(\log C)\min\{nm\log(n^2/m),\ n^2\log U + nm\})$ if the fastest known maximum flow algorithm [1,14] is used. The latter bound was also obtained later by Bland and Jensen [3] using a somewhat different cost-scaling approach. A generalization of the cost-scaling approach has been proposed in [12,13,15].

These results left open the question of whether there is a strongly polynomial algorithm

for the problem. A strongly polynomial algorithm is one with a time bound polynomial in n and m if arithmetic operations take unit time, and with a time bound polynomial in $n, m, \log U$, and $\log C$ if arithmetic operations take time polynomial in the number of bits needed to represent the operands. Tardos [24] was the first to devise a strongly polynomial algorithm for the problem. Other strongly polynomial algorithms were later proposed by Orlin [19] and Fujishige [9]. The fastest known strongly polynomial algorithm is that of Galil and Tardos [11], which runs in $O(n^2(\log n)\ (m + n \log n))$ time.

For networks having integer arc costs that are not huge, the asymptotically fastest algorithm is the one previously proposed by us [13], which runs in $O(nm(\log(n^2/m))$ $\min\{\log(nC), m \log n\})$ time.

All the known polynomial-time algorithms for the problem, whether they are strongly polynomial or not, are somewhat elaborate. Our purpose in this paper is to show that a simple, classical algorithm for the problem becomes strongly polynomial if a careful choice is made among possible iterative steps. The algorithm we analyze was proposed by Klein [17]. We call it the *cycle-canceling algorithm.* This algorithm consists of repeatedly finding a residual cycle of negative cost and sending as much flow as possible around the cycle. Our rule for choosing cycles is to always select a cycle whose average arc cost is smallest. Such a cycle is called a *minimum-mean cycle.* A minimum-mean cycle can be found in $O(nm)$ time using an algorithm of Karp [16]. The cycle-canceling algorithm with minimum-mean cycle selection runs in $O(nm \min\{\log(nC), m \log n\})$ iterations and $O(n^2m^2 \min\{\log(nC), m \log n\})$ time. This algorithm is primal (i.e. maintains a feasible circulation), very simple, and does no scaling. The algorithm itself is described in Section 2 and analyzed in Section 3.

Although this algorithm seems to be of mostly theoretical interest, it has a variant that is more efficient. The variant combines more flexible cycle selection with the use of a sophisticated data structure for representing dynamic trees [22,23,25]. It has an asymptotic running time of $O(nm(\log n) \min\{\log(nC), m \log n\})$, which is competitive with the fastest previously known algorithms, at least on non-dense networks with arc costs that are not huge. This algorithm and its analysis are presented in Section 4. Section 5 contains some concluding remarks.

2 Minimum-Cost Circulations, Cycle Canceling, and Minimum-Mean Cycles

Our framework for discussing the minimum-cost circulation problem is the same as in our previous paper [13]. Let $G = (V, E)$ be a directed graph with vertex set V containing n vertices and arc set E containing m arcs. We require G to be *symmetric*, i.e. $(v, w) \in E$ if and only if $(w, v) \in E$. For a vertex v, we denote by $E(v)$ the set $\{w|(v, w) \in E\}$. Graph G is a *circulation network* if each arc (v, w) has a *capacity* $u(v, w)$ and a *cost* $c(v, w)$, both real numbers. We require the cost function to be *antisymmetric*, i.e. $c(v, w) = -c(w, v)$ for all $(v, w) \in E$.

A *circulation* is a real-valued function f on arcs satisfying the following constraints:

$$f(v, w) \leq u(v, w) \quad \forall (v, w) \in E \quad \text{(capacity constraints)}, \tag{1}$$

$$f(v, w) = -f(w, v) \quad \forall (v, w) \in E \quad \text{(flow antisymmetry constraints)}, \tag{2}$$

$$\sum_{v \in E(w)} f(v, w) = 0 \quad \forall w \in V \quad \text{(conservation constraints)}. \tag{3}$$

The *cost* of a circulation f is given by the following expression:

$$cost(f) = \frac{1}{2} \sum_{(v,w) \in E} c(v, w) f(v, w).$$

The *minimum-cost circulation problem* is that of finding a circulation of minimum cost.

For a circulation f and an arc (v, w), the *residual capacity* of (v, w) is $u_f(v, w) = u(v, w) - f(v, w)$. An arc (v, w) is a *residual arc* if $u_f(v, w) > 0$. An arc that is not residual is *saturated*. We denote by E_f the set of residual arcs. A *residual cycle* is a simple cycle of residual arcs. The *capacity* of a residual cycle is the minimum of the residual capacities of its arcs. Note that the capacity of any residual cycle is positive. The *cost* of a cycle is the sum of the costs of its arcs. A residual cycle is *negative* if it has negative cost.

The following classical theorem characterizes minimum-cost circulations:

Theorem 2.1 [4]. *A circulation is minimum-cost if and only if there are no negative residual cycles.*

Theorem 2.1 suggests the following well-known algorithm due to Klein [17] for computing a minimum-cost circulation, which we call the *cycle-canceling algorithm.* Begin with any circulation f. (A starting circulation can be computed using any maximum flow algorithm, such as the algorithm of [14].) Repeat the following step until there are no negative residual cycles: Find a negative residual cycle Γ and *cancel* it by increasing the flow on each of its arcs by an amount equal to the capacity of Γ. (This saturates at least one arc on Γ.)

The cycle-canceling algorithm can run for an exponential number of iterations even if the capacities and costs are integers, and it need not even terminate if the capacities are irrational. A natural question is whether there exists a rule for selecting cycles to cancel that results in a number of iterations bounded by a polynomial in n and m. We answer this question in the affirmative.

Our selection rule is simple: always cancel a residual cycle whose average arc cost is as small as possible. In discussing this rule, we shall use the following terminology. The *mean cost* of a cycle is its cost divided by the number of arcs it contains. A *minimum-mean cycle* is a cycle whose mean cost is as small as possible. The *minimum cycle mean* of a graph with arc costs is the mean cost of a minimum-mean cycle. We call our selection rule *minimum-mean selection.*

The importance of minimum-mean cycles in the minimum-cost circulation problem is suggested by a result in our previous paper [13]. There we obtained a strongly polynomial algorithm by using a minimum cycle mean computation to update the dual variables.

Minimum-mean selection can be regarded as a generalization of the rule proposed by Dinic [5] and Edmonds and Karp [6] for selecting augmenting paths in the Ford-Fulkerson maximum flow algorithm [7]. Their rule is to always select an augmenting path containing as few arcs as possible. A standard way to convert a maximum flow problem into a minimum-cost circulation problem is to give every original arc a cost of zero and to add a *return arc* from the sink to the source. The capacity of the return arc is infinite and the cost is minus one. The minimum-mean cycle-canceling algorithm on this augmented network corresponds exactly to the Ford-Fulkerson maximum flow algorithm on the original network; minimum-mean selection algorithm corresponds exactly to the Edmonds-Karp algorithm.

A minimum-mean cycle can be found in $O(nm)$ time using an algorithm of Karp [16] or in $O(nm \log n)$ time using a new algorithm of Young [26]. Although the former method has a smaller worst-case time bound, the latter method might be preferable in practice, since

Karp's algorithm has a *best-case* running time of $\Omega(nm)$, whereas Young's method can run much faster. In the next section we show that the cycle-canceling algorithm with minimum-mean selection terminates after $O(nm \min\{\log(nC), m \log n\})$ cycles have been canceled, thereby establishing a strongly polynomial bound of $O(n^2m^2 \min\{\log(nC), m \log n\})$ on its running time.

3 Analysis of Minimum-Mean Cycle-Canceling

In order to analyze the cycle-canceling algorithm, we need to introduce notions from linear programming duality theory. A *price function* p is a real-valued function on the vertices of G. For a price function p, the *reduced cost* of an arc (v, w) is $c_p(v, w) = c(v, w) + p(v) - p(w)$. Observe that the cost of a residual cycle is the same whether the original arc costs or the reduced arc costs with respect to some price function are used. Furthermore, the flow conservation constraints (3) imply that the cost of any circulation is unaffected by replacing original costs by reduced costs.

The following classical result expresses linear programming duality for the special case of minimum-cost circulations.

Theorem 3.1 [7]. *A circulation f is minimum-cost if and only if there is a price function p such that, for all $(v, w) \in E$,*

$$u_f(v, w) > 0 \Rightarrow c_p(v, w) \geq 0 \quad \text{(optimality constraint)}. \tag{4}$$

The optimality constraints are more commonly called the *complementary slackness constraints.*

A notion of approximate optimality plays a crucial role in our analysis. The appropriate notion, called *ϵ-optimality*, is obtained by relaxing the complementary slackness constraints. The relaxed complementary slackness constraints were first described in print by Tardos [24] and were independently discovered by Bertsekas [2]. The notion of ϵ-optimality is the basis of several minimum-cost circulation algorithms [2,13,24]. For an $\epsilon \geq 0$, a circulation f is *ϵ-optimal* if there is a price function p such that, for all $(v, w) \in E$,

$$u_f(v, w) > 0 \Rightarrow c_p(v, w) \geq -\epsilon \quad (\epsilon\text{-optimality constraint}). \tag{5}$$

Note that 0-optimality is equivalent to optimality. Furthermore, if all arc costs are integers and ϵ is small enough, then an ϵ-optimal circulation is optimal.

Theorem 3.2 [2]. *If all arc costs are integers and $\epsilon < 1/n$, then any ϵ-optimal circulation is minimum-cost.*

A result from our previous paper establishes a connection between ϵ-optimality and minimum cycle means. For a circulation f, we denote by $\epsilon(f)$ the minimum ϵ such that f is ϵ-optimal, and by $\mu(f)$ the mean cost of a minimum-mean residual cycle.

Theorem 3.3 [13]. *For any circulation f, $\epsilon(f) = \max\{0, -\mu(f)\}$.*

An important concept concerning minimum-cost flows is that of the *admissible graph* $G(f,p) = (V, E(f,p))$. The admissible graph is the subgraph of the residual graph induced by the arcs with negative reduced cost:

$$E(f,p) = \{(v,w) \in E_f | c_p(v,w) < 0\}.$$

The following lemma provides a key insight into the problem. Although the lemma is not used until Section 4, it motivates the analysis of the current section.

Lemma 3.4 *Suppose a circulation f is ϵ-optimal with respect to a price function p and $G(f,p)$ is acyclic. Then f is $(1-1/n)\epsilon$-optimal.*

Proof: Let Γ be a simple cycle in G_f, and let l be the length of Γ. By the ϵ-optimality of f, the cost of every arc on Γ is at least $-\epsilon$. Since the admissible graph is acyclic, at least one arc on Γ has a nonnegative cost. Therefore the mean cost of Γ is at least

$$-\frac{(l-1)\epsilon}{l} \leq -\frac{(n-1)\epsilon}{n} = -(1-1/n)\epsilon.$$

Theorem 3.3 implies that f is $(1-1/n)\epsilon$-optimal. ∎

Now we have enough tools to analyze the minimum-mean cycle-canceling algorithm. As a measure of the quality of the current circulation f, we use $\epsilon(f)$: the smaller $\epsilon(f)$, the closer f is to optimal. Let f be an arbitrary circulation, let $\epsilon = \epsilon(f)$, and let p be a price function with respect to which f is ϵ-optimal. Holding ϵ and p fixed, we study the effect on $\epsilon(f)$ of minimum-mean cycle cancellations that modify f.

Lemma 3.5 *Canceling a minimum-mean cycle cannot increase $\epsilon(f)$.*

Proof: Let Γ be the minimum-mean cycle that is canceled. Before Γ is canceled, every residual arc satisfies $c_p(v,w) \geq -\epsilon$ by ϵ-optimality. The choices of ϵ and Γ imply that every arc (v,w) on Γ satisfies $c_p(v,w) = -\epsilon$ before canceling. By antisymmetry, every new residual arc created by canceling Γ has cost ϵ. (Every such arc is the reversal of an arc on Γ.) It follows that after cancellation of Γ, every residual arc still satisfies $c_p(v,w) \geq -\epsilon$. Thus after the cancellation $\epsilon(f) \leq \epsilon$. ∎

Lemma 3.6 *A sequence of m minimum-mean cycle cancellations reduces $\epsilon(f)$ to at most $(1 - /n)\epsilon$, i.e. to at most $1 - 1/n$ times its original value.*

Proof: Consider a sequence of m minimum-mean cycle cancellations. As f changes, the admissible graph $G(f,p)$ changes as well. Initially every arc $(v,w) \epsilon E(f,p)$ satisfies $c_p(v,w) \geq -\epsilon$. Canceling a cycle all of whose arcs are in $E(f,p)$ and adds only arcs of positive reduced cost to E_f and deletes at least one arc from $E(f,p)$. We consider two cases.

Case 1: None of the cycles canceled contains an arc of nonnegative reduced cost. Then each cancellation reduces the size of $E(f,p)$, and after m cancellations $E(f,p)$ is empty, which implies that f is optimal, i.e. $\epsilon(f) = 0$. Thus the lemma is true in this case.

Case 2: Some cycle canceled contains an arc of nonnegative reduced cost. Let Γ be the first such cycle canceled. By the proof of Lemma 3.4, the mean cost of Γ is at least $-(1 - 1/n)\epsilon$. Thus just before the cancellation of Γ, $\epsilon(f) \leq (1 - 1/n)\epsilon$ by Theorem 3.3. Since by Lemma 3.5 $\epsilon(f)$ never increases, the lemma is true in this case also. ∎

Lemmas 3.5 and 3.6 are enough to derive a polynomial bound on the number of iterations, assuming that all arc costs are integers.

Theorem 3.7 *If all arc costs are integers, then minimum-mean cycle-canceling terminates after $O(nm \log(nC))$ iterations.*

Proof: Let f be the circulation maintained by the algorithm. Initially $\epsilon(f) \leq C$. If $\epsilon(f) < 1/n$, then $\epsilon(f) = 0$ by Theorem 3.1. Lemmas 3.5 and 3.6 imply that if i is the total number of iterations, $(1 - 1/n)^{\lfloor (i-1)/m \rfloor} \geq 1/(nC)$. That is, $\lfloor (i-1)/m \rfloor \leq -\ln(nC)/\ln(1 - 1/n) \leq n \ln(nC)$, since $\ln(1 - 1/n) \leq -1/n$ for $n > 1$. It follows that $i = O(nm \log(nC))$. ∎

To obtain a strongly polynomial bound, we use an analysis closely following that of [13]. We say that an arc is *ϵ-fixed* if and only if the flow through this arc is the same for all

ϵ-optimal circulations. The following result is a generalization of a theorem of Tardos [24].

Theorem 3.8 ([13]) *Let $\epsilon > 0$, suppose a circulation f is ϵ-optimal with respect to a price function p, and suppose that for some arc (v,w), $\mid c_p(v,w) \mid \geq 2n\epsilon$. Then (v,w) is ϵ-fixed.*

Consider an execution of the cycle-canceling algorithm. Suppose an edge (v,w) becomes fixed at some point of the execution. Since, by Lemma 3.5, the error parameter $\epsilon(f)$ never increases, the flow through (v,w) henceforth remains the same. When all edges are fixed, the current circulation is optimal. To see this, observe that an optimal circulation is ϵ-optimal for any $\epsilon \geq 0$, and therefore it must agree with the current circulation on all (fixed) edges.

The following theorem bounds the number of iterations of the cycle-canceling algorithm in the case of real-valued costs. In the proof, we use the following inequality: $(1-\frac{1}{n})^{n(\ln n+1)} \leq \frac{1}{2n}$ for $n \geq 2$.

Theorem 3.9 *For arbitrary real-valued arc costs, the minimum-mean cycle-canceling algorithm terminates after $O(nm^2 \log n)$ iterations.*

Proof: Let $k = m(n\lceil \ln n+1 \rceil)$. Divide the iterations into groups of k consecutive iterations. We claim that each group of iterations fixes the flow on a distinct arc (v,w), i.e. iterations after those in the group do not change $f(v,w)$. The theorem is immediate from the claim.

To prove the claim, consider any group of iterations. Let f be the flow before the first iteration of the group, f' the flow after the last iteration of the group, $\epsilon = \epsilon(f)$, $\epsilon' = \epsilon(f')$, and let p' be a price function for which f' satisfies the ϵ'-optimality constraints. Let Γ be the cycle canceled in the first iteration of the group. The choice of k implies by Lemmas 3.5 and 3.6 that $\epsilon' \leq \epsilon(1-\frac{1}{n})^{n\lceil \ln n+1 \rceil} \leq \frac{\epsilon}{2n}$. Since the mean cost of Γ is $-\epsilon$, some arc on Γ, say (v,w), must have $c_p(v,w) \leq -\epsilon \leq -2n\epsilon'$. By Lemma 3.5 and Theorem 3.8, the flow on (v,w), will not be changed by iterations after those in the group. But $f(v,w)$ *is* changed by the first iteration in the group, which cancels Γ. Thus each group fixes the flow on a distinct arc. ∎

Theorem 3.10 *With the use of Karp's algorithm for finding a minimum-mean cycle, the minimum-mean cycle-canceling algorithm runs in $O(n^2m^3 \log n)$ time on networks with arbitrary real-valued arc costs, and in $O(n^2m^2 \min\{\log(nC), m \log n\})$ on networks with integer arc costs.*

Proof: Immediate from Theorems 3.7 and 3.9. ∎

4 A Faster Variant of Minimum-Mean Cycle-Canceling

Although Theorem 3.10 is an interesting theoretical result, because it shows that a classical minimum-cost circulation algorithm is strongly polynomial with a natural choice of iterative steps, the bounds on the performance of the algorithm are not competitive with those of most previous polynomial-time algorithms. In this section we describe a variant of the minimum-mean cycle-canceling algorithm for which the time per iteration is $O(\log n)$ instead of $O(nm)$. This improvement is based on a more flexible selection of cycles for canceling, explicit maintenance of a price function to help identify cycles for canceling, and a sophisticated data structure to help keep track of arc flows.

The algorithm, which we call the *cancel-and-tighten algorithm*, maintains a circulation f and a price function p. We denote by $\epsilon(f,p)$ the minimum ϵ such that f satisfies the ϵ-optimality constraints for p, i.e. $\epsilon(f,p) = \max\{0, -\min\{c_p(v,w) | u_f(v,w) > 0\}\}$. We call a residual arc (v,w) *admissible* if $c_p(v,w) < 0$, and a residual cycle *admissible* if all its arcs are admissible. Initially f is any circulation and p is the identically zero price function. (Thus $\epsilon(f,p) \leq C$ initially.) The algorithm consists of repeating the following two steps until the circulation f is optimal:

Step 1 [cancel cycles]. Repeatedly find and cancel admissible cycles until the admissible graph is acyclic.

Step 2 [tighten prices]. Modify p so that $\epsilon(f,p)$ decreases to at most $(1-1/n)$ times its former value.

Theorem 4.1 *The cancel-and-tighten algorithm is correct. Each iteration of Step 1 results in the canceling of at most m cycles. If all arc costs are integers, there are $O(n\log(nC))$ iterations of Steps 1 and 2.*

Proof: The first two statements of the theorem follow by the proof of Lemma 3.6. Namely, let $E(f,p)$ be the set of admissible arcs. Canceling an admissible cycle reduces the size of $E(f,p)$ by at least one and cannot increase $\epsilon(f,p)$, since all newly created residual arcs have positive cost. Since $E(f,p)$ has maximum size m and minimum size 0, after at most m cycle cancellations Step 1 terminates. Lemma 3.4 implies that after Step 1, the mean cost of a residual cycle is at least $-(1-1/n)\,\epsilon(f,p)$, which implies that p can be modified

to satisfy the requirement in Step 2. The third part of the theorem follows as in the proof of Theorem 3.7. ∎

Now we show how to implement Steps 1 and 2. Step 1 is the dominant part of the computation. A simple implementation runs in $O(nm)$ time ($O(n)$ per cycle canceled). A more complicated implementation runs in $O(m \log n)$ time ($O(\log n)$ per cycle canceled).

Performing Step 1 is essentially the same as converting an arbitrary flow into an acyclic flow by eliminating cycles of flow, and algorithms for the latter purpose, such as the $O(m \log n)$-time algorithm of Sleator and Tarjan [22], can be adapted to the former purpose. We shall describe the appropriately modified version of this algorithm. First, however, we describe the simple implementation of Step 1 on which the Sleator-Tarjan algorithm is based.

The simple method is analogous to a subroutine in Dinic's maximum flow algorithm for finding augmenting paths of a given length, once a layered residual network is constructed [5]. The method uses depth-first search to find admissible cycles. Each search advances only along admissible arcs. Whenever a search retreats from a vertex v, this vertex is marked as being on no admissible cycles. A search is allowed to visit only unmarked vertices. Whenever a search advances to a vertex it has already visited, an admissible cycle has been found. The cycle is canceled and a new search begun. A straightforward implementation of this algorithm has a running time of $O(m)$ plus $O(n)$ per cycle canceled, for a total of $O(nm)$ time.

The Sleator-Tarjan algorithm improves on this method by using a dynamic tree data structure [22,23,25] to avoid explicitly searching along the same path many times. The data structure allows the maintenance of a collection of vertex-disjoint rooted trees, each arc of which has an associated real value. The data structure supports the seven operations described in Figure 1. A sequence of l tree operations on trees of maximum size k takes $O(l \log k)$ time.

In the dynamic tree implementation of Step 1, if *parent* (v) is the parent of a vertex v, then $(v,$ *parent* $(v))$ is an admissible arc. The implementation of Step 1 is described in Figure 2.

A few extra data structures are needed in this method. To make Step 1b efficient, the algorithm maintains a list of all vertices and a current pointer into this list. All vertices preceding the pointer are marked. To find an unmarked vertex, the algorithm steps the

make-tree(v):	Make vertex v into a one-vertex dynamic tree. Vertex v must be in no other tree.
find-root(v):	Find and return the root of the tree containing vertex v.
find-value(v):	Find and return the value of the tree arc connecting v to its parent. If v is a tree root, return infinity.
find-min(v):	Find and return the ancestor w of v such that the tree arc connecting w to its parent has minimum value along the path from v to *find-root*(v). In case of a tie, choose the vertex w closest to the tree root. If v is a tree root, return v.
change-value(v, x):	Add real number x to the value of every arc along the path from v to *find-root*(v).
link(v, w):	Combine the trees containing v and w by making w the parent of v and giving the new tree arc joining v and w the value x. This operation does nothing if v and w are in the same tree or if w is not a tree root. A sequence of
cut(v):	Break the tree containing v into two trees by deleting the arc from v to its parent. This operation does nothing if v is a tree root.

Figure 1: Dynamic tree operations.

pointer through the list, stopping at the first unmarked vertex. Similarly, to find admissible arcs in Step 1c, the algorithm maintains lists of the arcs leaving each vertex and a current pointer into each such list. In addition, for each vertex v, the algorithm maintains *parent*(v) and a list of the vertices u such that *parent*(u) $= v$. A straightforward analysis of the algorithm shows that it requires $O(m)$ tree operations and runs in $O(m \log n)$ time.

Step 2 of the *cancel-and-tighten* algorithm is much easier to implement. The simple method described in Figure 3 has an $O(m)$ running time.

Since Step 2 is only performed when there are no admissible cycles, the level of every vertex is well-defined, and Step 2a can be performed in $O(m)$ time by computing the levels of vertices in a topological order, i. e. an order such that if (v, w) is an admissible arc, $L(v)$ is computed before $L(w)$. Step 2b obviously takes $O(m)$ time; Step 2c, $O(n)$ time. The following lemma shows that this implementation of Step 2 achieves the required decrease in $\epsilon(f, p)$:

Lemma 4.2 *Steps 2a-2c reduce $\epsilon(f, p)$ to at most $1 - 1/n$ times its former value.*

Proof: In Step 2b, the computed value of ϵ is $\epsilon(f, p)$. Every admissible arc (v, w) has $L(v) < L(w)$ by the definition of levels. Thus the only arcs participating in the definition of ρ are those of nonnegative reduced cost. It follows that $\rho \geq 0$.

Step 1a [initialize]. For each vertex v, unmark v and perform *make-tree* (v).

Step 1b [find starting vertex for a search]. If all vertices are marked, stop. Otherwise, select an unmarked vertex v and go to Step 1c.

Step 1c [find end of path]. Perform $r \leftarrow$ *find-root* (v). If there is no admissible arc (r, w) with w unmarked, go to step 1f. Otherwise, let (v, w) be such an arc and go to Step 1d.

Step 1d [extend path]. If *find-root* $(w) \neq r$, perform *link*$(r, w, u_f(r, w))$ and go to Step 1c. Otherwise, go to Step 1e.

Step 1e [cancel cycle]. Let $\delta = \min\{u_f(r, w),$ *find-value*(*find-min*$(w)\})$. Perform $f(r, w) \leftarrow f(r, w) + \delta$. If $u_f(v, w) = 0$, mark (r, w) inadmissible. Perform *change-value*$(w, -\delta)$. While *find-value*(*find* $-$ *min*$(w)) = 0$, do the following: $u \leftarrow$ *find-min*(w); $f(u,$ *parent*$(u)) \leftarrow 0$; *cut*(u). Go to Step 1b.

Step 1f [retract path]. Mark r. For each vertex u such that $r =$ *parent*(u), do the following: $f(u, r) \leftarrow u(v, w) -$ *find-value*(u); *cut*(u). Go to Step 1b.

Figure 2: Implementation of Step 1.

Step 2a [compute levels]. For each vertex v, compute a *level* $L(v)$, defined recursively as follows: If v has no incoming admissible arcs, $L(v) = 0$. Otherwise, $L(v) = \max\{L(u) + 1 | (u, v)$ is an admissible arc$\}$.

Step 2b [compute price increment]. Compute $\epsilon = -\min\{c_p(v, w) | (v, w)$ is admissible$\}$. Compute $\rho = \min\{(c_p(v, w) + \epsilon)/(L(v) - L(w) + 1) | u_f(v, w) > 0$ and $L(v) > L(w)\}$.

Step 2c [compute new prices]. For all $v \epsilon V$, replace $p(v)$ by $p(v) - \rho L(v)$.

Figure 3: Implementation of Step 2.

The price change in Step 2c increases the cost of each arc (v, w) such that $L(v) < L(w)$ by an amount $\rho(L(w) - L(v))$. This includes all the originally admissible arcs. Thus each such arc has cost at least $-\epsilon + \rho$ after Step 2c. Reduced cost of an arc (v, w) with $L(v) = L(w)$ is not changed by Step 2c, and thus the reduced cost remains nonnegative. Reduced cost of an arc (v, w) with $L(v) > L(w)$ is decreased by $\rho(L(v) - L(w))$. If $c_p(v, w)$ is its original reduced cost, its new reduced cost is $c_p(v, w) - \rho\ (L(v) - L(w)) \geq -\epsilon + \rho$ by the definition of ρ. Thus after Step 2c $\epsilon(f, p) \leq \epsilon - \rho$. The definition of ρ implies that $\rho \geq \epsilon/n$, which implies that, after Step 2c, $\epsilon(f, p) \leq (1 - 1/n)\epsilon$, as desired. ∎

The dynamic tree implementation of Step 1 and the above implementation of Step 2 yield an $O(m \log n)$ time bound per iteration of Steps 1 and 2. Theorem 4.1 gives an $O(nm \log n \log(nC))$ bound on the total time to find a minimum-cost circulation. This algorithm is not strongly polynomial, but the implementation of Step 2 can be modified to give a strongly polynomial method. Namely, every n^{th} iteration of Step 2 is performed differently. At such an iteration, we replace ϵ by $\epsilon(f)$ and then replace the price function

p by a price function p' such that f is $\epsilon(f)$-optimal with respect to p'. In our previous paper [13] we show how to find $\epsilon(f)$ and p' in $O(nm)$ time using an algorithm of Karp [16] for finding a minimum mean cost cycle and the Bellman-Ford algorithm (see e.g. [25]) for finding shortest paths from a single source. Since these computations only occur once every n iterations of Steps 1 and 2, they do not affect the $O(m \log n)$ bound per iteration, except that the bound becomes amortized instead of worst-case. With this change of Step 2, a bound of $O(nm \log n)$ on the number of iterations of Steps 1 and 2 follows as in the proof of Theorem 3.9. This bound is valid for arbitrary real-valued costs. Thus we obtain the following theorem:

Theorem 4.3 *The cancel-and-tighten algorithm, with the dynamic tree implementation of Step 1 and a minimum cycle mean computation after every n iterations, runs in $O(nm^2(\log n)^2)$ time on networks with arbitrary real-valued arc costs, and in $O(nm(\log n)\ \min\{\log(nC),\ m \log n\})$ time on networks with integer arc costs.*

5 Remarks

Open problems remain, concerning both the practical and the theoretical ramifications of our results. On the practical side, we believe that the practical performance of the algorithm described in Section 4 is worth investigating. There are some obvious modifications that should improve the practical performance of the method. For example, Step 1 need not be performed unless there is an admissible cycle, and Steps 2a-2c can be repeatedly performed until such a cycle exists. Perhaps some alternative implementation of Step 2 might be better in practice. It is not clear how much time should be spent between iterations of Step 1 in trying to reduce $\epsilon(f, p)$.

On the theoretical side, an open question is whether one can reduce the asymptotic running time of Step 1, and hence of the entire algorithm. Previous results [13,14] suggest the possibility of an $O(m \log(n^2/m))$ bound. More generally, it would be interesting to see what other well-known algorithms for the minimum-cost circulation problem and other problems can be made polynomial or strongly polynomial by choosing iterative steps carefully. Another question is to what extent the bounds in Theorems 3.10 and 4.3 can be improved in special cases. For example, if all capacities are zero or one, the simple implementation of Step 1 of the *cancel-and-tighten* algorithm runs in $O(m)$ time, and the bounds in Theorem 4.3 decrease by a factor of $\log n$. Probably even better bounds are

obtainable. See e.g. [10].

References

[1] R. K. Ahuja and J. B. Orlin. *A Fast and Simple Algorithm for the Maximum Flow Problem.* Technical Report 1905-87, Sloan School of Management, M.I.T., 1987.

[2] D. P. Bertsekas. *Distributed Asynchronous Relaxation Methods for Linear Network Flow Problems.* Technical Report LIDS-P-1986, Lab. for Decision Systems, M.I.T., September 1986. (Revised November, 1986).

[3] R. G. Bland and D. L. Jensen. *On the Computational Behavior of a Polynomial-Time Network Flow Algorithm.* Technical Report 661, School of Operations Research and Industrial Engineering, Cornell University, 1985.

[4] R. G. Busacker and T. L. Saaty. *Finite Graphs and Networks: An Introduction with Applications.* McGraw-Hill, New York, NY., 1965.

[5] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Dokl.*, 11:1277–1280, 1970.

[6] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. Assoc. Comput. Mach.*, 19:248–264, 1972.

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

[8] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *Proc. 25th IEEE Symp. on Foundations of Computer Science*, pages 338–346, 1984. (To appear in J. Assoc. Comput. Mach.).

[9] S. Fujishige. A capacity-rounding algorithm for the minimum-cost circulation problem: a dual framework of the tardos algorithm. *Math. Prog.*, 35:298–308, 1986.

[10] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. (To appear).

[11] Z. Galil and E. Tardos. *An $O(n^2 \log n(m + n \log n))$ Minimum Cost Flow Algorithm.* Technical Report 04518-86, Mathematical Science Research Institute, Berkeley, CA, 1986.

[12] A. V. Goldberg. *Efficient Graph Algorithms for Sequential and Parallel Computers.* PhD thesis, M.I.T., January 1987. (Also available as Technical Report TR-374, Lab. for Computer Science, M.I.T., 1987).

[13] A. V. Goldberg and R. E. Tarjan. *Finding Minimum-Cost Circulations by Successive Approximation.* Technical Report CS-TR 106-87, Department of Computer Science, Princeton University, 1987.

[14] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. In *Proc. 18th ACM Symp. on Theory of Computing*, pages 136–146, 1986. (To appear in J. Assoc. Comput. Mach.).

[15] A. V. Goldberg and R. E. Tarjan. Solving minimum-cost flow problems by successive approximation. In *Proc. 19th ACM Symp. on Theory of Computing*, pages 7–18, 1987.

[16] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Math.*, 23:309–311, 1978.

[17] M. Klein. A primal method for minimal cost flows with applications to the assignment and transportation problems. *Management Science*, 14:205–220, 1967.

[18] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Reinhart, and Winston, New York, NY., 1976.

[19] J. B. Orlin. *Genuinely Polynomial Simplex and Non-Simplex Algorithms for the Minimum Cost Flow Problem.* Technical Report No. 1615-84, Sloan School of Management, MIT, December 1984.

[20] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, NJ, 1982.

[21] H. Röck. Scaling techniques for minimal cost network flows. In U. Pape, editor, *Discrete Structures and Algorithms*, pages 181–191, Carl Hansen, Münich, 1980.

[22] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. System Sci.*, 26:362–391, 1983.

[23] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *J. Assoc. Comput. Mach.*, 32:652–686, 1985.

[24] E. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255, 1985.

[25] R. E. Tarjan. ***Data Structures and Network Algorithms.*** Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

[26] N. Young. Finding a minimum mean cycle: an $O(nm\log(n))$ time primal-dual algorithm. 1987. (Unpublished manuscript, Department of Computer Science, Princeton University).