# A Polynomial Combinatorial Algorithm for Generalized Minimum Cost Flow [*]

Kevin D. Wayne[†]

**Abstract**

We propose the first combinatorial solution to one of the most classic problems in combinatorial optimization: the generalized minimum cost flow problem (flow with losses and gains). Despite a rich history dating back to Kantorovich and Dantzig, until now, the only known way to solve the problem in polynomial-time was via general purpose linear programming techniques. Polynomial combinatorial algorithms were previously known only for the version of our problem without costs. We design the first such algorithms for the version with costs. Our algorithms also find provably good solutions faster than optimal ones, providing the first strongly polynomial approximation schemes for the problem.

Our techniques extend to optimize linear programs with two variables per inequality. Polynomial combinatorial algorithms were previously developed for testing the feasibility of such linear programs. Until now, no such methods were known for the optimization version.

## 1 Introduction

In the traditional minimum cost flow problem, the goal is to send a single commodity from supply nodes to demand nodes without violating capacity constraints, and do so as cheaply as possible. The *generalized minimum cost flow problem* is an important generalization in which each arc $e$ also has a positive multiplier $\gamma(e)$, called a *gain factor*, associated with it. For each unit of flow that enters the arc, $\gamma(e)$ units exit. We formally define the problem in § 2. The gain factors can represent physical transformations due to leakage, evaporation, breeding, theft, or interest rates. They

[*]An extended abstract of this paper appears in the Proceedings of the 31st Annual ACM Symposium of Theory of Computing 1999.

[†]Computer Science Department, Princeton University, Princeton, NJ 08544. Research supported while the author was at Cornell in part by ONR through grant AASERT N00014-97-1-0681. Email: `wayne@cs.princeton.edu`.

can also represent transformations from one commodity to another as a result of manufacturing, scheduling, or currency exchange. For example, a gain factor of 0.96 could represent the possibility of converting one U.S. dollar into 0.96 Euros. Many applications are described in [2, 11, 12].

The problem has a distinguished history. Kantorovich [23] introduced the problem in his landmark 1939 paper, where he justified the use of optimization as a tool for planning and production: his main applications were formulated as generalized flow problems. He also proposed a dual simplex type method for the problem. In the late 1950's Dantzig [8] extended the network simplex method to handle generalized flow. Around the same time, Jewell [22] designed a combinatorial primal-dual method for the problem.

Since our problem is a special case of linear programming, it can be solved in polynomial-time using the ellipsoid [25] or interior point methods [24, 37]. These general purpose linear programming methods rely on linear algebraic techniques, e.g., Gaussian elimination. We develop fast *combinatorial algorithms* for the problem. Our combinatorial algorithms directly manipulate the underlying graph. This approach has led to superior algorithms for a variety of optimization problems, including shortest path, maximum flow, minimum spanning tree, matching, and minimum cost flow.

We design the first polynomial combinatorial algorithms for the generalized minimum cost flow problem. Our algorithms actually solve the *generalized minimum cost circulation problem*, an equivalent problem in which all supplies and demands are zero. The basic scheme is to start with a circulation (e.g., the zero flow) and repeatedly augment flow along the generalized flow analogs of paths and cycles. Our first algorithm is a direct "augmenting path" or "cycle-canceling" style algorithm. Our second algorithm is a faster version that uses scaling.

Although the best interior point methods are faster than our combinatorial algorithms for computing optimal solutions, our algorithms are faster for computing approximately optimal solutions when the size of the input numbers is relatively large. The complexity of all known polynomial linear programming algorithms depends on the size of the costs, capacities, and/or gain factors, i.e., they are not strongly polynomial. In practice, interior point methods find approximately optimal solutions faster than optimal ones. However, no theorems are known to guarantee this, even for generalized flow. As a result, only weakly polynomial algorithms were previously known, even when computing a constant factor approximation. Our combinatorial algorithms are the first strongly polynomial approximation schemes for the generalized minimum cost circulation problem. However, we note that our combinatorial approach does not imply a strongly polynomial approximation scheme for the version with supplies and demands. The straightforward reduction from the version with supplies or demands to the circulation version requires finding a feasible solution that satisfies all of the demand; it is not known how to do this in strongly polynomial time.

Using an exponential length function in a packing framework, Oldham [30] and Wayne and Fleischer [41] designed combinatorial approximation schemes for a version of the generalized minimum cost flow problem. Their version has supplies and demands, but they do not insist on obtaining feasible solutions. Instead, they construct a solution that nearly satisfies all of the demand or whose cost is nearly optimal. The added flexibility makes the problem easier. Their methods have not led to exact algorithms for the problem since the running time is exponential in the binary encoding of the precision parameter.

We generalize the minimum ratio cycle-canceling algorithms of Wallacher [38], which in turn, generalizes the minimum mean cycle-canceling algorithm of Goldberg and Tarjan [15], which in turn generalizes the shortest augmenting path algorithm of Edmonds and Karp [9]. The recent Goldberg-Rao [14] maximum flow algorithm builds upon ideas from Wallacher's algorithm. Also, different extensions of Wallacher's algorithm have recently produced efficient methods for solving unimodular linear programs [27], submodular flow [39], and certain classes of integer programs [33]. Our problem has no integrality theorem, so it does not fit into any of these categories.

The *generalized maximum flow problem* is a special case of our problem in which there are no costs. In one version of the problem, the goal is to maximize the amount of flow sent through a given arc. Goldberg, Plotkin, and Tardos [13] designed the first combinatorial polynomial algorithms for this problem. Subsequently, researchers [7, 16, 17, 18, 31, 36, 41] proposed new polynomial algorithms, also using flow-based techniques. All of these are primal-dual style algorithms. Remarkably, the specializations of our algorithms to this well-studied problem are the first strictly primal methods that run in polynomial-time, i.e., they send flow only along "generalized augmenting paths," a nonbasic version of Dantzig's primal simplex method.

In this paper we also solve the related 2VPI optimization problem. Here the goal is to maximize an arbitrary linear objective function subject to linear constraints with at most two variables per inequality. Two important special cases of the TVPI optimization problem are: the shortest path problem, and the dual of (uncapacitated) generalized minimum cost flow. The TVPI problem has some applications of its own, but is probably most important because it is one of the simplest classes of linear programs that is not well understood. Previously, researchers [3, 4, 6, 21, 29, 35] developed combinatorial algorithms for the feasibility version of the problem. However, prior to this paper, there was no way to handle arbitary objective functions without resorting to general purpose linear programming techniques. We propose the first polynomial combinatorial algorithms for the TVPI optimization problem. All previous polynomial combinatorial 2VPI feasibility algorithms are extensions of the Bellman-Ford shortest path algorithm. In contrast, our 2VPI optimization algorithms build upon existing minimum cost flow techniques.

## 2 Preliminaries

The input to the generalized minimum cost circulation problem is a generalized network $G = (V, E, u, c, \gamma)$, where $(V, E)$ is a directed graph with node set $V$ and arc set $E$, $u\colon E \rightarrow \Re_{\geq 0}$ is a *capacity function*, $c\colon E \rightarrow \Re$ is a *cost function*, and $\gamma\colon E \rightarrow \Re_{>0}$ is a *gain function.* For notational convenience we assume that there are no parallel arcs so that each arc can be uniquely specified by its endpoints. We let $n = |V|$ and $m = |E|$. A *generalized circulation* $g\colon E \rightarrow \Re_{\geq 0}$ is a nonnegative function that satisfies the *flow conservation constraints:*

$$\forall v \in V : \quad \sum_{w \in V:(v,w)\in E} g(v,w) = \sum_{w \in V:(w,v)\in E} \gamma(w,v)g(w,v).$$

It is *feasible* if, in addition, it satisfies the *capacity constraints:*

$$\forall (v,w) \in E : \quad g(v,w) \leq u(v,w).$$

We abuse notation slightly, and denote the *cost of a circulation* $g$ by $c(g) = \sum_{(v,w)\in E} c(v,w)g(v,w)$. The generalized minimum cost circulation problem is to find a feasible generalized circulation of minimum cost. We are also interested in finding provably good (but not necessarily optimal) circulations. An *$\epsilon$-optimal generalized circulation* is a feasible generalized circulation that has value within a $(1+\epsilon)$ factor of the optimum value. An *approximation scheme* is a family of algorithms that finds a $\epsilon$-optimal generalized circulation in polynomial-time for every fixed constant $\epsilon > 0$. A *fully polynomial-time approximation scheme* is an approximation scheme whose running time is polynomial in the size of the input and in $1/\epsilon$. We assume the costs and capacities are integral and that the gain factors are ratios of two integers, and we denote the biggest of these integers by $B$. For notational convenience, we assume $B \geq m \geq 2$ and use $\tilde{\mathcal{O}}(f)$ to denote $f \log^{\mathcal{O}(1)} m$. Also, we sometimes omit the adjective "generalized" if its meaning is clear from context.

## 2.1 Residual Networks

For each $e \in E$, let $\bar{e}$ denote its reverse arc. Let $\bar{E} = \{\bar{e} : e \in E\}$ denote the set of reverse arcs. The gain factor of $\bar{e}$ is $1/\gamma(e)$ and its cost is $-c(e)/\gamma(e)$. Arc $\bar{e}$ represents the possibility of pushing flow back on arc $e$. Let $g$ be a generalized circulation in network $G$. The *residual capacity function* of arc $e$ and $\bar{e}$ is defined by $u_g(e) = u(e) - g(e)$ and $u_g(\bar{e}) = \gamma(e)g(e)$, respectively. Let $E_g \subseteq E \cup \bar{E}$ denote the subset of arcs with positive residual capacity. The *residual network* is $G_g = (V, E_g, u_g, c, \gamma)$. Solving the problem in the residual network is equivalent to solving it in the original network. *Residual cycles* and *residual circulations* are cycles and circulations in the residual network.

## 2.2 Optimality Conditions.

The optimality conditions for the problem have been known since at least the 1970's, and appear in Gondran and Minoux [19]. Recall, in non-generalized networks, a feasible circulation $f$ is optimal if and only if the residual network $G_f$ contains no negative cost cycles. An analogous result holds for generalized flows.

First, we review some basic definitions. The *gain of a cycle* is the product of the gain factors of arcs participating in that cycle. A *unit-gain cycle* is a cycle whose gain is equal to one. A *flow-generating (flow-absorbing) cycle* is a cycle whose gain is greater (less) than one. By sending flow around a flow-generating (flow-absorbing) cycle, we create (destroy) flow. A *bicycle* is shown in Figure 1. It is a flow-generating cycle, a flow-absorbing cycle, and a (possibly trivial) path from the first cycle to the second. We note that the arcs in the flow-generating and flow-absorbing cycles need not be edge-disjoint. Unit-gain cycles and bicycles play the same role that cycles do for non-generalized flows. Given a residual unit-gain cycle or bicycle, we can increase flow on these arcs so that the capacity and flow conservation constraints are maintained. If in addition, at least one arc becomes saturated, this operation is called *canceling* a unit-gain cycle or bicycle.

A *circuit* is a circulation that sends positive flow only along the arcs of a single residual unit-gain cycle or bicycle. The boxed values in Figure 1 represent a circuit corresponding to the bicycle. The following conformal decomposition lemma says that any generalized circulation can be decomposed into a small collection of circuits. It follows by standard flow decomposition arguments.

**Lemma 1 (e.g., see [19]).** *A generalized circulation $g$ can be decomposed into components $g_1, \ldots, g_k$ with $k \leq m$ such that $g = \sum_i g_i$ and each component $g_i$ is a circuit that is positive only on arcs $e$ with $g(e) > 0$.*

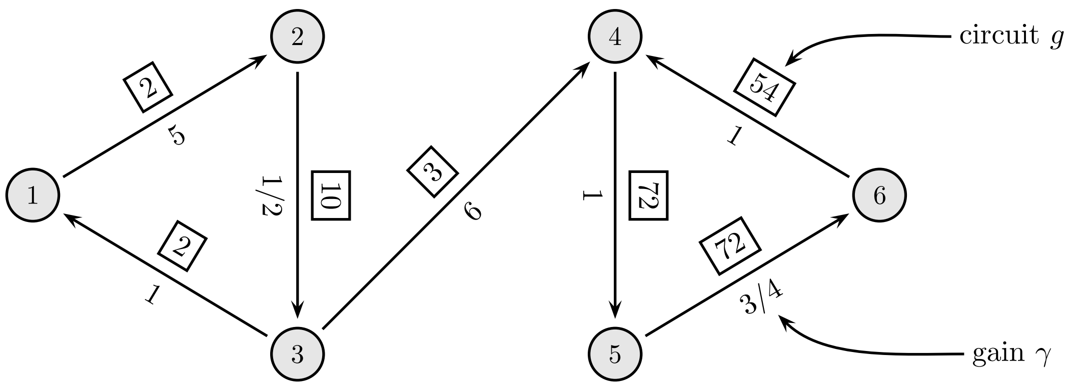

Figure 1: A bicycle and a corresponding circuit.

It is now easy to characterize the optimality conditions. Recall that the cost of a circuit $x$ is $c(x) = \sum_{e \in E} c(e)x(e)$. Note that even for circuits associated with unit-gain cycles, the cost is not in general equal to the sum of the arc costs in the cycle. Because of the gain factors, the circuit might send very different amounts of flow along different arcs in the cycle. Let $g$ be a feasible generalized circulation. Clearly the existence of a negative cost residual circuit implies that we can improve the current circulation. The converse is also true and its proof is straightforward using Lemma 1.

**Theorem 2 (e.g., see [19]).** *A feasible generalized circulation $g$ is optimal if and only if $G_g$ contains no negative cost circuits.*

## 3 Minimum-Ratio Circuit Canceling Algorithm

In this section we present a simple (but not the most efficient) "circuit-canceling" algorithm for the generalized minimum cost circulation problem.

The basic scheme is to start with a feasible circulation (e.g., the zero circulation), and then repeatedly cancel a negative cost residual circuit. This strictly improves the objective function, while maintaining feasibility. In non-generalized networks, this is known as Klein's [26] cycle-canceling algorithm. Klein's algorithm may require exponential time, but it can be refined to a polynomial algorithm by carefully choosing the negative cost cycles to cancel. Several efficient cycle selection rules are known for the traditional minimum cost circulation problem. See Shigeno, Iwata, and McCormick [34] for a recent survey of such cycle-canceling methods. The most famous of these is Goldberg and Tarjan's [15] minimum mean cost cycle rule. The *mean cost of a cycle* $\Gamma$ is $\sum_{e\in\Gamma} c(e)/|\Gamma|$, i.e., the ratio of its cost to the number of arcs. Repeatedly canceling the minimum mean cost residual cycle is a strongly polynomial algorithm for the problem. Although the algorithm extends to generalized flow, the complexity analysis does not.

Wallacher [38] proposed a different strategy, which is (apparently) more amenable for extensions to generalized flow. Given a nonnegative *time function* $t\colon E \to \Re_{\geq 0}$, the *ratio of a cycle* $\Gamma$ is defined to be $\sum_{e\in\Gamma} c(e)/\sum_{e\in\Gamma} t(e)$, i.e., the ratio of its cost to time. Finding such a cycle with arbitrary costs and nonnegative times is known as the *tramp steamer problem* and has many applications of its own [2]. If the times are chosen to be uniform for all arcs, minimum ratio cycles coincide with minimum mean cycles. Wallacher also considered choosing the time of an arc to be the reciprocal of its residual capacity. Using this novel rule, Wallacher [38] shows that repeatedly canceling a minimum ratio cycle is a (weakly) polynomial algorithm for the traditional minimum cost circulation problem. Intuitively, we would like to cancel a cycle that significantly improves the objective function. We hope to find a cycle with very negative cost and with large residual capacity; the minimum ratio cycle is a tradeoff between these two competing objectives.

A more direct way to achieve the same goal would be to cancel a *most-improving cycle*, i.e., a cycle whose cancelation improves the objective function by the maximum amount. This is Weintraub's algorithm [42]. However, it is NP-hard to find such a cycle. By solving a sequence of assignment problems, Barahona and Tardos [5] find a collection of cycles whose cancelation improves the objective function by at least as much as the most-improving cycle. However, we do not see how to find an analogous object for our problem. It is possible to find a most-improving object for generalized maximum flow [40].

Our generalized flow algorithm repeatedly cancels *minimum ratio circuits.* Motivated by the cost-to-time ratio for traditional networks, we define the *ratio of a circuit* $x$ to be:

$$\sum_{e\in E} c(e)x(e) \;\; / \;\; \sum_{e\in E} t(e)x(e)$$

and, as in [38], we choose the time of an arc to be the reciprocal of its residual capacity. Note that even for circuits corresponding to unit-gain cycles, the ratio is not simply the sum of the arc costs divided by the sum of the arc times. Because of the gain factors, canceling a circuit may send very different amounts of flow on different arcs in the unit-gain cycle. However, if every arc in the cycle has unit gain, our definition coincides with Wallacher's. In § 6 we describe an efficient method for computing a minimum ratio circuit. The method is based upon the Bellman-Ford shortest path algorithm.

The minimum ratio circuit canceling algorithm is described in Figure 2. We will show that the gap between the cost of the current circulation and the optimum geometrically decreases to zero. Because there is no integrality theorem for generalized flows and we may augment non-integral amounts of flow, we terminate our algorithm when this gap is small (but possibly nonzero). After a polynomial number of iterations, the gap will be sufficiently small, and this will guarantee that the current circulation is $\epsilon$-optimal. If we are interested in computing an optimal circulation, we can stop when $\epsilon = B^{-7m}$ and "round" the current circulation to an optimal one. Such a rounding procedure is described in § 5. Since the running time grows only logarithmically will $1/\epsilon$, this is a polynomial algorithm.

```
Input: generalized network $G$ and precision parameter $\epsilon$
Output: $\epsilon$-optimal circulation $g$
  Initialize $g \leftarrow 0$
  for $i = 1$ to $\lceil me^{-1}/2 \log(1/\epsilon) \rceil$ do
    $\forall e \in E_g : \ \ t(e) = 1/u_g(e)$
    Cancel a min ratio circuit in $G_g$
    Update $g$
  end for
```
**Figure 2:** Minimum Ratio Algorithm

The next two lemmas imply that the cost of the circulation $g$ that the algorithm maintains geometrically converges to the optimum value. The first lemma says that canceling a circuit with ratio $\mu$ improves the objective function by at least $\mu$. The second crucial lemma says that the minimum ratio circuit value is always within a factor of $m$ of the optimum value. As a result, each time we cancel a minimum ratio circuit, we capture at least a $1/m$ fraction of the remaining profit in the network. Our proofs of these lemmas are similar to those in Wallacher [38].

**Lemma 3.** *Let $g$ be a feasible circulation. Let $x$ be a negative cost circuit in $G_g$, and let $\mu$ denote its ratio. Canceling circuit $x$ improves the objective function value by at least $\mu$.*

*Proof.* First recall $x$ has negative cost so $\mu$ is negative. When $x$ is canceled, it is rescaled so that it satisfies the residual capacity constraints and saturates at least one arc. After rescaling,

$t(x) = \sum_{e \in E_g} x(e)/u_g(e) \geq 1$ since at least one residual arc is saturated. Rescaling does not affect the ratio; consequently $c(x) = \mu t(x) \leq \mu < 0$. $\square$

The key to the our complexity analysis is showing that there exists a circuit with a very negative ratio.

**Lemma 4.** *Let $g$ be a feasible circulation that is not optimal. Let $\mu^*$ denote the minimum ratio circuit value in $G_g$. Then* $\mathrm{OPT}(G_g) \leq \mu^* \leq \mathrm{OPT}(G_g)/m < 0$.

*Proof.* The first inequality follows from Lemma 3. The third inequality follows from the sub-optimality of $g$. Now, let $x^{\mathrm{OPT}}$ be an optimal circulation in $G_g$. Since any circulation can be decomposed into circuits (see Lemma 1), the value of the minimum ratio circulation is equal to the value of the minimum ratio circuit. Then, by definition of $\mu^*$, we have $\mu^* \leq c(x^{\mathrm{OPT}})/t(x^{\mathrm{OPT}})$. Recall $t(x^{\mathrm{OPT}}) = \sum_e x^{\mathrm{OPT}}(e)/u_g(e)$. Since $x^{\mathrm{OPT}}$ is feasible, we have $t(x^{\mathrm{OPT}}) \leq m$. Combining these two inequalities and recalling that $c(x^{\mathrm{OPT}})$ is negative yields:

$$\mu^* \leq c(x^{\mathrm{OPT}})/t(x^{\mathrm{OPT}}) \leq c(x^{\mathrm{OPT}})/m = \mathrm{OPT}(G_g)/m.$$

$\square$

Lemma 4 says that value of the minimum ratio circuit $\mu^*$ and the optimality gap $\mathrm{OPT}(G_g)$ are always within a factor of $m$ of each other. Since the optimality gap geometrically decreases to zero, so does $\mu^*$. So while our algorithm does not explicitly use a scaling parameter, it implicitly drives the parameter $\mu^*$ to zero.

**Theorem 5.** *The minimum ratio circuit-canceling algorithm computes an $\epsilon$-optimal generalized minimum cost circulation in $\tilde{\mathcal{O}}(m^2 n^3 \log(1/\epsilon))$ time. In $\tilde{\mathcal{O}}(m^3 n^3 \log B)$ time it computes an optimal circulation.*

*Proof.* Lemma 4 says that the minimum ratio circuit value is always within a factor of $m$ of the optimality gap. Lemma 3 says that if we cancel a circuit of ratio $\mu$, then the optimality gap decreases by at least that quantity. Combining these two lemmas, we see that each iteration captures at least a $1/m$ fraction of the remaining profit. In other words, each iteration decreases the optimality gap by at least a $1 - 1/m$ factor. Thus, there are $\mathcal{O}(m \log(1/\epsilon))$ iterations since $(1 - 1/m)^m \geq 1/(2e)$ for $m \geq 2$. The bottleneck computation in each iteration is computing a minimum ratio circuit; in § 6 we propose a $\tilde{\mathcal{O}}(mn^3)$ time subroutine. This proves the first part of the theorem.

We will show in § 5 that the value of a vertex in the underlying polytope requires only $\mathcal{O}(m \log B)$ bits of precision. The resulting Lemma 11 shows how a $B^{-7m}$-optimal circulation can be rounded to an optimal vertex by canceling at most $m$ additional circuits. This proves the second part of the theorem. $\square$

## 4 A Scaling-Version

In this section we propose a faster version of the minimum ratio algorithm. The overwhelming bottleneck in the original algorithm is computing minimum ratio circuits. The crucial idea needed for improving the complexity is that we do not really need to cancel the exact minimum ratio circuit. The same idea was used by Wallacher [38] for non-generalized flows. Canceling such a circuit would improve the objective function by at least $\mu^*$, where $\mu^*$ is the minimum ratio circuit value. However, this requires $\tilde{\mathcal{O}}(mn^3)$ time. Our improved version cancels circuits in phases. Each phase is characterized by a scaling parameter $\mu < 0$, which at the beginning of a phase is at least as small as $\mu^*/2$. Instead of canceling the minimum ratio circuit, we cancel one that has ratio at least as small as $\mu$. To do this we introduce the cost function $c_\mu(e) := c(e) - \mu t(e)$. We cancel negative cost circuits with respect to these modified costs $c_\mu$. It is easy to see that such negative cost circuits have ratio at least as small as $\mu$. Now, finding negative cost circuits requires only $\tilde{\mathcal{O}}(mn^2)$ time and speeds up the overall algorithm by a factor of $n$. We halve $\mu$ when it becomes an overestimate of the current minimum ratio circuit value. When $\mu$ is sufficiently close to zero, the optimality gap is small enough to guarantee an approximately optimal solution. The scaling algorithm is given in Figure 3.

**Lemma 6.** *There are at most $2m$ cancelations per scaling phase.*

*Proof.* In a $\mu$-phase, each circuit canceled has ratio at least as small as $\mu$. Lemma 3 implies that this improves the objective function by at least $\mu$.

At the end of a $2\mu$ scaling phase, there are no negative $c_{2\mu}$-cost residual circuits. This implies $2\mu \leq \mu^*$, where $\mu^*$ is the value of the minimum ratio circuit at the end of the phase. Combining this with Lemma 4, we obtain $2m\mu \leq \mathrm{OPT}(G_g) < 0$.

Now, at the beginning of a $\mu$-phase, we know that the objective function can be improved by at most $2m\mu$. Since, each cancelation improves the objective function by at least $\mu$, there can be at most $2m$ cancelations per phase. $\square$

```
Input: generalized network $G$ and precision parameter $\epsilon$
Output: $\epsilon$-optimal flow $g$
  Initialize $g \leftarrow 0, \mu \leftarrow$ min ratio circuit value
  $\forall e \in E_g : \quad t(e) = 1/u_g(e), \quad c_\mu(e) = c(e) - \mu t(e)$
  repeat
    while $\exists$ negative $c_\mu$-cost residual circuit do
      Cancel a negative cost (with respect to $c_\mu$) circuit in $G_g$
      Update $g$, $t$, and $c_\mu$
    end while
    $\mu \leftarrow \mu/2$
  until $\mu \geq \frac{\epsilon}{m} c(g)$     (g is $\epsilon$-optimal)
```

**Figure 3:** Scaling Min Ratio Circuit-Canceling Algorithm

The next lemma shows that when our algorithm terminates, the circulation $g$ is $\epsilon$-optimal.

**Lemma 7.** *Suppose at the end of a $\mu$ scaling phase we have $\mu \geq \frac{\epsilon}{2m} c(g)$, where $g$ is the current circulation. Then $g$ is $\epsilon$-optimal.*

*Proof.* Let $\mu^*$ denote the value of the minimum ratio circuit at the end of the $\mu$-scaling phase. As in the proof of Lemma 6 we have $2\mu \leq \mu^*$. Now,

$$\frac{\epsilon}{m} c(g) \quad \leq \quad 2\mu \leq \mu^* \quad \leq \quad \frac{\mathrm{OPT}(G_g)}{m}.$$

The third inequality follows from Lemma 4. Multiplying through by $m$ and using the fact that $\mathrm{OPT}(G) = \mathrm{OPT}(G_g) + c(g)$ we obtain $\mathrm{OPT}(G) \geq (1 + \epsilon)\, c(g)$ as claimed. □

**Theorem 8.** *The scaling algorithm finds an $\epsilon$-optimal circulation in $\tilde{\mathcal{O}}(m^2 n^2 \log(1/\epsilon))$ time. It finds an optimal circulation in $\tilde{\mathcal{O}}(m^3 n^2 \log B)$ time.*

*Proof.* The bottleneck computation is detecting a negative cost circuit. In § 6 we show that this can be done in $\tilde{\mathcal{O}}(mn^2)$ time using a Bellman-Ford style algorithm. There are at most $2m$ cancelations per phase.

We complete the proof by showing that there are $\mathcal{O}(\log(m/\epsilon))$ scaling phases. Let $\mu_0$ be the value of the minimum ratio circuit in the original network $G$, and let $g_0$ be circulation just after canceling the first circuit. By Lemma 3 we have $\mu_0 \geq c(g_0)$. The parameter $\mu$ halves in each phase, so after $\mathcal{O}(\log(2m/\epsilon))$ phases we have $\mu \geq \frac{\epsilon}{2m} c(g_0) \geq c(g)$, where the last inequality follows since the objective function is monotone decreasing throughout the algorithm. This implies that after $\mathcal{O}(\log(2m/\epsilon))$ iterations, the stopping condition $\mu \geq \frac{\epsilon}{2m} c(g)$ is satisfied. Lemma 7 guarantees $g$ is $\epsilon$-optimal when the stopping condition is satisfied. □

## 5 Rounding to a Vertex

In this section, we propose a method to convert one generalized circulation into another circulation that is a vertex of the underlying polytope. The cost of the vertex we obtain will be no larger than the cost of the original circulation. In linear programming terminology, this process is called *purification*. Our purification method uses flow-type techniques instead of linear algebra. This purification process is useful to "round" an essentially optimal circulation into an optimal one. First, it is easy to characterize the vertices of the underlying polytope.

**Lemma 9.** *Let $g$ be a feasible generalized circulation and let $A := \{e \in E_g : 0 < g(e) < u(e)\}$ be the subset of arcs that have residual capacity in both directions. Then $g$ is a vertex if and only if the subgraph induced by $A$ has no unit-gain cycles or bicycles.*

Now, we show how to "round" a generalized circulation into a vertex. Let $g$ be a feasible circulation, and let $A$ be defined as above. If there are no unit-gain cycles or bicycles in the subgraph induced by $A$, then $g$ is a vertex. Otherwise, we cancel such a unit-gain cycle or bicycle (in a direction that improves or does not change the objective function). We update $A$, and repeat this process until we obtain a vertex. There are at most $m$ iterations, since each cancelation saturates at least one new arc in $A$. The bottleneck computation is detecting unit-gain cycles or bicycles. In § 6, we describe how to do this in $\mathcal{O}(mn)$ time using two Bellman-Ford computations. This subroutine is a factor of $n$ faster finding negative cost unit-gain cycles or bicycles.

**Lemma 10.** *Given a feasible generalized circulation $g$ of cost $c(g)$, we can find another feasible circulation $g'$ that is a vertex and has cost no more than $c(g)$ in $\mathcal{O}(m^2 n)$ time.*

The following lemma establishes the precision necessary to be able to "round" to an optimal circulation.

**Lemma 11.** *Given a $B^{-7m}$-optimal generalized circulation $g$, we can determine an optimal circulation in $\mathcal{O}(m^2 n)$ time.*

*Proof.* First, applying Lemma 10, we find a vertex $g'$ that has cost at least as good as $g$. The optimum value can be bounded from below by $-mB^2$, since each arc has capacity at most $B$ and cost at least $-B$. Thus,

$$c(g') \leq c(g) \leq \frac{OPT}{1+B^{7m}} \leq OPT(1-B^{-7m}) \leq OPT + mB^2 B^{-7m} < OPT + B^{-4m}.$$

The last inequality follows from our assumption that $m \leq B$. By Kramer's rule, the cost of each vertex (including $g'$ and some optimal circulation) is a rational number, and these costs have a common denominator no bigger than $B^{4m}$. Thus, $c(g') = OPT$ and $g'$ is optimal. $\square$

## 6 Minimum Ratio Circuit Subroutine

In this section we describe a combinatorial method for computing minimum ratio circuits. First, we discuss how to detect circuits and negative cost circuits. As we will see, detecting negative cost circuits is intimately related to testing the feasibility of 2VPI (two variable per inequality) linear programs.

### 6.1 Detecting a Circuit.

We describe how to detect a circuit in $\mathcal{O}(mn)$ time using two Bellman-Ford style computations. This will provide some intuition for detecting a minimum ratio circuit. Also, the subroutine was used in § 5 when we showed how to "round" a nearly-optimal circulation into an optimal one.

The basic approach is to first identify a bicycle if one exists; if we don't find one, then we detect unit-gain cycles. We must do the computation in this order, since for general networks it is NP-hard to detect unit-gain cycles. The NP-hardness can be establish by reducing SUBSET-PRODUCT (see problem SP14 in Garey and Johnson [10]) to the unit-gain cycle problem.

**Step 1.** First, we describe how to detect bicycles. Recall, a bicycle is a flow-generating cycle and a flow-absorbing cycle connected by a path from the first cycle to the second. Let $N$ denote the subset of nodes that either participate in a flow-absorbing cycle or can reach one along a path. We compute the subset $N$ using a single Bellman-Ford style computation, with the following distance update rule along arc $(v, w)$:

$$\pi(v) \leftarrow \min\{\pi(v),\ \pi(w)\gamma(v,w)\}.$$

Initially we set $\pi(v) = 1$ for all nodes $v \in V$. Then we run $2n$ Bellman-Ford style iterations as shown in Figure 4. The subset $N$ consists of those node potentials that were changed between iterations $n$ and $2n$. Such vertices must either participate in a flow-absorbing cycle or have a path leading to one. This observation is well-known in traditional networks. I.e. if we use the lengths $l = +\log \gamma$ then, flow-absorbing cycles are in one-to-one correspondence with negative length cycles.

**Input:** generalized network $G$

**Output:** subset of nodes that can reach a flow-absorbing cycle via a directed path

```
Initialize $\forall v \in V :$   $\pi_0(v) \leftarrow 0$
for $i = 1$ to $2n$ do
  $\forall e \in E :$   $\pi_i(v) \leftarrow \min\{\pi_{i-1}(v),\ \pi_{i-1}(w)\gamma(v,w)\}$
end for
output $N \leftarrow \{\, v \,:\, \pi_{2n}(v) < \pi_n(v) \,\}$
```

**Figure 4:** Finding subset of nodes that participate in a flow-absorbing cycle or can reach one via a directed path.

Now, we detect flow-generating cycles in the subgraph induced by $N$. Obviously, if we find such a flow-generating cycle, we can find a bicycle by connecting it to a flow-absorbing cycle in $N$. By the definition of $N$, such a flow-absorbing cycle will exist. Again, detecting flow-generating cycles can be done with a single Bellman-Ford style computation. As before, we initialize $\pi(v) = 1$ for all nodes $v \in N$. Now, we use the following modified update rule along arc $(v, w)$ :

$$\pi(w) \leftarrow \max\{\pi(w),\ \pi(v)\gamma(v,w)\}.$$

We run $n$ Bellman-Ford style iterations. If we don't detect a flow-generating cycle after $n$ iterations, then we conclude no such cycles exist.

**Step 2.** Now, assuming there are no bicycles, we show how to detect unit-gain cycles in linear time, using the information extracted from the two previous Bellman-Ford computations. First, we detect unit-gain cycles in the subgraph induced by $V \setminus N$. This problem is equivalent to detecting zero length cycles in a traditional network using lengths $l = +\log \gamma(v, w)$. By the definition of $N$, this subgraph has no negative length cycles. Thus, there exists node potentials $\pi$ such that the *reduced lengths* $l_\pi(v, w) := \log \gamma(v, w) - \pi(v) + \pi(w) \geq 0$ for all arcs $(v, w) \in G(V \setminus N, E)$. Moreover, the first Bellman-Ford style computation produces them. Now, identifying a zero length cycle is equivalent to finding a cycle in the subgraph induced by zero reduced length arcs. Using depth first search, we can do this in linear time. The second case is to detect unit-gain cycles in the subgraph induced by $N$. In this subgraph, there are no flow-generating cycles, since otherwise we would have detected a bicycle. By a similar argument as in the first case, there exist node potentials $\pi'$ such that the reduced lengths $l'_{\pi'}(v,w) := -\log\gamma(v,w) - \pi'(v) + \pi'(w) \geq 0$. for all arcs $(v,w) \in G(N,E)$. The second Bellman-Ford computation produces these potentials. As above, we can use depth first search to detect unit-gain cycles in this subgraph.

## 6.2 Detecting a Negative Cost Circuit

Now, we describe how to detect a negative cost circuit. First, we note that it is sufficient to detect a negative cost circulation, since, by Lemma 2, any circulation can be decomposed into circuits. That is, we want to find a solution to system (I) below.

$$c(x) < 0, x \text{ circulation} \tag{I}$$

$$\forall (v,w) \in E : c^{\pi}(v,w) := c(v,w) + \pi(v) - \gamma(v,w)\pi(w) \geq 0. \tag{II}$$

Note that system(II) has two variables per inequality. The 2VPI system (II) is related to system (I) through linear programming duality. Linear programming weak duality asserts that for any cost function $c$ and gain function $\gamma$, systems (I) and (II) can't both be feasible. This is easy to verify, since if system (I) and (II) are both feasible, we have:

$$\begin{aligned}
0 &\leq \sum_{(v,w)\in E} c^{\pi}(v,w)x(v,w) \\
&= \sum_{(v,w)\in E} c(v,w)x(v,w) + \sum_{(v,w)\in E} \pi(v)x(v,w) - \sum_{(v,w)\in E} \gamma(v,w)\pi(w)x(v,w) \\
&= c(x) + \sum_{v\in V} \pi(v) \sum_{w\in V:(v,w)\in E} x(v,w) \ - \sum_{v\in V} \pi(v) \sum_{w\in V:(w,v)\in E} \gamma(w,v)x(w,v) \\
&= c(x) \\
&< 0.
\end{aligned}$$

The first inequality follows since both $c^{\pi}$ and $x$ are nonnegative. The last equality follows from flow conservation. Intuitively, variable $\pi(v)$ represents the market price for the commodity at node $v$.

The quantity $c^{\pi}(v,w)$ represent the cost of buying one unit at node $v$, shipping it to node $w$, and then selling off the $\gamma(v,w)$ units that arrive at node $w$. Linear programming strong duality asserts that if there are no negative cost circulations, then there exist a set of market prices for which there is no incentive to buy, ship, and sell. That is, there exists a negative cost circulation if and only if the 2VPI system (II) is infeasible. Thus, we could determine the existence of a negative cost circuit if we had a 2VPI feasibility subroutine. Fortunately, researchers have previously developed such specialized 2VPI feasibility algorithms. Moreover, all of these algorithms either return a feasible solution to (II) or output a minimal "certificate of infeasibility." The certificate corresponds to a negative cost circuit.

## 6.3 2VPI Feasibility

To gain intuition and perspective, we give a brief review of the fundamental idea behind these flow-based 2VPI feasibility algorithms. We refer the reader to [21, 6] for more details. They are all based on the Bellman-Ford shortest path algorithm. To see the connection, we formulate the shortest path problem as a 2VPI linear program. The decision variables are the node labels $\{\pi(v) : v \in V\}$. For each $(v,w) \in E$, the node labels must satisfy the metric inequality $\pi(w) \leq \pi(v)+d(v,w)$. Note that all of the coefficients are either 0, +1, or -1. Detecting a negative cost cycle is equivalent to testing the feasibility of the associated 2VPI linear program, and can be done using the Bellman-Ford algorithm.

Building upon the Bellman-Ford algorithm, researchers have designed algorithms to detect the feasibility of general 2VPI linear programs. For each 2VPI system, there is an associated directed network: variables correspond to nodes, and two-variable inequalities correspond to arcs between nodes of participating variables. The orientation of the arcs depend on the sign of the multiplier. The algorithms exploit the graph structure by "propagating inequalities" along paths and cycles, using a Bellman-Ford style approach. Recall, given distances $d(v,w)$, the Bellman-Ford algorithm computes the shortest path distances $\{\pi(v) : v \in V\}$ from the source $s$ to all other nodes. Initially $\pi(s) = 0$ and all other distance labels are infinite. Then it propagates distance information using the metric inequality: $\pi(w) \leq pi(v) + \pi(v,w)$.

The propagation method naturally extends to handle more general two variable inequalities. Suppose we have an upper bound on $\pi(v)$. Then we can use the inequality $\pi(w) \leq \gamma(v,w)\pi(v) + c(v,w)$ to propagate the distance information, and potentially obtain a new upper bound for $\pi(w)$.

When all coefficients are 0, $+1$, or $-1$, propagating inequalities around a cycle either leads to a redundant inequality or implies that the underlying system is infeasible. If arbitary coefficients are permitted, the analogous objects are unit-gain cycles and bicycles. That is, tracing inequalities around one of the objects leads to a redundant inequality or proves that the underlying system is infeasible. This fact follows from Lemma 1. Non-unit gain cycles also play an important role. For example, consider the three inequalities: $2\pi(x) \leq 6\pi(y)+5$, $6\pi(y) \leq 14\pi(z)+16$, and $14\pi(z) \leq \pi(x)$. They form a cycle in the associated graph, and summing up the three inequalities implies $\pi(x) \leq 21$. Tracing inequalities around cycles provides a method for obtaining upper and lower bounds on the variables. It is this difficulty that makes the 2VPI feasibility problem more difficult than the shortest path problem. Shostak [35] proved that the 2VPI feasibility problem can be solved by tracing only simple paths and cycles in the underlying graph; this has been the foundation for all subsequently considered algorithms for the problem. It was later refined to a polynomial algorithm by Aspvall and Shiloach [4]. Currently, the best known complexity bound is $\tilde{\mathcal{O}}(mn^2)$, due to Hochbaum and Naor [21] and Cohen and Megiddo [6]. The latter algorithm has a parallel running time of $\tilde{\mathcal{O}}(n)$ using $\mathcal{O}(mn)$ processors. Later, we will use this parallel version to speed up our sequential minimum ratio circuit algorithm. We also note that Cohen and Megiddo give an improved version of their algorithm using randomization, achieving a $\tilde{\mathcal{O}}(n^3+mn)$ expected running time. Using this randomized version, we can improve our complexity bound for the scaling algorithm accordingly.

## 6.4 Finding a Minimum Ratio Circuit

Now, we describe how to identify a minimum ratio circuit. This subroutine is used in our first algorithm (Figure 2), but is not essential to our faster version (Figure 3), except to obtain the strongly polynomial bound on computing approximately optimal flows. Our method combines a negative cost circuit detection subroutine with binary search. We obtain a faster and strongly-polynomial algorithm by using Megiddo's parametric search instead of binary search.

To find the optimal ratio $\mu^*$, we first estimate $\mu^*$ with a guess $\mu$, and then linearize the objective function with the parametric cost function $c_\mu(e) := c(e) - \mu t(e)$. It is easy to see that a circuit has ratio less than $\mu$ if and only if there exists a negative cost circuit with respect to the cost function $c_\mu$. In a preceding subsection, we discussed a $\tilde{\mathcal{O}}(mn^2)$ subroutine for detecting negative cost circuits. We use this subroutine in the network with cost function $c_\mu$. If there is a negative cost circuit, then $\mu$ overestimates $\mu^*$; otherwise, if every non-trivial circuit has positive cost, then $\mu$ is too small;

otherwise, $\mu^* = \mu$. This provides a binary search framework and we can find $\mu^*$ in $\tilde{\mathcal{O}}(m^2 n^2 \log B)$ time, since $\mathcal{O}(m \log B)$ bits of precision are required. By incorporating the parametric search techniques of Megiddo [28] and using the parallel implementation of the Cohen-Megiddo [6] 2VPI feasibility subroutine, we can improve this bound to $\tilde{\mathcal{O}}(mn^3)$.

## 7 Optimizing 2VPI Linear Programs

In this section we outline how our algorithms can be extended to optimize 2VPI linear programs of the form:

$$\begin{array}{ll} \min & \sum_{v \in V} b(v)\pi(v) \\ \forall (v,w) \in E: & \pi(v) - \gamma(v,w)\pi(w) \geq c(v,w), \end{array} \tag{$TVPI$}$$

where $\{\pi(v) : v \in V\}$ is the set of decision variables, $E$ indexes the set of constraints, $\gamma\colon E \to \Re$ is the vector of multipliers, $c\colon E \to \Re$ is the vector of right hand sides, and $b\colon V \to \Re$ is the vector of objective function coefficients. We first relate the uncapacitated generalized minimum cost flow problem (P) to (TVPI).

$$\begin{array}{llll} \max & \sum_{(v,w) \in E} c(v,w) g(v,w) & & \\ \forall v: & \sum\limits_{w:(v,w) \in E} g(v,w) - \sum\limits_{w:(w,v) \in E} \gamma(w,v) g(w,v) & = & b(v) \\ \forall (v,w) \in E: & g(v,w) & \geq & 0. \end{array} \tag{$P$}$$

(P) and (TVPI) are almost dual linear programs. The only obstacle is that we assumed the gain factors $\gamma$ in (P) are positive, whereas the multipliers in (TVPI) can be positive or negative. Therefore, the inequalities in the dual linear program of (P) are all *monotone*, i.e., each two variable inequality has one positive coefficient and one negative coefficient. Thus, our algorithms from §3 and §4 extend to optimize monotone 2VPI linear programs, since the algorithms simultaneously find primal and dual optimal solutions. We show they also extend to the non-monotone case.

We sketch below how to extend our algorithm to solve the 2VPI optimization problem. The critical idea needed to handle the non-monotone case is to allow negative gain factors. This is analogous to the "bidirected generalized networks" proposed by Cohen and Megiddo [7] for solving a generalized version of the shortest path problem. Intuitively, an arc with a negative gain factor represents the possibility of destroying flow at both of its endpoints. Its "residual arc" offers the possibility of creating flow at both of its endpoints. To handle this, we extend the generalized minimum cost flow problem to allow negative gain factors. To undo the flow shipped across a negative gain arc, we must ship a negative amount of flow back along the "residual arc." Thus, we allow arcs to have nonpositivity constraints, in addition to the usual nonnegativity constraints.

We also need to generalize our definition of the residual network. As before, for each $e \in E$, we let $\bar{e}$ denote its reverse arc and $\bar{E}$ denote the set of reverse arcs as before. Also, the gain factor of $\bar{e}$ is $1/\gamma(e)$ and its cost is $-c(e)/\gamma(e)$. Let $g$ be a generalized circulation in network $G$. We extend the definition of $E_g$ to be the subset of arcs with nonzero residual capacity; this includes arcs with negative residual capacity. The residual capacity function and residual network are defined as before. Solving the problem in the residual network is equivalent to solving it in the original network.

We assume without loss of generality that $b = 0$. If not, we could add artificial self-loop arcs and assign them very high cost. Then, a feasible solution is easy to identify. The residual problem is now of the desired form. We note, however, that this transformation is not approximation preserving.

Now, the algorithm is exactly as before except that we need to abstract our definition of circulation and circuit to allow for the negative gain factors and nonpositive capacity constraints. Our previous algorithms and analyses remain essentially unchanged.

**Theorem 12.** *There exists a polynomial combinatorial algorithm to optimize linear programs with two variables per inequality. It runs in $\tilde{O}(m^3 n^2 \log B)$ time, where $n$ is the number of variables and $m$ is the number of inequalities.*

In fact, Hochbaum et al. [20] gave a transformation which shows that optimizing general 2VPI linear programs is no harder than optimizing monotone 2VPI ones. Our algorithms easily extend to directly solve the seemingly more general version so we do not need to use their reduction.

## 8 An Open Problem

Currently, the existence of a strongly-polynomial algorithm (combinatorial or linear programming based) for generalized minimum cost flow and 2VPI optimization remains a challenging open question. The question is even unresolved for generalized maximum flow. On the other hand, strongly-polynomial algorithms are known for traditional minimum cost flow and the 2VPI feasibility problem.

McCormick and Shioura [27] showed that the minimum ratio algorithm is not a strongly polynomial algorithm for the (non-generalized) minimum cost circulation problem. They construct a network with irrational data and show the the minimum ratio algorithm does not terminate in finite time. We can modify our algorithm by rounding the circulation to a better vertex at each iteration, as in Section § 5. Now, our modified algorithm terminates, even for irrational data, since it can visit each vertex at most once. Thus, the counterexample in McCormick and Shioura [27] does not apply.

**Acknowledgments.** The author is thankful to Éva Tardos and Tom McCormick for helpful discussions and comments on an earlier version of this paper. The author also thanks the anonymous referees for many helpful suggestions for improving the presentation of this paper.

## References

[1] *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[3] B. Aspvall. *Efficient algorithms for certain satisfiability and linear programming problems.* PhD thesis, Department of Computer Science, Stanford University, 1980.

[4] B. Aspvall and Y. Shiloach. A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality. *SIAM Journal on Computing*, 9:827–845, 1980.

[5] F. Barahona and É. Tardos. Note on Weintraub's minimum-cost circulation algorithm. *SIAM Journal on Computing*, 18:579–583, 1989.

[6] E. Cohen and N. Megiddo. Improved algorithms for linear inequalities with two variables per inequality. *SIAM Journal on Computing*, 23:1313–1347, 1994.

[7] E. Cohen and N. Megiddo. New algorithms for generalized network flows. *Mathematical Programming*, 64:325–336, 1994.

[8] G. B. Dantzig. *Linear programming and extensions.* Princeton University Press, Princeton, NJ, 1963.

[9] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19:248–264, 1972.

[10] M. S. Garey and D.S. Johnson. *Computers and intractability: A guide to the theory of NP-compleness.* W. H. Freeman, New York, 1979.

[11] F. Glover, J. Hultz, D. Klingman, and J. Stutz. Generalized networks: A fundamental computer based planning tool. *Management Science*, 24:1209–1220, 1978.

[12] F. Glover, D. Klingman, and N. Phillips. Netform modeling and applications. *Interfaces*, 20:7–27, 1990.

[13] A. V. Goldberg, S. A. Plotkin, and É. Tardos. Combinatorial algorithms for the generalized circulation problem. *Mathematics of Operations Research*, 16:351–379, 1991.

[14] A. V. Goldberg and S. Rao. Beyond the flow decomposition barrier. *Journal of the ACM*, 45:753–782, 1998.

[15] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *Journal of the ACM*, 36:388–397, 1989.

[16] D. Goldfarb and Z. Jin. A faster combinatorial algorithm for the generalized circulation problem. *Mathematics of Operations Research*, 21:529–539, 1996.

[17] D. Goldfarb, Z. Jin, and Y. Lin. A polynomial dual simplex algorithm for the generalized circulation problem. Technical report, Department of Industrial Engineering and Operations Research, Columbia University, 1998.

[18] D. Goldfarb, Z. Jin, and J. B. Orlin. Polynomial-time highest gain augmenting path algorithms for the generalized circulation problem. *Mathematics of Operations Research*, 22:793–802, 1997.

[19] M. Gondran and M. Minoux. *Graphs and Algorithms.* Wiley, New York, 1984.

[20] D. S. Hochbaum, N. Megiddo, J. Naor, and A. Tamir. Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality. *Mathematical Programming*, 62:69–83, 1993.

[21] D. S. Hochbaum and J. Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM Journal on Computing*, 23-6:1179–1192, 1994.

[22] W. S. Jewell. Optimal flow through networks with gains. *Operations Research*, 10:476–499, 1962.

[23] L. V. Kantorovich. Mathematical methods of organizing and planning production. *Publication House of the Leningrad State University*, page 68, 1939. Translated in *Management Science*, 6:366–422, 1960.

[24] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[25] L. G. Khachiyan. Polynomial algorithms in linear programming. *Zh. Vychisl. Mat. Mat. Fiz.*, 20:53–72, 1980.

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

[27] S. T. McCormick and A. Shioura. Minimum ratio canceling is polynomial for linear programming, but not strongly polynomial, even for networks. Unpublished manuscript, 1999.

[28] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30:852–865, 1983.

[29] N. Megiddo. Toward a genuinely polynomial algorithm for linear programming. *SIAM Journal on Computing*, 12:347–353, 1983.

[30] J. D. Oldham. Combinatorial approximation algorithms for generalized flow problems. In ACM/SIAM [1], pages 704–714.

[31] T. Radzik. Faster algorithms for the generalized network flow problem. *Mathematics of Operations Research*, 23:69–100, 1998.

[32] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Chichester, 1986.

[33] A. S. Schulz and R. Weismantel. An oracle-polynomial time augmentation algorithm for integer programming. In ACM/SIAM [1], pages 967–968.

[34] M. Shigeno, S. Iwata, and S. T. McCormick. Relaxed most negative cycle and most positive cut canceling algorithms for minimum cost flow. Unpublished manuscript, 1998.

[35] R. Shostak. Deciding linear inequalities by computing loop residues. *Journal of the ACM*, 28:769–779, 1981.

[36] É. Tardos and K. D. Wayne. Simple generalized maximum flow algorithms. In *7th International Integer Programming and Combinatorial Optimization Conference*, pages 310–324, 1998.

[37] P. M. Vaidya. Speeding up linear programming using fast matrix multiplication. In *30th Annual IEEE Symposium on Foundations of Computer Science*, pages 332–337, 1989.

[38] C. Wallacher. A generalization of the minimum-mean cycle selection rule in cycle canceling algorithms. Technical report, Institute für Angewandte Mathematik, Technische Universität Braunschweig, 1989.

[39] C. Wallacher and U. T. Zimmermann. A polynomial cycle canceling algorithm for submodular flows. To appear in Mathematical Programming, 1999.

[40] K. D. Wayne. *Generalized Maximum Flow Algorithms*. PhD thesis, Department of Operations Research and Industrial Engineering, Cornell University, 1999.

[41] K. D. Wayne and L. Fleischer. Faster approximation algorithms for generalized flow. In ACM/SIAM [1], pages 981–982.

[42] A. Weintraub. A primal algorithm to solve network flow problems with convex costs. *Management Science*, 21:87–97, 1974.