Polynomial-Time Highest-Gain Augmenting Path Algorithms for the Generalized Circulation Problem

by

Donald Goldfarb
Zhiying Jin
James B. Orlin

WP #3909-96 June 1996

Polynomial-Time Highest-Gain Augmenting Path Algorithms for the Generalized Circulation Problem

Donald Goldfarb*

Zhiying Jin*

James B. Orlin†

June 24, 1996

Abstract

This paper presents two new combinatorial algorithms for the generalized circulation problem. After an initial step in which all flow-generating cycles are canceled and excesses are created, both algorithms bring these excesses to the sink via highest-gain augmenting paths. Scaling is applied to the fixed amount of flow that the algorithms attempt to send to the sink, and both node and arc excesses are used. The algorithms have worst-case complexities of $O(m^2(m + n\log n)\log B)$, where n is the number of nodes, m is the number of arcs, and B is the largest integer used to represent the gain factors and capacities in the network. This bound is better than the previous best bound for a combinatorial algorithm for the generalized circulation problem, and if $m = O(n^{4/3-\epsilon})$, it is better than the previous best bound for any algorithm for this problem.

Key Words: Network flows, generalized maximum flow, generalized circulation, highest-gain augmentating path, arc excess, excess scaling.

*Department of Industrial Engineering and Operations Research, Columbia University, New York, New York 10027. This research was supported in part by NSF Grants DMS 94-14438 and DMS 95-27124 and DOE Grant DE-FG02-92ER25126.

†Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139. This research was supported in part by ONR Grant N00014-96-1-0051 and a grant from the UPS foundation.

1 Introduction

The *generalized circulation problem* is a generalization of the maximum flow problem in which each arc (v, w) in the underlying network has a gain factor $\gamma(v, w)$ associated with it. If $g(v, w)$ units of flow are sent from node v to node w along arc (v, w), then $\gamma(v, w)g(v, w)$ units arrive at w. Specifically, let $G = (V, E, u, \gamma, s)$ be a directed network of n nodes and m arcs, where V is the set of nodes, E is the set of directed arcs, $u : E \rightarrow R^+ \cup \{+\infty\}$ is a capacity function, $\gamma : E \rightarrow R^+$ is a gain function, and $s \in V$ is the *sink*. The objective of the *generalized circulation problem* is to find a flow in G that maximizes the net amount of flow into s while satisfying flow conservation at each node and capacity constraints on each arc. Many applications of this problem are described and referenced in the books of Ahuja, Magnanti, and Orlin [1] and Lawler [15] and the survey papers of Glover, Hultz, and Klingman [6] and Glover, Klingman, and Phillips [5].

Since the generalized circulation problem can be formulated as a linear programming problem, it can be solved by general purpose linear programming algorithms, including simplex, ellipsoid [13] and interior point methods [12], which can be adapted to take advantage of network structure. Combinatorial algorithms based upon flow augmentations specifically designed to solve the generalized circulation problem have also been developed. These include the algorithms of Onaga [16] and Truemper [19], and the recently proposed polynomial-time algorithms of Goldberg, Plotkin and Tardos [8]. One of the latter algorithms, Algorithm MCF, is based on the repeated application of a minimum cost flow subroutine and has a complexity of $O(n^2m(m + n \log n) \log n \log B)$. For this bound, and throughout this paper, we assume that all capacities are integers, each gain is a ratio of two integers, and B is the largest of these integers. Improved version of Algorithm MCF have been proposed by Goldfarb and Jin [10], [11]. The algorithm given in [11] is a simplex version of the algorithm given in [10] and both algorithms have a complexity of $O(n^2m(m+n \log n) \log B)$.

Goldberg, Plotkin and Tardos [8] proposed a second algorithm, called the Fat-Path algorithm. That algorithm repeatedly cancels all flow-generating cycles and sends the excesses created by this process to the sink along fat (big improvement) paths. A modified version of the Fat-Path algorithm, proposed by Radzik [18], improves its complexity from $O(n^2m^2 \log n \log^2 B)$, to $O(m^2(m + n \log n \log(n \log B)) \log B)$, by only cancelling flow-generating cycles with relatively big gains.

In this paper, we propose two new simple combinatorial algorithms based on highest-gain generalized flow augmentations. Our algorithms cancel all flow-generating cycles only on the first iteration. On each subsequent iteration, our algorithms bring all excesses created at the first iteration to the sink without creating flow-generating cycles or more excess. They accomplish this by augmenting flow along highest-gain paths, and hence can be viewed as versions of Onaga's [16] algorithm. They can also be viewed as analogs of Orlin [17] excess scaling algorithm for the minimum cost network flow problem. By using both node and arc excesses and scaling, our algorithms achieve a complexity of $O(m^2(m + n \log n) \log B)$. This complexity is the best complexity known for any combinatorial algorithm for the generalized circulation problem.

Two other polynomial time algorithms for the generalized circulation problem should be mentioned. The first is a strongly polynomial approximation algorithm proposed by Cohen and Megiddo [2]. This algorithm runs in at most $O(n^2m^2 \log \epsilon^{-1}(\log m + \log^2 n))$ time for a $(1-\epsilon)$ approximation to an optimal solution and at most $O(n^2m^3(\log m + \log^2 n) \log B)$ time for an optimal solution. The second is an interior point algorithm due to Vaidya [20]. Using the techniques of Kapoor and Vaidya [14] for speeding up such algorithms on network flow problems, it solves the more general *generalized minimum cost network flow* problem in at most $O(n^2m^{1.5} \log B)$ time. While this is the fastest known polynomial-time algorithm for the generalized circulation problem when $m = O(n^{4/3+\epsilon})$, $\epsilon > 0$, our new algorithms are fastest when $m = O(n^{4/3-\epsilon})$.

Our paper is organized as follows. In Section 2, we present some notation and definitions. In Section 3, we briefly describe Goldberg, Plotkin and Tardos' Fat-Path algorithm, and give some of its properties. Our first algorithm and an analysis of its complexity is presented in Section 4. Our second algorithm and its complexity analysis are given in Section 5. Some conclusions are presented in Section 6.

2 Preliminaries

For convenience, we assume that G has no multiple arcs; if there is an arc from node v to node w, this arc is unique and is denoted by (v, w). Without loss of generality, we assume that for each arc $(v, w) \in E$, arc (w, v) is also in E and has a gain factor $\gamma(w, v) = 1/\gamma(v, w)$. A *generalized pseudoflow* is a function $g : E \rightarrow R$ that satisfies the capacity constraints: $g(v, w) \leq u(v, w), \forall (v, w) \in E$ and the generalized antisymmetry constraints: $g(v, w) = -\gamma(w, v) g(w, v), \forall (v, w) \in E$. Given a generalized pseudoflow g, the *residual capacity* is a function $u_g : E \rightarrow R$ defined by $u_g(v, w) = u(v, w) - g(v, w)$. The *residual network* with respect to a generalized pseudoflow g is the network $G_g = (V, E_g, u_g, \gamma, s)$, where $E_g = \{(v, w) \in E \mid u_g(v, w) > 0\}$. Without loss of generality, we assume that there is a directed path from each node $v \in V$ to node s in the residual network corresponding to the zero pseudoflow, since any node for which this is not the case, together with all arcs incident to it, can be deleted from the problem. For a given generalized pseudoflow g the *imbalance* at node v is defined as $e_g(v) = -\sum_{(v,w)\in E} g(v,w)$. Node v is said to have *excess* if $e_g(v)$ is positive and *deficit* if $e_g(v)$ is negative.

A *generalized circulation* is a generalized pseudoflow that satisfies $e_g(v) = 0$ at all nodes v other than node s. An optimal solution of the generalized circulation problem is a generalized circulation that gives the maximum excess $e_g^*(s)$.

A *flow-generating (absorbing) cycle* is a cycle for which the product of the gains of the arcs on the cycle is greater (less) than 1. Goldberg, Plotkin, and Tardos [8] call a generalized circulation problem *restricted* if, in the residual network with respect to zero flow, all flow-generating cycles pass through the sink, and prove that a generalized circulation problem is polynomially ($O(nm)$-time) reducible to a restricted problem. The polynomial-time algorithms proposed in [8], [10], and [11] and the new algorithms presented in this paper are all designed to solve restricted problems. Therefore, in the following, all generalized circulation problems that are referred to are assumed to be restricted. At the end of Section 4, we describe how to modify our algorithms so that they can be applied to problems that are not restricted.

The above-mentioned polynomial-time algorithms use a *relabeling* technique originally introduced by Glover and Klingman [7]. Given a function $\mu : V \rightarrow R^+$ and a network $G = (V, E, \gamma, u, s)$, the *relabeled network* is $G_\mu = (V, E, \gamma_\mu, u_\mu, s)$, where $\mu(v)$ is referred as the *label* of node v, the *relabeled gains* are defined as $\gamma_\mu(v,w) = \gamma(v,w)\mu(v)/\mu(w)$ and the *relabeled capacities* are defined as $u_\mu(v,w) = u(v,w)/\mu(v)$. Given a generalized pseudoflow g and a labeling μ, the *relabeled residual capacity* is defined as $u_{g,\mu}(v,w) = (u(v,w) - g(v,w))/\mu(v)$. It can be shown that $g_\mu(v,w) = g(v,w)/\mu(v)$ is a generalized pseudoflow in the relabeled network G_μ and that $g_\mu(v,w) = -\gamma_\mu(w,v)g_\mu(w,v)$ and the residual graphs of g and g_μ are the same. The imbalance at node v with respect to g_μ in the relabeled network is $e_{g,\mu}(v) = e_g(v)/\mu(v)$. The *relabeled residual network* is $G_{g,\mu} = (V, E_{g,\mu}, u_{g,\mu}, \gamma_\mu, s)$, where $E_{g,\mu} = \{(v,w) : u_{g,\mu}(v,w) > 0\}$.

Our algorithm, like the Fat-Path algorithm, uses what Goldberg, Plotkin and Tardos call *canonical relabeling to the sink.* For each node $v \in V$, the *canonical label* $\mu(v)$ is defined as the inverse of the gain of the highest-gain simple $v - s$ path in the residual graph. The gain of a path is a product of the gains of the arcs on the path. Note that $\mu(s) = 1$. The canonical labels μ can be found by solving a shortest path problem, where arc lengths are defined as $c(v,w) = -\log(\gamma(v,w))$

for all $(v, w) \in E$, because whenever μ is computed in our algorithms the residual network contains no flow-generating cycles.

3 The Fat-Path Algorithm

It is instructive to consider the Fat-Path algorithm since our algorithm is closely related to it and was, in fact, motivated by it. The Fat-Path algorithm first cancels all flow-generating cycles. This creates excesses at some nodes in the network. The algorithm then looks for augmenting paths from nodes with excess to the sink. If a flow augmentation is performed along a highest-gain residual path, then the resulting residual network will not contain any flow generating cycles. Therefore, if such flow augmentations are carried out until there is no excess at any node $v \in V - s$, an optimal flow will be obtained. This is essentially Onaga's [16] algorithm.

Unfortunately, if flow augmentations are done only along highest-gain paths, exponentially many augmentations may be necessary. Therefore the Fat-Path algorithm only augments flow along highest-gain paths that can bring to the sink at least Δ units of flow, i.e., the Δ-fat paths. Such paths can be found by solving a shortest path problem. The Fat-Path algorithm maintains a factor Δ, which is divided by 2 each time it is adjusted. In each Δ-phase, Δ is unchanged. By not using some highest-gain paths from the residual network because they are not Δ-fat, the Fat-Path algorithm may create flow-generating cycles during a Δ-phase. Therefore the Fat-Path algorithm cancels flow-generating cycles at the beginning of the next phase before again pushing flow from nodes with excess to the sink s along fat paths. It can be shown that the number of phases required by the Fat-Path algorithm is at most $O(m \log B)$ and each phase, the cost of which is dominated by cancelling flow-generating cycles, can be done in $O(mn^2 \log n \log B)$. This gives a complexity of $O(m^2n^2 \log n \log^2 B)$ for the Fat-Path algorithm.

Radzik [18] modified the Fat-Path algorithm so that it only cancels flow generating cycles with "big" gains, resulting in an algorithm with an improved complexity. Radzik's algorithm and the proof of its complexity are quite complicated.

4 A Highest-Gain Augmenting Path Algorithm

In this section we present the first of two new algorithms for the generalized circulation problem. Both of our algorithms start with a zero generalized pseudoflow. First, all flow-generating cycles are cancelled as in the Fat-Path algorithm, but this is done only once. In subsequent iterations, all of the excesses that were created during the initial step are brought to the sink along highest-gain augmenting paths. This is carried out in phases characterized by a scaling parameter Δ, which is reduced, as we shall show later, by at least a factor of two between phases. During a Δ-phase, if a chosen highest-gain path P is not Δ-fat (i.e., there is at least one arc in it whose relabeled residual capacity is less than Δ), it may not be possible to push Δ units of flow to s through P. To ensure that at most $O(m)$ pseudoflow augmentations are required in each phase, both of our algorithms make use of arc excesses $e_{g,\mu}(v,w)$ for all $(v,w) \in E$, as in [10]. Our two algorithms differ only in the way that the pseudoflow augmentations are done and how arc excesses are treated. When the total relabeled excess at nodes other than node s is less than 2^{-mK}, where K is the smallest integer such that $B \leq 2^K$, (i.e., $K = \lceil \log B \rceil$), our algorithms bring this remaining excess to s by computing a maximum flow in the relabeled residual network $G_{g,\mu}$ using only arcs (v,w) with $\gamma_\mu(v,w) = 1$. This is justified by the following:

Lemma 1. Suppose that the residual network $G_{g,\mu}$ has no flow-generating cycles and all arc excesses are zero. If $Ex_{g,\mu} = \sum_{v \in V-s} e_{g,\mu}(v) < B^{-m}$ and $e_{g,\mu}(v) \geq 0$ for all $v \in V - s$, then all of the total relabeled excess $Ex_{g,\mu}$ can be brought to s by computing a maximum flow from nodes $v \in V - s$ with $e_{g,\mu}(v) > 0$ to s in $G_{g,\mu}$, using only arcs with $\gamma_\mu(v,w) = 1$.

Proof. Let $\bar{g}$ be the pseudoflow when the maximum flow computation ends. Let R be the set of nodes from which s is not reachable using only arcs (v,w) with $\gamma_\mu(v,w) = 1$ in $G_{\bar{g},\mu}$ and let $L = \{(v,w) \in E \mid v \in R, \text{ and } \bar{g}(v,w) \neq 0\}$. Clearly $Ex_{\bar{g},\mu} = \sum_{v \in R} e_{\bar{g},\mu}(v) = -\sum_{(v,w)\in L} \bar{g}_\mu(v,w)$.

Now $L = L^1 \cup L^+ \cup L^-$, where $L^1 = \{(v,w) \in L \mid w \in R \text{ and } \gamma_\mu(v,w) = 1\}$, $L^+ = \{(v,w) \in L \backslash L^1 \mid \bar{g}(v,w) > 0\}$, and $L^- = \{(v,w) \in L \backslash L^1 \mid \bar{g}(v,w) < 0\}$. If $(v,w) \in L^1$ then so is (w,v) and $\bar{g}_\mu(w,v) = -\bar{g}_\mu(v,w)$; hence $\sum_{(v,w)\in L^1} \bar{g}_\mu(v,w) = 0$. Moreover, if $(v,w) \in L^+$, then $\bar{g}_\mu(v,w) = u_\mu(v,w) = u(v,w)/\mu(v)$, and if $(v,w) \in L^-$, then $\bar{g}_\mu(v,w) = -\gamma_\mu(w,v)\bar{g}_\mu(w,v) = -\gamma_\mu(w,v)u_\mu(w,v) = -\gamma(w,v)u(w,v)/\mu(v)$. Therefore,

$$Ex_{\bar{g},\mu} = -\sum_{(v,w)\in L^+} u(v,w)/\mu(v) + \sum_{(v,w)\in L^-} \gamma(w,v)u(w,v)/\mu(v).$$

For each node $v \in V - s$, $1/\mu(v)$ is the product of a subset of the gain factors corresponding to a tree T of highest-gain paths in $G_{g,\mu}$ from nodes in $V - s$ to s and no arc $(v,w) \in L^-$ can be in such a tree since $\gamma_\mu(v,w) < 1$. Since the capacities $u(v,w)$ are integral and the gain factors $\gamma(v,w) = p(v,w)/q(v,w)$, where $p(v,w)$ and $q(v,w)$ are integers less than or equal to B, it follows that if $Ex_{\bar{g},\mu} > 0$, then

$$Ex_{\bar{g},\mu} > 1/ \prod_{(v,w)\in T\cup L^-} q(v,w) \geq 1/B^m.$$

Consequently, $Ex_{\bar{g},\mu} = 0$ and the lemma is proved. □

The above lemma is a slightly tightened version of Lemma 7.1 in [8].

Main Algorithm

```
Step 0. Cancel all flow-generating cycles in $G$ (This yields a pseudoflow $g$).
        Compute the gain $Gain(v)$ of a highest-gain path from $v$ to $s$ in $G_g$
        and set $\mu(v) \leftarrow 1/Gain(v)$, $\forall v \in V - s$.
        Relabel the network with $\mu$.
        Set $e_{g,\mu}(v,w) \leftarrow 0$ for all $(v,w) \in E$.
Step 1. If $Ex_{g,\mu} \equiv \sum_{v\in V-s} e_{g,\mu}(v) < 2^{-mK}$, then
        begin
          compute a maximum flow $f$ from nodes with excess to the sink node $s$ in $G_{g,\mu}$,
          using only arcs with $\gamma_\mu(v,w) = 1$;
          set $g(v,w) \leftarrow g(v,w) + f(v,w)\mu(v)$, $\forall (v,w) \in E$ and STOP;
        end.
Step 2. Set $\Delta \leftarrow \sum_{v\in V-s} e_{g,\mu}(v)/(2(m+n))$ and call Phase($\Delta$);
        Go to Step 1.
```

The input to Procedure Phase(Δ) is a scaling parameter Δ, a generalized pseudoflow g, and node labels μ, such that the relabeled gain of every arc in $G_{g,\mu}$ is at most 1. Procedure Push-Flow(v) pushes pseudoflow from a node v with relabeled excess at least Δ along a highest-gain path from v to s. The procedure's output is a generalized pseudoflow g', together with updated labels μ', such that the relabeled gain of every arc in $G_{g',\mu'}$ is at most 1.

In Procedure Phase(Δ), each arc (v,w) has associated with it an *arc imbalance* at its tail node v, denoted by $e_g(v,w)$. The relabeled imbalance $e_g(v,w)/\mu(v)$ is denoted by $e_{g,\mu}(v,w)$. These arc imbalances are used to limit the number flow augmentations that can occur in a Δ-phase. In our algorithms, all arc imbalances are nonnegative; hence we will refer to them as *arc excesses.* Initially, all arc excesses are zero, and all arc excesses are transferred to node excesses at the end of each phase.

```
**Procedure Phase(Δ)**
begin
  set $D \leftarrow \{v \in V - s : e_{g,\mu}(v) \geq \Delta\}$;
  while $D \neq \emptyset$ do
  begin
    choose a node $v \in D$;
    call Push-Flow(v);
    using the relabeled gains $\gamma_\mu$, compute the gain $Gain(v)$ of a highest-gain path from $v$ to $s$ in $G_{g,\mu}$
    and set $\mu(v) \leftarrow \mu(v)/Gain(v)$, $\forall v \in V - s$;
    relabel the network with the new labels $\mu$ (all arcs on highest-gain paths have unit relabeled gains);
    reconstruct $D$;
  end;
  set $e_{g,\mu}(v) \leftarrow e_{g,\mu}(v) + \sum_{(v,w)} e_{g,\mu}(v,w)$ for each $v \in V - s$;
  set $e_{g,\mu}(v,w) \leftarrow 0$ for all $(v,w) \in E$;
end.
```

After cancelling all flow-generating cycles in Step 0 of the main algorithm, the residual network does not contain any flow-generating cycles. Since each flow augmentation computed by Procedure Push-Flow is along a highest-gain path in $G_{g,\mu}$, no flow-generating cycles are created (as first observed by Onaga [16]), and we have the following:

Lemma 2. After Step 0 of the main algorithm $G_{g,\mu}$ does not contain flow-generating cycles and $\gamma_\mu(v,w) \leq 1$ for each arc $(v,w) \in G_{g,\mu}$ after each relabeling of the network.

An immediate consequence of Lemma 2 is:

Lemma 3. Let μ^{old} and μ be the labels just before and after a relabeling of the network. Then $\mu(v) \geq \mu(v)^{old}$ and $e_{g,\mu}(v) \leq e_{g,\mu^{old}}(v)$ for all $v \in V$ and $e_{g,\mu}(v,w) \leq e_{g,\mu^{old}}(v,w)$ for all $(v,w) \in E$.

Proof. By Lemma 2, since $\gamma_{\mu^{old}}(v,w) \leq 1$ for all $(v,w) \in E$, $Gain(v) \leq 1$ for all $v \in V$. Therefore $\mu^{old}(v)/\mu(v) = e_{g,\mu}(v)/e_{g,\mu^{old}}(v) = e_{g,\mu}(v,w)/e_{g,\mu^{old}}(v,w) = Gain(v) \leq 1$. □

¿From Lemma 3, the total relabeled excess never increases due to relabeling. In fact, the total relabeled excess can decrease due to flow shrinking (label increase) and/or a flow augmentation to the sink s. Note that a Δ-phase can end even though no flow has been pushed into s. We now give our first algorithm for pushing flow in Procedure Phase(Δ). The pseudoflow augmentations that are computed by the following procedure do not necessarily terminate in the sink node s.

```
Procedure Push-Flow($v$)
begin
  while $v \neq s$ and $e_{g,\mu}(v) \geq \Delta$ do
  begin
    let $(v, w)$ be the first arc on the highest-gain path from $v$ to $s$;
    set $e_{g,\mu}(v) \leftarrow e_{g,\mu}(v) - \Delta$, $e_{g,\mu}(v, w) \leftarrow e_{g,\mu}(v, w) + \Delta$,
    set $\alpha \leftarrow \min\{e_{g,\mu}(v, w), u_{g,\mu}(v, w)\}$, $g_{\mu}(v, w) \leftarrow g_{\mu}(v, w) + \alpha$,
    $e_{g,\mu}(v, w) \leftarrow e_{g,\mu}(v, w) - \alpha$, $e_{g,\mu}(w, v) \leftarrow e_{g,\mu}(w, v) + \alpha$, $g_{\mu}(w, v) \leftarrow -g_{\mu}(v, w)$;
    compute $\beta = \min\{e_{g,\mu}(w, v), \Delta\}$;
    set $e_{g,\mu}(w, v) \leftarrow e_{g,\mu}(w, v) - \beta$, $e_{g,\mu}(w) \leftarrow e_{g,\mu}(w) + \beta$;
    set $v \leftarrow w$;
  end;
end;
```

Procedure Push-Flow terminates when the condition $v = s$ or $e_{g,\mu}(v) < \Delta$ is detected (i.e., the Δ-augmentation of flow is *blocked*). Note that this procedure is somewhat different ¿From the one that is used in the algorithm developed in [10]. The following lemma shows that arc excesses stay bounded by Δ during each Δ-phase.

Lemma 4. In each Δ-phase, both immediately before and after calling Procedure Push-Flow in Procedure Phase(Δ), $e_{g,\mu}(v, w) \geq 0$, $e_{g,\mu}(w, v) \geq 0$ and $e_{g,\mu}(v, w) + e_{g,\mu}(w, v) < \Delta$ for all pairs of arcs (v, w) and $(w, v) \in E$. Furthermore, $e_{g,\mu}(v, w) < u_{g,\mu}(v, w)$ for all $(v, w) \in G_{g,\mu}$.

Proof. Since $e_{g,\mu}(v, w) = 0$ for all $(v, w) \in E$ at the start of a Δ-phase, the lemma holds at this point. Moreover, it is easily verified from Procedure Push-Flow that $e_{g,\mu}(v, w) \geq 0$ for all $(v, w) \in E$ throughout each Δ-phase.

Now suppose that the statements of the lemma hold at the beginning of a step of the inner loop of Procedure Push-Flow in which flow is pushed from node v to node w along arc (v, w). This affects only $e_{g,\mu}(v, w)$, $e_{g,\mu}(w, v)$, $u_{g,\mu}(v, w)$ and $u_{g,\mu}(w, v)$. Let us distinguish quantities at the end of the step from those at the beginning by an overbar.

¿From Procedure Push-Flow we have the following results.

(i) If $\beta < \Delta$, then $\bar{e}_{g,\mu}(w, v) = 0$.

(ii) If $\beta = \Delta$, then $\bar{e}_{g,\mu}(v, w) + \bar{e}_{g,\mu}(w, v) = e_{g,\mu}(v, w) + \Delta - \alpha + e_{g,\mu}(w, v) + \alpha - \beta = e_{g,\mu}(v, w) + e_{g,\mu}(w, v) < \Delta$. Moreover, since $e_{g,\mu}(w, v) < \Delta$ and $u_{\bar{g},\mu}(w, v) = u_{g,\mu}(w, v) + \alpha \geq \alpha$, we have $\bar{e}_{g,\mu}(w, v) = e_{g,\mu}(w, v) + \alpha - \beta < \Delta + \alpha - \Delta \leq u_{\bar{g},\mu}(w, v)$.

(iii) If $\alpha < u_{g,\mu}(v, w)$, then $\bar{e}_{g,\mu}(v, w) = 0$.

(iv) If $\alpha = u_{g,\mu}$, then $\bar{e}_{g,\mu}(v, w) = e_{g,\mu}(v, w) + \Delta - u_{g,\mu}(v, w) < \Delta$ since $e_{g,\mu}(v, w) < u_{g,\mu}(v, w)$. Moreover, $(v, w) \notin G_{\bar{g},\mu}$ since $u_{\bar{g},\mu}(v, w) = 0$.

It is easy to see from the above that for each of the four possible combinations of α and β, the statements of the lemma hold for the quantities with overbars. Hence, if the statements of the lemma hold immediately before Procedure Push-Flow is called, they hold immediately after as well. Since, by Lemma 3, arc excesses do not increase when the network is relabeled in Procedure Phase(Δ), the proof is complete. $\square$

¿From Lemma 4, we have the following:

Lemma 5. At the end of each Δ-phase, the total excess at nodes except node s is at most $(n + m)\Delta$.

The next lemma gives a bound on the number of flow augmentations that can occur in each phase.

Lemma 6. In each phase, the number of flow augmentations (i.e., calls of Procedure Push-Flow) is at most $2(m + n)$.

Proof. Consider the potenial function $F = \sum_{v \in V-s} \lfloor e_{g,\mu}(v)/\Delta \rfloor$. At the beginning of a Δ-phase, $F \leq 2(m + n)$ by Lemma 5. During Procedure Push-Flow when flow is pushed from node v with $e_{g,\mu}(v) \geq \Delta$ to node w, F does not increase since $\beta \leq \Delta$. On the last push along an arc (v, w) in the Procedure Push-Flow, F is reduced by at least 1 since $e_{g,\mu}(v)$ is reduced by Δ and with either $w = s$ or $w \neq s$ and $\bar{e}_{g,\mu}(w) < \Delta$ after the push. Therefore F decreases by at least 1 on every call of Procedure Push-Flow. The lemma follows. $\square$

Let Δ' and Δ be two consecutive scaling factors with $\Delta' < \Delta$ and Ex' and Ex be total relabeled excess at all nodes except node s at the start of Δ'-phase and Δ-phase, respectively. Then from the definitions of Δ' and Δ and Lemma 5, which states that $Ex' \leq (n+m)\Delta$, it follows that $\Delta' = Ex'/(2(m+n)) < (n+m)\Delta/(2(n+m)) = \Delta/2$, and $Ex' < (n+m)\Delta = (n+m)Ex/(2(n+m)) = Ex/2$. Thus, we have:

Lemma 7. Both the scaling factor and total relabeled excess excluding $e_g(s)$ are reduced by at least a factor 2 in each phase.

Since the initial relabeled excess is at most mB^n, we have from the stopping criterion in the main algorithm and from Lemma 7:

Lemma 8. The total number of phases is at most $O(m \log B)$.

There are at most $2(m+n)$ flow augmentations in each of the $O(m \log B)$ phases, and each flow augmentation and network relabeling is dominated by a shortest path computation, which can be accomplished in $O(m+n\log n)$ time by using Fredman and Tarjan's [4] implementation of Dijkstra's algorithm [3]. In Step 0 cancelling all flow-generating cycles can be done in $O(mn^2 \log n \log B)$ time as described in [8], and the maximum flow computation in Step 1 can be done in $O(nm \log(n^2/m))$ time [9]. Consequently, we have:

Theorem 1. The complexity of our algorithm is bounded by $O(m^2(m + n\log n)\log B)$.

If in the main algorithm, the maximum flow computation in Step 1 is done on every iteration before calling Procedure Phase(Δ), and the algorithm is terminated when the relabeled excess $E_{g,\mu}$ is zero, the worst-case complexity of the algorithm is unaffected. If either this algorithm or the original algorithm is applied to problems that are not restricted, there may be nodes that have excesses from which there is no path in the residual graph to s. In this case, the set $V_g = \{v \in V \mid \text{there is a directed path from } v \text{ to } s \text{ in } G_g\}$ and the subgraph of $G_{g,\mu}$ restricted to V_g should be used in place of V and $G_{g,\mu}$, respectively, throughout these algorithms. At termination, the optimal pseudoflow that is obtained can be converted into an optimal flow by pushing any excesses that remain backwards around the flow-generating cycles that created them.

5 An Alternative Algorithm

We now describe an alternative and conceptually simpler way to use arc excesses to limit the number of flow augmentations in each phase to $O(m)$. In contrast with the approach presented in the last section, in this alternative approach, flow augmentations always terminate at node s and once node excess is transferred into arc excess it is never transferred back until the end of a Δ-phase.

To prevent arc excesses from becoming too large, in this alternative algorithm, all arcs (v, w) with $e_{g,\mu}(v,w) \geq u_{g,\mu}(v,w)$ are deleted from $G_{g,\mu}$ before computing highest-gain paths. Such arcs, will be called *self-saturating arcs*, since they can be saturated using only their own arc excesses. When the network is relabeled, the relabeled gains of self-saturating arcs can be greater than 1 since these arcs are ignored when computing the new labels. Consequently at the end of each phase our algorithm saturates all self-saturating arcs that have relabeled gains greater than 1. This is done by calling Procedure Saturate-Arcs given below just before transferring arc excesses to node excesses at the end of a Δ-phase.

```
**Procedure Saturate-Arcs**
**begin**
  **for** each arc $(v,w) \in G_{g,\mu}$ with $\gamma_\mu(v,w) > 1$ **do**
  **begin**
    set $e_{g,\mu}(v,w) \leftarrow e_{g,\mu}(v,w) - u_{g,\mu}(v,w)$, $e_{g,\mu}(w,v) \leftarrow e_{g,\mu}(w,v) + \gamma_\mu(v,w)u_{g,\mu}(v,w)$,
    $g_\mu(v,w) \leftarrow g_\mu(v,w) + u_{g,\mu}(v,w)$, $g_\mu(w,v) \leftarrow -\gamma_\mu(v,w)g_\mu(v,w)$;
  **end**;
**end**.
```

Since all self-saturating arcs with relabeled gains greater than 1 are saturated at the end of each phase, $\gamma_\mu(v,w) \leq 1$ for each arc in the relabeled residual network at the end of each phase. Thus, the relabeled residual network at the end of each phase does not contain flow-generating cycles.

The following version of Procedure Push-Flow pushes pseudoflow ¿From a node whose excess is greater than Δ along a highest-gain path to node s. In contrast with the version of Procedure

Push-Flow in Section 4, the augmentation always terminates in node s. We use $\bar{G}_{g,\mu}$ to denote the subgraph of the relabeled residual graph $G_{g,\mu}$ that is obtained by deleting all self-saturating arcs from $G_{g,\mu}$.

```
Procedure Push-Flow($v$)
begin
  while $v \neq s$ do
  begin
    let $(v,w)$ be the first arc on the highest-gain path from $v$ to $s$ in $\bar{G}_{g,\mu}$;
    compute $\alpha = \min\{e_{g,\mu}(v), \Delta\}$ and $\delta = \min\{u_{g,\mu}(v,w), \alpha\}$;
    set $e_{g,\mu}(v) \leftarrow e_{g,\mu}(v) - \alpha$, $g_\mu(v,w) \leftarrow g_\mu(v,w) + \delta$, $g_\mu(w,v) \leftarrow -g_\mu(v,w)$;
    set $e_{g,\mu}(v,w) \leftarrow e_{g,\mu}(v,w) + \alpha - \delta$, $e_{g,\mu}(w) \leftarrow e_{g,\mu}(w) + \delta$;
    set $v \leftarrow w$;
  end;
end;
```

The pseudoflow augmentation terminates when the condition $v = s$ is detected. Note that if $\delta = \alpha$ then arc excess $e_{g,\mu}(v,w)$ remains unchanged. Also, when flow is pushed from v to w arc excess $e_{g,\mu}(w,v)$ does not change. The pseudoflow augmentation starts from a node v with $e_{g,\mu}(v) \geq \Delta$ and pushes flow all of the way to node s whether or not $e_{g,\mu}(w) \geq \Delta$ for those nodes w on the path from v to s. Thus, each flow augmentation always sends a positive amount of flow into s and reduces the total node excess excluding $e_{g,\mu}(s)$ by at least Δ. These Δ units of flow are either sent to s or distributed as arc excesses along the path from v to s. The following lemma gives the bounds on the arc excess and the total excess created after saturating arcs at the end of each phase.

Lemma 9. In each Δ-phase, both immediately before and after calling Procedure Push-Flow in Procedure Phase(Δ),

(i) $0 \leq e_{g,\mu}(v,w) < \Delta$, for all $(v,w) \in E$;

(ii) $e_{g,\mu}(v,w) - u_{g,\mu}(v,w) + \gamma_\mu(v,w)u_{g,\mu}(v,w) < \Delta$ for all self-saturating arcs (v,w).

Proof. At the start of a Δ-phase, all arc excesses are zero and there are no self-saturating arcs; hence the statements of the lemma hold at this point. Now suppose that the statements of the lemma hold prior to flow being pushed from v to w along arc (v, w). Since $\bar{e}_{g,\mu}(v, w) = e_{g,\mu}(v, w) + \alpha - \delta$ and $\delta \leq \alpha$, the arc excess $\bar{e}_{g,\mu}(v, w)$ is nonnegative. If $\delta = \alpha$, then $\bar{e}_{g,\mu}(v, w) = e_{g,\mu}(v, w) < \Delta$. If $\delta < \alpha$, then $\bar{e}_{g,\mu}(v, w) = e_{g,\mu}(v, w) + \alpha - u_{g,\mu}(v, w) \leq e_{g,\mu}(v, w) + \Delta - u_{g,\mu}(v, w) < \Delta$, since $e_{g,\mu}(v, w) < u_{g,\mu}(v, w)$ (flow is only pushed through nonsaturating arcs). If as a result of this push arc (v, w) becomes self-saturating (i.e., $\bar{e}_{\bar{g},\mu}(v, w) \geq u_{\bar{g},\mu}(v, w)$), then since $\gamma_\mu(v, w) = 1$ and $\bar{e}_{\bar{g},\mu}(v, w) < \Delta$, it follows that $\bar{e}_{\bar{g},\mu}(v, w) - u_{\bar{g},\mu}(v, w) + \gamma_\mu(v, w) u_{\bar{g},\mu}(v, w) < \Delta$. Pushing flow in arc (v, w) increases $u_{g,\mu}(w, v)$. Hence, if (w, v) was self-saturating it may become non-self-saturating. If it remains self-saturating (ii) will still hold after the push because $\gamma_\mu(w, v) = 1$. Hence, statement (ii) in the lemma holds for all self-saturating arcs immediately after the push. Now $e_{g,\mu}(v, w) = e_g(v, w)/\mu(v) \geq 0$ and $\gamma_\mu(v, w) u_{g,\mu}(v, w) = \gamma(v, w)(u(v, w) - g(v, w))/\mu(w) \geq 0$, and, if (v, w) is self-saturating, $e_{g,\mu}(v, w) - u_{g,\mu}(v, w) = (e_g(v, w) - u_g(v, w))/\mu(v) \geq 0$. Therefore, since by Lemma 3 the node labels never decrease, (ii) continues to hold after relabelling. □

¿From Lemma 9, we have that the total relabeled excess excluding $e_{g,\mu}(s)$ is at most $(n + m)\Delta$ at the end of each Δ-phase; so Lemma 5 holds for this variant of our main algorithm. Since each flow augmentation reduces the relabeled *node* excess (excluding $e_{g,\mu}(s)$) by at least Δ, the total number of flow augmentations in each Δ-phase is at most $2(n + m)$; i.e., Lemma 6 applies. Consequently, Lemmas 7 and 8 and Theorem 1 also apply to this variant. We note that the bound on the total amount of arc excess created by our first algorithm in each phase is half of what it is for the alternative algorithm. Specifically, in Lemma 4 it is proved that $e_{g,\mu}(v, w) + e_{g,\mu}(w, v) < \Delta$, while in Lemma 9 $e_{g,\mu}(v, w) < \Delta$ is proved. In fact m in the statements of Lemmas 5 and 6 can be replaced by $m/2$ when these lemmas refer to the algorithm given in Section 4 (i.e., our first algorithm).

6 Conclusions

We have developed new combinatorial algorithms for solving the generalized circulation problem that are based on excess scaling and highest-gain path augmentations. These algorithms are simple and have a better complexity than any previously proposed combinatorial algorithm for this problem. We believe that our algorithms will also be fast in practice. It will be interesting to see if our algorithms can be extended to solve the generalized minimum cost network flow problem. Whether there exists a strongly polynomial time algorithm for the generalized circulation problem remains an open question.

References

[1] Ahuja, R., Magnanti, T., and Orlin, J. *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, 1993.

[2] Cohen, E. and Megiddo, N. "New algorithms for generalized network flows", *Math. Programming*, 64(1994), 325-336.

[3] Dijkstra, E. "A note on two problems in connexion with graphs", *Numeriche Mathematics*, 1(1959), 269-271.

[4] Fredman, M. L., and Tarjan, R. E. "Fibonacci heaps and their uses in improved network optimization algorithms", *Journal of ACM*, 34(1987), 596-615.

[5] Glover, F., Klingman, D. and Phillips, N. "Netform modeling and applications", *Interfaces*, 20(1990), 7-27.

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

[7] Glover, F., and Klingman, D. "On the equivalence of some generalized network problems to pure network problems", *Math. Programming*, 4(1973), 269-278.

[8] Goldberg, A. V., Plotkin, S. A., and Tardos, E. "Combinatorial algorithms for the generalized circulation problem", *Math. Oper. Res.*, 16(1991), 351-379.

[9] Goldberg, A.V. and Tarjan, R.E. " A New Approach to the Maximum Flow Problem," *Journal of the Association of Computing Machinery*, 35(1988), 921-940

[10] Goldfarb, D. and Jin, Z. "A faster combinatorial algorithm for the generalized circulation problem", To appear in *Math. Oper. Res.*

[11] Goldfarb, D. and Jin, Z. "A polynomial dual simplex algorithm for the generalized circulation problem", Tech. Report, Dept. of Ind. Engr. and Oper. Res., Columbia University, 1995.

[12] Karmarkar, N. "A new polynomial-time algorithm for linear programming", *Combinatorica*, 4(1984), 373-395.

[13] Khachian, L. G. "Polynomial algorithms in linear programming", *Zhurnal Vychislitelnoi Matematiki i Matematicheskoi Fiziki*, 20(1980), 53-72.

[14] Kapoor, S. and Vaidya, P. M. "Fast algorithms for convex quadratic programming and multi-commodity flows", *Proc. 18th Annual ACM Symposium on Theory of Computing*, 147-159.

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

[16] Onaga, K. "Dynamic programming of optimum flows in lossy communication nets", *IEEE Trans. Circuit Theory*, 13(1966), 308-327.

[17] Orlin, J. B. "A faster strongly polynomial minimum cost flow algorithm", *Oper. Res.*, 41(1993), 338-350.

[18] Radzik, T. "Faster algorithms for the generalized network flow problem", *Proc. 34th IEEE Annual Symposium on Foundations of Computer Science*, 1993, 438-448.

[19] Truemper, K. "On max flows with gains and pure min-cost flows", *SIAM J. Appl. Math.*, 32(1977), 450-456.

[20] Vaidya, P.M. "Speeding up linear programming using fast matrix multiplication", *Proc. 30th IEEE Annual Symposium on Foundations of Computer Science*, 1989.