Finding Minimum-Cost Circulations by Canceling Negative Cycles

ANDREW V. GOLDBERG

Stanford University, Stanford, California

AND

ROBERT E. TARJAN

Princeton University, Princeton, New Jersey, and AT&T Bell Laboratories, Murray Hill, New Jersey

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.

Categories and Subject Descriptors: G.2.2 [**Discrete Mathematics**]: Graph Theory—*network problems*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Cycle canceling, dynamic tree, minimum-cost flow, network optimization, transportation problem, transshipment problem

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 [10, 22, 27].

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 [16]; we summarize some of the previous results here.

A preliminary version of this paper was published in the *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, ACM, New York, 1988, pp. 388–397.

Part of this research was done while A. V. Goldberg was at the M.I.T. Laboratory for Computer Science, supported by DARPA contract no. N00014-80-C-0622.

The work of R. E. Tarjan was partially supported by the National Science Foundation, grant no. DCR 86-05961, and the Office of Naval Research, contract no. N00014-87-K-0467.

Authors' present addresses: A. V. Goldberg, Department of Computer Science, Stanford University, Stanford, CA 94305; R. E. Tarjan, Department of Computer Science, Princeton, NJ 08544.

The important parameters by which the running time of an algorithm is measured 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 [9], 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 $O(m(\log U)S(n, m, C))$. Here $S(n, m, C)$ is the time required for a single-source shortest path computation on a network with nonnegative arc lengths. The best-known strongly polynomial bound is $S(n, m, C) = O(m + n \log n)$ [11]; when the length function is integral and C is not too large, better bounds on $S(n, m, C)$ can be obtained [2, 34]. Röck [28] exhibited another capacity-scaling algorithm achieving the same time bound and also presented a cost-scaling algorithm with a running time of $O(n(\log C)M(n, m, U))$. Here $M(n, m, U)$ is the time required for a maximum flow computation; the best current bound is $M(n, m, U) = \min\{nm \log(n^2/m), nm \log((n/m)(\log U)^{1/2} + 2)\}$ [3, 17]. The $O(n(\log C)M(n, m, U))$ bound was also obtained later by Bland and Jensen [6] using a somewhat different cost-scaling approach. A generalization of the cost-scaling approach has been proposed in [15], [16], and [18].

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 [32] was the first to devise a strongly polynomial algorithm for the problem. Other strongly polynomial algorithms were later proposed by Orlin [23], Fujishige [12], and Gail and Tardos [14]. The fastest known strongly polynomial algorithm is that of Orlin [25], which runs in $O(m(\log n)(m + n \log n))$ time.

For networks having integer arc costs that are not huge, the asymptotically fastest algorithms are those of Goldberg and Tarjan [16], with a running time of $O(nm(\log(n^2/m))\min\{\log(nC), m \log n\})$, and of Ahuja et al. [1], with a running time of $O(nm \log \log U \log nC))$.

All the known polynomial-time algorithms for the problem, whether 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 [21]. 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. 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 [10]. 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. We call such a cycle a *minimum-mean* cycle, and the above selection rule the *minimum-mean* selection rule. A minimum-mean cycle can be found in $O(nm)$ time using an algorithm of Karp [19] or in $O(\sqrt{n}m \log(nC))$ time using an algorithm of Orlin and Ahuja [26]. 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 $O(n^2m^3\{\log n\})$ on its running time.

The minimum-mean cycle-canceling strategy can be viewed as a greedy strategy. Interpret arcs of the network as unit-time transactions involving a certain commodity, with profit per unit of commodity given by the negation of the cost function. A valid collection of transactions corresponds to a valid augmentation of flow along a cycle. Then minimum-mean selection corresponds to selecting a change that gives biggest improvement in profit per unit of time.

The minimum-mean cycle-canceling algorithm is closely related to a classical maximum flow method. Namely, the minimum-mean selection rule is a natural generalization of the rule proposed by Edmonds and Karp [9] and Dinic [8] for selecting augmenting paths in the Ford–Fulkerson maximum-flow algorithm [10]. Their rule is to always select a shortest augmenting path. If a maximum-flow problem is formulated as a minimum-cost circulation problem in a standard way, the cycle-canceling algorithm corresponds exactly to the Ford–Fulkerson maximum-flow algorithm, and the minimum-mean selection algorithm corresponds exactly to the Edmonds–Karp algorithm.

The key to the analysis of the minimum-mean selection algorithm is the observation that the minimum-mean cycle cost is a good measure of the quality of a circulation (Theorem 3.3). This observation leads to a new method for proving strongly polynomial bounds on the running time of minimum-cost flow algorithms. In addition to the cycle-canceling algorithm, this method can be applied to any minimum-cost circulation algorithm that uses generalized cost scaling [16].

Although the minimum-mean cycle-canceling 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 [30, 31, 33]. 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 on nondense networks with arc costs that are not huge. This algorithm and its analysis are presented in Section 4.

The minimum-mean cycle selection strategy is not the only one that leads to polynomial-time algorithms. Weintraub [35] describes an algorithm that at each iteration cancels a collection of negative cycles and significantly improves the objective function value. Although Weintraub's algorithm does not run in polynomial time, a modification of it, suggested by Barahona and Tardos [4], does. Section 5 contains some concluding remarks, including a discussion of Weintraub's algorithm.

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

Our framework for discussing the minimum-cost circulation problem is as follows. 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*, that is, $(v, w) \in E$ if and only if $(w, v) \in E$. For a vertex v, we denote by $E(v)$ the set $\{w \mid (v, w) \in E\}$. 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*, that is, $c(v, w) = -c(w, v)$ for all $(v, w) \in E$.

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

$$f(v, w) \leq u(v, w) \qquad \forall (v, w) \in E \quad \text{(capacity constraints)}, \tag{1}$$
$$f(v, w) = -f(w, v) \qquad \forall (v, w) \in E \quad \text{(flow antisymmetry constraints)}, \tag{2}$$
$$\sum_{v \in E(w)} f(v, w) = 0 \qquad \forall w \in V \quad \text{(conservation constraints)}. \tag{3}$$

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 [7]. *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 [21] 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 algorithms of [3, 17].) 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.)

We show that a careful choice of the next cycle to cancel leads to a strongly polynomial algorithm. 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.*

A minimum-mean cycle can be found in $O(nm)$ time using an algorithm of Karp [19] or in $O(\sqrt{n}m \log(nC))$ time using a recent algorithm of Orlin and Ahuja. (Although these two algorithms give the best running time bounds, an algorithm of Karp and Orlin [20] may turn out to be faster in practice.) 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 $O(n^2m^3 \log n)$ 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 [10]. *A circulation f is minimum-cost if and only if there is a price function p such that*:

$$u_f(v, w) > 0 \Rightarrow c_p(v, w) \geq 0 \qquad \forall (v, w) \in E \quad \text{(optimality constraints)}. \qquad (4)$$

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 [32] and were independently discovered by Bertsekas [5]. The notion of ϵ-optimality is the basis of several minimum-cost circulation algorithms [1, 5, 16, 32]. For an $\epsilon \geq 0$, a circulation f is *ϵ-optimal* if there is a price function p such that

$$u_f(v, w) > 0 \Rightarrow c_p(v, w) \geq -\epsilon \qquad \forall (v, w) \in E \quad (\epsilon\text{-optimality constraints}). \qquad (5)$$

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

PROOF. Consider a simple cycle in G_f. The ϵ-optimality of f implies that the reduced cost of the cycle is at least $n\epsilon > -1$. The reduced cost of the cycle equals its original cost, which must be integral and hence nonnegative. Theorem 2.1 implies that f 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 [16]. *Suppose f is a nonoptimal circulation. Then $\epsilon(f) = -\mu(f)$.*

PROOF. Consider any cycle Γ in G_f. Let the length of Γ be l. For any ϵ, define $c^{(\epsilon)}$ by $c^{(\epsilon)}(v, w) = c(v, w) + \epsilon$ for $(v, w) \in E_f$. Since f is $\epsilon(f)$-optimal, we have $0 \leq c^{(\epsilon)}(\Gamma) = c(\Gamma) + l\epsilon$, i.e., $c(\Gamma)/l \geq -\epsilon(f)$. Since this is true for any cycle Γ, $\mu(f) \geq -\epsilon(f)$, i.e., $\epsilon(f) \geq -\mu(f)$.

Conversely, let Γ be the minimum-mean cost residual cycle; then the mean cost of Γ is equal to $\mu(f)$. Since f is not optimal, there is a negative-cost residual cycle, and therefore $\mu(f) < 0$. Fix a price function p and an ϵ such that $\epsilon > -\mu(f)$. Since the cost of Γ is equal to the sum of the reduced costs of the arcs on Γ, for the minimum reduced cost arc (v, w) on Γ we have $c_p(v, w) \leq \mu(f)$ (the minimum is at most the average). Therefore $c_p(v, w) < -\epsilon$, so f is not ϵ-optimal with respect to p. Thus $\epsilon(f) \leq -\mu(f)$. □

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:

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

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

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

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

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. By Theorem 3.3, 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 $\epsilon(f) \leq \epsilon$. $\square$

LEMMA 3.6. *A sequence of m minimum-mean cycle cancellations reduces $\epsilon(f)$ to at most $(1 - 1/n)\epsilon$, that is, 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 $(v, w) \in E(f, p)$ satisfies $c_p(v, w) \geq -\epsilon$. Canceling a cycle all of whose arcs are in $E(f, p)$ 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, that is $\epsilon(f) = 0$. 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. Every arc of Γ has a reduced cost of at least $-\epsilon$, one arc of Γ has a nonnegative reduced cost, and the number of arcs in Γ is at most n. Therefore, 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. $\square$

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 the minimum-mean cycle-canceling algorithm 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 $i = O(nm \log(nC))$. $\square$

To obtain a strongly polynomial bound, we use an analysis closely following that of [16]. 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 [32].

THEOREM 3.8 [16]. *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), $|c_p(v, w)| \geq 2n\epsilon$. Then (v, w) is ϵ-fixed.*

PROOF. By antisymmetry, it is enough to prove the theorem for the case $c_p(v, w) \geq 2n\epsilon$. Let f' be a circulation such that $f'(v, w) \neq f(v, w)$. Since $c_p(v, w) > \epsilon$, the flow through the arc (v, w) must be as small as the capacity constraints allow, namely $-u(w, v)$, and therefore $f'(v, w) \neq f(v, w)$ implies $f(v, w) > f(v, w)$. We show that f' is not ϵ-optimal, from which the theorem follows.

Consider $G_{>} = (V, \{(x, y) \in E \mid f'(x, y) > f(x, y)\})$. Note that $G_{>}$ is a subgraph of G_f, and (v, w) is an arc of $G_{>}$. Since f and f^1 are circulations, $G_{>}$ must contain a simple cycle Γ that passes through (v, w). Let l be the length of Γ. Since all arcs of Γ are residual arcs, the cost of Γ is at least

$$c_p(v, w) - (l - 1)\epsilon \geq 2n\epsilon - (n - 1)\epsilon > n\epsilon.$$

Now consider a cycle $\bar{\Gamma}$ obtained by reversing the arcs on Γ. Note that $\bar{\Gamma}$ is a cycle in $G_{<} = (V, \{(x, y) \in E \mid f'(x, y) < f(x, y)\})$ and therefore a cycle in $G_{f'}$. By antisymmetry, the cost of $\bar{\Gamma}$ is less than $-n\epsilon$ and thus the mean cost of $\bar{\Gamma}$ is less than $-\epsilon$. Theorem 3.3 implies that f' is not ϵ-optimal. $\square$

Consider an execution of the cycle-canceling algorithm. Suppose an edge (v, w) becomes fixed at some point in 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) arcs.

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: $(1 - 1/n)^{n(\ln n+1)} \leq 1/(2n)$ 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), that is, 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 $\epsilon' \leq \epsilon(1 - 1/n)^{n(\ln n+1)} \leq \epsilon/(2n)$. 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. *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 $O(n^{3/2}m^2\min\{\log^2(nC), \sqrt{n}\log(nC), \sqrt{nm}\log n\})$ time on networks with integer arc costs.*

PROOF. Immediate from Theorems 3.7 and 3.9. □

4. *A Faster Cycle-Canceling Algorithm*

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 the best previous polynomial-time algorithms. In this section we describe a variant of the minimum-mean cycle-canceling algorithm for which the time per cycle cancellation 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, that is, $\epsilon(f, p) = \max\{0, -\min\{c_p(v, w) | u_f(v, w) > 0\}\}$. 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.

Note that, by Lemma 3.4, a suitable price function can always be found in Step 2.

THEOREM 4.1. *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. 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 $O(m \log n)$ 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 [30], 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 Sleater–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 [8]. 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 [30, 31, 33] 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 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, that is, an order such that

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, x):	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 v is not a tree root.
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.

FIG. 1. Dynamic tree operations.

Step 1a [*initialize*].
For each vertex v, unmark v and perform *make-tree*(v).
Step 1b [*finding starting vertex for a search*].
If all vertices are marked, go to Step 1g.
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:
$z \leftarrow$ *find-min*(w); $f(z, \mathit{parent}(z)) \leftarrow u(z, \mathit{parent}(z))$; *cut*($z$).
Go to Step 1b.
Step 1f [*retract path*].
Mark r.
For each vertex z such that $r = \mathit{parent}(z)$, do the following:
$f(z, r) \leftarrow u(z, r) -$ *find-value*(z); *cut*(z).
Go to Step 1b.
Step 1g [*extract flow values of tree arcs*].
For each vertex v, if *find-root*(v) $\neq v$, perform $f(v, \mathit{parent}(v)) \leftarrow u(v, \mathit{parent}(v)) -$ *find-value*(v).
Stop.

FIG. 2. Implementation of Step 1.

if (v, w) is an admissible arc, $L(v)$ is computed before $L(w)$. Step 2b obviously takes $O(n)$ time. The following lemma shows that this implementation of Step 2 achieves the required decrease in $\epsilon(f, p)$:

LEMMA 4.2. *Steps* 2a–2b *reduce* $\epsilon(f, p)$ *to* $1 - 1/n$ *times its former value.*

PROOF. The price change in Step 2b increases the reduced cost of each arc (v, w) such that $L(v) < L(w)$ by an amount $(\epsilon/n)(L(w) - L(v)) \geq \epsilon/n$. This includes all the originally admissible arcs. Thus each such arc has a reduced cost of at least $-\epsilon + \epsilon/n$ after Step 2b. The reduced cost of an arc (v, w) with $L(v) = L(w)$ is not changed by Step 2b, and thus the reduced cost remains nonnegative. The reduced cost of an arc (v, w) with $L(v) > L(w)$ is decreased by $(\epsilon/n)(L(v) - L(w))$.

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, $L(v) = \max\{L(u) + 1 \mid (u, v) \text{ is an admissible arc}\}$.

Step 2b [*compute new prices*].
For all $v \in V$, replace $p(v)$ by $p(v) - (\epsilon/n)L(v)$.

FIG. 3. Implementation of Step 2.

If $c_p(v, w) \geq 0$ is its original reduced cost, its new reduced cost is $c_p(v, w) - (\epsilon/n)(L(v) - L(w)) \geq -\epsilon(1 - 1/n)$, as desired. □

Remark. An alternative implementation of Step 2 is described in Figure 4. The proof of Lemma 4.2 can be modified to show that the modified implementation reduces $\epsilon(f)$ to at most $1 - 1/n$ times its former value. One expects, however, that the modified implementation will do better in practice.

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 nth iteration of Step 2 is performed differently. In 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 [16] we show how to find $\epsilon(f)$ and p' in $O(nm)$ time using an algorithm of Karp [19] for finding a minimum mean cycle and the Bellman–Ford algorithm (see, e.g., [33]) 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 $O(m \log n)$ 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*

As mentioned in the introduction, the algorithm of Weintraub [35] is another example of a cycle-canceling algorithm. Although Weintraub's paper deals with networks with convex costs, we shall discuss his method in the special case of linear costs.

Weintraub's approach is as follows: Consider the improvement in the flow cost obtained by canceling a negative cycle. Since a symmetric difference of two circulations can be decomposed into at most m cycles, canceling the cycle that gives the best improvement reduces the difference between the current and the optimal values of the flow cost by a factor of $(1 - 1/m)$. If the input data is integral, only a polynomial number of such improvements can be made until an optimal solution is obtained. However, finding the cycle that gives the best improvement is NP-hard. Weintraub shows how to find a collection of cycles whose cancellation reduces the flow cost by at least as much as the best improvement achievable by

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, $L(v) = \max\{L(u) + 1 \mid (u, v) \text{ is an admissible arc}\}$.

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

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

FIG. 4. Alternative Implementation of Step 2.

canceling a single cycle. His method requires a superpolynomial number of applications of an algorithm for the assignment problem. Each such application yields a minimum-cost collection of vertex-disjoint cycles. Barahona and Tardos [4] have shown that the algorithm can be modified so that the required collection of cycles is found in at most m assignment computations. The resulting algorithm runs in polynomial time.

Orlin (personal communication) has suggested an alternative proof of Theorem 3.9. His proof combines the result of Theorem 3.7 and the techniques of [24]. Consider an instance $(G = (V, E), u, c)$ of the minimum-cost circulation problem. Let c' be a cost function; for any cycle Γ in G, let $\mu(\Gamma)$ denote the mean cost of Γ with respect to c, and let $\mu'(\Gamma)$ denote the mean cost of Γ with respect to c'. Define c' to be *equivalent* to c if for every pair of simple cycles Γ and Δ in G, we have $\mu(\Gamma) > \mu(\Delta)$ implies $\mu'(\Gamma) > \mu'(\Delta)$ and $\mu(\Gamma) = \mu(\Delta)$ implies $\mu'(\Gamma) = \mu'(\Delta)$. Note that if c is equivalent to c', there is a one-to-one correspondence between executions of the minimum-mean cycle-canceling algorithm on (G, u, c) and on (G, u, c'). (Different executions correspond to different ways of breaking ties among minimum-mean cost cycles.) By an argument similar to the one in [24], it can be shown that for every real-valued cost function c there is an equivalent cost function c' whose values are integers in the range $[-C', \ldots, C']$, where C' is not too large, namely $\log(C') = O(m \log n)$. This fact, combined with Theorem 3.7, implies that the number of iterations in any execution of the algorithm on (G, u, c') is $O(nm^2 \log n)$. Therefore, the number of iterations in any execution of the algorithm on (G, u, c) is $O(nm^2 \log n)$.

Open problems remain, concerning both the practical and the theoretical ramifications of our results. On the practical side, we believe that the actual performance of the algorithm described in Section 4 is worth investigating. There are some obvious modifications that should improve the performance of the method. For example, Step 1 need not be performed unless there is an admissible cycle, and Step 2 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 [16, 17] suggest the possibility of an $O(m \log(n^2/m))$ bound. (Note, however, that this bound requires a much more involved analysis. Compare, for example the arguments in [16] and [17] with those in [15], [29], and [30].) 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, for example, [13].

Although the minimum-mean cycle-canceling algorithm is primal and does not use scaling, the concepts of duality and scaling are crucial in our analysis. The use of these concepts in the analysis is natural; in fact, our discovery of Theorem 3.3 lead us to the statement of the algorithm. It is possible, however, that there is a "purely combinatorial" analysis of the algorithm that does not use these concepts. For the special case of the maximum flow problem, Edmonds and Karp [9] give such an analysis. Such an analysis would be interesting from the theoretical point of view.

We have exhibited cycle-canceling strategies that yield polynomial-time cycle-canceling algorithms. Barahona and Tardos have given another such strategy. The discovery of additional strategies of this kind would be of theoretical and potentially of practical interest.

ACKNOWLEDGMENTS. We are grateful to Éva Tardos for bringing Weintraub's paper to our attention. We also would like to thank Serge Plotkin and David Shmoys for many helpful comments on preliminary versions of this paper.

REFERENCES

1. AHUJA, R. K., GOLDBERG, A. V., ORLIN, J. B. AND TARJAN, R. E. Finding minimum-cost flows by double scaling. Tech. Rep. CS-TR-164-88. Department of Computer Science, Princeton Univ., Princeton, N.J., 1988.
2. AHUJA, R. K., MEHLHORN, K., ORLIN, J. B., AND TARJAN, R. E. Faster algorithms for the shortest path problem. Tech. Rep. CS-TR-154-88. Department of Computer Science, Princeton Univ., Princeton, N.J., 1987.
3. AHUJA, R. K., ORLIN, J. B., AND TARJAN, R. E. Improved time bounds for the maximum flow problem. *SIAM J. Comput.*, to appear.
4. BARAHONA, F. AND TARDOS, É. Note on Weintraub's minimum cost circulation algorithm. *SIAM J. Comput. 18* (1989), 579–583.
5. BERTSEKAS, D. P. Distributed Asynchronous Relaxation Methods for Linear Network Flow Problems. Tech. Rep. LIDS-P-1986, Lab. for Decision Systems. M.I.T., Cambridge, Mass., Sept. 1986 (Revised November, 1986).
6. BLAND, R. G., AND JENSEN, D. L. On the computational behavior of a polynomial-time network flow algorithm. Tech. Rep. 661. School of Operations Research and Industrial Engineering, Cornell Univ., Ithaca, N.Y. 1985.
7. BUSACKER, R. G., AND SAATY, T. L. *Finite Graphs and Networks: An Introduction with Applications.* McGraw-Hill, New York, 1965.
8. DINIC, E. A. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Dokl. 11* (1970), 1277–1280.
9. EDMONDS, J., AND KARP, R. M. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM* 19, 2 (Apr. 1972), 248–264.
10. FORD, JR., L. R., AND FULKERSON, D. R. *Flows in Networks.* Princeton Univ. Press, Princeton, N.J., 1962.
11. FREDMAN, M. L., AND TARJAN, R. E. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM* 34, 3 (July 1987), 596–615.
12. FUJISHIGE, S. A capacity-rounding algorithm for the minimum-cost circulation problem: A dual framework on the Tardos algorithm. *Math. Prog.* 35 (1986), 298–308.
13. GABOW, H. N., AND TARJAN, R. E. Faster scaling algorithms for network problems. *SIAM J. Comput.*, to appear.
14. GALIL, Z., AND TARDOS, É. An $O(n^2(m + n \log n)\log n)$ minimum cost flow algorithm. *J. ACM* 35, 2 (Apr. 1988), 374–386.
15. GOLDBERG, A. V. Efficient graph algorithms for sequential and parallel computers. Tech. Rep. TR-374, Lab. for Computer Science, M.I.T., Cambridge, Mass., 1987.

16. GOLDBERG, A. V., AND TARJAN, R. E. Finding minimum-cost circulations by successive approximation. *Math. Oper. Res.*, to appear.
17. GOLDBERG, A. V., AND TARJAN, R. E. A new approach to the maximum flow problem. *J. ACM 35*, 4 (Oct, 1988), 921–940.
18. GOLDBERG, A. V., AND TARJAN, R. E. Solving minimum-cost flow problems by successive approximation. In *Proceedings of the 19th ACM Symposium on Theory of Computing* (New York, N.Y., May 25–27). ACM, New York, 1987, pp. 7–18.
19. KARP, R. M. A characterization of the minimum cycle mean in a digraph. *Discrete Math. 23* (1978), 309–311.
20. KARP, R. M., AND ORLIN, J. B. Parametric shortest path algorithms with an application to cyclic staffing. *Discrete Appl. Math. 3* (1981), 37–45.
21. KLEIN, M. A primal method for minimal cost flows with applications to the assignment and transportation problems. *Manage. Sci. 14* (1967), 205–220.
22. LAWLER, E. L. *Combinatorial Optimization: Networks and Matroids.* Holt, Reinhart, and Winston, New York, N.Y., 1976.
23. ORLIN, J. B. Genuinely polynomial simplex and non-simplex algorithms for the minimum cost flow problem. Tech. Rep. No. 1615-84. Sloan School of Management, M.I.T., Cambridge, Mass., Dec. 1984.
24. ORLIN, J. B. On the simplex algorithm for networks and generalized networks. *Math. Prog. Studies 24* (1985), 166–178.
25. ORLIN, J. B. A faster strongly polynomial minimum cost flow algorithm. In *Proceedings of the 20th ACM Symposium on Theory of Computing* (Chicago, Ill, May 2–4). ACM, New York, 1988, pp. 377–387.
26. ORLIN, J. B., AND AHUJA, R. K. New scaling algorithms for assignment and minimum cycle mean problems. Sloan Working Paper 2019-88. Sloan School of Management, M.I.T., Cambridge, Mass., 1988.
27. PAPADIMITRIOU, C. H., AND STEIGLITZ, K. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, N.J., 1982.
28. RÖCK, H. Scaling techniques for minimal cost network flows. In *Discrete Structures and Algorithms*, U. Pape, Ed. Carl Hansen, Münich, W. Germany, 1980, pp. 181–191.
29. SLEATOR, D. D. An $O(nm \log n)$ algorithm for maximum network flow. Tech. Rep. STAN-CS-80-831. Computer Science Department, Stanford Univ., Stanford, Calif., 1980.
30. SLEATOR, D. D., AND TARJAN, R. E. A data structure for dynamic trees. *J. Comput. Syst. Sci. 26* (1983), 362–391.
31. SLEATOR, D. D., AND TARJAN, R. E. Self-adjusting binary search trees. *J. ACM 32*, 3 (July 1985), 652–686.
32. TARDOS, Ê. A strongly polynomial minimum cost circulation algorithm. *Combinatorica 5*, 3 (1985), 247–255.
33. TARJAN, R. E. *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, Pa., 1983.
34. VAN EMDE BOAS, P., KAAS, R., AND ZIJLSTRA, E. Design and implementation of an efficient priority queue. *Math. Syst. Theory 10* (1977), 99–127.
35. WEINTRAUB, A. A primal algorithm to solve network flow problems with convex costs. *Manage. Sci. 21* (1974), 87–97.

RECEIVED AUGUST 1987; REVISED NOVEMBER 1988; ACCEPTED JANUARY 1989