Simple Generalized Maximum Flow Algorithms

Éva Tardos[⋆] and Kevin D. Wayne[⋆⋆]

Cornell University

Abstract. We introduce a *gain-scaling* technique for the generalized maximum flow problem. Using this technique, we present three simple and intuitive polynomial-time combinatorial algorithms for the problem. Truemper's augmenting path algorithm is one of the simplest combinatorial algorithms for the problem, but runs in exponential-time. Our first algorithm is a polynomial-time variant of Truemper's algorithm. Our second algorithm is an adaption of Goldberg and Tarjan's preflow-push algorithm. It is the first polynomial-time preflow-push algorithm in generalized networks. Our third algorithm is a variant of the Fat-Path capacity-scaling algorithm. It is much simpler than Radzik's variant and matches the best known complexity for the problem. We discuss practical improvements in implementation.

1 Introduction

In this paper we present new algorithms for the *generalized maximum flow problem*, also known as the *generalized circulation problem*. In the traditional maximum flow problem, the objective is to send as much flow through a network from one distinguished node called the source to another called the sink, subject to capacity and flow conservation constraints. In generalized networks, a fixed percentage of the flow is lost when it is sent along an arc. Specifically, each arc (v, w) has an associated *gain factor* $\gamma(v, w)$. When $g(v, w)$ units of flow enter arc (v, w) at node v then $\gamma(v, w)g(v, w)$ arrive at w. The gains factors can represent physical transformations due to evaporation, energy dissipation, breeding, theft, or interest rates. They can also represent transformations from one commodity to another as a result of manufacturing, blending, or currency exchange. They may also represent arc failure probabilities. Many applications are described in [1, 3, 5].

Since the generalized maximum flow problem is a special case of linear programming, it can be solved using simplex, ellipsoid, or interior-point methods.

[⋆] Computer Science Department and Operations Research and Industrial Engineering Department, Cornell University, Ithaca, NY 14853, USA. Research supported in part by an NSF PYI award DMI-9157199, by NSF through grant DMS 9505155, and by ONR through grant N00014-96-1-0050. Email: `eva@cs.cornell.edu`.

[⋆⋆] Operations Research and Industrial Engineering Department, Cornell University, Ithaca, NY 14853, USA. Research supported in part by ONR through grant AASERT N00014-97-1-0681. Email: `wayne@orie.cornell.edu`.

Many general purpose linear programming algorithms can be tailored for the problem. The network simplex method can handle generalized flows. Kapoor and Vaidya [16] showed how to speed up interior-point methods on network flow problems by exploiting the structured sparsity in the underlying constraint matrix. Murray [18] and Kamath and Palmon [15] designed different interior-point algorithms for the problem. We note that these simplex and interior-point methods can also solve the generalized minimum cost flow problem.

The first *combinatorial* algorithms for the generalized maximum flow problem were the augmenting path algorithms of Jewell [14] and Onaga [19] and exponential-time variants. Truemper [22] observed that the problem is closely related to the minimum cost flow problem, and that many of the early generalized maximum flow algorithms were, in fact, analogs of pseudo-polynomial minimum cost flow algorithms. Goldberg, Plotkin and Tardos [7] designed the first two combinatorial polynomial-time algorithms for the problem: Fat-Path and MCF. The Fat-Path algorithm uses capacity-scaling and a subroutine that cancels flow-generating cycles. The MCF algorithm performs minimum cost flow computations. Radzik [20] modified the Fat-Path algorithm, by canceling only flow-generating cycle with sufficiently large gains. Goldfarb and Jin [12] modified the MCF algorithm by replacing the minimum cost flow subroutine with a simpler computation. Goldfarb and Jin [11] also presented a dual simplex variant of this algorithm. Recently, Goldfarb, Jin and Orlin [13] designed a new capacity-scaling algorithm, motivated by the Fat-Path algorithm. Tseng and Bertsekas [23] proposed an ϵ-relaxation method for solving the more general *generalized minimum cost flow problem with separable convex costs.* However, their running time may be exponential in the input size.

Researchers have also developed algorithms for the *approximate generalized maximum flow problem.* Here, the objective is to find a ξ-optimal flow, i.e., a flow that generates excess at the sink that is within a $(1 - \xi)$ factor of the optimum, where ξ is an input parameter. Cohen and Megiddo [2] showed that the approximate generalized maximum flow problem can be solved in strongly polynomial-time. Their algorithm uses a subroutine which tests feasibility of a linear system with two variables per inequality. Radzik [20] observed that the Fat-Path algorithm can be used to compute approximates flow faster than optimal flows. His Fat-Path variant, that cancels only flow-generating cycles with large gain, is the fastest algorithm for computing approximate flows. Subsequently, Radzik [21] gave a new strongly polynomial-time analysis for canceling all flow-generating cycles, implying that the original Fat-Path algorithm computes an approximate flow in strongly polynomial-time. For the linear programming algorithms, it is not known how to improve the worst-case complexity of the exact algorithms to find approximate flows.

We present a new rounding technique for generalized flows, which can be viewed as a type of *gain-scaling.* Using this technique, we propose three simple combinatorial algorithms for the generalized maximum flow problem. Our first algorithm is a polynomial-time variant of Truemper's [22] algorithm. Truemper's algorithm is a very simple maximum flow based augmenting path algorithm, analogous to Jewell's primal-dual algorithm for the minimum cost flow problem. Truemper's algorithm may require exponential-time, but by applying our new gain-scaling technique, we develop a polynomial-time variant. Our second algorithm is an adaption Goldberg and Tarjan's [10] preflow-push algorithm for the minimum cost flow problem. Using gain-scaling, we establish the first polynomial-time preflow-push algorithm for generalized flows. Our third algorithm is a simple variant of the Fat-Path capacity-scaling algorithm. By using gain-scaling, our Fat-Path variant improves the complexity of canceling flow-generating cycles, and hence of the overall algorithm. In contrast, Radzik's Fat-Path variant modifies this subroutine, canceling only flow-generating cycles with sufficiently large gain. Both Fat-Path variants have the same complexity, but Radzik's variant and proof of correctness are quite complicated.

2 Preliminaries

2.1 Generalized Networks

Since some of our algorithms are iterative and recursive, it is convenient to solve a seemingly more general version of the problem which allows for multiple sources. An instance of the generalized maximum flow problem is a *generalized network* $G = (V, E, t, u, \gamma, e)$, where V is an n-set of nodes, E is an m-set of directed arcs, $t \in V$ is a distinguished node called the *sink*, $u\colon E \to \Re_{\geq 0}$ is a *capacity function*, $\gamma\colon E \to \Re_{>0}$ is a *gain function*, and $e\colon V \to \Re_{\geq 0}$ is an *initial excess function*. A *residual arc* is an arc with positive capacity. A *lossy network* is a generalized network in which no residual arc has gain factor exceeding one. We consider only simple directed paths and cycles. The *gain of a path P* is denoted by $\gamma(P) = \prod_{e \in P} \gamma(e)$. The *gain of a cycle* is defined similarly. A *flow-generating cycle* is a cycle whose gain is more than one.

For notational convenience we assume that G has no parallel arcs. Our algorithms easily extend to allow for parallel arcs and the running times we present remain valid. Without loss of generality, we assume the network is *symmetric* and the gain function is *antisymmetric*. That is, for each arc $(v, w) \in E$ there is an arc $(w, v) \in E$ (possibly with zero capacity) and $\gamma(w, v) = 1/\gamma(v, w)$. We assume the capacities and initial excesses are given as integers between 1 and B, and the gains are given as ratios of integers which are between 1 and B. To simplify the running times we assume $B \geq m$, and use $\tilde{\mathcal{O}}(f)$ to denote $f \log^{\mathcal{O}(1)} m$.

2.2 Generalized Flows

A *generalized pseudoflow* is a function $g\colon E \to \Re$ that satisfies the *capacity constraints* $g(v, w) \leq u(v, w)$ for all $(v, w) \in E$ and the *antisymmetry constraints* $g(v, w) = -\gamma(w, v) g(w, v)$ for all $(v, w) \in E$. The *residual excess* of g at node v is $e_g(v) = e(v) - \sum_{(v,w) \in E} g(v, w)$, i.e., the initial excess minus the the net flow out of v. If $e_g(v)$ is positive (negative) we say that g has *residual excess (deficit)* at node v. A *flow* g is a pseudoflow that has no residual deficits; it may have residual excesses. A *proper flow* is a flow which does not generate any additional residual excess, except possibly at the sink. We note that a flow can be converted into a proper flow, by removing flow on useless paths and cycles. For a flow g we denote its *value* $|g| = e_g(t)$ to be the residual excess at the sink. Let $\mathrm{OPT}(G)$ denote the maximum possible value of any flow in network G. A flow g is *optimal* in network G if $|g| = \mathrm{OPT}(G)$ and ξ*-optimal* if $|g| \geq (1-\xi)\,\mathrm{OPT}(G)$. The *(approximate) generalized maximum flow problem* is to find a (ξ-) optimal flow. We sometimes omit the adjective *generalized* when its meaning is clear from context.

2.3 Residual and Relabeled Networks

Let g be a generalized flow in network $G = (V, E, s, u, \gamma, e)$. With respect to the flow g, the *residual capacity function* is defined by $u_g(v,w) = u(v,w) - g(v,w)$. The *residual network* is $G_g = (V, E, s, u_g, \gamma, e_g)$. Solving the problem in the residual network is equivalent to solving it in the original network.

Our algorithms use the technique of *relabeling*, which was originally introduced by Glover and Klingman [4]. A *labeling function* is a function $\mu\colon V \to \Re_{>0}$ such that $\mu(t) = 1$. The *relabeled network* is $G_\mu = (V, E, t, u_\mu, \gamma_\mu, e_\mu)$, where the *relabeled capacity*, *relabeled gain* and *relabeled initial excess* functions are defined by: $u_\mu(v,w) = u(v,w)/\mu(v)$, $\gamma_\mu(v,w) = \gamma(v,w)\mu(v)/\mu(w)$, and $e_\mu(v) = e(v)/\mu(v)$. The relabeled network provides an equivalent instance of the generalized maximum flow problem. Intuitively, node label $\mu(v)$ changes the local units in which flow is measured at node v; it is the number of old units per new unit. The inverses of the node labels correspond to the linear programming dual variables, for the primal problem with decision variables $g(v,w)$. With respect to a flow g and labels μ we define the the *relabeled residual network* by $G_{g,\mu} = (V, E, t, u_{g,\mu}, \gamma_\mu, e_{g,\mu})$, where the *relabeled residual capacity* and *relabeled residual excess* functions are defined by $u_{g,\mu}(v,w) = (u(v,w) - g(v,w))/\mu(v)$ and $e_{g,\mu}(v) = e_g(v)/\mu(v)$. We define the *canonical label* of a node v in network G to be the inverse of the highest gain residual path from v to the sink. If G has no residual flow-generating cycles, then we can compute the canonical labels using a single shortest path computation with costs $c(v,w) = -\log \gamma(v,w)$.

2.4 Optimality Conditions

An *augmenting path* is a residual path from a node with residual excess to the sink. A *generalized augmenting path* (GAP) is a residual flow-generating cycle, together with a (possibly trivial) residual path from a node on this cycle to the sink. By sending flow along augmenting paths or GAPs we increase the net flow into the sink. The following theorem of Onaga [19] says that the nonexistence of augmenting paths and GAPs implies that the flow is optimal.

Theorem 1. *A flow g is optimal in network G if and only if there are no augmenting paths or GAPs in G_g.*

2.5 Finding a Good Starting Flow

Our approximation algorithms require a rough estimate of the optimum value in a network. Radzik [20] proposed a $\tilde{\mathcal{O}}(m^2)$ time greedy augmentation algorithm that finds a flow that is within a factor n of the optimum. His greedy algorithm repeatedly sends flow along highest-gain augmenting paths, but does *not* use arcs in "backward" direction. Using this algorithm, we can determine an initial parameter Δ_0 which satisfies $\mathrm{OPT}(G) \leq \Delta_0 \leq n\,\mathrm{OPT}(G)$.

2.6 Canceling Flow-Generating Cycles

Subroutine CancelCycles converts a generalized flow g into another generalized flow g' whose residual network contains no flow-generating cycles. In the process, the net flow into every node, including the sink, can only be increased. It also finds node labels μ so that $G_{g',\mu}$ is a lossy network. This subroutine is used by all of our algorithms. CancelCycles was designed by Goldberg, Plotkin, and Tardos and is described in detail in [7]. It is an adaptation of Goldberg and Tarjan's [9] cancel-and-tighten algorithm for the minimum cost flow problem using costs $c(v,w) = -\log_b \gamma(v,w)$ for any base $b > 1$. Note that negative cost cycles correspond to flow-generating cycles. In Section 7 we discuss practical implementation issues.

Theorem 2. *Let $b > 1$. If all of the costs are integral and at least $-C$, then* CancelCycles *runs in $\tilde{\mathcal{O}}(mn \log C)$ time.*

In generalized flows, the costs will typically not be integral. In this case, the next theorem is useful.

Theorem 3. *If the gains are given as ratios of integers between 1 and B, then* CancelCycles *requires $\tilde{\mathcal{O}}(mn^2 \log B)$ time.*

Radzik [21] showed that CancelCycles runs in strongly polynomial-time. We have a variant that limits the relabeling increases, allowing a simpler proof of the strongly polynomial running time.

2.7 Nearly Optimal Flows

The following lemma derived from [7] says that if a flow is ξ-optimal for sufficiently small ξ, then we can efficiently convert it into an optimal flow. It is used to provide termination of our exact algorithms. The conversion procedure involves one call to CancelCycles and a single (nongeneralized) maximum flow computation.

Lemma 1. *Given a B^{-4m}-optimal flow, we can compute an optimal flow in $\tilde{\mathcal{O}}(mn^2 \log B)$ time.*

3 Gain-Scaling

In this section we present a rounding and scaling framework. Together, these ideas provide a technique which can be viewed as a type of *gain-scaling.* By rounding the gains, we can improve the complexity of many generalized flow computations (e.g., canceling flow-generating cycles above). However, our approximation from rounding creates error. Using an iterative or recursive approach, we can gradually refine our approximation, until we obtain the desired level of precision.

3.1 Rounding Down the Gains

In our algorithms we round down the gains so that they are all integer powers of a base $b = (1 + \xi)^{1/n}$. Our rounding scheme applies in lossy networks, i.e., networks in which no residual arc has a gain factor above one. This implies that the network has no residual flow-generating cycles. We round the gain of each residual arc down to $\bar{\gamma}(v, w) = b^{-\bar{c}(v,w)}$ where $\bar{c}(v, w) = -\lfloor \log_b \gamma(v, w) \rfloor$, maintaining antisymmetry by setting $\bar{\gamma}(w, v) = 1/\bar{\gamma}(v, w)$. Note that if both (v, w) and (w, v) are residual arcs, then each has unit gain, ensuring that $\bar{\gamma}$ is well-defined. Let H denote the resulting *rounded network.* H is also a lossy network, sharing the same capacity function with G. Let h be a flow in network H. The *interpretation of flow* h as a flow in network G is defined by: $g(v, w) = h(v, w)$ if $g(v, w) \geq 0$ and $g(v, w) = -\gamma(w, v)h(w, v)$ if $g(v, w) < 0$. Flow interpretation in lossy networks may create additional excesses, but no deficits. We show that approximate flows in the rounded network induce approximate flows in the original network. First we show that the rounded network is close to the original network.

Theorem 4. *Let G be a lossy network and let H be the rounded network constructed as above. If $0 < \xi < 1$ then $(1 - \xi)\,\mathrm{OPT}(G) \leq \mathrm{OPT}(H) \leq \mathrm{OPT}(G)$.*

Proof. Clearly $\mathrm{OPT}(H) \leq \mathrm{OPT}(G)$ since we only decrease the gain factors of residual arcs. To prove the other inequality, we consider the path formulation of the maximum flow problem in lossy networks. We include a variable x_j for each path P_j, representing the amount of flow sent along the path. Let x^* be an optimal path flow in G. Then x^* is also a feasible path flow in H. From path P_j, $\gamma(P_j)x_j^*$ units of flow arrive at the sink in network G, while only $\bar{\gamma}(P_j)x_j^*$ arrive in network H. The theorem then follows, since for each path P_j,

$$\bar{\gamma}(P_j) \geq \frac{\gamma(P_j)}{b^{|P|}} \geq \frac{\gamma(P_j)}{1+\xi} \geq \gamma(P_j)(1 - \xi).$$

Corollary 1. *Let G be a lossy network and let H be the rounded network constructed as above. If $0 < \xi < 1$ then the interpretation of a ξ'-optimal flow in H is a $\xi + \xi'$-optimal flow in G.*

Proof. Let h be a ξ'-optimal flow in H. Let g be the interpretation of flow h in G. Then we have

$$|g| \geq |h| \geq (1-\xi')\,\mathrm{OPT}(H) \geq (1-\xi)(1-\xi')\,\mathrm{OPT}(G) \geq (1-\xi-\xi')\,\mathrm{OPT}(G).$$

3.2 Error-Scaling and Recursion

In this section we describe an *error-scaling* technique which can be used to speed up computations for generalized flow problems. Radzik [20] proposed a recursive version of error-scaling to improve the complexity of his Fat-Path variant when finding nearly optimal and optimal flows. We use the technique in a similar manner to speed up our Fat-Path variant. We also use the idea to convert constant-factor approximation algorithms into fully polynomial-time approximation schemes.

Suppose we have a subroutine which finds a 1/2-optimal flow. Using error-scaling, we can determine a ξ-optimal flow in network G by calling this subroutine $\log_2(1/\xi)$ times. To accomplish this we first find a 1/2-optimal flow g in network G. Then we find a 1/2-optimal flow h in the residual network G_g. Now $g+h$ is a 1/4-optimal flow in network G, since each call to the subroutine captures at least half of the remaining flow. In general, we can find a ξ-optimal flow with $\log_2(1/\xi)$ calls to the subroutine.

The following lemma of Radzik [20] is a recursive version of error-scaling. It says that we can compute an ξ-optimal flow by combining two appropriate $\sqrt{\xi}$-optimal flows.

Lemma 2. *Let g be a $\sqrt{\xi}$-optimal flow in network G. Let h be a $\sqrt{\xi}$-optimal flow in network G_g. Then the flow $g+h$ is ξ-optimal in G.*

4 Truemper's Algorithm

Truemper's maximum flow based augmenting path algorithm is one of the simplest algorithms for the generalized maximum flow problem. We apply our gain-scaling techniques to Truemper's algorithm, producing perhaps the cleanest and simplest polynomial-time algorithms for the problem. In this section we first review Truemper's [22] algorithm. Our first variant runs Truemper's algorithm in a rounded network. It computes a ξ-optimal flow in polynomial-time, for any constant $\xi > 0$. However, it requires exponential-time to compute optimal flows, since we would need ξ to be very small. By incorporating error-scaling, we show that a simple variant of Truemper's algorithm computes an optimal flow in polynomial-time.

A natural and intuitive algorithm for the maximum flow problem in lossy networks is to repeatedly send flow from excess nodes to the sink along highest-gain (most-efficient) augmenting paths. Onaga observed that if the input network has no residual flow-generating cycles, then the algorithm maintains this property. Thus, we can find a highest-gain augmenting path using a single shortest path computation with costs $c(v,w) = -\log\gamma(v,w)$. By maintaining canonical labels, we can ensure that all relabeled gains are at most one, and a Dijkstra shortest path computation suffices. Unit gain paths in the canonically relabeled network correspond to highest gain paths in the original network. This is essentially Onaga's [19] algorithm. If the algorithm terminates, then the resulting flow is optimal by Theorem 1. However, this algorithm may not terminate in finite time if the capacities are irrational. Truemper's algorithm [22] uses a (nongeneralized) maximum flow computation to simultaneously augment flow along all highest-gain augmenting paths. It is the generalized flow analog of Jewell's primal-dual minimum cost flow algorithm.

Theorem 5. *In Truemper's algorithm, the number of maximum flow computations is bounded by n plus the number of different gains of paths in the original network.*

Proof. After each maximum flow computation, $\mu(v)$ strictly increases for each excess node $v \neq t$.

4.1 Rounded Truemper (RT)

Algorithm RT computes a ξ-optimal flow by running Truemper's algorithm in a rounded network. The input to Algorithm RT is a lossy network G and an error parameter ξ. Algorithm RT first rounds the gains to integer powers of $b = (1+\xi)^{1/n}$, as described in Section 3.1. Let H denote the rounded network. Then RT computes an optimal flow in H using Truemper's algorithm. Finally the algorithm interprets the flow in the original network. Algorithm RT is described in Figure 1.

```
Input: lossy network $G$, error parameter $0 < \xi < 1$
Output: $\xi$-optimal flow $g$
  Set base $b = (1+\xi)^{1/n}$ and round gains in $G$ to powers of $b$
  Let $H$ be resulting network
  Initialize $h \leftarrow 0$
  while $\exists$ augmenting path in $H_h$ do
    $\mu \leftarrow$ canonical labels in $H_h$
    $f \leftarrow$ max flow from excess nodes to the sink in $H_{h,\mu}$ using only gain one relabeled
        residual arcs
    $h(v,w) \leftarrow h(v,w) + f(v,w)\mu(v)$
  end while
  $g \leftarrow$ interpretation of flow $h$ in $G$
```

Figure 1: RT(G, ξ)

The gain of a path in network G is between B^{-n} and B^n. Thus, after rounding to powers of b, there are at most $1 + \log_b B^{2n} = \mathcal{O}(n^2\xi^{-1}\log B)$ different gains of paths in H. Using Goldberg and Tarjan's [8] preflow-push algorithm, each (nongeneralized) maximum flow computation takes $\tilde{\mathcal{O}}(mn)$ time. Thus, by

Theorem 5, RT finds an optimal flow in H in $\tilde{\mathcal{O}}(mn^3\xi^{-1}\log B)$ time. The following theorem follows using Corollary 1.

Theorem 6. *Algorithm RT computes a ξ-optimal flow in a lossy network in $\tilde{\mathcal{O}}(mn^3\xi^{-1}\log B)$ time.*

4.2 Iterative Rounded Truemper (IRT)

RT does not compute an optimal flow in polynomial-time, since the precision required to apply Lemma 1 is roughly $\xi = B^{-m}$. In Algorithm IRT, we apply error-scaling, as described in Section 3.2. IRT iteratively calls RT with error parameter 1/2 and the current residual network. Since RT sends flow along highest-gain paths in the rounded network, not in the original network, it creates residual flow-generating cycles. So, before calling RT in the next iteration, we must first cancel all residual flow-generating cycles with subroutine CANCEL-CYCLES, because the input to RT is a lossy network. Intuitively, this can be interpreted as rerouting flow from its current paths to highest-gain paths, but not all of the rerouted flow reaches the sink.

Theorem 7. *Algorithm IRT computes a ξ-optimal flow in $\tilde{\mathcal{O}}(mn^3\log B\log\xi^{-1})$ time. It computes an optimal flow in $\tilde{\mathcal{O}}(m^2n^3\log^2 B)$ time.*

In the full paper we prove that Algorithm IRT actually finds an optimal flow in $\tilde{\mathcal{O}}(m^2n^3\log B + m^2n^2\log^2 B)$ time.

5 Preflow-Push

In this section we adapt Goldberg and Tarjan's [10] preflow-push algorithm to the generalized maximum flow problem. This is the first polynomial-time preflow push algorithm for generalized network flows. Tseng and Bertsekas [23] designed a preflow push-like algorithm for the generalized minimum cost flow problem, but it may require more than B^n iterations. Using our rounding technique, we present a preflow-push algorithm that computes a ξ-optimal flow in polynomial-time for any constant $\xi > 0$. Then by incorporating error-scaling, we show how to find an optimal flow in polynomial-time.

5.1 Rounded Preflow-Push (RPP)

Algorithm RPP is a generalized flow analog of Goldberg and Tarjan's preflow push algorithm for the minimum cost flow problem. Conceptually, RPP runs the minimum cost flow algorithm with costs $c(v,w) = -\log\gamma(v,w)$ and error parameter $\epsilon = \frac{1}{n}\log b$ where $b = (1+\xi)^{1/n}$. This leads to the following natural definitions and algorithm. An *admissible arc* is a residual arc with relabeled gain above one. The *admissible graph* is the graph induced by admissible arcs. An *active node* is a node with positive residual excess and a residual path to the sink. We note that if no such residual path exists and an optimal solution sends flow through this node, then that flow does not reach the sink. So we can safely disregard this useless residual excess. (Periodically RPP determines which nodes have residual paths to the sink.) Algorithm RPP maintains a flow h and node labels μ. The algorithm repeatedly selects an active node v. If there is an admissible arc (v, w) emanating from node v, IPP pushes $\delta = \min\{e_h(v), u_h(v, w)\}$ units of flow from node v to w. If $\delta = u_h(v, w)$ the push is called *saturating*; otherwise it is *nonsaturating*. If there is no such admissible arc, RPP increases the label of node v by a factor of $2^\epsilon = b^{1/n}$; this corresponding to an additive potential increase for minimum cost flows. This process is referred to as a *relabel* operation. Relabeling node v can create new admissible arcs emanating from v. To ensure that we do not create residual flow-generating cycles, we only increase the label by a relatively small amount.

The input to Algorithm RPP is a lossy network G and error parameter ξ. Before applying the preflow-push method, IPP rounds the gains to powers of $b = (1 + \xi)^{1/n}$, as described in Section 3.1. The method above is then applied to the rounded network H. Algorithm RPP is described in Figure 2.

We note that our algorithm maintains a pseudoflow with excesses, but no deficits. In contrast, the Goldberg-Tarjan algorithm allows both excesses and deficits. Also their algorithm scales ϵ. We currently do not see how to improve the worst-case complexity by a direct scaling of ϵ.

```
Input: lossy network $G$, error parameter $0 < \xi < 1$
Output: $\xi$-optimal flow $g$
  Set base $b = (1 + \xi)^{1/n}$ and round gains in network $G$ to powers of $b$.
  Let $H$ be resulting network
  Initialize $h \leftarrow 0, \mu \leftarrow 1$
  while $\exists$ active node $v$ do
    if $\exists$ admissible arc $(v, w)$ then
      Push $\delta = \min\{e_h(v), u_h(v, w)\}$ units of flow from $v$ to $w$ and update $h$   {push}
    else
      $\mu(v) \leftarrow b^{1/n}\mu(v)$                                                        {relabel}
    end if
  end while
  $g \leftarrow$ interpretation of flow $h$ in $G$
```

Figure 2: RPP(G, ξ)

The bottleneck computation is performing nonsaturating pushes, just as for computing minimum cost flows with the preflow-push method. By carefully choosing the order to examine active nodes (e.g., the wave implementation), we can reduce the number of nonsaturating pushes. A dual approach is to use more clever data structures to reduce the amortized time per nonsaturating push. Using a version of dynamic trees specialized for generalized networks [6], we obtain the following theorem.

Theorem 8. *Algorithm RPP computes a ξ-optimal flow in $\tilde{\mathcal{O}}(mn^3\xi^{-1}\log B)$ time.*

5.2 Iterative Rounded Preflow-Push (IRPP)

RPP does not compute an optimal flow in polynomial time, since the precision required is roughly $\xi = B^{-m}$. Like Algorithm IRT, Algorithm IRPP adds error-scaling, resulting in the following theorem.

Theorem 9. *IRPP computes a ξ-optimal flow in $\tilde{\mathcal{O}}(mn^3\log B\log\xi^{-1})$ time. It computes an optimal flow in $\tilde{\mathcal{O}}(m^2n^3\log^2 B)$ time.*

6 Rounded Fat Path

In this section we present a simple variant of Goldberg, Plotkin, and Tardos' [7] Fat-Path algorithm which has the same complexity as Radzik's [20] Fat-Path variant. Our algorithm is intuitive and its proof of correctness is much simpler than Radzik's. The Fat-Path algorithm can be viewed as an analog of Orlin's capacity scaling algorithm for the minimum cost flow problem. The original Fat-Path algorithm computes a ξ-optimal flow in $\tilde{\mathcal{O}}(mn^2\log B\log\xi^{-1})$ time, while Radzik's and our variants require only $\tilde{\mathcal{O}}(m(m+n\log\log B)\log\xi^{-1})$ time.

The bottleneck computation in the original Fat-Path algorithm is canceling residual flow-generating cycles. Radzik's variant reduces the bottleneck by canceling only residual flow-generating cycles with big gains. The remaining flow-generating cycles are removed by decreasing the gain factors. Analyzing the precision of the resulting solution is technically complicated. Instead, our variant rounds down the gains to integer powers of a base b, which depends on the precision of the solution desired. Our rounding is done in a lossy network, which makes the quality of the resulting solution easy to analyze. Subsequent calls to CancelCycles are performed in a rounded network, improving the complexity.

We first review the FatAugmentations subroutine which finds augmenting paths with sufficiently large capacity. Then we present Algorithm RFP, which runs the Fat-Path algorithm in a rounded network. It computes approximately optimal and optimal flows in polynomial-time. We then present a recursive version of RFP, which improves the complexity when computing nearly optimal and optimal flows.

6.1 Fat Augmentations

The FatAugmentations subroutine was originally developed by Goldberg, Plotkin, and Tardos for their Fat-Path algorithm and is described in detail in [7]. The input is a lossy network and fatness parameter δ. The subroutine repeatedly augments flow along *highest-gain δ-fat paths*, i.e. highest-gain augmenting paths among paths that have enough residual capacity to increase the excess at the sink by δ, given sufficient excess at the first node of the path. This process is repeated until no δ-fat paths remain. There are at most $n + \mathrm{OPT}(G)/\delta$ augmentations. By maintaining appropriate labels μ, an augmentation takes $\tilde{\mathcal{O}}(m)$ time, using an algorithm based on Dijkstra's shortest path algorithm. Upon termination, the final flow has value at least $\mathrm{OPT}(G) - m\delta$.

6.2 Rounded Fat Path (RFP)

Algorithm RFP runs the original Fat-Path algorithm in a rounded network. The idea of the original Fat-Path algorithm is to call FatAugmentations and augment flow along δ-fat paths, until no such paths remain. At this point δ is decreased by a factor of 2 and a new phase begins. However, since FatAugmentations selects only paths with large capacity, it does not necessarily send flow on overall highest-gain paths. This creates residual flow-generating cycles which must be canceled so that we can efficiently compute $\delta/2$-fat paths in the next phase.

The input to Algorithm RFP is a lossy network and an error parameter ξ. First, RFP rounds down the gains as described in Section 3.1. It maintains a flow h in the rounded network H and an upper bound $\varDelta$ on the *excess discrepancy*, i.e., the difference between the value of the current flow $|h|$ and the optimum $\mathrm{OPT}(H)$. The scaling parameter $\varDelta$ is initialized using Radzik's greedy augmentation algorithm, as described in Section 2.5. In each phase, $\varDelta$ is decreased by a factor of 2. To achieve this reduction, Algorithm RFP cancels all residual flow-generating cycles in H_h, using the CancelCycles subroutine. By Theorem 2 this requires $\tilde{\mathcal{O}}(mn \log C)$ time where C is the biggest cost. Recall $c(v, w) = -\lfloor \log_b \gamma(v, w) \rfloor$ so $C \leq 1 + \log_b B = \mathcal{O}(n\xi^{-1} \log B)$. Then subroutine FatAugmentations is called with fatness parameter $\delta = \varDelta/(2m)$. After this call, the excess discrepancy is at most $m\delta = \varDelta/2$, and $\varDelta$ is decreased accordingly. Since each δ-fat augmentation either empties the residual excess of a node or increases the flow value by at least δ, there are at most $n + \varDelta/\delta = n + 2m$ augmentations per $\varDelta$-phase, which requires a total of $\tilde{\mathcal{O}}(m^2)$ time. Algorithm RFP is given in Figure 3.

Theorem 10. *Algorithm RFP computes a 2ξ-optimal flow in a lossy network in $\tilde{\mathcal{O}}((m^2 + mn \log(\xi^{-1} \log B)) \log \xi^{-1})$ time.*

Proof. To bound the running time, we note that there are at most $\log_2(n/\xi)$ phases. FatAugmentations requires $\tilde{\mathcal{O}}(m^2)$ time per phase, and CancelCycles requires $\tilde{\mathcal{O}}(mn \log C)$ time, where we bound $C = \mathcal{O}(n\xi^{-1} \log B)$ as above. The algorithm terminates when $\varDelta \leq \xi\, \mathrm{OPT}(H)$. At this point h is ξ-optimal in network H, since we maintain $\varDelta \geq \mathrm{OPT}(H) - |h|$. The quality of the resulting solution then follows using Theorem 4.

6.3 Recursive Rounded Fat-Path (RRFP)

Algorithm RFP computes a ξ-optimal flow in $\tilde{\mathcal{O}}(m^2 + mn \log \log B)$ time when $\xi > 0$ is inversely polynomial in m. However it may require more time to

```
Input: lossy network $G$, error parameter $0 < \xi < 1$
Output: $2\xi$-optimal flow $g$
  Set base $b = (1+\xi)^{1/n}$ and round gains in network $G$ to powers of $b$
  Let $H$ be resulting network
  Initialize $\Delta \leftarrow \Delta_0$ and $h \leftarrow 0$          $\{\mathrm{OPT}(H) \leq \Delta_0 \leq n\,\mathrm{OPT}(H)\}$
  repeat
    $(h', \mu) \leftarrow$ CancelCycles$(H_h)$
    $h \leftarrow h + h'$
    $h' \leftarrow$ FatAugmentations$(H_h, \mu, \Delta/(2m))$          $\{\mathrm{OPT}(H_h) - |h'| \leq \Delta/2\}$
    $h \leftarrow h + h'$
    $\Delta \leftarrow \Delta/2$
  until $\Delta \leq \xi\,\mathrm{OPT}(H)$
  $g \leftarrow$ interpretation of $h$ in network $G$
```
Figure 3: $\mathrm{RFP}(G, \xi)$

compute optimal flows than the original Fat-Path algorithm. By using the recursive scheme from Section 3.2, we can compute nearly optimal and optimal flows faster than the original Fat-Path algorithm. In each recursive call, we reround the network. We cancel flow-generating cycles in an already (partially) rounded network. The benefit is roughly to decrease the average value of C from $\mathcal{O}(n\xi^{-1}\log B)$ to $\mathcal{O}(n \log B)$.

Theorem 11. *Algorithm RRFP computes a ξ-optimal flow in a lossy network in $\tilde{\mathcal{O}}(m(m + n\log\log B)\log\xi^{-1})$ time. If the network has residual flow-generating cycles, then an extra $\tilde{\mathcal{O}}(mn^2 \log B)$ preprocessing time is required. Algorithm RRFP computes an optimal flow in $\tilde{\mathcal{O}}(m^2(m + n\log\log B)\log B)$ time.*

7 Practical Cycle-Canceling

We implemented a version of the preflow-push algorithm, described in Section 5, in C++ using Mehlhorn and Näher's [17] Library of Efficient Data types and Algorithms (LEDA). We observed that as much as 90% of the time was spent canceling flow-generating cycles. We focused our attention on reducing this bottleneck.

Recall, CancelCycles is an adaption of Goldberg and Tarjan's cancel-and-tighten algorithm using costs $c(v, w) = -\log\gamma(v, w)$. Negative cost cycles correspond to flow-generating cycles. The underlying idea of the Goldberg-Tarjan algorithm is to cancel the most negative mean cost cycle until no negative cost cycles remain. To improve efficiency, the actual cancel-and-tighten algorithm only approximates this strategy. It maintains a flow g and node potentials (corresponding to node labels) π that satisfy *ϵ-complementary slackness.* That is, $c_\pi(v, w) = c(v, w) - \pi(v) + \pi(w) \geq -\epsilon$ for all residual arcs (v, w). The subroutine makes progress by reducing the value of ϵ, using the following two computations which comprise a phase: (i) canceling residual cycles in the subgraph induced by negative reduced cost arcs and (ii) updating node potentials so that finding new negative cost (flow-generating) cycles is efficient. The cancel-and-tighten algorithm uses the following two types of potential updates. Loose updating uses a computationally inexpensive topological sort, but may only decrease ϵ by a factor of $(1-1/n)$ per phase. Tight updating reduces ϵ by the maximum possible amount, but involves computing the value of the minimum mean cost cycle ϵ^*, which is relatively expensive.

We observed that the quality of the potential updates was the most important factor in the overall performance of CancelCycles. So, we focused our attention on limiting the number of iterations by better node potential updates. Using only loose updates, we observed that CancelCycles required a large number of phases. By using tight updates, we observed a significant reduction in the number of phases, but each phase is quite expensive. The goal is to a reach a middle ground. We introduce a *medium updating* technique which is much cheaper than tight updating, yet more effective than loose updating; it reduces the overall running time of the cycle canceling computation. Our implementation uses a combination of loose, medium, and tight potential updates.

Tight updating requires $\tilde{\mathcal{O}}(mn)$ time in the worst case, using either a dynamic programming or binary search method. We incorporated several heuristics to improve the actual performance. These heuristics are described in the full paper. However, we observed that tight relabeling was still quite expensive.

We introduce a *medium potential updating* which is a middle ground between loose and tight updating. In medium updating, we find a value ϵ' which is close to ϵ^*, without spending the time to find the actual minimum mean cost cycle. In our algorithm, we only need to estimate ϵ^* in networks where the subgraph induced by negative cost arcs is acyclic. To do this efficiently, we imagine that the in addition to the original arcs, a zero cost link exists between every pair of nodes. We can efficiently find a minimum mean cost cycle in this modified network by computing a minimum mean cost path in the acyclic network induced by only negative cost arcs, without explicitly considering the imaginary zero cost arcs. Let ϵ' denote the value of the minimum mean cost cycle in the modified network. Clearly $\epsilon' \leq \epsilon^*$, and it is not hard to see that $\epsilon' \geq (1-1/n)\epsilon$. We can binary search for ϵ' using a shortest path computation in *acyclic* graphs. This requires only $\mathcal{O}(m)$ time per iteration. If we were to determine ϵ' exactly, in $\tilde{\mathcal{O}}(n \log B)$ iterations the search interval would be sufficiently small. If the gains in the network are rounded to powers of $b = (1+\xi)^{1/n}$ then $\tilde{\mathcal{O}}(\log C)$ iterations suffice, where $C = \mathcal{O}(n\xi^{-1} \log B)$. In our implementation we use an approximation to ϵ'.

References

1. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, New Jersey, 1993.
2. Edith Cohen and Nimrod Megiddo. New algorithms for generalized network flows. *Math Programming*, 64:325–336, 1994.
3. F. Glover, J. Hultz, D. Klingman, and J. Stutz. Generalized networks: A fundamental computer based planning tool. *Management Science*, 24:1209–1220, 1978.

4. F. Glover and D. Klingman. On the equivalence of some generalized network flow problems to pure network problems. *Math Programming*, 4:269–278, 1973.
5. F. Glover, D. Klingman, and N. Phillips. Netform modeling and applications. *Interfaces*, 20:7–27, 1990.
6. A. V. Goldberg, S. A. Plotkin, and É Tardos. Combinatorial algorithms for the generalized circulation problem. Technical Report STAN-CS-88-1209, Stanford University, 1988.
7. A. V. Goldberg, S. A. Plotkin, and É Tardos. Combinatorial algorithms for the generalized circulation problem. *Mathematics of Operations Research*, 16:351–379, 1991.
8. A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 35:921–940, 1988.
9. A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *Journal of the ACM*, 36:388–397, 1989.
10. A. V. Goldberg and R. E. Tarjan. Solving minimum cost flow problems by successive approximation. *Mathematics of Operations Research*, 15:430–466, 1990.
11. D. Goldfarb and Z. Jin. A polynomial dual simplex algorithm for the generalized circulation problem. Technical report, Department of Industrial Engineering and Operations Research, Columbia University, 1995.
12. D. Goldfarb and Z. Jin. A faster combinatorial algorithm for the generalized circulation problem. *Mathematics of Operations Research*, 21:529–539, 1996.
13. D. Goldfarb, Z. Jin, and J. B. Orlin. Polynomial-time highest gain augmenting path algorithms for the generalized circulation problem. *Mathematics of Operations Research*, to appear.
14. W. S. Jewell. Optimal flow through networks with gains. *Operations Research*, 10:476–499, 1962.
15. Anil Kamath and Omri Palmon. Improved interior point algorithms for exact and approximate solution of multicommodity flow problems. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 502–511, 1995.
16. S. Kapoor and P. M. Vaidya. Speeding up Karmarkar's algorithm for multicommodity flows. *Math Programming*, to appear.
17. K. Mehlhorn and S. Näher. A platform for combinatorial and geometric computing. *CACM*, 38–1:96–102, `http://ftp.mpi-sb.mpg.de/LEDA/leda.html`, 1995.
18. S. M. Murray. *An interior point approach to the generalized flow problem with costs and related problems*. PhD thesis, Stanford University, 1993.
19. K. Onaga. Dynamic programming of optimum flows in lossy communication nets. *IEEE Trans. Circuit Theory*, 13:308–327, 1966.
20. T. Radzik. Faster algorithms for the generalized network flow problem. *Mathematics of Operations Research*, to appear.
21. T. Radzik. Approximate generalized circulation. Technical Report 93-2, Cornell Computational Optimization Project, Cornell University, 1993.
22. K. Truemper. On max flows with gains and pure min-cost flows. *SIAM J. Appl. Math*, 32:450–456, 1977.
23. P. Tseng and D. P. Bertsekas. An ϵ-relaxation method for separable convex cost generalized network flow problems. In *5th International Integer Programming and Combinatorial Optimization Conference*, 1996.