Approximating fractional multicommodity flow independent of the number of commodities

Lisa K. Fleischer*

May 31, 1999

Abstract

We describe fully polynomial time approximation schemes for various multicommodity flow problems in graphs with m edges and n vertices. We present the first approximation scheme for maximum multicommodity flow that is independent of the number of commodities k, and our algorithm improves upon the runtime of previous algorithms by this factor of k, performing in $\mathcal{O}^*(\epsilon^{-2}m^2)$ time. For maximum concurrent flow, and minimum cost concurrent flow, we present algorithms that are faster than the current known algorithms when the graph is sparse or the number of commodities k is large, i.e. $k > m/n$. Our algorithms build on the framework proposed by Garg and Könemann [4]. They are simple, deterministic, and for the versions without costs, they are strongly polynomial.

Our maximum multicommodity flow algorithm extends to an approximation scheme for the maximum weighted multicommodity flow, which is faster than those implied by previous algorithms by a factor of $k/\log W$ where W is the maximum weight of a commodity.

*Department of Industrial Engineering and Operations Research, Columbia University, New York, NY 10027. Email: lisa@ieor.columbia.edu. Part of this paper was written while the author was at Center for Operations Research and Econometrics, Louvain-la-Neuve, Belgium.

1 Introduction

A multicommodity flow problem is defined on a directed network $G = (V, E)$ with capacities $u : E \to \mathbf{R}$ and k source-sink terminal pairs (s_i, t_i), $1 \leq i \leq k$. The problem is to find flows f_i from s_i to t_i that satisfy node conservation constraints and meet some objective function criteria so that the sum of flows on any edge does not exceed the capacity of the edge. Let $|f_i|$ denote the amount of flow sent from s_i to t_i in f_i. For the *maximum multicommodity flow* problem, the objective is to maximize the sum of the flows: $\max \sum_i |f_i|$. For the *maximum concurrent flow* problem, there are demands d_i associated with each commodity i, and the objective is to satisfy the maximum possible proportion of all demands: $\max \lambda$, $|f_i| \geq \lambda d_i \ \forall i$. If there are costs $c(e)$ associated with a unit of flow on edge e, the minimum cost concurrent flow problem is to find a maximum concurrent flow of minimum cost, where the cost of a flow is the cost of sending flow on each edge of the graph, summed over all the edges carrying flow.

Our goal is to find an *ϵ-approximate solution* for any error parameter $\epsilon > 0$. For the maximization problems, an ϵ-approximate solution is a flow that has value at least $(1 - \epsilon)$ times the maximum value. For versions with costs, an ϵ-approximate solution is flow that has value at least $(1 - \epsilon)$ times the maximum value and has cost at most the optimal cost. For each problem discussed in this paper, we describe a *fully polynomial-time approximation scheme* (FPTAS) to solve the problem. A FPTAS is a family of algorithms that finds an ϵ-approximate solution in time polynomial in the size of the input and $1/\epsilon$. Here, the size of the input is specified by the number of nodes n, the number of arcs m, and the largest integer M used to specify any of the capacities, costs, and demands. To simplify the run times, we use $\mathcal{O}^*(f)$ to denote $f \log^{\mathcal{O}(1)} m$.

There has been a series of papers providing FPTAS's for multicommodity flow problems and generalizations. These approximation schemes are all iterative algorithms modeled on Lagrangian relaxations and linear programming decomposition techniques. Shahrokhi and Matula [19] give the first polynomial time, combinatorial algorithm for approximating the maximum concurrent flow problem with uniform capacities, and introduced the use of an exponential length function to model the congestion of flow on an edge. Klein, Plotkin, Stein, Tardos [12] improve the complexity of this algorithm using randomization. Leighton et al. [13] extend [12] to handle graphs with arbitrary capacities, and give improved run times when capacities are uniform. None of these papers considers the versions with costs. Grigoriadis and Khachiyan [8] describe approximation schemes for block angular linear programs that generalize the uniform capacity maximum and minimum cost concurrent flow problems. Plotkin, Shmoys, and Tardos [17] formulate more general approximation schemes for fractional packing and covering problems, and describe an approximation scheme for the minimum cost concurrent flow with general capacities. They also obtain improved run times when all capacities are uniform. These last three papers all observe that it is not necessary to exactly solve the subproblems generated at each iteration; it suffices to obtain an approximate solution. The most theoretically efficient algorithms discussed in the above references are randomized algorithms, although several also describe slower deterministic versions. Radzik [18] gives the first deterministic algorithm to match the run times of the fastest existing randomized algorithms for the maximum concurrent flow problem. His algorithm uses a "round-robin" approach to routing commodities. Karger and Plotkin [11] use this idea to obtain deterministic algorithms for minimum cost concurrent flow that reduce the dependence on ϵ, to improve upon the fastest randomized algorithms. Grigoriadis and Khachiyan [9] reduce the dependence on ϵ further to the current best known ϵ^{-2} with a specialized version of their initial algorithm. To get this improvement, they use the fact that it is sufficient to solve the subproblems approximately.

problem	previous best	this paper
max multiflow	$\mathcal{O}^*(\epsilon^{-2}km^2)$ [10, 4][1]	$\mathcal{O}^*(\epsilon^{-2}m^2)$
max concurrent flow	$\mathcal{O}^*(\epsilon^{-2}m(m+k)+k$ max flows) [4][1] $\mathcal{O}^*(\epsilon^{-2}kmn)$ [13, 18]	$\mathcal{O}^*(\epsilon^{-2}m(m+k))$
min cost concurrent flow	$\mathcal{O}^*((\epsilon^{-2}m(m+k)+kmn)\log M)$ [4][1] $\mathcal{O}^*(\epsilon^{-2}kmn\log M)$ [9]	$\mathcal{O}^*(\epsilon^{-2}m(m+k)\log M)$

Figure 1: Comparison of multicommodity flow FPTAS. $\mathcal{O}^*()$ hides polylog(m).

All of the above algorithms compute initial flows, and then reroute flow from more congested paths to less congested paths. Young [21] proposes the first scheme that does not explicitly use the residual graph. This is a randomized algorithm. In contrast to the earlier algorithms, the subproblem in Young's scheme is a shortest path computation, instead of a single commodity minimum cost flow computation.

Shmoys [20] explains how the framework of [17] can be used to approximately solve the maximum multicommodity flow problem. The subproblem he uses is also a shortest path problem. Grigoriadis and Khachiyan [10] reduce the run time for approximately solving this problem by a factor of $1/\epsilon$ using a logarithmic instead of exponential potential function. Their algorithm can also provide approximation guarantees for the minimum cost version.

Recently, Garg and Könemann [4] give simple, deterministic algorithms to solve both the maximum multicommodity flow problem and the concurrent flow problems. Like the work in [21], their algorithms do not maintain a residual graph and use shortest path computations as a subproblem. For the maximum multicommodity flow problem, their algorithm matches the complexity in [10]. They obtain a small improvement in run time for the concurrent flow problems. Their main contribution is to provide a very simple analysis for the correctness of their algorithms, and a simple framework for positive packing problems.

We present faster approximation schemes for both maximum multicommodity flow and concurrent flow problems. For the maximum multicommodity flow problem we give the first approximation scheme with run time that is independent of the number of commodities. It is faster than the best previous approximation schemes by the number of commodities. For the maximum concurrent flow problem, we describe an algorithm that is faster than the best previous approximation schemes when the graph is sparse or there are a large number of commodities. In particular, our algorithm is faster when $k > m/n$. See Figure 1 for comparison with previous work. Our algorithms build on the framework proposed in [4]. They are deterministic and do not use the residual graph.

Fractional multicommodity flow problems can be solved in polynomial time by linear programming techniques. However, these problems are typically quite large and can take a long time to solve using these techniques. As discussed below, there are many applications of such problems. For this reason, developing faster algorithms that deliver solutions that are provably close to optimal is of interest. Experimental results to date suggest that these techniques can lead to significantly faster solution times. For example, Bienstock [3] has reported speedups on the order of 20 times to obtain approximate solutions via these methods versus the time needed to obtain similar approximate solutions using standard linear programming techniques. (The speed up over the time to obtain exact solutions is on the order of 1000). These experiments build on earlier experimental work

[1]The extended abstract [4] makes stronger claims, but the actual achievable run times are correctly stated here [5].

including [15, 9, 7].

One area of application for obtaining quick approximate solutions to fractional multicommodity flow problems is the field of network design. Given pairwise demands, it is desired to build a network to route all demand. The network design problems encountered in practice are typically NP-hard, and difficult to solve. In addition, it is often desired to develop many solutions for many different scenarios. A fast multicommodity flow algorithm permits the designer to quickly test if the planned network is feasible, and proceed accordingly. These algorithms can also be incorporated in a branch-and-cut framework to optimize network design problems. For example, see Bienstock [2].

Another area of application is interest in solving the dual problems. The integer version of the dual to the maximum multicommodity flow problem is the multicut problem. The *multicut problem* is, given (s_i, t_i) pairs, to find a minimum capacity set of edges whose removal disconnects the graph so that s_i is in a different component than t_i, for all pairs i. Garg, Vazirani, and Yannakakis [6] describe an algorithm to round the fractional solution to the LP relaxation that is the dual of the maximum multicommodity flow problem to obtain a solution to the multicut problem that is provably close to the optimal solution. The integer version of the dual to the maximum concurrent flow problem is the sparsest cut problem. The *sparsest cut problem* is to find the set S so that the ratio of the capacity of edges leaving S divided by the sum of demands separated by S is minimized. There have been several approximation results for this problem that again round the fractional solution to the LP relaxation of the sparsest cut problem, most recently an algorithm of London, Linial, and Rabinovich [16]. This, in turn can be used to approximate the *balanced cut problem* [14]: the problem of partitioning the vertices of a graph into two sets of size at least $|V|/3$, so that the total capacity of the edges that have one endpoint in each set is minimized.

2 Maximum multicommodity flow

Our work builds directly on the simple analysis given in [4]. We use the linear programming path formulation of the maximum multicommodity flow problem. Let $\mathcal{P}_i$ denote the set of paths from s_i to t_i, and let $\mathcal{P} := \cup_i \mathcal{P}_i$. Variable $x(P)$ equals the amount of flow sent along path P.

$$\begin{array}{lll} \max & \sum_{P\in\mathcal{P}} x(P) & \\ \forall e: & \sum_{P:e\in P} x(P) & \leq \quad u(e) \\ \forall P: & x(P) & \geq \quad 0. \end{array} \qquad (\mathbf{P})$$

The dual LP problem is to assign lengths to the edges of the graph so that the length of the shortest path from s_i to t_i is at least 1 for all commodities i. The length of an edge represents the marginal cost of using an additional unit of capacity of the edge.

$$\begin{array}{lll} \min & \sum_e u(e)l(e) & \\ \forall P: & \sum_{e\in P} l(e) & \geq \quad 1 \\ \forall e: & l(e) & \geq \quad 0. \end{array} \qquad (\mathbf{D})$$

The algorithm starts with length function $l \equiv \delta$ for an appropriately defined δ depending on m and ϵ, and with a primal solution $x \equiv 0$. While there is a path in $\mathcal{P}$ of length less than 1, the algorithm selects such a path and augments flow along this path. The amount of flow sent along path P is determined by the bottleneck capacity of the path, using the *original* capacities. Denote

this capacity by u. The primal solution is updated by setting $x(P) = x(P) + u$. Note that this solution may be infeasible, since it will likely violate capacity constraints. However, it satisfies all other primal constraints. At the end of the algorithm, we will scale the final primal solution so that it is feasible.

After updating $x(P)$, the dual variables are updated by increasing the lengths of the edges exponentially in relation to the congestion of the edge. For edge e on P, the update is $l(e) = l(e)(1+\frac{\epsilon u}{u(e)})$. The lengths of edges not on P remain unchanged. This update ensures that the length of the bottleneck edge on P increases by a factor of $(1+\epsilon)$.

The key to the efficiency of our algorithm lies in our selection of P. Garg and Könemann [4] show that if P is selected so that $P = \text{argmin}_{P\in\mathcal{P}}\, l(P)$, where $l(P) = \sum_{e\in P} l(e)$, then the following series of lemmas hold.

Lemma 2.1 *The algorithm terminates after* $\mathcal{O}(m \log_{1+\epsilon} \frac{1+\epsilon}{\delta})$ *augmentations.*

Lemma 2.2 *The flow obtained by scaling the final flow by* $\log_{1+\epsilon} \frac{1+\epsilon}{\delta}$ *is feasible.*

Lemma 2.3 *If* f_t *is the final flow value, then* $\frac{f_t}{\log_{1+\epsilon} \frac{1+\epsilon}{\delta}} \geq (1-2\epsilon)\,OPT.$

Since k shortest path computations are required to determine the minimum length path in $\mathcal{P}$, this implies an $\mathcal{O}^*(\epsilon^{-2}km^2)$ time approximation scheme for maximum multicommodity flow.

2.1 Our Improvement

We improve upon this algorithm by selecting P in a less costly manner. Instead of finding the shortest path in $\mathcal{P}$, we settle for some path within a factor of $(1+\epsilon)$ of the shortest, and show that we can obtain a similar approximation guarantee. Given a length function l, define $\alpha(l) := \min_{P\in\mathcal{P}} l(P)$. Denote the length function at the end of iteration i by l_i, and for ease of notation let $\alpha(i) = \alpha(l_i)$. We show below in Theorem 2.4 that by augmenting in iteration i along a path P such that $l(P) \leq (1+\epsilon)\alpha(i)$, the number of iterations is unchanged, and the final scaled flow has value at least $(1-4\epsilon)$OPT.

This enables us to modify the algorithm by reducing the number of shortest path computations. Instead of looking at all commodities to see which source sink pair is the closest according to the current length function, we cycle through the commodities, sticking with a commodity until the shortest source to sink path for that commodity is above a $1+\epsilon$ factor times a lower bound estimate of the overall shortest path. Let $\hat{\alpha}(i)$ be a lower bound on $\alpha(i)$. To start, we set $\hat{\alpha}(0) = \delta$. As long as there is some $P \in \mathcal{P}$ with $P \leq \min\{1, (1+\epsilon)\hat{\alpha}(i)\}$, we augment flow along P, and set $\hat{\alpha}(i+1) = \hat{\alpha}(i)$. When this no longer holds, we know that the length of the shortest path is at least $(1+\epsilon)\hat{\alpha}(i-1)$, and so we set $\hat{\alpha}(i) = (1+\epsilon)\hat{\alpha}(i-1)$. Thus, throughout the course of the algorithm, $\hat{\alpha}$ takes on values in the set $\{\delta(1+\epsilon)^i\}_{i\in\mathcal{N}}$. Since $\alpha(0) \geq \delta$ and $\alpha(t-1) < 1$, $\alpha(t) < 1+\epsilon$. Thus, when we stop, $\hat{\alpha}(t)$ is between 1 and $1+\epsilon$. Since each time we increase $\hat{\alpha}$, we increase it by a $1+\epsilon$ factor, the number of times that we increase $\hat{\alpha}$ is $\log_{1+\epsilon} \frac{1+\epsilon}{\delta}$ (which implies that the final value of i is $\log_{1+\epsilon} \lfloor \frac{1+\epsilon}{\delta} \rfloor$).

Between updates to $\hat{\alpha}$, the algorithm proceeds by considering each commodity one by one. As long as the shortest path for commodity j has length less than the minimum of $1+\epsilon$ times the

Input: network G, capacities $u(e)$, commodity pairs (s_i, t_i), $1 \leq i \leq k$, accuracy ϵ
Output: primal (infeasible) and dual solutions x and l
Initialize $l(e) = \delta \ \forall e, \ x \equiv 0$.
for $i = 1$ to $\log_{1+\epsilon} \frac{1+\epsilon}{\delta}$
for $j = 1$ to k **do**
$P \leftarrow$ shortest path in $\mathcal{P}_j$ using l.
while $l(P) < \min\{1, \delta(1+\epsilon)^i\}$
$u \leftarrow \min_{e \in P} u(e)$
$x(P) \leftarrow x(P) + u$
$\forall e \in P, \ l(e) \leftarrow l(e)(1 + \frac{\epsilon u}{u(e)})$
$P \leftarrow$ shortest path in $\mathcal{P}_j$ using l.
end while
Return (x, l).

Figure 2: FPTAS for maximum multicommodity flow

current value of $\hat{\alpha}$ and 1, flow is augmented along such a shortest path. When $\min_{P \in \mathcal{P}_j} \geq (1+\epsilon)\hat{\alpha}$, commodity $j+1$ is considered. After all k commodities are considered, $\hat{\alpha}$ is updated. Between each update, there is at most one shortest path computation per commodity that does not lead to an augmentation. We charge these computations to the update of $\hat{\alpha}$. Thus a total of at most $k \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$ shortest path computations are used to update α over the course of the algorithm. This algorithm is presented in Figure 2.

We prove Lemmas 2.1 and 2.2 for this modified algorithm.

Proof of Lemma 2.1: At start, $l(e) = \delta$ for all edges e. The last time the length of an edge is updated, it is on a path of length less than one, and it is increased by at most a factor of $1+\epsilon$. Thus the final length of any edge is at most $1+\epsilon$. Since every augmentation increases the length of some edge by a factor of least $1+\epsilon$, the number of possible augmentations is at most $m \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$. ■

Proof of Lemma 2.2: Every time the total flow on an edge increases by a fraction $0 < a_i \leq 1$ of its capacity, its length is multiplied by $1 + a_i\epsilon$. Since $1 + a\epsilon \leq (1+\epsilon)^a$ for all $0 \leq a \leq 1$, $\Pi_i(1 + a_i\epsilon) \geq (1+\epsilon)^{\sum_i a_i}$, when $0 \leq a_i \leq 1$ for all i. Thus, every time the flow on an edge increases by its capacity, the length of the edge increases by a factor of at least $1+\epsilon$. Since initially $l(e) = \delta$ and at the end $l(e) < 1+\epsilon$, the total flow on edge e cannot exceed $u(e) \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$. ■

Let L be the maximum number of arcs in an augmenting path used in the algorithm. We will use $\delta = (1+\epsilon)/((1+\epsilon)L)^{1/\epsilon}$. Thus, these lemmas together with our observations on the number of times $\hat{\alpha}$ is recomputed, imply a runtime of $\mathcal{O}^*(\epsilon^{-2}(m^2 + km))$. This can be reduced to $\mathcal{O}^*(\epsilon^{-2}m^2)$ by observing that for this problem we can group commodities by a common source node, and compute shortest paths from the source nodes, both in our computation of $\hat{\alpha}$, and in our selection of an augmenting path. Below, we show that this algorithm actually finds a flow that is close to optimal by comparing the scaled primal solution to the dual solution that is a byproduct of the algorithm. Our choice of δ is determined by getting this ratio to be at least $(1-\epsilon)$.

Theorem 2.4 *An ϵ-approximate maximum multicommodity flow can be computed in $\mathcal{O}(\frac{1}{\epsilon^2} m(m + n \log m) \log L)$ time.*

Proof: We show that the scaled flow in Lemma 2.2 has value within $1/(1+4\epsilon)$ of the dual optimal value, and hence the optimal value for the primal. By choosing an initial value $\epsilon' = \epsilon/4$, this then implies the theorem. At the end of iteration i, $\alpha(i)$ is the exact shortest path over all commodities using length function l_i. Given length function l, define $D(l) := \sum_e l(e)u(e)$, and let $D(i) := D(l_i)$. $D(i)$ is the dual objective function value of l_i and $\beta = \min_l D(l)/\alpha(l)$ is the optimal dual objective value.

For each iteration $i \geq 1$,

$$D(i) = \sum_e l_i(e)u(e) = \sum l_{i-1}(e)u(e) + \epsilon \sum_{e \in P} l_{i-1}(e)u \leq D(i-1) + \epsilon(f_i - f_{i-1})(1+\epsilon)\alpha(i-1),$$

which implies that

$$D(i) \leq D(0) + \epsilon(1+\epsilon) \sum_{j=1}^{i} (f_j - f_{j-1})\alpha(j-1). \tag{1}$$

Consider the length function $l_i - l_0$. Note that $D(l_i - l_0) = D(i) - D(0)$. For any path used by the algorithm, the length of the path using l_i versus $l_i - l_0$ differs by at most δL. Since this holds for the shortest path using length function $l_i - l_0$, we have that $\alpha(l_i - l_0) \geq \alpha(i) - \delta L$. Hence

$$\beta \leq \frac{D(l_i - l_0)}{\alpha(l_i - l_0)} \leq \frac{D(i) - D(0)}{\alpha(i) - \delta L}.$$

Substituting this bound on $D(i) - D(0)$ in equation (1) gives

$$\alpha(i) \leq \delta L + \frac{\epsilon(1+\epsilon)}{\beta} \sum_{j=1}^{i} (f_j - f_{j-1})\alpha(j-1).$$

Observe that, for fixed i, this right hand side is maximized by setting $\alpha(j)$ to its maximum possible value, for all $0 \leq j < i$. Call this maximum value $\alpha'(i)$. Hence

$$\begin{aligned} \alpha(i) \leq \alpha'(i) &= \delta L + \frac{\epsilon(1+\epsilon)}{\beta} \sum_{j=1}^{i-1} (f_j - f_{j-1})\alpha'(j-1) + \frac{\epsilon(1+\epsilon)}{\beta}(f_i - f_{i-1})\alpha'(i-1) \\ &= \alpha'(i-1)(1 + \epsilon(1+\epsilon)(f_i - f_{i-1})/\beta) \\ &\leq \alpha'(i-1)e^{\epsilon(1+\epsilon)(f_i - f_{i-1})/\beta}, \end{aligned}$$

where this last inequality uses the fact that $1 + x \leq e^x$ for $x \geq 0$. Since $\alpha'(0) = \delta L$, this implies that

$$\alpha(i) \leq \delta L e^{\epsilon(1+\epsilon)f_i/\beta} \tag{2}$$

By the stopping condition,

$$1 \leq \alpha(t) \leq \delta L e^{\epsilon(1+\epsilon)f_t/\beta}$$

and hence

$$\frac{\beta}{f_t} \leq \frac{\epsilon(1+\epsilon)}{\ln(\delta L)^{-1}}. \tag{3}$$

Let γ be the ratio of the dual and primal solutions. $\gamma := \frac{\beta}{f_t} \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$. By substituting the bound on β/f_t from (3), we obtain

$$\gamma \leq \frac{\epsilon(1+\epsilon)\log_{1+\epsilon}\frac{1+\epsilon}{\delta}}{\ln(L\delta)^{-1}} = \frac{\epsilon(1+\epsilon)}{\ln(1+\epsilon)}\frac{\ln\frac{1+\epsilon}{\delta}}{\ln(L\delta)^{-1}}.$$

Letting $\delta = (1+\epsilon)((1+\epsilon)L)^{-1/\epsilon}$, we have

$$\gamma \leq \frac{\epsilon(1+\epsilon)}{(1-\epsilon)\ln(1+\epsilon)} \leq \frac{\epsilon(1+\epsilon)}{(1-\epsilon)(\epsilon-\epsilon^2/2)} \leq \frac{(1+\epsilon)}{(1-\epsilon)^2}.$$

This is at most $(1+4\epsilon)$, for $\epsilon < .15$. The choice of δ together with Lemma 2.1 and the discussion of the algorithm then give the run time. ∎

Above, we have shown that the framework of Garg and Könemann [4] can be modified to obtain a more efficient algorithm, by showing that on average, it is sufficient to solve the shortest path subproblem for a single commodity. We note that the same is not easily said of the algorithm described in [10]. This algorithm requires that the shortest path subproblem for each commodity be solved (albeit approximately) at each iteration.

Although it is simple to extend this algorithm to approximately solve the minimum cost maximum multicommodity flow problem, the technique for doing so does not differ from the technique of extending the maximum concurrent flow problem to the minimum cost concurrent flow problem. For this reason, we omit this discussion and refer the reader to the section on minimum cost concurrent flows.

2.2 Maximum Weighted Multicommodity Flow

The maximum multicommodity flow approximation scheme can be extended to handle weights in the objective function. Let w_j be the weight of commodity j. We wish to find flows f_j maximizing $\sum_j w_j|f_j|$. By scaling, we assume $\min_j w_j = 1$, and then the maximum ratio of weights of any two commodities W can be taken to be equal to $\max_j w_j$.

Corollary 2.5 *A solution of value within $(1-\epsilon)$ of the optimal solution for the weighted maximum multicommodity flow problem problem can be computed in $\mathcal{O}^*(\epsilon^{-2}m^2 \min\{\log W, k\})$ time.*

Proof: We alter the FPTAS for maximum multicommodity flow by redefining $\alpha(i) = \min_{1\leq j\leq k}$ (length of shortest path connecting commodity pair $j)/w_j$, where w_j is the weight for commodity j, and halting the algorithm when $\alpha \geq 1/W$. For this modified algorithm, we choose $\delta = (1+\epsilon)/((1+\epsilon)LW)^{1/\epsilon}$. The rest of the algorithm remains unchanged. The analysis of the algorithm changes slightly, now that we have modified α. Now, the value of $\alpha(0)$ can be as low as δ/W, and as large as $(1+\epsilon)/W$. Hence, the number times α can increase by a factor of $(1+\epsilon)$ remains the same function of δ as before, $\log_{1+\epsilon}\frac{1+\epsilon}{\delta}$. The length of an edge starts at δ and can be as large as 1, so the number of iterations, and the scale factor to create a feasible solution do not change as a function of δ. To obtain the guarantee on the ratio of feasible dual and primal solutions, we set $\delta = (1+\epsilon)/((1+\epsilon)LW)^{1/\epsilon}$. This implies an $\mathcal{O}^*(\epsilon^{-2}m^2 \log W)$ run time.

The strongly polynomial run time follows directly from the analysis of the packing LP algorithm in [4]. ∎

3 Maximum concurrent flow

As with the maximum multicommodity flow, we let $x(P)$ denote the flow quantity on path P. The maximum concurrent flow problem can be formulated as the following linear program.

$$\begin{array}{lll} \max & \lambda & \\ \forall e: & \sum_{P:e\in P} x(P) & \leq u(e) \\ \forall j: & \sum_{P\in\mathcal{P}_j} x(P) & \geq \lambda d_j \\ \forall P: & x(P) & \geq 0. \end{array}$$

The dual LP problem is to assign lengths to the edges of the graph, and weights z_j to the commodities so that the length of the shortest path from s_j to t_j is at least z_j for all commodities j, and the sum of the product of commodity weights and demands is at least 1. The length of an edge represents the marginal cost of using an additional unit of capacity of the edge, and the weight of a commodity represents the marginal cost of not satisfying another unit of demand of the commodity.

$$\begin{array}{lll} \min & \sum_e u(e)l(e) & \\ \forall j,\ \forall P\in\mathcal{P}_j: & \sum_{e\in P} l(e) & \geq z_j \\ & \sum_{1\leq j\leq k} d_j z_j & \geq 1 \\ \forall e: & l(e) & \geq 0 \\ \forall j: & z_j & \geq 0 \end{array}$$

When the objective function value is ≥ 1, Garg and Könemann [4] describe an $\mathcal{O}^*(\epsilon^{-2}m(m+k))$ time approximation scheme for the maximum concurrent flow problem. If the objective is less than one, then they describe a procedure to scale the problem so that the objective is at least one. This procedure requires computing k maximum flows, thus increasing the runtime of their algorithm, to match previously known algorithms. We describe a different procedure that requires k maximum bottleneck path computations which can be performed in $\mathcal{O}(m)$ time each, and thus produces an algorithm to solve the problem in $\mathcal{O}^*(\epsilon^{-2}m(m+k))$ time. For the cost bounded problem, the required subroutine in [4] is a minimum cost flow, whereas in our case, it is a cost bounded maximum bottleneck path, which can be solved in $\mathcal{O}(m\log m)$ time. Thus our procedure improves the run time for the minimum cost concurrent flow problem as well.

We first describe the approximation algorithm in [4]. Initially, $l(e) = \delta/u(e)$, $z_j = \min_{P\in\mathcal{P}_j} l(P)$, $x \equiv 0$. The algorithm proceeds in phases. In each phase, there are k iterations. In iteration j, the objective is to route d_j units of flow from s_j to t_j. This is done in steps. In one step, a shortest path P from s_j to t_j is computed using the current length function. Let u be the capacity of this path. Then the minimum of u and the remaining demand is sent along this path. The dual variables l are updated as before, and z_j is set equal to the length of the new minimum length path from s_j to t_j. The entire procedure stops when the dual objective function value is at least one: $D(l) := \sum_e u(e)l(e) \geq 1$. Garg and Könemann [4] prove the following sequence of lemmas, for $\delta = (\frac{m}{1-\epsilon})^{-1/\epsilon}$.

Lemma 3.1 *If $\beta \geq 1$, the algorithm terminates after at most $t := 1 + \frac{\beta}{\epsilon}\log_{1+\epsilon}\frac{m}{1+\epsilon}$ phases.*

Input: network G, capacities $u(e)$, vertex pairs (s_i, t_i) with demands d_i, $1 \leq i \leq k$, accuracy ϵ
Output: primal (infeasible) and dual solutions x and l

```
Initialize l(e) = δ/u(e) ∀e, x ≡ 0.
while D(l) < 1
    for j = 1 to k do
        d'_j ← d_j
        while D(l) < 1 and d'_j > 0
            P ← shortest path in P_j using l
            u ← min{d'_j, min_{e∈P} u(e)}
            d'_j ← d'_j − u
            x(P) ← x(P) + u
            ∀e ∈ P, l(e) ← l(e)(1 + εu/u(e))
        end while
end while
Return (x, l).
```

Figure 3: FPTAS for maximum concurrent flow

Lemma 3.2 *After $t-1$ phases, $(t-1)d_j$ units of each commodity j have been routed. Scaling the final flow by $\log_{1+\epsilon} 1/\delta$ yields a feasible primal solution of value $\lambda = \frac{t-1}{\log_{1+\epsilon} 1/\delta}$.*

Lemma 3.3 *If $\beta \geq 1$, then the final flow scaled by $\log_{1+\epsilon} 1/\delta$ has a value at least $(1-3\epsilon)$ OPT.*

The veracity of these lemmas rely on $\beta \geq 1$. Notice also that the run time depends on β. Thus we need to insure that β is at least one and not too large.

Let ζ_j denote the maximum flow value of commodity j in the graph when all other commodities have zero flow. Let $\zeta = \min_j \zeta_j/d_j$. Since at best all single commodity flows can be routed simultaneously, ζ is an upper bound on the value of the optimal solution. The feasible solution that routes $1/k$ fraction of each commodity flow of value ζ_j demonstrates that ζ/k is a lower bound on the optimal solution. Once we have these bounds, we can scale the original demands so that this lower bound is at least one. However, now β can be as large as k.

To reduce the dependence of the number of phases on β, we use a popular technique developed in [17] that is also used in [4]. We run the algorithm, and if it does not stop after $T := 2\frac{1}{\epsilon}\log_{1+\epsilon}\frac{m}{1-\epsilon}$ phases, then $\beta > 2$. We then multiply demands by 2, so that β is halved, and still at least 1. We continue the algorithm, and again double demands if it does not stop after T phases. After repeating this at most k times, the algorithm stops. The total number of phases is $T \log k$. The number of phases can be further reduced by first computing a solution of cost at most twice the optimal, using this scheme. This takes $\mathcal{O}(\log k \log m)$ phases, and returns a value $\hat{\beta}$ such that $\beta \leq \hat{\beta} \leq 2\beta$. Thus with at most T additional phases, an ϵ-approximation is obtained. The following lemma follows from the fact that there are at most k iterations per phase.

Lemma 3.4 *The total number of iterations required by the algorithm is at most $2k \log m(\log k + \frac{1}{\epsilon^2})$.*

It remains to bound the number of steps. For each step except the last step in an iteration, the algorithm increases the length of some edge (the bottleneck edge on P) by $1+\epsilon$. Since each

variable $l(e)$ has initial value $\delta/u(e)$ and value at most $\frac{1}{u(e)}$ before the final step of the algorithm (since $D(t-1) < 1$), the number of steps in the entire algorithm exceeds the number of iterations by at most $m \log_{1+\epsilon} \frac{1}{\delta} = m \log_{1+\epsilon} \frac{m}{1-\epsilon}$.

Theorem 3.5 *Given ζ_j, an ϵ-approximate solution to the maximum concurrent flow problem can be obtained in $\mathcal{O}^*(\epsilon^{-2}m(k+m))$ time.*

Our contribution is to reduce the dependence of the run time of the algorithm on the computation of ζ_j, $1 \leq j \leq k$, by observing that it is not necessary to obtain the exact values of ζ_j, since they are just used to get an estimate on β. Computing an estimate of ζ_j that is at most a factor m away from its true value increases the run time by only a $\log m$ factor. That is, if our estimate $\hat{\zeta_j}$ is $\geq \frac{1}{m}\zeta_j$, then we have upper and lower bounds on β that differ by a factor of at most mk.

We compute estimates $\hat{\zeta_j} \geq \frac{1}{m}\zeta_j$ as follows. Any flow can be decomposed into at most m path flows. Hence flow sent along a maximum capacity path is an m-approximation to a maximum flow. Such a path can be computed easily in $\mathcal{O}(m \log m)$ time by binary searching for the capacity of this path. This run time can be improved to $\mathcal{O}(m)$ using a linear time median finding algorithm. In fact, by grouping commodities by their common source node, we can compute all such paths in $\mathcal{O}(\min\{k,n\}m)$ time.

Theorem 3.6 *An ϵ-approximate solution to the maximum concurrent flow problem can be obtained in $\mathcal{O}^*(\epsilon^{-2}m(k+m))$ time.*

4 Minimum cost concurrent flow

By adding a budget constraint to the multiple commodity problem, it is possible to find a concurrent flow within $(1-\epsilon)$ of the maximum concurrent flow within the given budget. Since there is a different variable for each commodity-path pair, this budget constraint can easily incorporate different costs for different commodities.

In the dual problem, let ϕ be the dual variable corresponding to the budget constraint. In the algorithm, the initial value of ϕ is δ/B. The subroutine to find a most violated primal constraint is a shortest path problem using length function $l + \phi c$, where c is the vector of cost coefficients, and ϕ is the dual variable for the budget constraint. (This can be easily adapted to multiple budget constraints with corresponding dual variables ϕ_i and cost vectors c_i using length function $l + \sum_i \phi_i c_i$.) The amount of flow sent on this path is now determined by the minimum of the capacity of this path, the remaining demand to be sent of this commodity in this iteration, and $B/c(P)$, where $c(P) = \sum_{e \in P} c(e)$ is the cost of sending one unit of flow along this path. Call this quantity u. The length function is updated as before, and ϕ is updated by $\phi = \phi(1 + \epsilon\frac{uc(P)}{B})$. The correctness of this algorithm is demonstrated in [4] and follows a similar analysis as for the maximum concurrent flow algorithm.

Again, we improve on the run time in [4] by efficient approximation of the values ζ_j. To get estimates on ζ_j, as needed to delimit β, we need to compute polynomial factor approximate solutions to $\min\{n^2, k\}$ maximum-flow-with-budget problems. To do this, it is sufficient to compute a maximum bottleneck path of cost at most the budget. This can be done by binary searching for the capacity

of this path among the m candidate capacities, and performing a shortest path computation at each search step, for a run time of $\mathcal{O}(m \log m)$.

To find a $(1-\epsilon)$-maximum concurrent flow of cost no more than the minimum cost maximum concurrent flow, we can then binary search for the correct budget.

Theorem 4.1 *There exists a FPTAS for the minimum cost concurrent flow problem that requires* $\mathcal{O}^*(\epsilon^{-2}m(m+k)\log M)$ *time.*

Acknowledgment

We thank Cynthia Phillips for suggesting a simpler presentation of the maximum multicommodity flow algorithm, and Kevin Wayne and Jeffrey Oldham for helpful comments.

References

[1] *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1995.

[2] Daniel Bienstock. Experiments with a network design algorithm using ϵ-approximate linear programs. Submitted for publication, 1996-98.

[3] Daniel Bienstock, January 1999. Talk at Oberwolfach, Germany.

[4] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th Annual Symp. on Foundations of Comp. Sci.*, pages 300–309, Palo Alto, CA, November 1998. IEEE.

[5] Naveen Garg, January 1999. Personal communication.

[6] Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SJC*, 25(2):235–251, 1996.

[7] A. V. Goldberg, J. D. Oldham, S. Plotkin, and C. Stein. An implementation of a combinatorial approximation algorithm for minimum-cost multicommodity flow. 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 338–352, Berlin, 1998. Springer.

[8] M. D. Grigoriadis and L. G. Khachiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. *SIAM Journal on Optimization*, 4:86–107, 1994.

[9] M. D. Grigoriadis and L. G. Khachiyan. Approximate minimum-cost multicommodity flows. *Mathematical Programming*, 75:477–482, 1996.

[10] M. D. Grigoriadis and L. G. Khachiyan. Coordination complexity of parallel price-directive decomposition. *Mathematics of Operations Research*, 21:321–340, 1996.

[11] D. Karger and S. Plotkin. Adding multiple cost constraints to combinatorial optimization problems, with applications to multicommodity flows. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 00–00, 1995.

[12] P. Klein, S. Plotkin, C. Stein, and É. Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. *SIAM Journal on Computing*, 23:466–487, 1994.

[13] T. Leighton, F. Makedon, S. Plotkin, C. Stein, É. Tardos, and S. Tragoudas. Fast approximation algorithms for multicommodity flow problems. *Journal of Computer and System Sciences*, 50:228–243, 1995.

[14] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with application to approximation algorithms. In *29th Annual IEEE Symposium on Foundations of Computer Science*, pages 422–431, 1988.

[15] Tishya Leong, Peter Shor, and Clifford Stein. Implementation of a combinatorial multicommodity flow algorithm. In David S. Johnson aand C. McGoech, editor, *DIMACS Series in Discrete Mathenatics and Theoretical Computer Science: The First DIMACS IMplementation Challenge: Network Flows and Matchings*, volume 12, pages 387–405. 1993.

[16] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15:215–246, 1995.

[17] S. A. Plotkin, D. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20:257–301, 1995.

[18] T. Radzik. Fast deterministic approximation for the multicommodity flow problem. In ACM/SIAM [1].

[19] F. Shahrokhi and D. W. Matula. The maximum concurrent flow problem. *Journal of the ACM*, 37:318–334, 1990.

[20] D. B. Shmoys. Cut problems and their application to divide-and-conquer. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, chapter 5. PWS Publishing Company, Boston, 1997.

[21] N. Young. Randomized rounding without solving the linear program. In ACM/SIAM [1], pages 170–178.