Combinatorial Approximation Algorithms for Generalized Flow Problems

Jeffrey D. Oldham*

Abstract

Generalized network flow problems generalize normal network flow problems by specifying a flow multiplier $\mu(a)$ for each arc a. For every unit of flow entering the arc, $\mu(a)$ units of flow exit. Flow multipliers permit modelling transforming one type into another and modification of the amount of flow. For example, currency exchange and water evaporation from canals can be modelled.

We present a strongly polynomial algorithm for a single-source generalized shortest paths problem, also called the restricted generalized uncapacitated transshipment problem. We present a left-distributive closed semiring which permits use of the Bellman-Ford algorithm to solve this problem given a guess for the value of the optimal solution. Using Megiddo's parametric search scheme, we can compute the optimal value in strongly polynomial time. The algorithm's running time $O(mn^2 \log n)$ matches the previously best known, but the algorithm is simpler, is based on the well-known theory of closed semirings, and directly works with the given graph. All previous polynomial-time algorithms were based on interior-point methods or directly solved the dual problem and translated the solution back to the primal problem.

Using this generalized shortest paths algorithm, we present fully polynomial-time approximation schemes for the generalized versions of the maximum flow, the nonnegative-cost minimum-cost flow, the concurrent flow, the multicommodity maximum-flow, and the multicommodity nonnegative-cost minimum-cost flow problems. For all of these problems except the maximum flow variant, these combinatorial algorithm schemes are the first polynomial-time algorithms not based on interior point methods. All running times are independent of the size of the flow multipliers' representation. Also, the generalized ¡concurrent flow and the generalized multicommodity maximum flow approximation schemes are the first known strongly polynomial algorithms.

1 Introduction

Ordinary network flow models require flow conservation on all arcs: The amount of flow entering an arc equals the amount of flow leaving the arc. Generalized network flow models modify this conservation by associating a flow multiplier $\mu(v, w)$ with each arc (v, w). For each unit of flow sent from vertex v along the arc, $\mu(v, w)$ units of flow arrive at w. Using flow multipliers permits two types of modelling not possible with canonical models. Flow multipliers can represent transformations from one type of object to another. For example, Hong Kong dollars can be converted into South African rands, and trees can be converted into reams of paper. Multipliers can also modify the amount of flow. Thus, one can model evaporation from a network of water canals and breakage caused during transport through a delivery network.

The generalized flow model has been studied [Dan63, Jew62] since the publication of Ford and Fulkerson's network flows book [FF62] defined flows as an area of research, but the only previously known combinatorial polynomial-time algorithms solved the generalized versions of shortest paths and maximum flows. In this paper, we present a Bellman-Ford approach [Bel58, For56] for solving the single-source generalized shortest path problem (GSP), also called the *restricted generalized uncapacitated transshipment problem*. Previous approaches [AC91, CM94, HN94] solved the dual problem as a linear program with two variables per inequality and then converted the solution back to the original problem. Our approach has exactly the same running time but is simpler, directly uses the given graph, avoids the dual-to-primal conversion, and requires less space.

Using the GSP algorithm as a subroutine in the Cohen-Megiddo, Garg-Könemann, Grigoriadis--Khachiyan frameworks [CM94, GK98, GK96], we obtain fully polynomial-time approximation schemes for all variants of generalized network flow problems with nonnegative costs. Excepting the generalized maximum-flow problem [GPT91, Rad93, TW98], these are the first combinatorial polynomial-time algorithms known to the authors. Furthermore, the generalized concurrent flow and the generalized multicommodity maximum-flow flow algorithms ex-

*Department of Computer Science, Stanford University, Stanford, CA 94305-9045, oldham@cs.stanford.edu, http://theory.stanford.edu/~oldham, +1 650.723.1787 (voice), +1 650.725.4671 (fax). Research partially supported by ARO Grant DAAG55-98-1-0170 and NSF Grant CCR9307045. This work was completed while working at AT&T Labs.

single-source generalized shortest paths (GSP)		
[CR66, WF99] ($\mu(a) \leq 1$ only)	$O(m + n \log n)$	Dijkstra
Hochbaum and Naor [HN94]	$O(mn^2 \log n)$	Fourier-Motzkin + [AC91]
this paper	$O(mn^2 \log n)$	Bellman-Ford + [Meg79]
generalized maximum flow		
Tardos and Wayne [TW98]	$O(\log \epsilon^{-1} m(m + n \log\log B))$	
	$O(m^2(m + n \log\log B) \log B))$	capacity scaling
Goldfarb, Jin, and Orlin [GZO97]	$O(m^2(m + n \log n) \log B)$	augmented paths
Vaidya [Vai89]	$O(m^{1.5} n^2 \log B)$	interior point
Radzik [Rad93]	$O(\log \epsilon^{-1} m^2 n \log^2 m)$	capacity scaling
this paper	$O(\epsilon^{-2} m^2 n^2 \log^2 m)$	GSP + [GK98]
generalized minimum-cost flow		
Vaidya [Vai89]	$O(m^{1.5} n^2 \log(nB))$	interior point
Tseng and Bertsekas [TB96]	exponential in input size	Lagrangian relaxation
this paper (nonnegative costs)	$O((\epsilon^{-2} \log \epsilon^{-1} + \log m) m^2 n^2 \log m \log(mCU))$	GSP + [GK96]
generalized concurrent flow		
Vaidya [Vai89]	$O(k^{2.5} m^{1.5} n^2 \log(nB))$	interior point
Kamath and Palmon [KP95a]	$O(\log \epsilon^{-1}(k^{0.5} m^3 + k m^{1.5} n^{1.5}) n \log B)$	
	$O(\log \epsilon^{-1} k^{2.5} m^{1.5} n^2 \log B)$	interior point
this paper	$O(\log \epsilon^{-1} k m^2 n^2 \log m)$	[CM94] + k commodities
	$O(k m^3 n^2 \log m \log(nB))$	[CM94] + k commodities
generalized multicommodity maximum flow		
interior point algorithms	same as for generalized concurrent flow	
this paper	$O(\epsilon^{-2} k m^2 n^2 \log^2 m)$	GSP + [GK98]
generalized multicommodity minimum-cost flow		
interior point algorithms	same as for generalized concurrent flow	
this paper (nonnegative costs)	$O((\epsilon^{-2} \log \epsilon^{-1} + \log m) k m^2 n^2 \log m \log(mCU))$	GSP + [GK96]

Table 1: The best running times for exact and approximate generalized flow algorithms. To the right of each running times is a short description of the algorithm. B is the largest integer used to represent the input assuming the arc capacities and vertex supplies/demands are integral and assuming flow multipliers are ratios of integers. C and U represent the absolute value of the largest arc cost and capacity, respectively.

tend the class of problems for which strongly polynomial algorithms (assuming a fixed approximation factor) are known.

In the single-source generalized shortest paths problem, one is given a directed graph with arc costs, arc multipliers, and a source vertex s. The objective is to find a minimum-cost augmented path consuming one unit of flow starting at s. An augmented path is a path connected to a lossy cycle, i.e., a cycle with flow multiplier less than one. If one unit of flow enters a lossy cycle, traversing the cycle will yield less than one unit of flow.We show that the vertex potentials of the dual problem form a left-distributive, but not right-distributive, closed semiring. Fully-distributive closed semirings are algebraic structures underlying path algorithms, e.g., the Floyd-Warshall algorithm, in directed graphs. However, the Bellman-Ford algorithm requires only left-distributivity. Given an initial value for the source's vertex potential, one Bellman-Ford computation will indicate whether the potential was smaller than, equal to, or greater than the problem's optimal value. Thus, we can perform binary search to solve the problem. Using this comparison routine and binary search, we can solve the GSP in $O(mn(\log C + m\bar{\mu}))$ time, where C and $\bar{\mu}$ are the largest arc cost and arc multiplier, respectively.

To obtain a $O(mn^2 \log n)$ strongly polynomial running time, we use Megiddo's parametric search technique [Meg79] to compute the optimal value. Instead of representing the vertex potentials as numbers, we use lines that are a function of the source vertex's potential. Throughout the algorithm, each iteration narrows the possible range for the optimal value and consists of one Bellman-Ford iteration. An iteration may invoke the comparison subroutine, which narrows the range for the optimal value. At the algorithm's termination, the instance's optimal value is the smallest one in the possible range of optimal values.

Three recent frameworks [CM94, GK98, GK96] permit us to use the GSP algorithm to produce fully polynomial approximation schemes for all other generalized flow problems with nonnegative costs: generalized maximum flow, generalized nonnegative-cost minimum-cost flow, generalized concurrent flow, generalized multicommodity maximum flow, and generalized multicommodity nonnegative-cost minimum-cost flow. (Table 1 contains the schemes' running times.) The frameworks permit computing approximate solutions by repeatedly solving GSP problems with different arc costs. These arc costs reflect the current violation of the problem's capacity and cost constraints. The Cohen-Megiddo framework [CM94] repeatedly computes a GSP in the residual graph with respect to arc costs that reflect the flow's use of arc capacities. The Garg-Könemann framework [GK98] uses a greedy approach. Each iteration yields an augmented path routing as much flow as the path's capacity constraints permit. The arc costs are exponentially related to the ratio of their flow to capacity. The Grigoriadis-Khachiyan framework [GK96] repeatedly reroutes flow until it is ϵ-optimal. A potential function, which is closely related to the sum of the capacity and cost constraints' dual variables, yields the arc costs for a GSP iteration. The current flow is replaced by its convex combination with the GSP flow scaled to satisfy the source vertex's supply. Similar frameworks [KP95b, PST95] have proven practical [GOPS98] giving a good indication these frameworks will yield practical algorithms.

Running times for the approximation schemes and the best extant generalized flow algorithms are listed in Table 1. Both exact and approximation algorithms are presented. All the interior point and several combinatorial algorithms depend on the size of the input numbers. B is the largest integer in the problem's input assuming the arc capacities and vertex supplies/demands are represented as integers and the flow multipliers are ratios of integers. C and U represent the largest arc cost and capacity, respectively. Excepting generalized maximum flow, this paper's algorithms have the smallest running time for certain parameter values.

The running times of *strongly polynomial algorithms* depend only on the size of the instance's underlying structure, not the size of the input data. Tardos [Tar86] presented a strongly polynomial algorithm for a large class of combinatorial linear programs with bounded constraint matrix entries. Generalized flow problems can have arbitrary size flow multipliers so Tardos's algorithm does not solve these problems, but they are among the next natural set of problems to consider. In 1994, Cohen and Megiddo [CM94] presented a strongly polynomial approximation scheme for the generalized maximum flow problem to which we add the generalized concurrent flow and generalized multicommodity maximum flow problems.

In the next section, we formally define the single-source generalized shortest paths problem and prove its solution consists of an augmented path. In §3, the problem's dual variables are shown to be linear functions of the source vertex's dual value. We define generalized reduced costs and prove a set of linear functions form a left-distributive closed semiring. Subsequently, we present a Bellman-Ford comparison subroutine indicating whether the source vertex's initial potential is smaller than, equal to, or larger than the problem instance's optimal value. Using the Bellman-Ford subroutine, a binary-search algorithm can solve the GSP problem. In §5, this algorithm is modified to use Megiddo's parametric search technique to yield a strongly polynomial-time algorithm. Using this algorithm, we derive approximation algorithms for all the other generalized flow problems.

2 Single-Source Generalized Shortest Paths Problem

The *single-source generalized shortest paths problem* (GSP) is to find a minimum-cost flow function obeying flow conservation, obeying the arc multipliers, and starting at a source vertex. The input consists of a directed graph $G = (V, A)$, an arc multiplier function $\mu : A \to \mathbf{R} > 0$, an arc cost function $c : A \to \mathbf{R}$, and a source vertex $s \in V$. The resulting flow function $f : A \to \mathbf{R} \geq 0$ must obey flow conservation at the vertices, the multiplier function, remove one unit of flow from s, and minimize the flow's cost.

$$\text{Min} \sum_{a \in A} c(a) f(a)$$

$$\text{s.t. for all vertices } v \in V,$$

$$\sum_{(v,w)} f_{(v,w)} - \sum_{(w,v)} \mu_{(w,v)} f_{(w,v)} = [v = s] \quad (2.1)$$

$$(\forall \text{ arcs } a \in A)(f(a) \geq 0).$$

The constraints' equalities ensures flow is conserved at vertices, but the flow multipliers μ change the flow along arcs. (We use Iverson notation [*predicate*] which is 1 if *predicate* is true and 0 otherwise [Knu92].) Without loss of generality, the cost of a flow on an arc is the product of the arc's cost and the flow entering the arc.

The GSP has no sink vertices (vertices with demands) and exactly one source vertex s with unit supply. Conceptually, sink vertices can be eliminated by adding a zero-cost lossy self-looping arc so restricting the generalized problem does not limit our ability to model problems. Thus, any flow reaching the sink will be consumed by the arc. If multiple source vertices are present, the problem can be solved for each source and then the solutions can be combined. We can assume, without loss of generality, a unit supply at the source because any solution can be scaled by a positive scalar and still remain a solution. Also, without loss of generality, we assume all vertices are reachable from s.

Flow multipliers and costs can be defined for any walk. The *flow multiplier* $\mu(W)$ *of a walk* W is the product of its arcs' flow multipliers. The definition ensures flow conservation at the vertices. A *lossy cycle* C has flow multiplier $\mu(C)$ less than one. *Breakeven* and *gainy* cycles have multipliers equal to and greater than one, respectively. The *cost of a walk* is the cost of sending a unit of flow along the walk starting at its initial vertex. For example, the cost of the path $v \to w \to x$ is $1 \cdot c(v \to w) + \mu(v \to w)c(w \to x)$. The multiplier and cost of an empty walk is 1 and 0, respectively.

An *augmented path* $s \rightsquigarrow v \rightsquigarrow w \to v$ is a nonempty path $s \rightsquigarrow v \rightsquigarrow w$ with an *extra arc* $w \to v$ forming a lossy cycle $v \rightsquigarrow w \to v$. An augmented path is a solution to the GSP because its path transports the source's unit supply to a lossy cycle which "consumes" the flow reaching it.

LEMMA 2.1. *For any path P and any initial flow supply θ at the path's initial vertex, the flow on each of the path's arcs is determined, and the supply at the path's terminus is $\mu(P)\theta$.*

COROLLARY 2.1. *The flow on each arc of an augmented path $s \rightsquigarrow v \rightsquigarrow w \to v$ is a scalar multiple of the supply θ at the path's initial vertex.*

COROLLARY 2.2. *The cost of an augmented path is a scalar multiple of the supply θ.*

Augmented paths are the only solutions to the GSP.

LEMMA 2.2. *All solutions of the single-source generalized shortest paths problem are augmented paths.*

3 The Dual Variables and Left-Distributive Closed Semirings

In this section, we present the GSP's dual linear program, which has two variables per inequality. After defining reduced costs using the vertices' dual variables, we will show these variables' values are linear functions of a path's origin's value. Using this fact, we can determine whether the source vertex's value is less than, equal to, or greater than an augmented path's flow cost. We conclude with a left-distributive closed semiring for these dual variables. The semiring is isomorphic to lines with positive slope in the Cartesian plane. The Bellman-Ford algorithm of §4 uses the closed semiring, while the parametric algorithm of §5 uses its linear form.

The GSP dual linear program is

$$\begin{aligned}&\max \pi(s)\\&\quad \text{s.t. } (\forall (v,w) \in A)\ (\pi(v) - \mu(v,w)\pi(w) \le c(v,w))\end{aligned}$$

where $\pi(v)$ represents v's dual variable and can attain any real value. The objective function is so simple because the flow is conserved at all vertices except the source vertex s. By the duality theorem of linear programming, $\pi(s)$ equals the cost of the minimum-cost augmented path. Thus, determining the maximum value of $\pi(s)$ yields the cost of the minimum-cost augmented path.

Just as for ordinary shortest paths, we define the *reduced cost* $c^{\pi}(v,w)$ of an arc (v,w) as $c^{\pi}(v,w) = c(v,w) + \mu(v,w)\pi(w) - \pi(v)$. The dual program's constraints can be written as requiring nonnegative reduced costs for all arcs. Complementary slackness implies flow on an arc is positive only if its reduced cost is zero.

The vertex potentials along any path of arcs with zero reduced cost are linear functions of the initial vertex's potential. The same is also true for augmented paths having only arcs with zero reduced cost except possibly the extra arc.

LEMMA 3.1. *Given any path $v \rightsquigarrow w$ having only arcs with zero reduced cost and vertex potential $\pi(v) = \varpi$, the vertex potential $\pi(x)$ for vertex x*

on the path is

$$\frac{1}{\mu(v \rightsquigarrow x)} \left(\varpi - c(v \rightsquigarrow x)\right).$$

The lemma shows vertex potentials π are linear functions of a path's initial vertex's potential ϖ. Thus, we can view them as lines in (ϖ, π) Cartesian plane. Furthermore, the omitted proof indicates how to calculate one vertex's potential from its predecessor's potential and the arc's cost and multiplier. To form a left-distributive closed semiring, we only additional tool we need is an operator, e.g., maximum, to compare vertex potentials.

Closed semirings are an algebraic structure for solving path problems in directed graphs. For example, the Floyd-Warshall algorithm [Flo62, War62], transitive closure algorithms, and the Bellman-Ford algorithm [Bel58, For56] are all based on these structures. (Algorithms based on semirings usually use algebraic techniques. For example, the Bellman-Ford algorithm is based on the Jacobi iteration method [Jac45].) See [CLR90, §26.4] or [AHU74, §5.6] for details. A semiring consists of a domain, a summary operator $\oplus$, an extension operator $\odot$, and identities for these operators. The definition of closed semirings requires both left and right distributivity:

$$\begin{aligned} \text{left}: &\quad (b \oplus c) \odot a = (b \odot a) \oplus (c \odot a) \\ \text{right}: &\quad a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c), \end{aligned}$$

but the closed semiring for generalized flow will be only left-distributive, which is sufficient for the Bellman-Ford algorithm.

The domain of the generalized flow left-distributive closed semiring $(S, \oplus, \odot, \bar{0}, \bar{1})$ has ordered pairs of costs and flow multipliers for paths. To translate an ordered pair (c, μ) into a vertex potential, we will use $(\varpi - c)/\mu$, where ϖ is the vertex potential of the source vertex s. The domain $S = (\mathbf{R}, \mathbf{R} > 0) \cup \{(-\infty, 0)\}$ contains all possible path costs and multipliers augmented by a bottom element $(-\infty, 0)$. The summary operator $\oplus$ yields the larger vertex potential when comparing two paths: $(c_1, \mu_1) \oplus (c_2, \mu_2)$ equals the operand yielding the larger of $(\varpi - c_1)/\mu_1$ and $(\varpi - c_2)/\mu_2$. If a multiplier is zero, the division is defined to yield $-\infty$. This operation is analogous to comparing two lines in the (ϖ, π) Cartesian plane at a particular value of ϖ. The extension operator $\odot$ computes the cost and flow multiplier of the concatenation of two paths: $(c_1, \mu_1) \odot (c_2, \mu_2)$ equals $(c_1 + \mu_1 c_2, \mu_1 \mu_2)$. This operation is analogous to *reversed functional composition*, i.e., the functional composition of the second operand's line with the first operand's line. The summary identity $\bar{0} = (-\infty, 0)$ is the "least" domain element according to the summary operator. Extending a line with the extension identity $\bar{1} = (0, 1)$ returns the same line under reversed functional composition.

LEMMA 3.2. *For every value of ϖ, there is an isomorphism between the system $(S, \oplus, \odot, \bar{0}, \bar{1})$ and a system containing the line $\pi = -\infty$ and lines with positive slope in the (ϖ, π) Cartesian plane. (c, μ) is isomorphic to the line $\pi = (\varpi - c)/\mu$ if $\mu > 0$. $(-\infty, 0)$ is isomorphic to $\pi = -\infty$. The (ϖ, π) plane's operators are maximum value with respect to fixed ϖ and reversed functional composition. Its identities are the $\pi = -\infty$ line and the identity line.*

LEMMA 3.3. *The system $(S, \oplus, \odot, \bar{0}, \bar{1})$ is a left-distributive closed semiring. That is, it is a closed semiring with only left distributivity, not right distributivity.*

4 The Bellman-Ford Comparison Subroutine

In this section, we present a Bellman-Ford comparison subroutine for the GSP using the generalized flow left-distributive closed semiring presented in the previous section. Given an initial vertex potential $\pi(s)$ for the source vertex s, the algorithm indicates whether the guess is smaller than, equal to, or larger than the cost of the minimum-cost augmented path, and it also yields the path. Thus, we can perform binary search on $\pi(s)$ to obtain a GSP algorithm with running time polynomial in the logarithm of the largest flow multiplier. In §5, we present a strongly polynomial algorithm.

Suppose we are given an augmented path with unit supply at its initial vertex and we guess its cost ϖ (which is uniquely determined by Corollary 2.2). If all arcs except possibly the extra arc have zero reduced cost, then the extra arc's reduced cost indicates where the guess was too small, correct, or too large.

LEMMA 4.1. *Consider an augmented path $s \rightsquigarrow v \rightsquigarrow w \rightarrow v$ having only arcs with zero reduced cost except possibly the extra arc $w \rightarrow v$. Let $\pi(s)$ denote the path's cost. If vertex s has potential ϖ,*

Input: a guess ϖ for the cost $\pi^*(s)$ of the minimum-cost augmented path with initial vertex s
Output: the guess ϖ was smaller than, equal to, or larger than the minimum cost $\pi^*(s)$
// Initialization
  for all vertices v **do**
    $\pi^0(v) \leftarrow \begin{cases} \bar{1} & \text{if } s = v \\ \bar{0} & \text{otherwise} \end{cases}$
// Recurrences
  for iterations $i = 1 \ldots n-1$ **do**
    for all vertices v **do**
      $\pi^i(v) \leftarrow \pi^{i-1}(v) \oplus \bigoplus_{\text{arcs } w \to v} \left(\pi^{i-1}(w) \odot (c(w \to v), \mu(w \to v))\right)$
// Check for correctness of guess ϖ.
  if there exists an arc with negative reduced cost **then**
    return "$\varpi > \pi^*(s)$"
  else if there does not exist an augmented path in the subgraph of zero reduced cost arcs **then**
    return "$\varpi < \pi^*(s)$"
  else
    return "$\varpi = \pi^*(s)$"

Subroutine 1: Bellman-Ford Comparison Subroutine for the GSP

then,

$$\begin{aligned} c^\pi(w \to v) < 0 &\iff \varpi > \pi(s) \\ c^\pi(w \to v) = 0 &\iff \varpi = \pi(s) \\ c^\pi(w \to v) > 0 &\iff \varpi < \pi(s). \end{aligned}$$

The Bellman-Ford comparison subroutine (Subroutine 1) indicates whether the source vertex's potential is smaller than, equal to, or larger than the cost of the minimum-cost augmented path. The algorithm works by repeatedly increasing vertex potentials. At the end of the ith iteration, the algorithm has a tree of arcs with zero reduced cost maximizing the vertex potentials subject to the tree's depth being at most i. Since the longest path in the directed graph has $n-1$ arcs, the algorithm terminates with a tree containing all the vertices connected by arcs with zero reduced cost.

The algorithm requires only left-distributivity, not right-distributivity. This is because the right operand of the recurrence equation's extension operator $\odot$ is a single arc, not the summary of more than one path.

LEMMA 4.2. *The Bellman-Ford comparison subroutine (Subroutine 1) yields a correct answer.*

Proof. The Bellman-Ford subroutine guarantees all arcs of the minimum-cost augmented path except possibly the extra arc have zero reduced cost. By the previous lemma, this extra arc's reduced cost will be negative if and only if ϖ is too large. If the guess is too small for the minimum-cost augmented path, it is too small for all augmented paths. Using the previous lemma, the subgraph of arcs with zero reduced cost will be cycle-free. Otherwise, all arcs in the minimum-cost augmented path have zero reduced cost, and ϖ is the problem's optimal value. The minimum-cost augmented path is any augmented path in the subgraph of arcs with zero reduced cost and can be found using depth-first search. □

LEMMA 4.3. *The Bellman-Ford comparison subroutine (Subroutine 1) requires $O(mn)$ time and $O(m+n)$ space.*

An algorithm can perform binary search using the comparison subroutine to find the optimal answer of the GSP. The maximum possible value is $O(C\bar{\mu}^n)$, where C and $\bar{\mu}$ represent the largest arc cost and the largest multiplier, respectively. Thus, the worst-case running time is $O(mnp(\log C + n\bar{\mu}))$, where an accuracy of 2^{-p} is desired. In the next section, we will combine the Bellman-Ford comparison subroutine with ideas from Megiddo's parametric search to obtain a strongly polynomial algorithm for the GSP.

5 Parametric Search for the Optimal Value

We present a strongly polynomial algorithm for the single-source generalized shortest paths prob-

// Initialization
  $[b,e] \leftarrow$ largest possible range for the instance's minimum cost $\pi^*(s)$
  for all vertices v **do**
    $\pi^0(v) \leftarrow \begin{cases} \bar{1} & \text{if } s = v \\ \bar{0} & \text{otherwise} \end{cases}$
// Recurrences
  for iterations $i = 1 \ldots n-1$ **do**
    for all vertices v **do**
      $IP(v) \leftarrow$ sorted sequence of intersection points from
        $\pi^{i-1}(v) \oplus \bigoplus_{\text{arcs } w \to v} \left(\pi^{i-1}(w) \odot (c(w \to v), \mu(w \to v))\right)$
    $IP \leftarrow$ mergesort of all $IP(v)$ $\{IP$ is an ordered list of intersection points.$\}$
    Binary search with Subroutine 1 on IP to determine an intersection-free interval containing $\pi^*(s)$.
    $[b,e] \leftarrow$ intersection-free interval containing $\pi^*(s)$
    for all vertices v **do**
      Determine $\pi^i(v)$ which is the linear function of the interval containing $[b,e]$.
  return b

Algorithm 2: Strongly Polynomial GSP Algorithm

lem (GSP). The algorithm sketched at the end of the previous section consisted of binary search to choose ϖ with an inner loop indicating whether a particular guess was too small, correct, or too large. Our strongly polynomial algorithm will interleave searching for the optimal value with iterations of recurrences. This will eliminate the dependence of the number of Bellman-Ford comparisons on the flow multipliers.

We represent each vertex potential as a line per the isomorphism of Lemma 3.2. Initially, we know the answer $\pi^*(s)$ is in a finite range $[b,e]$ or the problem is infeasible. Each iteration will narrow the range by invoking the Bellman-Ford comparison subroutine for several values in the range.

Each Bellman-Ford recurrence requires using $\oplus$, and this operator requires having ϖ sufficiently close to the optimal value $\pi^*(s)$ to compute the correct answer. Consider operating on two distinct lines in the positive-slope (ϖ, π) Cartesian plane. If the lines intersect, they do so only at one point. Using the Bellman-Ford comparison subroutine at this point will indicate on which side of the point the optimal value resides and which line has larger value (or, if we are lucky, we will discover the optimal value). We can use this idea on all the $\oplus$ operands for one recurrence equation. The operands' lines form a piecewise linear monotonic function. We can use binary search on the function's intersection points.

We can extract more parallelism during each iteration of the algorithm by performing binary search on all of the vertices' intersection points simultaneously. Pseudocode appears in Algorithm 2. The algorithm only invokes the Bellman-Ford comparison subroutine (Subroutine 1) when needed to resolve which of an intersecting set of lines has the maximum value. When Algorithm 2 finishes, the smallest value in $[b,e]$ is the answer. One more Bellman-Ford subroutine invocation using this value will yield the augmented path.

LEMMA 5.1. *The parametric search Algorithm 2 solves the GSP problem and requires $O(mn^2 \log m)$ time and $O(m+n)$ space.*

6 Generalized Flow Approximation Schemes

Using the strongly polynomial single-source generalized shortest paths algorithm (Algorithm 2), we develop fully polynomial-time approximation schemes for the generalized versions of maximum flow, nonnegative-cost minimum-cost flow, concurrent flow, multicommodity maximum flow, and multicommodity nonnegative-cost minimum-cost flow problems. For all but generalized maximum flow, these approximation schemes yield the first polynomial-time algorithms not based on interior-point methods. Also, all the schemes' running times are independent of the flow multipliers μ. The generalized concurrent flow and multicommodity maximum flow approximation schemes are the first strongly polynomial approximation schemes known to the authors. This is of interest because no strongly polynomial exact algorithms are known. The only generalized problem known to have a strongly polynomial exact algorithm is the GSP. In 1994, Cohen and Megiddo [CM94] presented a fully strongly polynomial approximation scheme for the generalized maximum flow problem.

To create the approximation schemes, we use three different packing frameworks [CM94, GK98, GK96] and a GSP algorithm. Writing the generalized flow problems listed above as linear programs requires more than m constraints. The dual linear program, however, assigns one dual variable to each constraint and has only m constraints, each specifying the cost of an arc so a GSP algorithm can be used. Each framework yields an approximation algorithm that iteratively computes the flow and the dual variables' values. Each iteration, a GSP algorithm yields a minimum-cost flow w.r.t. the dual variables' values, the flow is combined with the previously computed flow, and new dual variable values are computed. The frameworks differ according to how the flows are combined and how the costs are computed.

The remainder of this section is split into three parts. Generalized problems with nonnegative costs are solved using the Grigoriadis-Khachiyan framework [GK96]. The generalized concurrent flow problem is solved using the technique of Cohen and Megiddo [CM94]. Generalized maximum flow problems are solved using the Garg-Könemann framework [GK98]. Pseudocode for the approximation algorithms follow the references.

6.1 Problems with Nonnegative Costs First, we will consider the generalized minimum-cost flow and generalized multicommodity minimumcost flow problems. Of all generalized flow problems, the *generalized minimum-cost flow problem* appears most frequently in the generalized flow literature. For example, the first generalized flow paper [Jew62] and the generalized flow chapter in [AMO93] concentrate on this problem. In addition to GSP's input, the nonnegative-cost version requires an arc capacity function $u : A \to \mathbf{R} > 0$ and restricts costs to be nonnegative. The goal is to find a flow of minimum-cost that obeys the capacity constraints. A flow on an arc obeys the arc's capacity constraint if the flow entering the arc is less than or equal to the arc's capacity. An ϵ-approximate solution exactly satisfies the supplies but may violate arc capacity constraints by a factor of $1+\epsilon$ and the flow's cost may be $1+\epsilon$ larger than the minimum-possible cost for a flow strictly obeying the arc capacity constraints.

(For simplicity, we will consider problems with only one source vertex. Exact algorithms do not differentiate between single source and multiple source problems, but this approximation scheme does. One approach to solve multiple source problems, each source having its own supply, is to add a supersource vertex connected to each source by an arc with capacity equal to the source's supply. The approximation scheme will yield a solution ensuring the total supply entering the network equals the total of the source vertices' supplies, but some sources may send up to $1+\epsilon$ their supply into the network. This technique's running time is the same as for a single source. Alternatively, each source can be treated as a separate commodity. The approximation scheme ensures each source sends exactly its supply into the network, but the running time and space requirements are multiplied by the number of sources.)

LEMMA 6.1. *A combinatorial ϵ-approximation algorithm (Algorithm 3 specialized to one commodity) solves the generalized nonnegative-cost minimum-cost flow problem in $O((\epsilon^{-2}\log\epsilon^{-1} + \log m)m^2n^2\log m\log(mCU))$ time and $O(m+n)$ space. (C and U represent the absolute value of the largest arc cost and capacity, respectively.)*

Proof. We consider the *generalized cost-bounded flow problem*, which is the same as the generalized minimum-cost flow problem except the latter's objective function is replaced by a constraint bounding the maximum permitted flow cost. The cost-bounded problem's goal is to find a feasible flow, i.e., one obeying the arc capacities and having cost at most the bound. Using binary search on the possible cost range $[-mCU, mCU]$, one can solve the minimum-cost problem.

Instead of directly dealing with arc capacity and cost constraints, the Grigoriadis-Khachiyan framework [GK96] uses their dual variables to determine arc costs. A GSP flow with respect to these prices is computed, scaled to satisfy the source's supply, and then a convex combination of the current flow and this minimum-cost flow replaces the current flow's value. Grigoriadis and Khachiyan present an exponential potential function argument showing a solution violating the constraints by at most a $1+\epsilon$ factor can be computed in $O((\epsilon^{-2}\log\epsilon^{-1}+\log m)m)$ iterations and $O(m)$ space. Using the strongly polynomial GSP of the previous section proves the lemma. □

The *generalized multicommodity minimum-cost flow problem* models sharing of network resources by several different commodities. The generalized minimum-cost flow problem specifies one source vertex and its supply, while the multicommodity version specifies k sources each with its own supply. All of the commodities' flows must share the network simultaneously while the total flow, over all the commodities, must still obey the arc capacity constraints and minimizing the total cost. (For details, see, e.g., [AMO93, Chapter 17].)

LEMMA 6.2. *The generalized multicommodity minimum-cost flow problem can be ϵ-solved in $O((\epsilon^{-2}\log\epsilon^{-1} + \log m)km^2n^2\log m\log(mCU))$ time and $O(k(m+n))$ space.*

6.2 Generalized Concurrent Flow The *generalized concurrent flow problem* models sharing of network resources while omitting a cost function. The objective function is to maximize the fraction of all the commodities' supplies that can be simultaneously satisfied given the network's arc capacity constraints.

LEMMA 6.3. *The generalized concurrent flow problem can be solved within a factor of $1-\epsilon$ of the optimal answer in $O(\log\epsilon^{-1}km^2n^2\log m)$ time and exactly solved in $O(km^3n^2\log m\log(nB))$ time. The space requirement is $O(k(m+n))$.*

6.3 Maximizing Flow We solve the generalized maximum flow and generalized multicommodity maximum flow problems using the framework of Garg and Könemann [GK98]. The *generalized maximum flow problem* is to maximize the flow out of a source vertex. That is, given a directed graph $G = (V, A)$, an arc multiplier function $\mu : A \to \mathbf{R} > 0$, an arc capacity function $u : A \to \mathbf{R} > 0$, and a source vertex $s \in V$, find a flow function $f : A \to \mathbf{R} \geq 0$ obeying flow conservation at all vertices except the source, the multiplier function, and the capacity function while maximizing the flow out of the source vertex. An ϵ-optimal solution is one within at least a $1-\epsilon$ factor of the optimal value.

The *generalized multicommodity maximum flow problem* models sharing of the network by several different commodities. Each commodity has its own source vertex. The goal is to maximize the total flow out of the source vertices. Note the problem is distinct from its single-commodity version only if the arcs have different arc multipliers (and possibly capacities) for different commodities.

LEMMA 6.4. *The generalized maximum flow problem can be ϵ-approximately solved in $O(\epsilon^{-2}m^2n^2\log^2 m)$ time and $O(m+n)$ space. The generalized multicommodity maximum flow problem can be ϵ-approximately solved in $O(\epsilon^{-2}km^2n^2\log^2 m)$ time and $O(k(m+n))$ space.*

The running times of Radzik's and Tardos and Wayne's approximation algorithms [Rad93, TW98] for the generalized maximum flow problem dominate the running time in the previous lemma, but we have included the algorithm for completeness.

LEMMA 6.5. *The generalized cost-bounded flow, generalized concurrent flow, generalized multicommodity maximum flow, and generalized multicommodity cost-bounded flow problems can be ϵ-approximately solved in strongly polynomial time.*

References

[AC91] I. Adler and S. Cosares. A strongly polynomial algorithm for a special class of linear programs. *Operations Research*, 39(6):955–960, November–December 1991.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, Reading, MA, 1974.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[Bel58] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1990.

[CM94] Edith Cohen and Nimrod Megiddo. New algorithms for generalized network flows. *Mathematical Programming*, 64(3):325–336, May 1994.

[CR66] A. Charnes and W. M. Raike. One-pass algorithms for some generalized network problems. *Operations Research*, 14:914–924, September–October 1966.

[Dan63] George Bernard Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

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

[Flo62] Robert W. Floyd. Algorithm 245 (SHORTEST PATH). *Communications of the Association for Computing Machinery*, 5(6):345, 1962.

[For56] L. R. Ford, Jr. Network flow theory. Technical Report P-923, The Rand Corporation, Santa Monica, CA, August 1956.

[GK96] Michael D. Grigoriadis and Leonid G. Khachiyan. Coordination complexity of parallel price-directive decomposition. *Mathematics of Operations Research*, 21(2):321–340, May 1996.

[GK98] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, November 1998.

[GOPS98] Andrew V. Goldberg, Jeffrey D. Oldham, Serge Plotkin, and Cliff Stein. An implementation of a combinatorial approximation algorithm for minimum-cost multicommodity flow. In Robert E. Bixby, E. Andrew Boyd, and Roger Z. Ríos-Mercado, editors, *Integer Programming and Combinatorial Optimization*, volume 1412 of *Lecture Notes in Computer Science*, pages 338–352, Berlin, 1998. Springer. Technical Report available as http://theory.stanford.edu/˜oldham/publications/-MCMCF-TR/MCMCF-TR.ps.

[GPT91] Andrew V. Goldberg, Serge A. Plotkin, and Éva Tardos. Combinatorial algorithms for the generalized circulation problem. *Mathematics of Operations Research*, 16(2):351–381, May 1991.

[GZO97] Donald Goldfarb, Jin Zhiying, and James B. Orlin. Polynomial-time highest-gain augmenting path algorithms for the generalized circulation problem. *Mathematics of Operations Research*, 22(4):793–802, November 1997.

[HN94] Dorit S. Hochbaum and Joseph (Seffi) Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM Journal on Computing*, 23(6):1179–1192, December 1994.

[Jac45] C. G. J. Jacobi. Über eine neue auflösungsart der bei der methode der kleinsten quadrate vorkommenden linearen gleichungen. *Astronomische Nachrichten*, 22:297–303, 1845.

[Jew62] William S. Jewell. Optimal flow through networks with gains. *Operations Research*, 10(4):476–499, July–August 1962.

[Knu92] Donald E. Knuth. Two notes on notation. *American Mathematical Monthly*, 99(5):403–422, May 1992.

[KP95a] Anil Kamath and Omri Palmon. Improved interior point algorithms for exact and approximate solution of multicommodity flow problems. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 502–511. Association for Computing Machinery, January 1995.

[KP95b] David Karger and Serge Plotkin. Adding multiple cost constraints to combinatorial optimization problems, with applications to multicommodity flows. In *Symposium on the Theory of Computing*, volume 27, pages 18–25. Association for Computing Machinery, ACM Press, May 1995.

[Meg79] Nimrod Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4(4):414–424, November 1979.

[PST95] Serge A. Plotkin, David B. Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20(2):257–301, May 1995.

[Rad93] Tomasz Radzik. Approximate generalized circulation. Technical Report 93-2, Cornell Computational Optimization Project, Ithaca, NY, January 1993.

[Tar86] Éva Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, March–April 1986.

[TB96] P. Tseng and D. P. Bertsekas. An ϵ-relaxation method for separable convex cost generalized network flow problems. In W. C. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Proceedings of the Fifth IPCO Conference on Integer Programming and Combinatorial Optimization*, June 1996. ftp://ftp.math.washington.edu/-pub/tseng/papers/erelaxg.ps.Z, to appear in *Mathematical Programming*.

[TW98] Éva Tardos and Kevin D. Wayne. Simple generalized maximum flow algorithms. In R. E. Bixby, E. A. Boyd, and R. Z. Ríos-Mercado, editors, *Integer Programming and Combinatorial Optimization*, volume 1412 of *Lecture Notes in Computer Science*, pages 310–324, Berlin, June 1998. Springer.

[Vai89] Pravin M. Vaidya. Speeding-up linear programming using fast matrix multiplication. In *Thirtieth Annual Symposium on Foundations of Computer Science*, volume 30, pages 332–337, Los Alamitos, CA, 30 October–01 November 1989. IEEE Computer Society Press.

[War62] Stephen Warshall. A theorem on boolean matrices. *Journal of the Association for Computing Machinery*, 9(1):11–12, January 1962.

[WF99] Kevin D. Wayne and Lisa Fleischer. Faster approximation algorithms for generalized flow. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, January 1999. http://www.cs.princeton.edu/ wayne/packing.ps.

Pseudocode for the Approximation Algorithms

Input: desired accuracy ϵ and initial flow f satisfying supplies
Output: flow f satisfying supplies and obeying constraints up to $1+\epsilon$ factor
loop
  Compute arc costs $c(f)$ as an exponential function of constraints under the current flow f.
  for all commodities i **do**
    $f_i^* \leftarrow \text{GSP}_i(c(f))$
    Scale f_i^* to satisfy supply d_i.
  if the difference between f and f^*'s costs is small **then**
    return f
  else
    $f \leftarrow (1-\theta)f + \theta f^*$

Algorithm 3: ϵ-Generalized Multicommodity Cost-Bounded Flow Algorithm

flow $f \leftarrow 0$
for $O(m \log \epsilon^{-1})$ iterations **do**
  Compute arc costs c as a function of residual arc capacities.
  for all commodities i **do**
    $f_i^* \leftarrow \text{GSP}_i(c)$
    Scale f_i^* to satisfy supply d_i.
  $f^* \leftarrow \sum_{\text{commodities } i} f_i^*$
  Scale f^* so all arc capacities obeyed and at least one arc is saturated.
  $f \leftarrow f + f^*$

Algorithm 4: ϵ-Generalized Concurrent Flow Algorithm

$(\forall \text{ arcs } a)(c(a) \leftarrow \iota)$ $\{\iota$ is a very small positive number.$\}$
flow $f \leftarrow 0$
loop
  $f^* \leftarrow \text{GSP}(c)$
  Scale f^* so all arc capacities obeyed and at least one capacity constraint becomes tight.
  $(\forall \text{ arcs } a)(c(a) \leftarrow c(a)(1+\alpha f^*(a)/u(a))$
  if $c \cdot f^* \geq 1$ **then**
    return f scaled to be feasible.
  $f \leftarrow f + f^*$

Algorithm 5: ϵ-Generalized Maximum Flow Algorithm