MIT/LCS/TM-358

# COMBINATORIAL ALGORITHMS FOR THE GENERALIZED CIRCULATION PROBLEM

Andrew V. Goldberg
Serge A. Plotkin
Eva Tardos

May 1988

# Combinatorial Algorithms for the Generalized Circulation Problem

*Andrew V. Goldberg*[*]
Computer Science Department
Stanford University
Stanford, CA 94305

*Serge A. Plotkin*[†]
Laboratory for Computer Science
M.I.T.
Cambridge, MA 02139

*Éva Tardos*[‡]
Department of Mathematics
M.I.T.
Cambridge, MA 02139
and
Computer Science Department
Eötvös University
Budapest

**Abstract**

We consider a generalization of the maximum flow problem in which the amounts of flow entering and leaving an arc are linearly related. More precisely, if $x(e)$ units of flow enter an arc $e$, $x(e)\gamma(e)$ units arrive at the other end. For instance, nodes of the graph can correspond to different currencies, with the multipliers being the exchange rates. We require conservation of flow at every node except a given *source* node. The goal is to maximize the amount of flow excess at the source.

This problem is a special case of linear programming, and therefore can be solved in polynomial time. In this paper we present the first polynomial time *combinatorial* algorithms for this problem. The algorithms are simple and intuitive.

*Keywords*: Combinatorial optimization, network flow, graph algorithms.

[*]Supported by NSF Presidential Young Investigator Award.

[†]Supported by DARPA Contract N00014-87-K-825 and by ONR Contract N00014-86-K-0593.

[‡]Research partially supported by Air Force contract AFOSR-86-0078.

## 1 Introduction

Consider the following problem in financial analysis. An investor wants to take advantage of the discrepancies in prices of securities at different stock exchanges and of the currency conversion rates. His objective is to maximize his profit by trading at different exchanges and by converting currencies. The generalized circulation problem, considered in this paper, models the above situation, assuming that a bounded amount of money is available to the investor and that bounded amounts of securities can be traded without affecting the prices.

The *generalized circulation problem* is a generalization of the maximum flow problem. Each arc $(v, w)$ in the network has a *gain* factor $\gamma(v, w)$ and a *capacity* $u(v, w)$. If $x$ units of flow enter the arc at $v$, then $x \cdot \gamma(v, w)$ units of flow arrive at $w$. The graph has a special node, called the *source*. The objective is to find a flow that maximizes the excess at the source. (Alternative formulations of the problem are discussed in Section 2.4.) In the model of the above financial analysis problem, nodes correspond to different currencies and securities, and arcs correspond to possible transactions. Gain factors represent the prices or the exchange rates. For example, an arc with gain factor of 113 from a vertex representing IBM stocks to a vertex representing U.S. dollars models the possibility of selling the stocks at \$113 per share.

The generalized circulation problem is important both from practical and theoretical points of view. First of all, many problems in fields ranging from science and engineering to operations research and financial analysis can be modeled as generalized circulation problems (see *e.g.*, [Law76]). From a theoretical viewpoint, the generalized circulation problem is, probably, the simplest, most combinatorial linear programming problem for which no strongly polynomial algorithm is known. The construction of a strongly polynomial algorithm for linear programming is an outstanding open problem in the theory of algorithms. The only types of linear programs currently known to be solvable in strongly polynomial time are those that either have two variables per inequality [Meg83], or have $(0, +1, -1)$ constraint matrix [Tar86]. The strongly polynomial algorithm for the second type of linear programs grew out of ideas developed for the first strongly polynomial algorithm for the minimum cost flow problem [Tar85]. We believe that developing a strongly polynomial algorithm for the generalized circulation problem may lead to a strongly polynomial algorithm for the linear programming problem.

In this paper we give the first polynomial-time combinatorial algorithms[1] for the generalized circulation problem. The importance of designing such algorithms for this problem is twofold. Combinatorial methods may lead to algorithms that are more efficient, both in theory and in practice, than algorithms based on general linear programming techniques. (For example, this is the case for the minimum-cost circulation problem.) Furthermore, we feel that a combinatorial approach is more likely to yield the insight needed to design a strongly polynomial algorithm for the generalized circulation problem, and we view our results as a significant first step in this direction.

The generalized circulation problem is in many respects similar to the minimum-cost circulation problem, where each arc has a cost per unit of flow in addition to capacity. Both problems can be viewed as problems of transporting a commodity from a producer to a consumer. Intuitively, the only difference between the two problems lies in the method of payment for shipping

[1]By combinatorial algorithms we mean algorithms that exploit the combinatorial structure of the underlying network (as opposed to being based on analytic ideas like the interior point methods for linear programming).

costs: in the case of the minimum-cost circulations, these costs are paid with money; in the case of the generalized circulations – with the commodity itself. The similarity is not only intuitive; the linear programming dual of these problems, as well as optimality conditions, are very similar. Both problems are special linear programs, and therefore can be solved in polynomial time using general-purpose linear programming algorithms, such as the ellipsoid method [Kha80] or Karmarkar's algorithm [Kar84]. Although closely related to the minimum-cost circulation problem, the generalized circulation problem is not yet as well understood. Polynomial-time combinatorial algorithms for the minimum-cost circulation problem were known for a long time [EK72]; more recently, strongly polynomial combinatorial algorithms were developed as well [Tar85].

The running time bounds of the best combinatorial algorithms for the minimum-cost circulation problem are far superior to the running time bounds of the algorithms based on the general linear programming techniques. Under the assumption that the input numbers are not too large (have size $\Theta(\log n)$), the fastest known algorithm for the problem [GT87] runs in $O(nm \log(n^2/m) \log(nC))$ time[2], where $C$ is the absolute value of the biggest arc cost (see also [AGOT87]). The fastest strongly polynomial algorithm for the problem [Orl88] runs in $O(m(m + n \log n) \log n)$ time.

In spite of the similarity between the minimum-cost circulation and the generalized circulation problems, the current techniques for design of efficient combinatorial algorithms for the former are not sufficient to yield efficient algorithms for the latter. The only previously known polynomial-time algorithms for the generalized circulation problem are based on polynomial-time linear programming algorithms.

Although the previous work on combinatorial algorithms for the generalized flow problem did not yield polynomial-time algorithms, it did produce useful insights into the structure of the problem. These insights lead to combinatorial algorithms that run in finite time. Special variants of the linear programming simplex method have been developed for the generalized flow problem (see *e.g.*, [EGK79]). These variants give combinatorial methods for solving the generalized network flow problem. Algorithms that iteratively augment the current flow have also been studied (see *e.g.*, [Jew58,Law76,PE84]). However, no polynomial-time bounds are known for these algorithms.[3]

We present two algorithms, one based on the repeated application of a minimum-cost flow subroutine, and another based on the idea of augmenting along a biggest improvement path [EK72] and the idea of canceling negative cycles [GT88,Kle67]. We assume that the capacities are integers represented in binary, each gain is given as a ratio of two integers, and denote the value of the biggest integer used to represent the gains and capacities by $B$. Under these assumptions, the first algorithm runs in $O(n^2 m(m + n \log n) \log n \log B)$ time and the second in $O(n^2 m^2 \log n \log^2 B)$ time.

The fastest general-purpose linear-programming algorithm currently known [Vai87], when applied to the generalized circulation problem, runs in $O(m^4 \log n \log B)$ time. Using techniques from [KV86], this algorithm can be modified to take advantage of the special structure of the problem; the resulting algorithm runs in $O(n^{2.5} m^{1.5} \log n \log B)$ time (Vaidya, personal communication). The running time bounds of our algorithms are better than the bounds achieved by the general-purpose

[2]In this paper, we denote the number of nodes in the input network by $n$ and the number of arcs in the network by $m$.

[3]Pulat and Elmaghraby [PE84] claim to have developed a strongly polynomial algorithm based on iterative augmentation of flow. However, the proof in the paper has a major gap, and the method used for the augmenting path selection does not seem to be sophisticated enough to yield a polynomial-time algorithm.

linear programming algorithms. Although our time bounds are slightly worse than the time bound of Vaidya's specialization of his linear programming algorithm mentioned above, our algorithms lose by only a logarithmic factor on sparse graphs. Unlike Vaidya's algorithm, our approach exploits the combinatorial structure of the problem. Therefore we feel that it will lead to bounds that are better than those arising from implementations of general-purpose linear programming methods.

In this paper we introduce new tools for the design of combinatorial algorithms for the generalized circulation problem. Analysis of our algorithms is based on new insights into the combinatorial structure of the problem. We believe that these tools and insights will lead to faster algorithms and to a better understanding of combinatorial structure of various network flow problems.

This paper is organized as follows. Section 2 defines the generalized circulation problem and closely related concepts, and gives their important algorithmic and combinatorial properties. Section 3 introduces the vertex labels, which are used to convert a given instance of the problem into an equivalent instance with several useful combinatorial properties. Section 4 describes two simple combinatorial algorithms for the generalized circulation problem. Section 5 describes our first polynomial time algorithms, which is based on a minimum-cost flow subroutine. In Section 6 we present our second algorithm, based on the idea of augmenting the flow along a "big improvement" path. The last section contains concluding remarks.

## 2 Definitions and Background

In this section we define the generalized circulation problem, present a number of fundamental facts about it, and discuss several related concepts. In addition, we describe several variants of the generalized circulation problem and discuss the relationship among them. We also review the minimum-cost circulation problem and discuss the close relationship between this problem and the generalized circulation problem.

### 2.1 Minimum-Cost Circulation Problem

First we discuss the minimum-cost circulation problem. A *circulation network* is a directed graph $G = (V, E)$ with arc capacities given by a non-negative capacity function $u : E \rightarrow \mathbf{R}_\infty$.[4] We assume that $G$ has no multiple arcs, *i.e.*, $E \subset V \times V$. If there is an arc from a node $v$ to a node $w$, this arc is unique by the assumption, and we will denote it by $(v, w)$. This assumption is for notational convenience only. We also assume, without loss of generality, that the input graph $G$ is symmetric: $(v, w) \in E \Longleftrightarrow (w, v) \in E$.

In the context of the minimum-cost circulation problem we need the following definitions. A *pseudoflow* is a function $f : E \rightarrow \mathbf{R}$ that satisfies the following constraints:

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

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

[4] $\mathbf{R}_\infty = \mathbf{R} \bigcup \{\infty\}$.

*Remark*: To gain intuition, it is often useful to think only about the non-negative components of a pseudoflow (or a generalized pseudoflow, defined in the next section). The antisymmetry constrains reflect the fact that a flow of value $x$ going from $v$ to $w$ can be thought of as a flow of value $(-x)$ from $w$ to $v$. The negative flow values are introduced only for notational convenience. Note, for example, that one does not have to distinguish between lower and upper capacity bounds: the capacity of the arc $(v, w)$ represents a lower bound on the flow value on the opposite arc.

Given a pseudoflow $f$, the *residual capacity* function $u_f : E \rightarrow \mathbf{R}$ is defined by $u_f(v, w) = u(v, w) - f(v, w)$. The *residual graph* with respect to a pseudoflow $f$ is given by $G_f = (V, E_f)$, where $E_f = \{(v, w) \in E | u_f(v, w) > 0\}$. An *excess* $Ex_f(v)$ at a node $v$ is equal to

$$Ex_f(v) = -\sum_{(u,v)\in E} f(u, v). \tag{3}$$

We will say that a node $v$ has *excess* if $Ex_f(v)$ is positive, and has *deficit* if it is negative. A *circulation* is a pseudoflow with zero excess at every node:

$$Ex_f(v) = 0 \quad \text{(flow conservation constraint).} \tag{4}$$

A *cost function* is a real-valued function on arcs $c : E \rightarrow \mathbf{R}$. Without loss of generality, we assume that costs are antisymmetric:

$$c(v, w) = -c(w, v) \quad \forall (v, w) \in E \quad \text{(cost antisymmetry constraint).} \tag{5}$$

A cost of a circulation $f$ is given by

$$c(f) = \sum_{(v,w)\in E: f(v,w)\geq 0} f(v, w)c(v, w).$$

The *minimum-cost circulation* problem is to find a minimum-cost (*optimal*) circulation in an input network.

Next we state two criteria for optimality of a circulation. Define the cost of a cycle to be the sum of costs of arcs along the cycle.

**Theorem 2.1** ([BS65]) *A circulation is optimal if and only if its residual graph contains no negative-cost cycles.*

To state the second criterion, we need the notions of the price function and the reduced cost function. A *price function* is a labeling on nodes $p : V \rightarrow \mathbf{R}$. A reduced cost function with respect to a price function $p$ is defined by $c_p(v, w) = c(v, w) + p(v) - p(w)$. These notions, which originate in the theory of linear programming, are crucial for many minimum-cost flow algorithms. As linear programming dual variables, node prices have a natural economic interpretation: they can be interpreted as current market prices of the commodity. We can interpret reduced cost $c_p(v, w)$ as the cost of buying a unit of commodity at $v$, transporting it to $w$, and then selling it.

**Theorem 2.2** ([FF62]) *The cost of a circulation $f$ is minimum if and only if there is a price function $p$ such that, for each arc $(v, w)$,*

$$c_p(v,w) < 0 \Rightarrow f(v,w) = u(v,w) \quad \text{(complementary slackness constraint).} \tag{6}$$

Another useful fact about circulations is the decomposition theorem, which states that a circulation can be viewed as a collection of flows along cycles.

**Theorem 2.3** ([FF62]) *For every circulation $f$, there exists a collection of $k \le m$ simple cycles $C_1, \dots, C_k$ in $G$, and $k$ positive numbers, $\delta_1, \dots, \delta_k$, such that an arc $(v,w)$ appears on one of the cycles only if $f(v,w) > 0$, and for every $(v,w) \in E$*

$$f(v,w) = \sum_{i:(v,w) \text{ is on } C_i} \delta_i - \sum_{i:(w,v) \text{ is on } C_i} \delta_i.$$

## 2.2 The Generalized Circulation Problem

In this section we formally define generalized pseudoflows, generalized circulations, and the generalized circulation problem. We study combinatorial properties of the generalized pseudoflows and generalized circulations, and describe the correspondence between these properties and the properties of regular circulations and pseudoflows discussed in the previous section.

In the *generalized circulation problem*, every arc has a *gain* factor associated with it, with the gains given by a gain function $\gamma : E \to \mathbf{R}^+$.[5] We assume (without loss of generality) that the gain function is antisymmetric:

$$\gamma(v,w) = \gamma^{-1}(w,v) \quad \forall (v,w) \in E \quad \text{(gain antisymmetry constraints).} \tag{7}$$

*Remark*: Recall that the gain factors in our financial analysis problem correspond to currency exchange rates. Though the exchange rates are usually not antisymmetric, it is easy to use multiple arcs to describe them by a network that satisfies gain antisymmetry constraints.

In the case of ordinary flows, if $f(v,w)$ units of flow are shipped from $v$ to $w$, $f(v,w)$ units arrive at $w$. In the case of generalized flows, if $g(v,w)$ units of flow are shipped from $v$ to $w$, $\gamma(v,w)g(v,w)$ units arrive at $w$. A *generalized pseudoflow* is a function $g : E \to \mathbf{R}$ that satisfies the capacity constraints (1) and the generalized antisymmetry constraints:

$$g(v,w) = -\gamma(w,v)g(w,v) \quad \forall (v,w) \in E \quad \text{(generalized antisymmetry constraints).} \tag{8}$$

If $\gamma(v,w) > 1$, then $(v,w)$ is a *gain* arc; if $\gamma(v,w) < 1$, then $(v,w)$ is a *loss* arc. A gain of a path (cycle) is a product of gains of arcs on the path (cycle). Given a generalized pseudoflow $g$, the definitions of residual capacities, residual graph, and excesses are the same as for the ordinary pseudoflows.

A *generalized circulation* is a generalized pseudoflow that satisfies conservation constraints (4) at all nodes except at the *source* node.

The input to a *generalized circulation problem* is a tuple $(G = (V,E), u, \gamma, s)$, where $G$ is a directed graph, $u$ is a capacity function, $\gamma$ is a gain function, and $s$ is the *source*. For simplicity

[5] $\mathbf{R}^+$ denotes the set of positive real numbers.

we assume that the capacities are finite and non-negative. As before, for notational convenience, we assume that $G$ is symmetric, has no multiple arcs, and that the residual graph of the zero flow is strongly connected. A *value* of a generalized pseudoflow $g$ is the excess $Ex_g(s)$. The output of a generalized circulation algorithm is a generalized circulation of the highest possible value (an *optimal* generalized circulation).

Unless mentioned otherwise, we assume that the capacities are given as integers, each gain is given as a ratio of two integers, and that all integers are represented in binary. We denote the biggest integer used to represent capacities and gains by $B$. In addition, we denote the maximum of a product of gain numerators on a simple path or cycle in the input graph by $T$, and the least common multiple of numerators and denominators of all gains in the problem by $L$. Clearly, $T \leq B^n$ and $L \leq B^{2m}$. However, $T$ and $L$ are often significantly smaller than implied by the above bounds. In particular, this is the case for some of the problems obtained via reductions.

A *flow-generating cycle* is a cycle whose gain is greater than 1, and a *flow-absorbing cycle* is a cycle whose gain is less than 1. Observe that if one unit of flow leaves a node $v$ and travels along a flow-generating cycle, more than one unit of flow arrives at $v$. Thus we can *augment* the flow along this cycle, that is, we can increase the excess at any node of the cycle while preserving excesses at other nodes by increasing flow along the arcs in the cycle and correspondingly decreasing flow on the opposite arcs, to satisfy the generalized antisymmetry constraints.

Recall the financial analysis interpretation of the generalized circulation problem, discussed in the introduction. From the investor's point of view, a residual flow-generating cycle is an opportunity to make profit. However, it is possible to take advantage of this opportunity only if there is a way to transfer the profit to the investor's bank account (the source node). This motivates the following definition.

A *generalized augmenting path (GAP)* is a residual flow-generating cycle and a (possibly trivial) residual path from a node on the cycle to the source. Given a generalized circulation and a GAP in the residual graph, we can augment the flow along the GAP, increasing the value of the current circulation. The role of GAPs in the generalized circulation problem is similar to the role of negative-cost cycles in the minimum-cost circulation problem: both can be used to augment the flow and thus improve the value of the current solution.

In the case of the maximum-flow problem, the flow is optimal if and only if the corresponding residual graph contains no augmenting paths [FF62]. A similar result holds for generalized circulations.

**Theorem 2.4** [Ona67] *A generalized circulation is optimal if and only if its residual graph contains no GAPs.*

Using the linear programming dual of the problem, it is possible to specify an alternate criterion of optimality, similarly to the way it is done for the minimum-cost circulation problem. We refer to the dual variables as prices. As in the context of the minimum-cost circulation problem, a *price function* $p$ is a labeling of nodes by real numbers. In addition, in the context of the generalized circulation problem, we require $p(s) = 1$. Node prices can also be interpreted as market prices of the commodity at nodes, which motivates the definition of the reduced cost function. If a unit of flow is purchased at $v$ and shipped to $w$, then $\gamma(v, w)$ units arrive at $w$. Buying the unit at $v$ costs $p(v)$, and selling $\gamma(v, w)$ units at $w$ returns $p(w)\gamma(v, w)$. Thus the reduced cost of $(v, w)$ is defined as

$$c_p(v,w) = p(v) - p(w)\gamma(v,w).$$

Linear programming duality theory provides the following optimality criterion, which is similar to the one given by Theorem 2.2 for minimum-cost circulations.

**Theorem 2.5** *A generalized circulation is optimal if and only if there exists a price function p, such that the complementary slackness conditions (6) hold for each arc $(v,w) \in E$.*

## 2.3 Decomposition of Generalized Pseudoflows

Theorem 2.3 states that a circulation can be decomposed into cycles. Next we prove a similar result for generalized pseudoflows. We show how to decompose a generalized pseudoflow into elements, where each element is a "simple" generalized pseudoflow.

We start by defining five types of such elements. Given a generalized pseudoflow $g$, let $\mathcal{E}(g)$ denote the set of nodes with excess and let $\mathcal{D}(g)$ denote the set of nodes with deficits. Each element of the decomposition is a simple pseudoflow and belongs to one of the five types, where the type is defined according to the graph induced by the set of arcs on which $g$ is positive.

| | |
|---|---|
| Type I | A path from $\mathcal{E}(g)$ to $\mathcal{D}(g)$. Creates a deficit and an excess at the two ends of the path. |
| Type II | A flow-generating cycle and a path connecting this cycle to a node in $\mathcal{E}(g)$. Creates excess at the end of the path. (If the path ends at the source, then this corresponds to a GAP.) |
| Type III | A flow-absorbing cycle and a path connecting this cycle to a node in $\mathcal{D}(g)$. Creates deficit at the end of the path. |
| Type IV | A cycle with unit gain. Does not create any excesses or deficits. |
| Type V | A pair of cycles connected by a path, where one of the cycles generates flow and the other one absorbs it. Does not create any excesses or deficits. |

**Theorem 2.6** *A generalized pseudoflow $g$ can be decomposed into components $g_1, \ldots, g_k$ with $k \leq m$ such that*

$$g(v,w) = \sum_i g_i.$$

*where each component $g_i$ is a generalized pseudoflow that is positive only on arcs $(v,w)$ with $g(v,w) > 0$, and that belongs to one of the above mentioned types. This decomposition can be found in $O(nm)$ time.*

*Proof*: We prove the theorem by induction on the number of arcs with non-zero flow value. Let $G'$ denote the subgraph of $G$ consisting of the arcs with positive flow value. If $G'$ is acyclic, then a decomposition consisting only of paths (generalized pseudoflows of type I) can be easily found by tracing the flow from some node with deficit to some node with excess. Otherwise, let $\mathcal{C}$ be a cycle in $G'$. If this cycle has gain 1, then we can subtract flow around the cycle until one arc on the cycle has zero flow value. The subtracted flow is of type IV, and the theorem follows by induction.

Now consider the case when the gain of $\mathcal{C}$ is more than 1 (the case when the gain is less than 1 can be treated similarly). Decrease the flow along this cycle until one of the arcs along the cycle has zero flow value, creating deficit at one of the nodes $v$ on the cycle, and denote the removed flow by $h$. Decompose $g - h$ according to the induction hypothesis. This decomposition will include a number of components that create the deficit at $v$. These components can be either of Type I or of Type III, and each is responsible for some amount of deficit, such that all the amounts sum up to the deficit at $v$ in $g - h$. Observe that each one of these components, together with an appropriate fraction of $h$, corresponds to an element of Type II or V in the decomposition of $g$.

For establishing the running time, observe that the above procedure decreases the number of arcs with positive flow value in $O(n)$ time. ▮

## 2.4 Alternative Formulations

The generalized circulation problem can be stated in several ways. In this section we present different formulations of the problem and discuss the relationship among them. Understanding this relationship is important since different practical problems can be modeled in different terms, and certain algorithms seem more natural when applied to certain formulations.

**Generalized Flow Problem** (See [Law76] under the name of "flows with losses and gains".) The input to the problem is a graph $G = (V, E)$, a gain function $\gamma : E \rightarrow \mathbf{R}$, a source $s$, and a sink $t$. A *generalized flow* if a function on arcs that satisfies the antisymmetry constraints on all arcs and the conservation constraints on all nodes except $s$ and $t$. The value of the flow is defined to be the amount of flow into the sink. Among all the generalized pseudoflows of maximum value, the goal is to find a generalized pseudoflow that minimizes the flow out of the source. The generalized flow problem is reducible to the generalized flows with profit.

**Generalized Flows with Profit Problem (GFP Problem)** The input to this problem is the same as the input to the generalized flow problem plus a number $r \in \mathbf{R}^+$ that gives the ratio of the price per unit of commodity at the sink to the price per unit of commodity at the source. The goal is to find a generalized flow that maximizes the profit. For a generalized flow $g$, the profit is $rEx_g(t) + Ex_g(s)$. To reduce a generalized flow problem to the GFP problem, chose $r$ to be large enough (but finite); for example, $r = B^n + 1$.

The following linear-time reductions show that the GFP problem is equivalent to the generalized circulation problem. Given an instance $(G = (V, E), \gamma, s, t, r)$ of the GFP problem, we define an equivalent instance of the generalized circulation problem by adding an arc $(t, s)$ with very large ($\geq nB^2$) capacity and a gain of $r$; and defining $s$ to be the source. Given an instance $(G = (V, E), \gamma, s)$ of the generalized circulation problem, add a new node $t$ to the graph, along with the arcs $(s, t)$ and $(t, s)$, assign unit gain and very high ( $\geq nB^2$) capacity to these arcs, and let $r = 1$.

## 2.5 The Restricted Problem

Instead of solving the generalized circulation problem directly, we will solve a restricted version of the problem. The *restricted problem* has the same input as the generalized circulation problem, but with an additional assumption that all flow-generating cycles in the residual graph of the zero flow pass through the source. This restricted problem has a simpler combinatorial structure and leads to simpler algorithms. In this subsection we shall give an $O(nm)$-time reduction of the general problem to the restricted one. Unless stated otherwise, in the subsequent sections we will refer to the restricted version of the problem under the name of the generalized circulation problem.

One of the nice facts about the restricted problem is that the optimality condition given by Theorem 2.4 simplifies in this case, and becomes very similar to Theorem 2.1. To prove this we need a lemma that will also be useful later.

**Lemma 2.7** *Let $g$ be a generalized pseudoflow in a restricted problem.*

1. *If the excess of every node other than the source is non-negative, then for every node $v$ there exists an $(v, s)$-path in the residual graph $G_g$.*
2. *If the residual graph of a generalized pseudoflow $g$ has no flow-generating cycles and the excess of every node other then $s$ is non-positive, then for every node $v$ there exists an $(s, v)$-path in the residual graph $G_g$.*

*Proof*: The proof is by induction on the number of arcs with positive flow value. Let $G'$ be the subgraph induced by these arcs. Both statements are easy to prove when $G'$ is acyclic. Now suppose $G'$ contains a cycle $C$. This cycle is in the residual graph of the zero flow and therefore if its gain is more than 1, it has to pass through the source. Consider the case when the gain of $C$ is at most 1. Decreasing the flow along $C$ can only decrease the deficit at some node. The first claim follows by induction. Observe that if there are no flow-generating cycles in the residual graph of $g$, then the gain of $C$ has to be at least 1, which proves the second claim. The case of $C$ having a gain of more than 1 can be checked in a similar way. ∎

**Theorem 2.8** [Ona66] *Given a restricted problem, generalized circulation $g$ is optimal if and only if the residual graph of $g$ contains no flow-generating cycles.*

*Proof*: Clearly, if the residual graph of the generalized pseudoflow has no flow generating cycles, then it has no GAPs, and therefore it is optimal. To see the converse, consider a flow-generating cycle in the residual graph. We have to show that there is a path in the residual graph connecting this cycle to the source. The generalized circulation $g$ has non-negative excess at every node, and hence, by the the above lemma, there is a path from every node to the source. ∎

**Corollary 2.9** *Given a restricted problem, a generalized circulation $g$ is optimal if and only if the residual graph of $g$ has no negative-cost cycles, with $c = -\log \gamma$ as the cost function.*

This corollary implies that the optimality of a solution for an instance of the restricted problem can be tested in one shortest path computation.

Finally, we show how to reduce the generalized circulation problem to the restricted version of this problem. The reduction works as follows. First, we saturate all gain arcs. More formally, define a generalized pseudoflow $h$ by $h(v, w) = u(v, w)$ if $(v, w)$ is a gain arc, $h(v, w) = -\gamma(w, v)u(w, v)$ if $(v, w)$ is a loss arc, and $h(v, w) = 0$ otherwise. For every $v \in V : Ex_h(v) < 0$, add an arc $(v, s)$ of gain $\gamma(v,s) = B^n + 1$ and capacity $u(v,s) = -Ex_h(v)$. (We also add reverse arcs with capacity 0 to preserve symmetry.) For every $v \in V : Ex_h(v) > 0$, add an arc $(s,v)$ of gain $\gamma(s,v) = B^n + 1$ and capacity $u(s,v) = Ex_h(v)/\gamma(s,v)$ (and the reverse arcs with capacity 0 to preserve symmetry). Finally, define $h$ to be zero on the new arcs. Let $\hat{G}$ be the extended graph. Then the transformed problem is $(\hat{G}, u_h, \gamma, s)$. Intuitively, the new arcs assure that nodes that have positive excess with respect to $h$ are supplied with an adequate amount of "almost free" commodity, and nodes that have deficits with respect to $h$ send adequate amount of commodity to $s$, whenever possible.

**Theorem 2.10** *Given an optimal generalized circulation $\hat{g}$ in the network $(\hat{G}, u_h, \gamma, s)$ one can construct an optimal generalized circulation in $(G, u, \gamma, s)$ in $O(nm)$ time.*

*Proof*: Define a pseudoflow $g'$ by $g'(v,w) = \hat{g}(v,w) + h(v,w)$ for every $(v,w) \in E$. Observe that the residual graph of $g'$ is the restriction of the residual graph of $\hat{g}$ to arcs in $E$. A node $v \in V - s$ has deficit in $g'$ if and only if the arc $(v,s)$ with gain $B^n + 1$ is in the residual graph of $\hat{g}$. Similarly, the node $v \in V - s$ has excess in $g'$ if and only if the arc $(s,v)$ with gain $B^n + 1$ is in the residual graph of $\hat{g}$. By Theorem 2.8, there are no flow-generating cycles in the residual graph of $\hat{g}$, which implies that there are no paths from nodes with excess to nodes with deficit in the residual graph of $g'$. Let $S$ be the subset of nodes from which $s$ is reachable in the residual graph of $g'$. Note, that there are no nodes with excess in $S - s$.

Decompose $g'$ as described in Theorem 2.6. Since there is no path in the residual graph from nodes with excess to nodes with deficit, the decomposition does not include elements of type I (paths). Therefore, excesses are created by flow-generating cycles, and deficits are created by flow-absorbing cycles. The existence of a flow-absorbing cycle in the decomposition implies that there are flow-generating cycles in the residual graph of $g'$, which, by Theorem 2.8, contradicts the optimality of $\hat{g}$. This implies that $g'$ cannot have deficits.

Subtract from the pseudoflow $g'$ the elements of the decomposition that create the excesses at nodes other than $s$. Let $g$ be the resulting generalized circulation. We claim that $g$ is optimal. Observe that on arcs $(v,w)$ inside $S$ or entering $S$, the flow value $g(v,w) = g'(v,w) = \hat{g}(v,w) + h(v,w)$, and therefore there are no flow-generating cycles contained in $S$. Furthermore, the source $s$ is not reachable from nodes outside $S$. Therefore the residual graph of $g$ cannot contain a GAP. ▮

Note that if $B$ is the biggest integer in the input problem, then the biggest integer in the transformed problem can be as high as $B^n + 1$. On the other hand, the transformation can not cause a significant increase in $T$ and $L$: the corresponding parameters $T'$ and $L'$ of the transformed problem are bounded by $B^n(B^n + 1) \le B^{2n+1}$ and $B^{2m}(B^n + 1) \le B^{3m}$, respectively.

## 3 Vertex Labels and Equivalent Problems

Recall the financial analysis interpretation of the generalized circulation problem, described in the introduction, where vertices correspond to different securities or currencies, and arcs correspond to possible transactions. Suppose one country decides to change the unit of currency (for example, Great Britain could decide to introduce the penny as the basic currency unit, instead of the £, or Italy could erase a couple 0's at the end of its bills). This causes an appropriate update of the exchange rates. Some of the capacities are changed as well (for example, a million $\mathcal{L}$ limit on the exchange from $\mathcal{L}$ to DM would now read as a limit of 100 million pennies).

The operation of changing the units of measure is called *relabeling* and the equivalent problem obtained by relabeling is called the *relabeled* problem. Let $\mu : V \longmapsto \mathbf{R}^+$ be a function denoting the number of old units per each new unit at nodes of the network. Given a function $\mu$, we shall refer to $\mu(v)$ as the *label* of $v$.

**Definition:** Given a function $\mu : V \longmapsto \mathbf{R}^+$ and a network $N = (V, E, \gamma, u)$, the *relabeled network* is $N_\mu = (V, E, \gamma_\mu, u_\mu)$, where the *relabeled capacities* and the *relabeled gains* are defined by

$$\begin{aligned} u_\mu(v,w) &= u(v,w)/\mu(v) \\ \gamma_\mu(v,w) &= \gamma(v,w)\mu(v)/\mu(w). \end{aligned}$$

Given a generalized pseudoflow $g$ and a labeling $\mu$, the *relabeled residual capacity* is defined by:

$$u_{g,\mu}(v,w) = \frac{u(v,w) - g(v,w)}{\mu(v)}.$$

The following lemma relates pseudoflows in the original and the relabeled networks.

**Lemma 3.1** *If $N$ is a generalized network and $g$ is a generalized pseudoflow in $N$, then $g_\mu(v,w) = g(v,w)/\mu(v)$ is a generalized pseudoflow in the relabeled network $N_\mu$. Moreover, the residual graphs of $g$ and $g_\mu$ are the same.*

The idea of relabeling was used by Glover and Klingman [GK73] (under the name "scaling") to show that the generalized network flow problem in which the $n$ flow conservation constraints are linearly dependent is equivalent to the maximum-flow problem. The name "scaling" comes from the fact, that relabeling corresponds to multiplying the columns of the corresponding linear program by scalars.

## 3.1 Canonical Relabeling

Let $g$ be a generalized pseudoflow whose residual graph has the property that all flow-generating cycles go through $s$ (for example, the zero flow in the restricted problem). We shall use two symmetric ways to relabel a residual network. One is the *canonical relabeling from the source*, and the other is the *canonical relabeling to the source*. We use the first relabeling when we want to push additional flow from $s$, and the second when we want to push additional flow into $s$.

The canonical relabeling from the source applies when every node $v \in V$ is reachable from the source $s$ via a path in the residual graph of the generalized pseudoflow $g$. For every node $v \in V$, the canonical label $\mu(v)$ is defined to be equal to the gain of the highest-gain simple $s$-$v$ path in the residual graph. That is, one new unit corresponds to the amount of flow that can reach the node $v$ if one old unit of flow is pushed along the most efficient simple path in the residual graph from $s$ to $v$, ignoring capacity restrictions along the path.

Observe that the highest-gain path is the shortest path when arc lengths are defined to be $c(v,w) = -\log(\gamma(v,w))$. Because of the assumption that all flow-generating cycles pass through the source, the highest-gain path can be easily found by a single shortest-path computation, since deleting the arcs entering $s$ from the residual graph yields a graph with no negative cycles.

The canonical relabeling to the source is defined similarly. It applies if there is a path in the residual graph from every node to the source $s$. The canonical label $\mu$ is defined as the inverse of the gain of the highest-gain simple $v$-$s$ path in the residual graph.

The following important properties of the canonically relabeled problem are easy to prove.

**Theorem 3.2** *After a canonical relabeling from the source:*

1. *Every arc $e$ with non-zero residual capacity, other than the arcs entering the node $s$, has $\gamma_\mu(e) \leq 1$.*
2. *For every node $v$, there exists a path from $s$ to $v$ in the residual graph with $\gamma_\mu(e) = 1$ for all arcs $e$ on the path.*
3. *The most efficient flow-generating cycles consist of an $(s, v)$-path for some $v \in V$ with $\gamma_\mu(e) = 1$ along the path, and the arc $(v, s) \in E_g$ such that $\gamma_\mu(v, s) = \max(\gamma_\mu(e) : e \in E_g)$.*

**Theorem 3.3** *After a canonical relabeling to the source:*

1. *Every arc $e$ with non-zero residual capacity, other than the arcs leaving the node $s$, has $\gamma_\mu(e) \leq 1$.*
2. *For every node $v$, there exists a path from $v$ to $s$ in the residual graph with $\gamma_\mu(e) = 1$ for all arcs $e$ on the path.*
3. *The most efficient flow-generating cycles consist of an $(v, s)$-path for some $v \in V$ with $\gamma_\mu(e) = 1$ along the path, and the arc $(s, v) \in E_g$ such that $\gamma_\mu(s, v) = \max(\gamma_\mu(e) : e \in E_g)$.*

There is a simple correspondence between the units used for a relabeling and the prices from the linear programming dual of the problem. Intuitively, making the unit at a vertex $v$ smaller, while keeping the price per unit constant, corresponds to increasing the price at $v$. In other words, changing the unit at $v$ to $\mu(v)$ and keeping the price per unit ($p(v)$) constant has the same effect as keeping the size of the unit and setting the price to $p(v) = p(v)/\mu(v)$. If the price at every node is 1, then canonical relabeling from the source corresponds to changing the prices to $p(v) = (\mu(v))^{-1}$. Note that, ignoring the arcs entering $s$, these prices are the marginal costs of the commodity, that is, the minimum prices (per unit) for which one could get some additional amount of the commodity to the nodes.

We can reformulate the optimality conditions of Theorem 2.5 to use labels instead of prices.

**Lemma 3.4** *A generalized circulation $g$ in a restricted problem is optimal if and only if there exists a labeling $\mu$ such that every arc in the residual graph of the generalized circulation has $\gamma_\mu(e) \leq 1$.*

We say that a labeling $\mu$ is *optimal* if there exist a generalized circulation $g$ such that $g$ and $\mu$ satisfy the conditions of Lemma 3.4.

**Lemma 3.5** *The optimality of a labeling $\mu$ for a restricted problem can be checked in one maximum-flow computation.*

*Proof*: Relabel the network with labels $\mu$. Suppose $g$ is a generalized circulation that satisfies the optimality conditions with $\mu$, and let $g_\mu$ be the corresponding generalized circulation in the relabeled network. If $\gamma_\mu(v,w) > 1$ then, by the optimality conditions, $g(v,w) = u(v,w)$, that is, $g_\mu(v,w) = u_\mu(v,w)$. Due to the symmetry, this uniquely defines the flow value on every arc with $\gamma_\mu(e) \neq 1$. The labeling $\mu$ is optimal if and only if $g_\mu$ can be extended to a feasible generalized circulation in the relabeled network. Hence, it is sufficient to solve a network flow feasibility problem on the subgraph of the relabeled network induced by the arcs with unit relabeled gain. ▌

During a computation one has to keep the size of the numbers under control. The following lemma provides bounds on some of the numbers occurring in the algorithms.

**Lemma 3.6** *Let $g$ be a generalized pseudoflow such that the canonical relabeling from the source (or to the source) in the residual graph applies. Let $\mu$ denote the canonical labels. Then $T^{-1} \leq \mu(v) \leq T$ and both numerator and denominator of $\mu(v)$ are divisors of $L$, for every $v \in V$.*

*Proof*: The label $\mu(v)$ is equal to the gain of an $s$-$v$ path in $G$, and hence satisfies the above claim, by the definitions of $T$ and $L$. ▌

The following theorem will be used to prove termination for both of our polynomial-time algorithms. For a node $v$, a generalized pseudoflow $g$, and labels $\mu$, let $Ex_{g,\mu}(v)$ denote the *relabeled excess* of the node $v$, that is, the total excess of the pseudoflow corresponding to $g$ in the relabeled network with labels $\mu$ ($Ex_{g,\mu}(v) = Ex_g(v)/\mu(v)$). Consider the subgraph $G'$ of $G$ induced by the arcs with unit relabeled gain. By Lemma 3.6, in a canonically relabeled network the relabeled capacity of every arc with relabeled gain different from 1 is a multiple of $L^{-1}$. The next theorem states that the same is almost true for the arcs in $G'$. For any subset $S$, the relabeled capacity of arcs of $G'$ entering $S$ plus the relabeled excesses of the nodes in $S$ is an integer multiple of $L^{-1}$.

**Theorem 3.7** *Let $g$ be a generalized pseudoflow whose residual graph contains no flow-generating cycles. Let $\mu$ denote the canonical labels when relabeling from the source (or to the source) in the residual graph of a generalized pseudoflow $g$. For any subset $S \subseteq V$,*

$$X = \sum_{v \in S} Ex_{g,\mu}(v) + \sum_{v \in S; w \notin S \text{ and } \gamma_\mu(w,v)=1} u_{g,\mu}(w,v)$$

*is an integer multiple of $L^{-1}$.*

*Proof*: By Lemma 3.6, the labels, relabeled gain factors, and relabeled capacities, are multiples of $L^{-1}$. However, this is not necessarily true for the relabeled residual capacities. First consider an arc with relabeled gain higher than 1. By assumption, the flow on this arc is equal to the capacity, so the flow is a multiple of $L^{-1}$. By symmetry, the flow value on an arc $(v,w)$, with relabeled gain of less than 1 is equal to $(-\gamma_\mu(w,v)u_\mu(w,v))$, so it is also a multiple of $L^{-1}$. This might not be true for arcs with unit relabeled gain.

The main observation is that the value of $X$ is unchanged when the value of the flow is changed on an arc (and its opposite, to preserve the antisymmetry) with a unit relabeled gain. Indeed, if the arc $e$ is contained in $S$, then changing the flow on $e$ changes the excess at the two ends of the arc by opposite amounts. On the other hand, if $e$ is outside of $S$, then it does not affect the expression.

Consider an arc $e$ entering $S$. A change in the flow on $e$ results in the change in the excess at the head of $e$ and in the corresponding change in the residual capacity of $e$ (in the opposite way). Consequently, one can replace the flow value on all arcs with unit relabeled gain by zero without changing the value of the expression. In the resulting generalized pseudoflow all flow values are multiples of $L^{-1}$, and therefore every term in the expression is an integer multiple of $L^{-1}$. ▮

## 4 Simple Algorithms

In this section we present two simple algorithms for the generalized circulation problem. Both algorithms are based on the natural approach of augmenting along flow-generating cycles. Though the algorithms are not efficient, we describe them in order to give some intuition for the polynomial-time algorithms presented in the subsequent sections.

The first algorithm, due to Onaga [Ona66], is similar to the minimum-cost flow method that augments the flow along a cheapest augmenting path [BG61,Jew58]. In the input network, all flow-generating cycles pass through the source. Onaga observed that this property is retained if the augmentation is done along the highest-gain flow-generating cycle. Onaga's algorithm iteratively augments the generalized circulation along highest-gai flow-generating cycles in the residual graph, until there are no such cycles left. By Theorem 3.2, if all flow-generating cycles pass through the source, then the highest-gain flow-generating cycle can be found by a shortest path computation. Therefore, each iteration of this algorithm consists of a shortest-path computation followed by an augmentation along a flow-generating cycle through the source.

By Theorem 2.8, we know that when the algorithm terminates, the resulting generalized circulation is optimal. However, similarly to the minimum-cost flow algorithm that augments the flow along the cheapest path, this algorithm does not run in finite time [FF62]. In order to make the running time finite, we use a maximum-flow algorithm as a subroutine to augment flow along all of the highest-gain flow-generating cycles at once, instead of augmenting the flow on a cycle-by-cycle basis. The resulting algorithm is similar to Jewell's [Jew58] algorithm that solves the minimum-cost flow problem by repeatedly applying a maximum-flow subroutine to the arcs with zero reduced cost and then changing the prices.

More precisely, consider the residual graph after the canonical relabeling from the source. Let $\alpha = \max\{\gamma_\mu(v,s) : (v,s) \in E_g\}$, and let $N_\alpha$ be the generalized network with underlying graph $G_\alpha$ that is induced by the arcs with unit relabeled gains and arcs $\{(v,s) : (v,s) \in E_g, \gamma_\mu(v,s) = \alpha\}$ and their opposites. Then, by Theorem 3.2, all flow-generating cycles with the highest gain lie in $G_\alpha$. Observe that any *non-generalized* augmentation of flow in $G_\alpha$, *i.e.*, augmentation that disregards the gain factors and views $G_\alpha$ as a standard network, corresponds to a valid *generalized* augmentation in $G$.

**Lemma 4.1** *A (ordinary) flow in $G_\alpha$ that maximizes the sum of the flow values on the arcs entering $s$ with the gain factor $\alpha$ corresponds to an optimal generalized circulation in $G_\alpha$.*

Therefore, by a single maximum-flow computation, we can augment the flow so that there are no flow-generating cycles that pass through the source in $G_\alpha$, and therefore there are no flow-generating cycles with gain $\alpha$ in the residual graph.

```
Step 1 Find $\mu$, canonical labeling from the source.
       If $\gamma_\mu(v,w) \leq 1$ on every arc of the residual graph, then halt (the current circulation
       is optimal).
       Otherwise let $\alpha = \max_{e \in E_g} \gamma_\mu(e)$.
Step 2 Let $G_\alpha = (V, E_\alpha)$ be the network induced by residual arcs with either unit reduced
       gain or reduced gain of $\alpha$, and their opposites.
       Find a circulation $f'$ in $G_\alpha$ using the residual relabeled capacities, that maximizes.
       the flow on the arcs with reduced gain $\alpha$ in to $s$
Step 3 Update the current solution by setting $g(v,w) = g(v,w) + f'(v,w)\mu(v)\ \forall(v,w) \in$
       $E; v \neq s$, and set the value $g(s,w)\ \forall w \in V$ to keep the symmetry.
```

Figure 1: A single iteration of the maximum-flow based algorithm.

The maximum-flow based algorithm proceeds in iterations, where at each iteration we compute the canonical relabeling from the source, construct $G_\alpha$, compute the appropriate maximum-flow, and interpret the result as an augmentation of the current generalized circulation. See Figure 1 for a more formal description. All flow-generating cycles in the residual graphs of the generalized circulations found throughout the algorithm pass through the source, which guarantees that the canonical relabeling is applicable. The optimality of the solution produced when the algorithm terminates follows from Theorem 2.8.

**Theorem 4.2** *The highest gain of a flow-generating cycle in the residual graph is strictly decreasing from one iteration to another, and therefore the number of iterations is bounded by the number of different gains of simple flow-generating cycles in the original network.*

**Corollary 4.3** *If the gain factors in the input are all integer powers of 2 (or of any other constant), then the above algorithm runs in polynomial time.*

Observe, that the first algorithm described in this section may not terminate, whereas the second algorithms terminates in exponential time. In the next two sections we describe more efficient algorithms for the problem.

## 5 Algorithm MCF

In this section we present our first algorithm that solves the restricted version of the generalized circulation problem in polynomial time. Unless stated otherwise, we will refer to the restricted problem as the generalized circulation problem. This algorithm is based on a minimum-cost flow subroutine, so we call it Algorithm MCF. The main idea of the algorithm is best described by contrasting Algorithm MCF with the maximum-flow based algorithm, presented in the previous section. At each iteration, both algorithms solve a simpler flow problem, and interpret the result as an augmentation in the generalized circulation network. The maximum-flow based algorithm is slow because at each iteration it considers only arcs with unit relabeled gains and some of the arcs adjacent to the source, disregarding the rest of the graph completely. The algorithm presented

```
Step 1 Find $\mu$, canonical labeling from the source.
        If $\gamma_\mu(v,w) \le 1$ on every arc of the residual graph and $\forall v \in (V-s) : Ex_{g,\mu}(v) = 0$,
        then halt (the current circulation is optimal).
Step 2 Introduce costs $c(v,w) = -\log \gamma_\mu(v,w)$ on the arcs of the network.
        Find a minimum cost pseudoflow $f'$ in the residual relabeled network $G$ that has
        excess $Ex_{f'}(v) = - Ex_{g,\mu}(v)$ for every node $v \in V - s$.
Step 3 Let $g'$ be the interpreted version of $f'$.
        Update the current solution by setting $g(v,w) = g(v,w) + g'(v,w)\mu(v)\ \forall (v,w) \in E$.
```

Figure 2: Inner loop of Algorithm MCF.

in this section considers all arcs, assigns cost $c(e) = -\log \gamma_\mu$ to each arc, and solves the resulting minimum-cost circulation problem (disregarding flow gains and losses).

The *interpretation* of a pseudoflow $f$ is a generalized pseudoflow $g$, such that $g(v,w) = f(v,w)$ if $f(v,w) \ge 0$ and $g(v,w) = -\gamma_\mu(w,v)f(w,v)$ otherwise, as in the maximum-flow based algorithm. Note that, as opposed to the case of the maximum-flow based algorithm, where the interpretation of a feasible circulation is a feasible generalized circulation, the interpretation of a minimum-cost circulation leads to a generalized pseudoflow. Whenever the circulation uses arcs with relabeled gain of less than 1, its interpretation has deficits. A connection between a pseudoflow $f$ and its interpretation is given by the following lemma.

**Lemma 5.1** *The residual graphs of a pseudoflow $f$ and its interpretation $g$ as a generalized pseudoflow are the same.*

The algorithm starts with the zero generalized pseudoflow, which has flow generating cycles in the residual graph. By the definition of the restricted problem, all these cycles path through the source. Using this fact, we will show that the only iteration that creates a positive excesses is the first one, and that the only positive excess created is the one at the source. Each subsequent iteration tries to use this excess to balance deficits at various nodes of the graph, created by interpretation of flow through arcs with relabeled gain of less then 1. In each iteration we find a minimum-cost flow that satisfies the deficits that were left after the previous iteration.

Algorithm MCF, shown in Figure 2, maintains a generalized pseudoflow $g$ in the original (non-relabeled) network, such that the excess at every node other than the source is non-positive. The algorithm proceeds in iterations. At each iteration it canonically relabels the residual graph, solves the corresponding minimum-cost flow problem in the relabeled network, and interprets the result as a generalized augmentation.

## 5.1 Analysis of the Inner Loop of the Algorithm

In this section we prove that each iteration of Algorithm MCF can be implemented to run in in polynomial time, and show that the algorithm produces an optimal generalized circulation upon termination. The proof that the number of iterations is polynomial is deferred until the next section.

The most important property of a minimum-cost flow, for this application, is that its residual graph has no negative cycles. By Lemma 5.1 this yields the following corollary:

**Corollary 5.2** *The residual graphs of the generalized pseudoflows introduced by the algorithm in Step 3 have no flow-generating cycles.*

**Lemma 5.3** *The following statements are true for a generalized pseudoflow $g$ that is constructed by Step 4:*

1. *The canonical relabeling applies to the residual graph of $g$.*
2. *All excesses, except at the source, are non-positive.*

*Proof*: We will prove the lemma by induction on the number of iterations. Assume that at the end of an iteration we have a generalized pseudoflow $g$ that satisfies the statements of the lemma. We shall prove that the statements are satisfied at the end of the next iteration.

By Corollary 5.2, the residual graph of $g$ has no flow-generating cycles, and therefore, by Lemma 2.7, every node is reachable from the source in the residual graph of $g$. Hence, we can compute the canonical relabeling from the source at Step 1, which proves the first statement of the lemma. Moreover, the absence of flow-generating cycles in the residual graph means that there are no arcs with relabeled gain of more than 1 in this graph. Therefore, the interpretation of flow $f'$, computed at Step 2, can create only deficits, which proves the second statement.

To prove the correctness of the statements after the first iteration, recall the assumption that the residual graph of the zero generalized pseudoflow is strongly connected. Also, by definition of the restricted problem, all flow-generating cycles in the residual graph of the zero generalized pseudoflow pass through the source. Therefore we can apply canonical relabeling from the source. Moreover, after the relabeling, all arcs with relabeled gain of more than 1 enter the source. Therefore, the only node that can have a positive excess after the interpretation of the flow $f'$ is the source. ∎

**Lemma 5.4** *For a generalized pseudoflow $g$ and the labeling $\mu$ after Step 1, there exists a pseudoflow $f'$ in the relabeled residual network of $g$ with $Ex_{f'}(v) = -Ex_g(v)$ for every node $v \in V - s$.*

*Proof*: Let $g$ be a generalized pseudoflow at the beginning of an iteration. Decompose it according to Theorem 2.6. The only elements of the decomposition that can contribute to deficits are paths from nodes with deficits to the source and paths from nodes with deficits to flow-absorbing cycles. Existence of a flow-absorbing cycle in the decomposition implies that there are flow-generating cycles in the residual graph of $g$, which contradicts Corollary 5.2, and therefore there are no flow-absorbing cycles in the decomposition.

Consider a subset of the nodes $S$ containing the source. By the previous lemma, the only node that can have a positive excess is the source, and therefore it is sufficient to prove that the relabeled residual capacity of the cut defined by $S$ exceeds the sum of the relabeled deficits.

Consider an $(s, v)$-path in the decomposition where $v \notin S$, and let $(w_1, w_2)$ be an arc on this path that leaves $S$. Recall that one of the properties of the decomposition is that $(w_2, w_1)$ is an arc of the residual graph. By Corollary 5.2, there are no flow-generating cycles in the residual graph of

$g$, and hence the relabeled gain of every arc in the residual graph of $g$ is at most 1. Therefore, the sum of deficits created by the paths in the decomposition that use this arc, is at most the relabeled residual capacity of the opposite arc $(w_2, w_1)$. ▮

The above lemmas show that the algorithm can proceed until a generalized circulation is found. By Corollary 5.2 and Theorem 2.8, the generalized circulation found is optimal. Hence, we have proved the following theorem.

**Theorem 5.5** *Each iteration of the algorithm can be implemented in polynomial time, and the generalized circulation $g$, produced by the algorithm upon termination, is optimal.*

## 5.2 Bounding the Number of Iterations

Consider a generalized pseudoflow $g$ at the beginning of an iteration. The fact that there are deficits and that there are no flow-generating cycles in the residual graph means that the current excess at the source is an overestimate on the value of the maximum possible excess. It is easy to see that the sum of the deficits (after the relabeling) at all the nodes except at the source is a lower bound on the amount of the overestimation. This suggests to use this value, $Def(g, \mu) = \sum_{v \neq s}(-Ex_{g,\mu}(v))$, as a measure of the proximity of a generalized pseudoflow to an optimal generalized circulation.

First we show that if $Def(g, \mu)$ is very small, then the algorithm terminates after one more iteration.

**Theorem 5.6** *If $Def(g, \mu) < L^{-1}$ before Step 2 of an iteration, then the algorithm produces an optimal generalized circulation at the end of Step 3 of this iteration.*

*Proof*: We claim that the pseudoflow $f'$, computed at Step 2 of this iteration, uses zero cost arcs only. To see this, consider a set of nodes $S$ containing the source $s$. We have to argue that the sum of the relabeled deficits of the nodes not in $S$ is at most equal to the sum of the relabeled residual capacities of the zero cost arcs entering $S$. Clearly, the difference is at most $Def(g, \mu) < L^{-1}$. Applying Theorem 3.7 to the complement of $S$ shows that this difference is an integer multiple of $L^{-1}$. Therefore, it is non-positive.

The interpretation $g'$ of a pseudoflow $f'$ that uses zero cost arcs only, is equal to the pseudoflow. Therefore, no deficits are left after Step 3. The lemma follows because the residual graph contains no flow-generating cycles throughout the algorithm (by Corollary 5.2). ▮

An important observation is that the labels $\mu$ are monotonically decreasing during the algorithm. The next lemma relates the decrease in the labels to the price function from the minimum-cost flow computation. Let $p'$ denote the optimal price function associated with the pseudoflow $f'$ found in Step 2. Assume, without loss of generality, that $p'(s) = 0$.

**Lemma 5.7** *Let $f'$ be the minimum-cost pseudoflow found in Step 2 of an iteration, and let $p'$ be the associated price function. For each node $v$, the canonical relabeling in Step 1 of the next iteration decreases the label $\mu(v)$ by at least a factor of $2^{p'(v)}$.*

*Proof*: The decrease in the label of a node $v$ is computed by finding a shortest path in the residual graph from $s$ to $v$. The price function $p'$ is optimal, and hence the reduced costs of the arcs in the residual graph are non-negative. Therefore, the length of any $(s, v)$ path is at least $p'(v) - p'(s)$. ∎

For the analysis of the algorithm we decompose the minimum cost flow into paths, and consider the paths one by one. The decomposition is based on the following lemma.

**Lemma 5.8** ([FF62]) *Let $f$ be a minimum-cost flow that satisfies given (non-negative) demands at every node other than $s$ and let $p$ be an optimal price function such that $p(s) = 0$. The flow $f$ can be decomposed into cycles and paths from $s$ to the other nodes, such that the cycles and the paths are in the residual graph of the zero flow, the cycles have non-positive cost, and the cost of the paths ending at a node $v$ is at most $p(v)$.*

*Proof*: For every node $v \in V$ add the arcs $(v, s)$ and $(s, v)$. Extend the flow $f$ to these arcs, so that it becomes a circulation. Define the cost of an arc $(v, s)$ to be equal to $c(v, s) = -p(v)$. The new arcs have zero reduced cost, and hence the circulation is of minimum cost.

The lemma is proved by applying Theorem 2.3 to decompose this circulation into cycles. Note that the arcs opposite to the ones that belong to the cycles of the theorem are in the residual graph of the circulation. Therefore, the cycles have non-positive cost. Any cycle that that uses a new arc $(v, s)$ corresponds to $(s, v)$-paths in the original graph. ∎

The key idea of the analysis is to distinguish two cases: Case 1, where the flow $f'$ can be decomposed into "cheap" paths, (*e.g.*, $p'(v)$ is small, say $p(v) < \log 1.5$, for every $v \in V$); and Case 2, where there exists $v \in V$ such that $p'(v)$ is "large" ($\geq \log 1.5$). We show that in the first case $Def(f, \mu)$ decreases significantly, while in the second case, by Lemma 5.7, some of the labels decrease significantly. Using Theorem 5.6 and Lemma 3.6, we prove that neither case can occur too many times.

The following lemma is used to estimate the total deficit created when interpreting a flow as a generalized pseudoflow.

**Lemma 5.9** *Let $f'$ be a flow along a simple path $P$ from $s$ to some other node $v$ that satisfies one unit of deficit at $v$. Let $g'$ be the interpretation of $f'$ as a generalized pseudoflow. Assume that all relabeled gains along the path $P$ are at most 1, and denote them by $\gamma_1, \ldots, \gamma_k$. Then after augmenting by $g'$, the unit of deficit at $v$ is replaced by deficits that sum up to at most $(\prod_{1 \leq i \leq k} \gamma_i)^{-1} - 1$.*

*Proof*: The deficit created at the $i$th node of the path is $(1 - \gamma_i)$, for $i = 1, \ldots, k$. Using the assumption that the gain factors along the path are at most 1, the sum of the deficits can be bounded by

$$\sum_{i=1}^{k}(1 - \gamma_i) \leq \sum_{i=1}^{k} \frac{1 - \gamma_i}{\prod_{j=1}^{i} \gamma_j} = \frac{1}{\prod_{i=1}^{k} \gamma_i} - 1.$$

∎

Using this lemma and Theorem 2.6, we bound the value of $Def(g, \mu)$ after an application of Step 3. Let $p'$ be an optimal price function associated with the flow $f'$, such that $p'(s) = 0$. Let $\beta = \max_{v \in V} p'(v)$ denote the maximum price.

**Lemma 5.10** *The value of $Def(g, \mu)$ after an application of Step 3 can be bounded by $2^{\beta} - 1$ times its value before the Step.*

*Proof*: Use Lemma 5.8 to decompose $f'$ into paths and cycles. Interpret the pseudoflow $f'$ by interpreting the cycles and the paths one by one. The costs associated with the arcs of the residual graph of the generalized pseudoflow $g$ are non-negative, and therefore the cycles in the decomposition consist of zero-cost arcs only. Interpretation of the flow along these cycles does not change the flow value; in particular it does not create deficits. By the above lemma, when interpreting a path from $s$ to $v$, the deficit at $v$ is replaced by deficits that sum up to at most $2^{p'(v)} - 1 \leq 2^{\beta} - 1$ times the deficit at $v$ satisfied by this path. ∎

**Corollary 5.11** *If $p'(v) < \log 1.5$ for every node $v$, then $Def(g, \mu)$ decreases by a factor of 2 during the application of Step 3.*

The remaining difficulty is the fact that the function $Def(g, \mu)$ can increase, either when the canonical relabeling is done in Step 1 or in Step 3 when Case 2 applies. However, if $Def(g, \mu)$ has increased by some factor during relabeling (Step 1), or after interpretation of $f'$ (Step 3), then at least one of the nodes is relabeled by the same factor. More precisely,

**Lemma 5.12** *If either during Step 3 of one iteration or Step 1 of the next one, $Def(g, \mu)$ increases by a factor of $\alpha$, then the label of some node decreases by at least a factor of $\alpha$ during Step 1 of the next iteration.*

*Proof*: The increase in $Def(g, \mu)$ during Step 1 is due to relabeling, and hence the deficit at a node can increase during this Step by a factor of $\alpha$ if and only if the label at this node decreases by the same factor. By Lemma 5.10, the increase in $Def(g, \mu)$ during Step 3 is bounded by $2^{\beta}$, where $\beta = \max p'(v)$. Hence, if $Def(g, \mu)$ increases by $\alpha$ during this step, there exists a node $v$ such that $p'(v) \geq \log \alpha$. By Lemma 5.7, this means that $\mu(v)$ decreases by at least $\alpha$ during Step 1 of the next iteration. ∎

Combining the above results, we can bound the overall growth of $Def(g, \mu)$ during an execution of the algorithm.

**Lemma 5.13** *The growth of the function $Def(g, \mu)$ throughout an execution of the algorithm is bounded by a factor of $T^{O(n)}$.*

*Proof*: The function $Def(g, \mu)$ can increase either as a result of Step 1 or Step 3. By the above lemma, such increase is followed by a decrease in the label of one of the nodes by at least the same factor during the subsequent relabeling. The claim follows from Lemma 3.6, that limits the decrease of the label of any node. ∎

Due to Lemma 3.6, Case 2 cannot occur too often. The above lemma helps to bound the number of times Case 1 can occur.

**Theorem 5.14** *The algorithm terminates in $O(n \log T) = O(n^2 \log B)$ iterations.*

*Proof*: Lemma 3.6 shows that Case 2 cannot occur more than $O(n \log T)$ times. When Case 1 applies, the value of $Def(g, \mu)$ decreases by a factor of 2. The value of $Def(g, \mu)$ is at most $O(nBT)$ after the first iteration, and by Theorem 5.6, the algorithm terminates when this value decreases below $L^{-1}$. Lemma 5.13 limits its increase during the algorithm. Hence, Case 1 cannot occur more than $O(n \log T)$ times. ∎

To get a bound on the running time, we have to decide which minimum-cost flow algorithm to use as a subroutine. The best choice turns out to be Orlin's [Orl88] strongly polynomial algorithm.

**Theorem 5.15** *The above generalized flow algorithm can be implemented so that it will use at most $O(n^2 m(m+n \log n) \log n \log B)$ arithmetic operations on numbers whose size is bounded by $O(m \log B)$.*

*Remark*: The choice of a strongly polynomial minimum-cost flow algorithm is somewhat surprising, since our algorithm is not strongly polynomial. The reason for this choice is that, the current best scaling algorithms would give worse running times, even when the size of the numbers in the input is small. Observe that the number of bits needed to represent the capacities of the intermediate minimum-cost flow problems can be as high as $m \log B$, which would make a capacity-scaling algorithm too slow. The classical cost-scaling algorithms cannot be used directly because the costs are (irrational) logarithms of rational numbers. A cost-scaling algorithm can be constructed based on the idea of $\epsilon$-optimality, as done in [Gol87,GT87]. Note that the gain of a flow-generating cycle can be as small as $2^{-O(B^n)}$, and hence the absolute value of the cost of a negative cycle can be as small as $B^{-O(n)}$. This means that the required precision seems to be $\Omega(B^{-n})$, and therefore the cost-scaling algorithms are too slow as well.

## 6 The Fat-Paths Algorithm

Recall that a GAP is defined to be a flow-generating cycle with an augmenting path connecting this cycle to the source. A natural way to make progress towards an optimal solution is to increase the flow along a GAP. We call this a *generalized augmentation.* Clearly, if the augmentations are done along arbitrary GAPs, we do not get a polynomial-time algorithm. One way to improve the running time is to execute only those generalized augmentations that result in a significant progress towards an optimal solution.

We divide a generalized augmentation into two parts: augmenting the flow along a flow-generating cycle and bringing the created excess to the source. Our algorithm first saturates *all* flow-generating cycles, creating excesses at various nodes of the graph, and only then looks for augmenting paths from nodes with excess to the source. Note that this method does not create deficits. A natural way to measure the progress of this algorithm is by the difference between the excess at the source in the current generalized pseudoflow and the excess at the source in the optimal solution. This difference is called the *excess discrepancy.* We shall show that if the excess discrepancy is very small, a single maximum-flow computation produces an optimal solution (similarly to Theorem 5.6).

**Stage 1** Cancel all flow-generating cycles by calling the *Cancel-Cycles* procedure.

**Stage 2** Compute $\mu$, the canonical labeling to the source. Let $E'$ be the set of all arcs with unit relabeled gain. Consider nodes with excess as sources with capacity bounded by the value of their relabeled excess, and compute maximum flow $f'$ from these nodes to the source in the graph induced by the arcs in $E'$ with the relabeled residual capacities. Update the generalized pseudoflow $g$ by setting $g(v,w) = g(v,w) + f'(v,w)\mu(v)\ \forall(v,w) \in E'$.
Terminate if $Ex_g(v) = 0\ \forall v \in V - s$.

**Stage 3** Repetitively call *Fat-Augmentation* procedure to augment the flow along $\Delta$-fat paths from nodes with excess to the source, as long as such paths exist.
Set $\Delta = \Delta/2$.

Figure 3: A single phase of the Fat-Path algorithm.

Consider an augmenting path with an excess in the beginning of the path. Even if this excess is very large, the increase in the excess at the source caused by an augmentation along this path is limited, and depends on both the capacities and the gains of arcs on the path. We call this limit the *fatness* of the path. A simple $v$-$s$ path is $\Delta$-*fat* if, given unlimited supply at $v$, at least $\Delta$ units of flow can be sent to $s$ along this path. An arc is $\Delta$-fat if it belongs to a $\Delta$-fat path. Saturating all the flow-generating cycles before bringing the excesses to the source facilitates the search for $\Delta$-fat paths.

The section consists of three parts. In the first part we describe the *Fat-Path* algorithm, assuming existence of the *Fat-Augmentation* and the *Cancel-Cycles* procedures, and discuss the correctness and the running time of the algorithm. The second part presents the *Fat-Augmentation* procedure. This procedure finds a $\Delta$-fat path in the residual graph and augments the flow along it. The third part of this section is devoted to the description of the *Cancel-Cycles* procedure, which transforms a generalized pseudoflow with flow-generating cycles in its residual graph into a generalized pseudoflow with no such cycles, without creating any deficits.

## 6.1 Fat-Path Algorithm - Overview

The Fat-Path algorithm, described in Figure 3, solves the restricted problem in phases. The goal of each phase is to decrease the bound on the excess discrepancy by at least a constant factor. The input and output of a phase is a generalized pseudoflow with non-negative excesses.

Each phase consists of three stages. The input to the first stage is a generalized pseudoflow with non-negative excesses and, possibly, with flow-generating cycles. This stage uses procedure *Cancel-Cycles*, described in Section 6.3, to cancel all such cycles, creating new excesses at various nodes of the graph.

In the second stage we test whether it is possible to bring all the excesses to the source over the most efficient residual paths. This is done by canonically relabeling to the source and then computing a maximum flow in the graph induced by the unit relabeled gain arcs. Observe that by Lemma 2.7, the canonical relabeling to the source applies. There are no flow-generating cycles in the residual graph after the first stage, and an augmentation of the flow along arcs with a unit relabeled gain does not create new ones. Therefore, if we succeed in bringing all excess to the source using these arcs, the resulting generalized circulation is optimal, and the algorithm terminates. This stage may be viewed as a variation of a single iteration of the maximum-flow based algorithm, described in Section 4.

The input to the third stage is a generalized pseudoflow with nonnegative excesses and no flow-generating cycles, and a "fatness" parameter $\Delta$ (initially set to $B^2$). This stage iteratively augments the flow on $\Delta$-fat paths from nodes with excess to the source, reducing the excess at the starting node of the path and increasing the excess at the source. The stage continues until there are no more $\Delta$-fat paths from nodes with excess to the source. Before the next phase, the value of $\Delta$ is decreased by a factor of 2. We shall show that this stage does not introduce flow-generating cycles in the graph induced by the $\Delta$-fat arcs.

The following lemma indicates that the excess discrepancy is a good measure of progress.

**Lemma 6.1** *If the excess discrepancy is below $L^{-1}$, and there are no negative excesses or flow-generating cycles, then all the excesses will be brought to the source by Stage 2 of the Fat-Path algorithm.*

*Proof*: We claim that after Stage 2 the excess discrepancy is an integer multiple of $L^{-1}$. This means that the excess discrepancy decreases by at least $L^{-1}$ in between any two iteration, which directly implies the lemma.

Consider the generalized pseudoflow $g$ before Stage 2. Let $(S, \overline{S})$ be the cut saturated by the maximum flow computation in Stage 2, such that there are no excesses left in $S$ and $s \in S$. Let $\mu$ denote the canonical labels computed during this stage. The excess at the source after this stage is equal to the sum of $Ex_{g,\mu}(v)$ for $v \in S$ plus the sum of the relabeled capacities of the arcs with relabeled gain 1 entering $S$. Applying Theorem 3.7 to $V' = S$, the pseudoflow $g$, and the labels $\mu$, we conclude that this sum is an integer multiple of $L^{-1}$. ▮

The following lemma bounds the excess discrepancy in terms of $\Delta$.

**Lemma 6.2** *The excess discrepancy at the end of a phase is at most $O(m\Delta)$.*

*Proof*: Let $g$ be a generalized pseudoflow at the end of a phase and let $g^*$ be an optimal generalized circulation. Let $E^+ = \{e | g^*(e) > g(e)\}$ be the arcs with positive residual flow, and let $G^+ = (V, E^+)$. The residual pseudoflow $g^* - g$ can be decomposed according to Theorem 2.6. The arcs opposite to the ones used by the cycles in the decomposition are in the residual graph of the generalized circulation $g^*$, and therefore the decomposition consists only of paths from nodes with excess to the source and GAPs. Note that each one of these paths and GAPs is in $G^+$. There are at most $O(m)$ paths and GAPs used, and therefore it is sufficient to show that each one of them contributes at most $\Delta$ to the excess at the source.

First, consider paths from excesses to the source. By construction, no path in $G_g$ from excess to the source is $\Delta$-fat. Also, we have $E^+ \subseteq E_g$. Hence, there are no $\Delta$-fat paths in $G^+$ either. Therefore, a path cannot contribute more than $\Delta$ to the excess at the source.

By Lemma 6.8, the subgraph of $G_g$ induced by the $\Delta$-fat arcs has no flow-generating cycles. We know that $E^+ \subseteq E_g$, and therefore every flow-generating cycle in $G^+$ has at least one arc

```
Initialize: Start with an empty tree.
            For all $v \in V - s$, set $Gain(v) = 0$.
            Set $Gain(s) = 1$ and $Father(s) = s$.
Step 1      while there are nodes outside the tree with non-zero Gain do
                a) Let $v$ be a node not in the tree with the maximum Gain.
                b) Add the arc $(v, Father(v))$ to the tree.
                c)For every neighbor $w$ of $v$ (i.e., $(w, v) \in E$), if
                  $u_g(w, v)\gamma_\mu(w, v)Gain(v) \geq \Delta$ and $Gain(w) < Gain(v)\gamma_\mu(w, v)$,
                  then update $Gain(w) = Gain(v)\gamma_\mu(w, v)$ and $Father(w) = v$.
Step 2      Relabel the nodes in the tree using $\mu(v)Gain(v)$ as the new label of node $v$ (so that
            the relabeled gain of the tree arcs becomes 1).
Step 3      If there exists a node with excess in the tree, augment the current generalized
            pseudoflow from this node to $s$ along the unique path in the tree.
```

Figure 4: The Fat-Augmentation algorithm.

that is not $\Delta$-fat. Consider a GAP in the decomposition and let $(v, w)$ be a non $\Delta$-fat arc along the flow-generating cycle in the GAP. The contribution of this GAP to the excess of the source is bounded by the fatness of the $(v, s)$-path in the GAP. The arc $(v, w)$ is not $\Delta$-fat, which implies that the fatness of this path is less than $\Delta$. ▮

In Sections 6.2 and 6.3, we prove that the *Fat-Augmentation* and the *Cancel-Cycles* procedures can be implemented to run in $O(m + n \log n)$ and $O(mn^2 \log n \log B)$ time, respectively. This leads to the following theorem:

**Theorem 6.3** *The* Fat-Path *algorithm runs in $O(m^2n^2 \log n \log^2 B)$ time.*

*Proof*: By Lemma 6.1, the algorithm terminates after $O(m \log B)$ phases. An augmentation either decreases the number of nodes with excess or increases the excess at the source by at least $\Delta$. By Lemma 6.2, this gives an $O(m)$ bound on the number of augmentations per phase. ▮

## 6.2 Fat-Augmentation

The procedure *Fat-Augmentation*, shown in Figure 4, is a crucial part of the *Fat-Path* algorithm. It finds a node with excess such that the source is reachable from this node through a $\Delta$-fat path, then finds the highest-gain $\Delta$-fat augmenting path from this node to the source, and augments.

First we describe a simple (but not the most efficient) version of the algorithm. Consider a highest-gain augmenting path from a node $v$ to the source. Either this path is $\Delta$-fat, or the capacity of the arc $(v, w)$ that would be saturated when the flow is augmented along this path, times the gain of the part of the path from $v$ to the source, is below $\Delta$. In this case we call $(v, w)$ *critical*. The observation that a critical arc cannot be $\Delta$-fat leads to a simple algorithm. First, find a highest-gain path from a node with excess, and check its fatness. If it is $\Delta$-fat, we are done. If not, look for a critical arc, delete this arc from the set of arcs considered when searching for a highest-gain path, and repeat.

It is important to note that if an augmentation is done along the highest-gain $\Delta$-fat path, then disregarded arcs do not become $\Delta$-fat.

The above algorithm runs in polynomial time, but its running time can be improved. A faster method, proposed to us by Robert Tarjan (personal communication), is the one described in Figure 4. The main idea is to recognize and disregard arcs that are not $\Delta$-fat while constructing a highest-gain path. Let $G_g^\Delta$ denote the graph induced by the $\Delta$-fat arcs in the residual graph of $g$. We search for a tree in $G_g^\Delta$ that is rooted at $s$, such that the path in this tree from any node to the source $s$ has the highest gain among all $\Delta$-fat paths from this node to the source. We call a tree with this property the "Max-Gain" tree. In order to improve the efficiency of finding this tree, we maintain a set of labels $\mu$, such that all $\Delta$-fat arcs have relabeled gains of at most 1. In particular, these labels guarantee that there are no flow-generating cycles in $G_g^\Delta$.

The input to *Fat-Augmentation* is a parameter $\Delta$, generalized pseudoflow $g$, and a set of labels $\mu$, such that the relabeled gain $\gamma_\mu$ of arcs in $G_g^\Delta$ is at most 1. The output is a generalized pseudoflow $g'$, together with updated labels $\mu'$, such that the relabeled gain of the arcs in $G_{g'}^\Delta$ with respect to $\mu'$ is at most 1.

The procedure can be viewed as a search for a shortest-path tree in $G_g^\Delta$, where the length of an arc with relabeled gain $\gamma_\mu$ is $-\log \gamma_\mu$. There are no $\Delta$-fat arcs with relabeled gain of above 1, and therefore the $\Delta$-fat tree can be "grown" like in the Dijkstra's shortest-path algorithm.

Initially, the $\Delta$-fat tree consists only of $s$. With each node $v$, we associate $Gain(v)$, which is the maximum gain of a $\Delta$-fat path found so far from $v$ to $s$. We initialize $Gain(v) = 0$, for all $v \in V - \{s\}$. Like in the Dijkstra's shortest path algorithm, at each iteration we find a node $v$ that has the largest *Gain* among the nodes not in the tree, and add it to the tree. The difference is in the way we update the *Gain* of its neighbors. An arc $(v, w)$ is disregarded by the algorithm if $u_g(v, w)Gain(w)\gamma_\mu(v, w) < \Delta$. We show below that an arc is disregarded if and only if it is not $\Delta$-fat.

After the tree is constructed, the labels $\mu$ are updated, so that the relabeled gains of the arcs in the tree are equal to 1. Then we find a node in the tree with positive excess and augment the flow on the path from this node to the source.

To prove the correctness of the algorithm, we need to show that each augmentation is done only along a $\Delta$-fat path, that the procedure finds a $\Delta$-fat path if such a path exists, and that the subgraph of the $\Delta$-fat arcs does not contain a flow-generating cycle.

**Lemma 6.4**

1. *The arcs used for updates in Step 1c are $\Delta$-fat.*
2. *Augmentation is done on a $\Delta$-fat path.*

*Proof*: Consider a path from some node to $s$ in the tree constructed by the algorithm, and let $(v, w)$ be a critical arc with respect to this path. This means that the fatness of this path is equal to $u_g(v, w)Gain(v) = u_g(v, w)Gain(w)\gamma_\mu(v, w)$. By construction, this is at least $\Delta$, and hence the path is $\Delta$-fat. A similar argument shows that all arcs used for updates are $\Delta$-fat. ▮

**Lemma 6.5** *If all $\Delta$-fat arcs have relabeled gains of less than or equal to 1 at the beginning of the procedure* Fat-Augmentation, *then Step 2 constructs a valid Max-Gain tree.*

*Proof*: By Lemma 6.4, the arcs used for updating in Step 1c are all $\Delta$-fat. Therefore the length $(-\log \gamma_\mu)$ of the arcs considered for updates is non-negative. Step 1 is an implementation of Dijkstra's shortest path algorithm on the graph induced in $G_g$ by the considered arcs. Therefore the constructed tree is the shortest-path tree in this graph.

To prove the lemma, we have to show that disregarded arcs are not $\Delta$-fat. Let $(v, w)$ be the first $\Delta$-fat arc disregarded by the algorithm. By definition there exists a path $P_1$ from $w$ to $s$ such that $(v, w)$ and $P_1$ form a $\Delta$-fat path. Denote by $P_2$ the path from $w$ to $s$ in the tree. The arc $(v, w)$ is the first $\Delta$-fat arc disregarded by the algorithm, and therefore, by construction, $P_2$ is the highest gain $\Delta$-fat path. Moreover, we have

$$Gain(w) = \prod_{e \in P_2} \gamma_\mu(e) \geq \prod_{e \in P_1} \gamma_\mu(e).$$

The path $(v, w)$ and $P_1$ is $\Delta$-fat, and therefore $u_g(v, w)\gamma_\mu(v, w) \prod_{e \in P_1} \gamma_\mu(e) \geq \Delta$. Hence, $u_g(v, w)Gain(v)\gamma_\mu(v, w) \geq \Delta$. This contradiction shows that no $\Delta$-fat arc will be disregarded by the algorithm. ∎

As an immediate corollary we get the following lemmas.

**Lemma 6.6** *If the procedure did not augment, the source is not reachable in $G_g^\Delta$ from any node with excess.*

**Lemma 6.7** *If all $\Delta$-fat arcs have relabeled gain less than or equal to 1 at the beginning of an iteration of the procedure* Fat-Augmentation, *then the same is true after the relabeling in Step 2.*

**Lemma 6.8** *In the graph $G_{g'}^\Delta$, where $g'$ is the generalized pseudoflow $g'$ returned by the procedure, there are no flow-generating cycles, and all arcs have relabeled gain of at most 1.*

*Proof*: The new labels $\mu$ are computed so that the augmentation is done along a path with relabeled gain 1. Therefore, the arcs whose residual capacity increases due to this augmentation all have relabeled gain 1. Since augmentation is done along the highest gain $\Delta$-fat path, arcs that do not lie along this path do not become $\Delta$-fat. ∎

The algorithm *Fat-Augmentation* can be implemented using Fibonacci heaps, similarly to the Fredman-Tarjan [FT87] implementation of the Dijkstra's shortest path algorithm.

**Theorem 6.9** *The* Fat-Augmentation *algorithm runs in $O(m + n \log n)$ time.*

## 6.3 Canceling Flow-Generating Cycles

The aim of the algorithm described in this section is to convert a generalized pseudoflow $g$ into a generalized pseudoflow $g'$ whose residual graph has no flow-generating cycles, without decreasing the excess at the nodes. The idea is to cancel flow-generating cycles in the residual graph, while increasing the excesses at some nodes of the graph and without creating any deficits. The algorithm is an adaptation of Goldberg and Tarjan's [GT88] minimum cost flow algorithm that iteratively cancels negative-cost cycles in the residual graph.

Throughout this section, we refer to a generalized pseudoflow with no flow-generating cycles in the residual graph as *passive*. The Cycle-Canceling algorithm of Goldberg and Tarjan is based on the idea of $\epsilon$-optimality. A circulation is said to be $\epsilon$-optimal if the mean of the arc costs along any cycle in its residual graph, *i.e.*, the sum of the costs divided by the number of arcs in the cycle) is at least $-\epsilon$. To adapt this algorithm for generalized pseudoflows, we introduce the cost $c(v, w) = -\log \gamma(v, w)$ on the arcs. We say that a generalized pseudoflow is *$\epsilon$-passive* if the mean cost of a cycle in its residual graph is at least $-\epsilon$. In other words, a generalized pseudoflow is $\epsilon$-passive if the geometric mean of the gain-factors along every residual cycle is at most $2^{\epsilon}$. Given a generalized pseudoflow $g$, let $\epsilon_g$ denote the minimum $\epsilon$ such that $g$ is $\epsilon$-passive. The following lemma corresponds to Theorem 3.3 in [GT88].

**Lemma 6.10** *A generalized pseudoflow $g$ can be relabeled so that the relabeled gain of any arc in the residual graph of $g$ is at most $2^{\epsilon_g}$.*

*Proof*: Define the length of an arc $(v, w)$ in the residual graph to be $l(v, w) = -\log \gamma(v, w) + \epsilon_g$. By definition of $\epsilon_g$, there are no negative-cost cycles with respect to this length. For $v \in V$, let $\rho(v)$ denote the length of the shortest path from $v$ to $s$ in the residual graph. (By Lemma 2.7 such path exist.) It can be seen that relabeling the graph using labels $\mu(v) = 2^{-\rho(v)}$ leads to a graph with the relabeled gain of every arc being below $2^{\epsilon_g}$. ▮

The following lemma corresponds to Theorem 3.2 in [GT88].

**Lemma 6.11** *If $\epsilon_g \leq 1/(nT)$, then any $\epsilon$-passive generalized pseudoflow is passive.*

*Proof*: Consider a cycle in the residual graph of a pseudoflow. By definition of $T$, if this cycle has gain above 1, then its gain is at least $1 + T^{-1}$. This means that the mean cost of this cycle is at most $-(1/n)\log(1 + T^{-1}) \leq 1/(nT)$. ▮

In the case of the minimum-cost flow problem, the Goldberg-Tarjan Cycle Canceling algorithm cancels cycles consisting of arcs with negative reduced cost. In the case of generalized pseudoflows we shall cancel cycles consisting of arcs with relabeled gain of above 1.

The algorithm is described in Figure 5. It starts with $\epsilon = \log B$ and proceeds in phases, where a phase consists of relabeling the graph so that the relabeled gain of every arc in the residual graph is at most $2^{\epsilon}$ (see Lemma 6.10), and cancelling all cycles in the graph induced by the arcs with relabeled gain above 1. Observe, that in the end of a phase any flow-generating cycle contains at least one arc with relabeled gain below 1, and the rest of the arcs have relabeled gain below $2^{\epsilon}$. Hence, we can set $\epsilon = (1 - 1/n)\epsilon$ and start the next phase.

**Lemma 6.12** *The algorithm terminates in $O(\log T) = O(n \log B)$ phases, producing a generalized pseudoflow with no flow-generating cycles.*

**Step 1** Relabel the residual network so that $\gamma(v,w) \leq 2^{\epsilon}$ for every $(v,w)$ in the residual graph of the generalized pseudoflow $g$.

**Step 2** Cancel all cycles in the residual graph of $g$ where the relabeled gain of every arc is above 1.
Set $\epsilon = (1 - \frac{1}{n})\epsilon$.

Figure 5: A single phase of the *Cancel-Cycle* algorithm.

*Proof*: Immediate from Lemma 6.11 and the above discussion. ∎

Each time we cancel a cycle, we saturate at least one resudual arc with relabeled gain above 1, and do not create any new such arcs. Therefore, we cancel at most $m$ cycles during a single phase. Canceling a single flow-generating cycle, while creating excess at one of the nodes of the cycle, can be done in $O(n)$ time. By marking nodes that do not belong to cycles, we can easily obtain a running time of $O(nm)$ per phase, which leads to a total running time of $O(n^3 m \log B)$.

This running time can be improved using a variant of the Dynamic Tree data structure. Dynamic trees were introduced in [ST83,Tar83,ST85], and implement the operations described in Figure 6, were each operation takes amortized $O(\log n)$ time. This data structure was used for speeding up several maximum flow and minimum cost flow algorithms [ST83,Gol87,GTar,GT87,GT88], including the Cycle-Canceling algorithm for minimum-cost flows. The main idea is that augmenting the flow along a path does not saturate all the arcs along this path. Instead, the path is subdivided into shorter paths. Storing these paths in a dynamic tree data structure allows to use the *add-value* operation to augment the flow in time wich is logarithmic in the length of the path.

Unfortunately, this technique can not be applied directly for generalized circulations. The problem arises when we want to cancel a cycle. In order to do this, we have to update the residual capacities of the arcs along the cycle. The amount of flow pushed along the arcs of the cycle is different in each arc and depends on the gains of the arcs. Therefore this update can not be done by subtracting the same value from all of them, as it is done in the case of flows.

We use a variant of the Dynamic Tree data structure, which we call the *Generalized Dynamic Tree* (GDT). The operations supported by this data structure are shown in Figure 7. The data structure is used to store and update the residual capacities of the arcs along the paths currently considered for augmentation. These arcs form a set of disjoint rooted trees. We shall show below that the Generalized Dynamic Tree operations can be implemented in $O(\log n)$ amortized time (Theorem 6.14).

Using Generalized Dynamic Trees, we can speed up Step 2 of the *Cancel-Cycle* algorithm to run in $O(m \log n)$ time. This implementation is shown in Figure 8. The algorithm picks a node $v$ that is not yet marked as useless, and finds the root $r$ of the tree that this node belongs to. If there are no arcs with relabeled gain above 1 from $r$ to some node which was not yet marked as useless, we mark $r$ as useless, remove it from the tree, and decompose the tree into a number of smaller trees. On the other hand, if there exists an arc $(r,w)$ with relabeled gain above 1 and $w$ was not yet marked, then $w$ belongs to some tree. Let $r'$ be the root of this tree. If $r'$ and $r$ are different nodes, then at this point we know that there is an augmenting path from $v$ to $r'$, that uses $(r,w)$.

| | |
|---|---|
| *make-tree*$(v)$: | Make node $v$ into a one-node dynamic tree. Node $v$ must be in no other tree. |
| *find-root*$(v)$: | Find and return the root of the tree containing node $v$. |
| *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 node $w$ 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. |

Figure 6: Dynamic tree operations.

| | |
|---|---|
| *make-tree*$(v)$: | Make node $v$ into a one-node dynamic tree. Node $v$ must be in no other tree. |
| *find-root*$(v)$: | Find and return the root of the tree containing node $v$. |
| *find-cap*$(v)$: | Find and return the residual capacity of the tree arc connecting $v$ to its parent. If $v$ is a tree root, return infinity. |
| *find-gain*$(v)$: | Find and return the gain of the path in the tree connecting node $v$ to to *find-root*$(v)$. |
| *find-sat*$(v)$: | Find and return node $w$, such that the arc between $w$ and its parent is the first one to get saturated if we increase the flow along the path from $v$ to *find-root*$(v)$. In case of a tie, choose the node $w$ closest to the tree root. If $v$ is a tree root, return $v$. |
| *change-cap*$(v, x)$: | Update the residual capacities of the arcs on the path from $v$ to *find-root*$(v)$. The residual capacity of each arc $(w, w')$ on this path is decreased by $x$ times the gain of the path from $v$ to $w$ in the tree. |
| *link*$(v, w, x, \gamma)$: | Combine the trees containing $v$ and $w$ by making $w$ the parent of $v$ and assigning $x$ and $\gamma$ to be the residual capacity and gain of this arc, respectively. 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. |

Figure 7: Generalized Dynamic Tree operations.

```
[Initialize]
for each node $v$, unmark $v$ and perform *make-tree* $(v)$;
while there exists an unmarked node $v$ do begin
    let $r = \textit{find-root}(v)$;
    if there exists an arc $(r, w)$ in the residual graph with $\gamma_\mu(r, w) > 1$
    then begin
        if $\textit{find-root}\ (w) \neq r$ then do [extend the path]
                                $\textit{link}(r, w, u_{g,\mu}(r, w), \gamma_\mu(r, w))$;
        else begin [cancel the cycle]
          $\delta = \min\{u_{g,\mu}(r, w), \textit{find-gain}(w)\textit{find-cap}(\textit{find-sat}(w))\}$;
          $g(r, w) = g(r, w) + \mu(r)\delta$;
          $\textit{change-cap}(w, \delta\gamma_\mu(r, w))$;
          while $\textit{find-cap}(\textit{find-sat}(w)) = 0$ do begin
                $z = \textit{find-sat}(w)$;
                $g(z, \textit{parent}(z)) = u(z, \textit{parent}(z))$;
                $\textit{cut}(z)$;
          end;
        end;
    else begin [remove $r$ from the path]
      mark $r$;
      for each node $z$ such that $r = \textit{parent}(z)$ do begin
         $g(z, r) = u(z, r) - \mu(z)\textit{find-cap}(z)$;
         $\textit{cut}(z)$;
      end;
    end;
end;
end.
```

Figure 8: Implementation of Step 2 of the *Cancel-Cycles* algorithm.

In this case we insert $(r, w)$ into our data structure by linking the two trees. If both $v$ and $w$ belong to the same tree, we have found a flow-generating cycle that consists of the arc $(r, w)$ and the path from $w$ to $r$ in the tree. In this case we augment the flow along this cycle, removing from the tree all arcs whose residual capacity becomes zero.

A straightforward analysis of the above procedure shows that it requires $O(m)$ tree operations. By Theorem 6.14, the use of Generalized Dynamic Trees leads to $O(m \log n)$ running time per phase. As it was already mentioned, after at most $O(n^2 \log B)$ phases the pseudoflow is passive, and therefore we have:

**Theorem 6.13** *The* Cycle-Canceling *algorithm runs in* $O(mn^2 \log n \log B)$ *time.*

## 6.4 Implementing the Generalized Dynamic Tree Data Structure

The implementation of Genaralized Dynamic Trees is based on two data structures. The first one is similar to the standard Dynamic Tree, and is used to store the gains on the arcs of the path. Using it, we can compute the gain of any path from a node to the root of the tree this node belongs to in amortized $O(\log n)$ time.

To implement the other operations that are supported by a Generalized Dynamic Tree data structure, we use a variant of Dynamic Trees that, in addition of supporting all the dynamic tree operations, supports the *multiply-value* operation that multiplies all the values stored in a tree by a constant. This data structure is used to store residual capacities that are *relabeled to the root of the tree.* In other words, instead of the capacity of the arc $(v, w)$, we store this capacity multiplied by the gain of the path from $v$ to the root. The advantage of storing the capacities in this way is that changing the flow along a path changes the residual capacities of the arcs along the path by different amounts, dependent on the gains, whereas the relabeled capacities are changed by the same amount. The *multiply-value* operation is needed to relabel the capacities of the new trees to the new roots when we *cut* or *join* the trees. We will show (Theorem 6.15) that a sequence of dynamic trees operations, possibly including *multiply-value* operations, can be implemented in $O(\log N)$ amortized time, where $N$ is the number of *make-tree* operations used. As a corollary, we get the following theorem.

**Theorem 6.14** *A sequence of $M$ Generalized Dynamic Tree operations takes $O(M \log N)$ time, where $N$ is the number of* make-tree *operations in the sequence.*

In order to be able to add the *multiply-value* operation to the standard dynamic tree operations, we change the way information is stored inside each element of the dynamic tree. Recall, that in a dynamic tree each tree is regarded as a collection of arc-disjoint paths. Dynamic tree operations are implemented in terms of operations on these paths. In order to implement a new operation, one has to add this operation to the set of operations supported by the underlying data structure that implements paths.

Each path in a Dynamic Tree is represented as a search tree, where the key is the distance from the end of the path. In the implementation suggested in [ST85] the search tree used is a splice tree. In order to distinguish between the trees in the graph (defined by the *link* and *cut* operations), and the search trees that represent paths, we call the latter trees *solid.*

Let $Minvalue(v)$ denote the minimum among the values of the descendants of $v$ in a solid tree. Each node of a solid tree holds the following information:

$$\begin{aligned}
\Delta value(v) &= value(v) - Minvalue(v) \\
\Delta min(v) &= \begin{cases} Minvalue(v) & \text{if } v \text{ is a solid tree root} \\ Minvalue(v) - Minvalue(parent(v)) & \text{otherwise.} \end{cases}
\end{aligned}$$

Each operation on a path, say *find-minimum*$(v)$, consists of scanning the solid tree representing the path from the node associated with $v$ to the root of the tree. $Minvalue(v)$ can be computed by summing the values of $\Delta min$ in the nodes encountered during this scan. Using this representation, adding a value to all the capacities on a path requires adding this value to $\Delta min$ of the root of the solid tree, which takes constant time.

In order to be able to multiply the stored values by a constant without scanning all the nodes on the path, each node contains the following information instead of the above data:

$$\begin{aligned}
\Psi value(v) &= \begin{cases} \Delta value(v) & \text{if } v \text{ is a solid tree root,} \\ \Delta value(v)/\Delta value(parent(v)) & \text{otherwise.} \end{cases} \\
\Psi min(v) &= \begin{cases} \Delta min(v) & \text{if } v \text{ is a solid tree root,} \\ \Delta min(v)/\Delta min(parent(v)) & \text{otherwise.} \end{cases}
\end{aligned}$$

It is easy to see, that by scanning the nodes on a path from a given node to the root of the solid tree, we can compute $\Delta min$ and $\Delta value$ of all the scanned nodes, and hence, we can also compute $Minvalue(v)$ and $value(v)$. More precisely, in order to be able to compute $\Delta min$ and $\Delta value$ even if some of them are zero, we keep the above ratios as tuples.

Observe that a change in the information stored in the root of a solid tree is sufficient in order to multiply $value(v)$ for every node in the solid tree by a constant. Adding a constant to $value(v)$ for all nodes $v$ in the solid tree can be done by changing the information stored in the root of the solid tree and in its sons only. The rest of the operations on paths, required for the dynamic tree operations, are implemented exactly as for the standard dynamic trees.

Using the analysis of the Dynamic Tree data structure by Sleator and Tarjan either in [ST83] or in [ST85], it is straight-forward to see that storing $\Psi value$ and $\Psi min$ instead of $\Delta value$ and $\Delta min$ in the nodes of the solid trees does not increase the running time. This means that it is possible to add the *multiply-value* operation to the set of the operations supported by dynamic trees without increase in the running time, which implies the following theorem.

**Theorem 6.15** *A sequence of $M$ Dynamic Tree operations, possibly including* multiply-value *operations, takes $O(M \log N)$ time, where $N$ is the number of* make-tree *operations in the sequence.*

## 7 Conclusions

We have presented two polynomial-time combinatorial algorithms for the generalized circulation problem. The first algorithm is based on the repeated application of a minimum-cost flow subroutine; the second algorithm is based on the idea of augmenting along the biggest improvement path [EK72] and the idea of canceling negative cycles [GT88,Kle67]. Previous polynomial-time algorithms for the problem were based on general-purpose linear programming techniques, and the combinatorial structure of the problem was used solely for improving the efficiency of computing the required matrix inversions. Our results show that the problem can be handled by methods that are closer to the combinatorial methods traditionally used for network flow problems.

Our algorithms use a combinatorial technique, which we call Canonical Relabeling. Although the labels produced by Canonical Relabeling are closely related to marginal prices from linear programming duality theory, we use them in a different way. In particular, we use the notion of "reduced gain", instead of using the natural notion of "reduced cost". An interesting question is to see whether an algorithm similar to the MCF Algorithm can be developed using the prices and the reduced costs instead of the labeles and the relabeled gains.

A by-product of our research is an increased appreciation of strongly polynomial algorithms. Our algorithms repeatedly use procedures to find shortest paths, maximum-flows, and minimum-cost flows. The input to these procedures may contain numbers which are much bigger than those in the input to the original problem. (See the remark at the end of the Section 5.) Because of this fact, we obtain better bounds by using strongly polynomial algorithms to solve these subproblems. This phenomena suggests that strongly polynomial algorithms may be important in practice as well as in theory: even though the numbers that occur in a statement of a problem may be relatively small, the numbers that occur in the intermediate problems can be large. This observation gives additional motivation to study strongly polynomial algorithms.

The methods used in [Tar85] for designing a strongly polynomial algorithm for the minimum cost circulation problem were extended in [Tar86] for linear programs with integer constraint matrices. This yields an algorithm for the generalized circulation problem, whose running time is independent of the size of the capacities, and polynomial in $n$, $m$, and the number of bits needed to represent the gains. Our algorithms exploit a similarity between the gains and certain corresponding costs. Unfortunately, we were unable to use this similarity to extend the methods of [Tar86] to construct a strongly polynomial algorithm for the generalized circulation problem.

An interesting observation is that despite the apparent similarity between gains in the generalized circulation problem and costs in the minimum-cost circulation problem, the roles of these numbers are quite different. The difference stems from the fact that the gains appear in the constraint matrix of the corresponding linear program, whereas the costs appear only in the objective function.

In some applications, the generalized circulation problem is naturally stated by giving logarithms of gains in the input. For example, in the context of electrical networks, it is customary to use decibels to measure power loss in transmission lines. We call this representation of the problem the *compact representation*. This representation is also natural from the theoretical point of view, because of the intuitive correspondence between a generalized circulation problem with a gain function $\gamma$ and the minimum-cost flow problem with the cost function $-\log \gamma$. On the compact representation, our algorithms do not run in polynomial time, and neither do the algorithms based on the linear programming techniques. In fact, all algorithms described in this paper that run in polynomial time on the original representation of the problem run in pseudo-polynomial time (*i.e.*, in polynomial time if the input numbers are given in unary) on the compact representation. Solving the problem in polynomial time assuming the compact representation is closely related to finding a strongly polynomial algorithm for the generalized circulation problem. It is interesting to note that if the input numbers of the compact representation are given in unary, the simple Maximum Flow Based algorithm has the best time among the algorithms discussed in this paper (see Corollary 4.3).

An important extension of the generalized circulation problem is the *generalized circulation with costs* problem. This problem has costs in addition to gains and capacities, and a fixed price $p(s)$ per unit of commodity of the source. The goal is to maximize profit, where the profit of a generalized circulation $g$ is defined in a natural way. Vaidya's algorithm handles this extended problem with no modifications. Our algorithms, however, cannot handle this problem. It would be interesting to see if our algorithms can be modified to handle the generalized circulation with costs problem. A promising approach is to use the linear programming prices and reduced costs, as discussed above.

## Acknowledgment

We would like to thank Bob Tarjan for allowing us to include his observation that improved the running time of the *Fat-Path* algorithm. We are grateful to David Shmoys for his help in improving the presentation. The second author wishes to express thanks to his advisor, Charles Leiserson, for providing support for this research.

## References

[AGOT87] R. K. Ahuja, A. V. Goldberg, J. B. Orlin, and R. E. Tarjan. Finding minimum-cost flows by double-scaling. 1987. Unpublished manuscript.

[BG61] R. G. Busacker and P. J. Gowen. *A Procedure for Determinimg a Family of Minimal-Cost Network Flow Patterns.* Technical Report 15, O.R.O., 1961.

[BS65] R. G. Busacker and T. L. Saaty. *Finite Graphs and Networks: An Introduction with Applications.* McGraw-Hill, New York, NY., 1965.

[EGK79] J. Elam, F. Glover, and D. Klingman. A strongly convergent primal simplex algorithm for generalized networks. *Math. of O. R.*, 4:39–59, 1979.

[EK72] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. Assoc. Comput. Mach.*, 19:248–264, 1972.

[FF62] L. R. Ford, Jr. and D. R. Fulkerson. *Flows in Networks.* Princeton Univ. Press, Princeton, NJ., 1962.

[FT87] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. Assoc. Comput. Mach.*, 34:596–615, 1987.

[GK73] F. Glover and D. Klingman. On the equivalence of some generalized network problems to pure network problems. *Math. Programming*, 4:269–278, 1973.

[Gol87] A. V. Goldberg. *Efficient Graph Algorithms for Sequential and Parallel Computers.* PhD thesis, M.I.T., January 1987. (Also available as Technical Report TR-374, Lab. for Computer Science, M.I.T., 1987).

[GT87] A. V. Goldberg and R. E. Tarjan. Solving minimum-cost flow problems by successive approximation. In *Proc. 19th ACM Symp. on Theory of Computing*, pages 7–18, 1987.

[GT88] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 388–397, 1988.

[GTar] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *JACM*, (To appear).

[Jew58] W. S. Jewell. *Optimal Flow through Networks.* Technical Report 8, M.I.T., 1958.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[Kha80] L. G. Khachian. Polynomial algorithms in linear programming. *Zhurnal Vychislitelnoi Matematiki i Matematicheskoi Fiziki*, 20:53–72, 1980.

[Kle67] M. Klein. A primal method for minimal cost flows with applications to the assignment and transportation problems. *Management Science*, 14:205–220, 1967.

[KV86] S. Kapoor and P. M. Vaidya. Fast algorithms for convex quadratic programming and multicommodity flows. In *Proc. 18th ACM Symp. on Theory of Computing*, pages 147–159, 1986.

[Law76] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Reinhart, and Winston, New York, NY., 1976.

[Meg83] N. Megiddo. Towards a genuinely polynomial algorithm for linear programming. *SIAM J. Comput.*, 12:347–353, 1983.

[Ona66] K. Onaga. Dynamic programming of optimum flows in lossy communication nets. *IEEE Trans. Circuit Theory*, 13:308–327, 1966.

[Ona67] K. Onaga. Optimal flows in general communication networks. *J. Franklin Inst.*, 283:308–327, 1967.

[Orl88] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. In *Proc. 20th ACM Symp. on Theory of Computing*, 1988. 377-387.

[PE84] P. S. Pulat and S. E. Elmaghraby. *On Maximizing Flow in Generalized Flow Networks.* Technical Report 202, NC State University, 1984.

[ST83] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. System Sci.*, 26:362–391, 1983.

[ST85] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *J. Assoc. Comput. Mach.*, 32:652–686, 1985.

[Tar83] R. E. Tarjan. *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

[Tar85] É. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255, 1985.

[Tar86] É. Tardos. A strongly polynomial algorithm for solving combinatorial linear programs. *Operations Research*, 250–256, 1986.

[Vai87] P. M. Vaidya. An algorithm for linear programmin that requires $O(((m+n)n^2+(m+n)^{1.5}n)L)$ arithmetic operations. In *Proc. 19th ACM Symp. on Theory of Computing*, pages 29–38, 1987.