APPROXIMATING FRACTIONAL MULTICOMMODITY FLOW INDEPENDENT OF THE NUMBER OF COMMODITIES[*]

LISA K. FLEISCHER [†]

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, running 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 in FOCS 1998. They are simple, deterministic, and for the versions without costs, they are strongly polynomial. The approximation guarantees are obtained by comparison with dual feasible solutions found by our algorithm.

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.

Key words. multicommodity flow, approximation algorithm, concurrent flow, VLSI routing

AMS subject classifications. 68Q25, 90C08, 90C27, 90C35, 90C59

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_j, t_j), $1 \leq j \leq k$. The problem is to find flows f_j from s_j to t_j 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 $|f_j|$ denote the amount of flow sent from s_j to t_j in f_j. For the *maximum multicommodity flow* problem, the objective is to maximize the sum of the flows: $\max \sum_j |f_j|$. For the *maximum concurrent flow* problem, there are demands d_j associated with each commodity j, and the objective is to satisfy the maximum possible proportion of all demands: $\max \lambda$, $|f_j| \geq \lambda d_j$, $\forall j$. 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 space needed in a binary representation of

[*]An extended abstract of this paper appeared in *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, 1999.

[†] Graduate School of Industrial Administration, Posner 239, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213. Email: `lkf@andrew.cmu.edu`. Partially supported by NSF through grant EIA-9973858. This paper was written while the author was at the Department of Industrial Engineering and Operations Research at Columbia University, and while on leave at Center for Operations Research and Econometrics, Louvain-la-Neuve, Belgium.

problem	**previous best**	**this paper**
max multiflow	$\mathcal{O}^*(\epsilon^{-2}km^2)$ [15, 10][1]	$\mathcal{O}^*(\epsilon^{-2}m^2)$
max concurrent flow	$\mathcal{O}^*(\epsilon^{-2}m(m+k) + k$ max flows) [10][1] $\mathcal{O}^*(\epsilon^{-2}kmn)$ [19, 24]	$\mathcal{O}^*(\epsilon^{-2}m(m+k))$
min cost concurrent flow	$\mathcal{O}^*((\epsilon^{-2}m(m+k) + kmn)I)$ [10][1] $\mathcal{O}^*(\epsilon^{-2}kmnI)$ [14]	$\mathcal{O}^*(\epsilon^{-2}m(m+k)I)$

FIG. 1.1. *Comparison of multicommodity flow FPTAS. $\mathcal{O}^*()$ hides polylog(m). $I := \log M$.*

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. We discuss these briefly below. These approximation schemes are all iterative algorithms modeled on Lagrangian relaxations and linear programming decomposition techniques. Preceeding this work is a 1973 paper by Fratta, Gerla, and Kleinrock that uses very similar techniques for solving minimum cost multicommodity flow problems [8]. While they do not discuss approximation guarantees, it appears that the flow deviation method introduced in their paper yields an approximation guarantee for this problem with only minor adjustments [6]. Shahrokhi and Matula [26] give the first polynomial time, combinatorial algorithm for approximating the maximum concurrent flow problem with uniform capacities, and introduce the use of an exponential length function to model the congestion of flow on an edge. Klein, Plotkin, Stein, Tardos [18] improve the complexity of this algorithm using randomization. Leighton, et al. [19] extend [18] 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 [13] describe approximation schemes for block angular linear programs that generalize the uniform capacity maximum and minimum cost concurrent flow problems. Plotkin, Shmoys, and Tardos [23] 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 [24] 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 [17] use this idea to obtain deterministic algorithms for minimum cost concurrent flow that reduce the dependence of deterministic algorithms on ϵ. For fixed ϵ, their algorithm also improves upon the fastest randomized algorithms. Grigoriadis and Khachiyan [14] 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.

All of the above algorithms compute initial flows, and then reroute flow from

[1]The extended abstract [10] makes stronger claims, but the actual achievable run times are correctly stated here [9].

more congested paths to less congested paths. Young [28] describes a randomized algorithm that works by augmenting flow along shortest paths using the exponential length function, instead of augmenting by single commodity minimum cost flows. Shmoys [27] explains how the framework of [23] can be used to approximately solve the maximum multicommodity flow problem. The subproblem he uses is also a shortest path problem. Grigoriadis and Khachiyan [15] 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 [10] give simple, deterministic algorithms to solve the maximum multicommodity flow problem, the concurrent flow problem, and the versions with costs. Like the work in [28], they augment flow on shortest paths. For the maximum multicommodity flow problem, their algorithm matches the complexity in [15]. 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 maximum multicommodity flow, maximum concurrent flow, and their minimum cost versions. 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, k. 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 $k > m/n$. We obtain similar improvements for the minimum cost versions. See Figure 1.1 for comparison with previous work. Our algorithms are deterministic and build on the framework proposed in [10].

Our algorithms provide their approximation guarantees by simultaneously finding solutions to the dual linear programs. We show that our primal solutions are within $1-\epsilon$ of the dual solutions obtained, and hence both the primal solutions and the dual solutions are ϵ-approximate. We discuss the interpretation of the dual problems at the very end of this section.

Fractional multicommodity flow problems can be solved in polynomial time by linear programming techniques. However, in many applications these problems are typically quite large and can take a long time to solve using these techniques. For this reason, it is useful to develop faster algorithms that deliver solutions that are provably close to optimal. Experimental results to date suggest that these techniques can lead to significantly faster solution times. For example, Bienstock [5] has reported significant speedups in obtaining approximate and often exact optimal solutions to block decomposable linear programs by building on the ϵ-approximation methods described in [23, 13]. There has also been earlier experimental work including [12, 14, 21]. In [12], they show that certain aspects of the approximation schemes need to be fine tuned to obtain good performance in practice. For example, one aspect is the choice of step size in each iteration. In the theoretical work [14, 23], each iteration involves computing a new flow for a commodity (an improving direction), and then moving to a new solution that is a convex combination of the old flow and the new flow. The step size is determined theoretically by parameters of the algorithm. Goldberg, et al. [12] show that in practice it is better to compute the optimal step size at each iteration.

The focus of the current paper is not to explore directly the experimental efficiency, but instead provide theoretical improvements that are simple to implement and thus may improve experimental performance. Thus we ignore details such as step size and other issues here so that our presentation may be easier to follow. However, since the publication of an extended abstract of this paper [7], these ideas have been tested and shown to lead to demonstrable improvements in practice [2, 25].

One area of application for obtaining quick approximate solutions to fractional multicommodity flow problems is the field of network design: in VLSI design [2], or in design of telecommunication networks [4]. Given pairwise demands, it is desired to build a network with enough capacity 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 [4].

Another area of application is interest in solving the dual problems. The integer versions of the dual problems to multicommodity flow problems are NP-hard. There are approximation algorithms for these problems that guarantee a solution within a logarithmic factor of the optimal solution. These algorithms all start with a solution to the linear program and round this solution. The guarantees are obtained by comparing the resulting integer solution with the LP bound. Since the approximation guarantees for these algorithms is much larger than ϵ, the worst-case performance guarantee does not erode by starting from an ϵ-approximate solution to the linear program instead of an exact solution. Since an ϵ-approximate solution is much easier to obtain, this improves the run time of these algorithms. We briefly describe the integer dual problems below.

The integer version of the dual to the maximum multicommodity flow problem is the multicut problem. The *multicut problem* is, given (s_j, t_j) pairs, to find a minimum capacity set of edges whose removal disconnects the graph so that s_j is in a different component than t_j, for all pairs j. Garg, Vazirani, and Yannakakis [11] describe an algorithm for the multicut problem that returns a solution that is provably close to the optimal solution. Their algorithm rounds the fractional solution to the LP relaxation that is the dual of the maximum multicommodity flow problem. 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 of demand pairs with one end in S and the other outside 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, the most recent algorithms are given by London, Linial, and Rabinovich [22], and Aumann and Rabani [3]. The sparsest cut problem arises as a subroutine in an algorithm for finding an approximately optimal balanced cut [20]. The *balanced cut problem* is to partition 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 [10]. We use the path-flow linear programming formulation of the maximum multicommodity flow problem. Let $\mathcal{P}_j$ denote the set of paths from s_j to t_j, and let $\mathcal{P} := \cup_j \mathcal{P}_j$. Variable $x(P)$ equals the amount of flow sent along path P. The linear programming formulation is then

$$\begin{array}{lllll} \max & \sum_{P\in\mathcal{P}} x(P) & & & \\ \forall e: & \sum\limits_{P:e\in P} x(P) & \leq & u(e) & \\ \forall P: & x(P) & \geq & 0. & \end{array} \qquad \textbf{(P)}$$

The dual to this linear program corresponds to the problem of assigning lengths to the edges of the graph so that the length of the shortest path from s_j to t_j is at least 1 for all commodities j. The length of an edge represents the marginal cost of using an additional unit of capacity of the edge.

$$\begin{array}{llll} \min & \sum_{e} u(e)l(e) & & \\ \forall P: & \sum\limits_{e\in P} l(e) & \geq & 1 \\ \forall e: & l(e) & \geq & 0. \end{array} \qquad \textbf{(D)}$$

While neither of these linear programs are polynomially-sized, they can both be solved in polynomial time. One way to see this is to realize that there are polynomially-sized formulations of both problems: for example, the arc-flow formulation of the maximum multiflow problem has only mk variables and $O((n+m)k)$ constraints. Another way to see this is to realize that the polytope described by the constraints in **D** has a polynomial time separation algorithm. This implies that **D** can be solved in polynomial time using the ellipsoid algorithm, via the equivalence of polynomial time separation and polynomial time optimization for convex polytopes, established by Grötschel, Lovász, and Schrijver [16]. This in turn implies that the primal can also be solve in polynomial time. Given a candidate vector l, the separation algorithm for **D** is to compute the shortest path for each commodity using l as the length function. If there is a path with length less than one, this is a violated inequality. Otherwise l satisfies all constraints.

For large problems it is often desirable to have faster algorithms. We now describe a fast algorithm for obtaining ϵ-approximate solutions to **P** and **D**. The Garg-Könemann algorithm starts with length function $l \equiv \delta$ for an appropriately small $\delta > 0$ 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 increases both the primal and the dual variables associated with this path as follows. For the primal problem, the algorithm 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. The *bottleneck* capacity of a path is the capacity of the minimum capacity edge in the path. 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 nonnegativity constraints. Once we determine the most violated capacity constraint, we can scale the primal solution so that it is feasible by dividing all variables by the appropriate scalar. Thus, at any point in the algorithm we can associate our current solution with a feasible solution. Since we will show that our algorithm has only a polynomial number of iterations, and each iteration increases $x(P)$ for just one P, it is also easy to compute the appropriate scalar since there are always only a polynomial number of non-zero variables. (Alternatively, we may keep track of flows by keeping track of the flow on each edge instead of the flow on each path. This is a natural approach to implementing our algorithm, and makes computing the scale factor for feasibility a very simple task. However, for simplicity of presentation, the algorithm described below will keep track of the values $x(P)$ only.)

```
Input: network $G$, capacities $u(e)$, commodity pairs $(s_j, t_j)$, $1 \leq j \leq k$,
        accuracy $\epsilon$
Output: primal (infeasible) and dual solutions $x$ and $l$

Initialize $l(e) = \delta\ \forall e,\ x \equiv 0$.
while there is a $P \in \mathcal{P}$ with $l(P) < 1$
    Select a $P \in \mathcal{P}$ with $l(P) < 1$
    $u \leftarrow \min_{e \in P} u(e)$
    $x(P) \leftarrow x(P) + u$
    $\forall e \in P,\ l(e) \leftarrow l(e)(1 + \frac{\epsilon u}{u(e)})$
end while
Return $(x, l)$.
```

FIG. 2.1. *Generic algorithm for maximum multicommodity flow*

After updating $x(P)$, the dual variables are updated so that the length of an edge is exponential in 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 $(1 + \epsilon)$. We refer to this algorithm as the *generic algorithm.* It is summarized in Figure 2.1.

At any point in the algorithm we can also find the most violated dual constraint (via shortest path computations using lengths $l(e)$) and scale the dual solution so that it is feasible for **D** by multiplying all variables by the proportion of violation. Thus, at the end of every iteration we have implicit primal and dual feasible solutions. While we use dual feasibility as a termination criterion, this is done only for simplicity of our arguments, and is not required for correctness of the algorithm. In practice, it would make sense to keep track of the best dual solution encountered in the algorithm, and terminate the algorithm when the ratio between this and the best primal solution is at most $1 + \epsilon$. Our analysis will show that this always happens by the time the length of every path in $\mathcal{P}$ is at least one. (See Proof of Theorem 2.4 and the ensuing discussion for further details.) The following two lemmas imply that the algorithm does not require too many iterations, and that the final primal solution is not too far from feasible.

LEMMA 2.1. *After $\mathcal{O}(m \log_{1+\epsilon} \frac{1+\epsilon}{\delta})$ augmentations, the generic algorithm terminates.*

Proof. 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 at least $1 + \epsilon$, the number of possible augmentations is at most $m \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$. □

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

Proof. 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 $1 + a\epsilon \geq (1+\epsilon)^a$ for all $0 \leq a \leq 1$, we have $\Pi_i(1 + a_i\epsilon) \geq (1 + \epsilon)^{\sum_i a_i}$, 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 $1 + \epsilon$. Initially $l(e) = \delta$ and at the end $l(e) < 1 + \epsilon$, so the total flow on edge e cannot exceed $u(e) \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$. □

The key to the efficiency of our algorithm lies in our selection of P. Garg and Könemann [10] show that if P is selected so that $P = \mathrm{argmin}_{P \in \mathcal{P}} l(P)$, where $l(P) = \sum_{e \in P} l(e)$, then the following theorem holds. We omit the proof here since our proof of Theorem 2.4 provided in the subsequent section is a straightforward modification of the proof of this theorem.

THEOREM 2.3. *If P is selected in each iteration to be the shortest (s_i, t_i) path among all commodities, then for a final flow value g_t we have that $\frac{g_t}{\log_{1+\epsilon} \frac{1+\epsilon}{\delta}} \geq (1 - 2\epsilon) OPT$.*

To determine the minimum length path in $\mathcal{P}$, it is necessary to compute a shortest path for each commodity. This takes a total of $\mathcal{O}^*(km)$ time. Choosing $\delta = (1 + \epsilon)/((1+\epsilon)n)^{1/\epsilon}$ then 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 one 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 $l(P) < \min\{1, (1 + \epsilon)\hat{\alpha}(i)\}$, 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 $(1 + \epsilon)\hat{\alpha}(i)$, and so we set $\hat{\alpha}(i + 1) = (1 + \epsilon)\hat{\alpha}(i)$. Thus, throughout the course of the algorithm, $\hat{\alpha}$ takes on values in the set $\{\delta(1 + \epsilon)^r\}_{r \in \mathcal{N}}$. Let t be the final iteration. Since $\alpha(0) \geq \delta$ and $\alpha(t - 1) < 1$, we have $\alpha(t) < 1 + \epsilon$. 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 r is $\lfloor \log_{1+\epsilon} \frac{1+\epsilon}{\delta} \rfloor$, where $\lfloor z \rfloor$ denotes the largest integer $\leq z$.).

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 current value of $\hat{\alpha}$ and 1, flow is augmented along such a shortest path. When $\min_{P \in \mathcal{P}_j} l(P)$ is at least $(1 + \epsilon)\hat{\alpha}$, commodity $j + 1$ is considered. After all k commodities are considered, we know that $\alpha(i) \geq (1 + \epsilon)\hat{\alpha}(i)$ so we update $\hat{\alpha}$ by setting $\hat{\alpha}(i + 1) = (1 + \epsilon)\hat{\alpha}(i)$. We implement this idea by defining phases determined by the values of $\hat{\alpha}$, starting with $\hat{\alpha} = \delta$ and ending with $\hat{\alpha} = \delta(1 + \epsilon)^r$ for r such that $1 \leq \delta(1 + \epsilon)^r < 1 + \epsilon$. This algorithm is presented in Figure 2.2.

Between each update to $\hat{\alpha}$, there is at most one shortest path computation per commodity that does not lead to an augmentation (For commodity j this is the computation that reveals that $\min_{P \in \mathcal{P}_j} l(P) \geq (1 + \epsilon)\hat{\alpha}$.) We charge these computations to the increase of $\hat{\alpha}$. Thus a total of at most $k \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$ shortest path computations are used to update $\hat{\alpha}$ over the course of the algorithm. The remaining shortest path computations all lead to augmentations. Lemma 2.1 enables us to bound the number

```
Input: network $G$, capacities $u(e)$, commodity pairs $(s_j, t_j)$, $1 \leq j \leq k$,
       accuracy $\epsilon$
Output: primal (infeasible) and dual solutions $x$ and $l$

    Initialize $l(e) = \delta \; \forall e, \; x \equiv 0$.
    for $r = 1$ to $\lfloor \log_{1+\epsilon} \frac{1+\epsilon}{\delta} \rfloor$
        for $j = 1$ to $k$ do
            $P \leftarrow$ shortest path in $\mathcal{P}_j$ using $l$.
            while $l(P) < \min\{1, \delta(1+\epsilon)^r\}$
                $u \leftarrow \min_{e \in P} u(e)$
                $x(P) \leftarrow x(P) + u$
                $\forall e \in P, \; l(e) \leftarrow l(e)(1 + \frac{\epsilon u}{u(e)})$
                $P \leftarrow$ shortest path in $\mathcal{P}_j$ using $l$.
            end while
    Return $(x, l)$.
```

FIG. 2.2. *FPTAS for maximum multicommodity flow. Here $\hat{\alpha}$ is represented implicitly as $\delta(1+\epsilon)^r$.*

of these computations by $\mathcal{O}^*(m \log_{1+\epsilon} \frac{1+\epsilon}{\delta})$. Using a Dijkstra shortest path algorithm, this implies a runtime of $\mathcal{O}^*(\epsilon^{-2}(m^2 + km))$ for this modified algorithm. This can be reduced to $\mathcal{O}^*(\epsilon^{-2}m^2)$ by observing that we can group commodities by a common source node, and compute shortest paths from a source node to all other nodes in the same asymptotic time as needed to compute a shortest path from a source to one specified node.

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. We choose δ so that this ratio is at least $(1 - O(\epsilon))$.

THEOREM 2.4. *An ϵ-approximate maximum multicommodity flow can be computed in $\mathcal{O}(\frac{1}{\epsilon^2} m(m + n \log m) \log n)$ 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. Let g_i be the primal objective function value at the end of iteration i.

For each iteration $i \geq 1$,

$$\begin{aligned} D(i) = \sum_e l_i(e)u(e) &= \sum l_{i-1}(e)u(e) + \epsilon \sum_{e \in P} l_{i-1}(e)u \\ &\leq D(i-1) + \epsilon(g_i - g_{i-1})(1+\epsilon)\alpha(i-1), \end{aligned} \tag{2.1}$$

which implies that

$$D(i) \leq D(0) + \epsilon(1+\epsilon) \sum_{j=1}^{i} (g_j - g_{j-1})\alpha(j-1) \tag{2.2}$$

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 δn. Since this holds for the shortest path using length function $l_i - l_0$, we have that $\alpha(l_i - l_0) \geq \alpha(i) - \delta n$. Hence

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

Substituting this bound on $D(i) - D(0)$ in equation (2.2) gives

$$\alpha(i) \leq \delta n + \frac{\epsilon(1+\epsilon)}{\beta} \sum_{j=1}^{i} (g_j - g_{j-1})\alpha(j-1).$$

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

$$\begin{aligned}\alpha(i) \leq \alpha'(i) &= \alpha'(i-1)(1 + \epsilon(1+\epsilon)(g_i - g_{i-1})/\beta) \\ &\leq \alpha'(i-1)e^{\epsilon(1+\epsilon)(g_i - g_{i-1})/\beta},\end{aligned}$$

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

$$\alpha(i) \leq \delta n e^{\epsilon(1+\epsilon)g_i/\beta} \tag{2.3}$$

By the stopping condition, in the final iteration t, we have

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

and hence

$$\frac{\beta}{g_t} \leq \frac{\epsilon(1+\epsilon)}{\ln(\delta n)^{-1}}. \tag{2.4}$$

Let γ be the ratio of the dual and primal solutions. $\gamma := \frac{\beta}{g_t}\log_{1+\epsilon}\frac{1+\epsilon}{\delta}$. By substituting the bound on β/g_t from (2.4), we obtain

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

Letting $\delta = (1+\epsilon)((1+\epsilon)n)^{-1/\epsilon}$, 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 imply the run time. □

Above, we have shown that the framework of Garg and Könemann [10] 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 [15]. This algorithm requires that the shortest path subproblem for each commodity be solved (albeit approximately) at each iteration.

This algorithm also finds ϵ-approximate dual solutions. Above, we have replaced D/α in (2.2), with the exact optimal dual solution β, and we have shown that (2.4) holds for this value. However, we can replace β throughout the proof with β', the best (lowest) value of $D(i)/\alpha(i)$ encountered in the algorithm, and we get the same bound for $\frac{\beta'}{g_t}$ as we have shown for $\frac{\beta}{g_t}$. This value β' corresponds to the dual solution $l_i/\alpha(i)$, which is feasible by definition of α. Thus our proof of ϵ-optimality for the primal solution really is a proof that the final scaled primal solution is within $1/(1+4\epsilon)$ of the best dual solution obtained during the algorithm. As noted earlier, instead of terminating the algorithm when $\min_{P\in\mathcal{P}_j} l(P) \geq 1$, we can terminate the algorithm once we have this proof.

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. Some Practical Modifications. We discuss briefly two aspects of the above analysis that could impact practical performance.

1. For large graphs and small ϵ, our expression for δ may be too small to implement easily. This is avoidable by choosing a larger value of δ, and modifying the termination criterion appropriately. Instead of showing that the algorithm will terminate by the time the length of the shortest path is at least 1, for any δ, it follows easily from the analysis that the algorithm will terminate by the time the length of the shortest path is at least $q(\delta) := \frac{\delta}{1+\epsilon}[n(1+\epsilon)]^{1/\epsilon}$. The number of augmentations remains $O(m\frac{1}{\epsilon^2}\log n)$. This also implies that if the lengths of edges become too large during the course of the algorithm, they can be scaled down without affecting the run time or accuracy of the algorithm.

2. Above we have described an algorithm that maintains a variable $l(e)$ for each edge, and modifies $l(e)$ whenever the flow on edge e is increased by $u \leq u(e)$ by multiplying it by $(1+\frac{u}{u(e)}\epsilon)$. This is not the only update that will result in the approximation guarantees described here. Another example that will work is to update $l(e)$ by multiplying by $e^{\epsilon u/u(e)}$. With this update, $l(e) = \delta e^{\epsilon x(e)/u(e)}$ at any point during the algorithm. Thus, scaling by $\log_{e^\epsilon} \frac{\max|l(e)|}{\delta}$ yields a feasible primal solution at any point in the algorithm, and the total number of augmentations is at most $O(m \log_{e^\epsilon}(q(e)e^\epsilon/\delta))$. To get an approximation guarantee, note that $e^a \leq 1+a+a^2$ for $0 \leq a \leq 1$. Then, the increase in $u(e)l(e)$ in one iteration is bounded from above by $\epsilon u + \epsilon^2 u\frac{u}{u(e)} \leq \epsilon(1+\epsilon)u$ as long as $u \leq u(e)$. Substituting this in the equation (2.1) turns it into an inequality, and results in a modification of inequality (2.3) to $\alpha(i) \leq \delta n e^{\epsilon(1+\epsilon)^2 g_i/\beta}$. This then implies that for an appropriate termination point that $\gamma \leq \left(\frac{1+\epsilon}{1-\epsilon}\right)^2$, which is at most $1+16\epsilon$ for small enough ϵ.

In fact, comparing this analysis with the second order Taylor series expansion reveals that we may use as the update factor $\phi(u)$, any convex, increasing function on $[0, u(e)]$ that satisfies $\phi(0) = 1$, $\phi'(0) = \epsilon/u(e)$, and $\phi''(0) \geq 0$. The theory shows that the number of iterations may depend on $\phi''(0)$. Experiments comparing $1+\epsilon\frac{u}{u(e)}$ with $e^{\epsilon\frac{u}{u(e)}}$ indicate that the latter update performs better [2].

2.3. 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 $W := \max_j w_j$ is the maximum ratio of weights of any two commodities. The resulting change in the dual problem is that the path constraints are now of the form $\sum_{e\in P\subset\mathcal{P}_j} l(e) \geq w_j$ for all $P \in \cup_j \mathcal{P}_j$.

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 defining $\overline{\alpha}(i) := \min_{1\leq j\leq k} \min_{P\in\mathcal{P}_j} \frac{l(P)}{w_j}$ and substituting this for α. As before, the algorithm terminates when $\overline{\alpha} \geq 1$. For this modified algorithm, we choose $\delta = (1+\epsilon)W/((1+\epsilon)nW)^{1/\epsilon}$. The rest of the algorithm remains unchanged. The analysis of the algorithm changes slightly, now that we are using $\overline{\alpha}$. The initial value of $\overline{\alpha}$ can be as low as δ/W, and the final value of $\overline{\alpha}$ is at most $(1+\epsilon)$. Hence, the number times $\hat{\alpha}$ can increase by a factor of $(1+\epsilon)$ is $\log_{1+\epsilon}\frac{(1+\epsilon)W}{\delta}$. The length of an edge starts at δ and can be as large as $(1+\epsilon)W$, so the number of iterations is at most $m\log_{1+\epsilon}\frac{(1+\epsilon)W}{\delta}$, and the scale factor to make the final solution primal feasible is at most $\log_{1+\epsilon}\frac{(1+\epsilon)W}{\delta}$. Choosing $\delta = (1+\epsilon)W/((1+\epsilon)nW)^{1/\epsilon}$ implies a run time of $\mathcal{O}^*(\epsilon^{-2}m^2\log W)$.

To establish that this modified algorithm finds an ϵ-approximate solution, we follow the analysis in the proof of Theorem 2.4 with the following modifications: let h_i be the value of the primal objective function at the end of iteration i, and let $D(i)$ be the dual objective function value, as before. For iteration $i \geq 1$ we have that

$$\begin{aligned} D(i) = \sum_e l_i(e)u(e) &= \sum l_{i-1}(e)u(e) + \epsilon\sum_{e\in P} l_{i-1}(e)u \\ &\leq D(i-1) + \epsilon(1+\epsilon)w_j\overline{\alpha}(i-1)\frac{h_i - h_{i-1}}{w_j} \\ &\leq D(i-1) + \epsilon(1+\epsilon)\overline{\alpha}(i-1)(h_i - h_{i-1}) \end{aligned}$$

Symmetric to the case without weights, a dual solution l_i is made dual feasible by dividing by $\overline{\alpha}(l_i)$. Since the bounds on $D(l_i - l_0)$ and $\overline{\alpha}(l_i - l_0)$ remain valid, the same analysis goes through to obtain the following bound on γ, the ratio of primal and dual solutions obtained by the algorithm:

$$\gamma \leq \frac{\epsilon(1+\epsilon)}{\ln(1+\epsilon)}\frac{\ln\frac{(1+\epsilon)W}{\delta}}{\ln(n\delta)^{-1}}.$$

With the above choice of δ, this is at most $(1+4\epsilon)$ for $\epsilon < .15$.

The strongly polynomial run time follows directly from the analysis of the packing LP algorithm in [10]. □

3. Maximum concurrent flow.

Recall the maximum concurrent flow problem has specified demands d_j for each commodity $1 \leq j \leq k$ and the problem is to find a flow that maximizes the ratio of met demands.

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.

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

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.

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

When the objective function value is ≥ 1, Garg and Könemann [10] 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, which increases the run time of their algorithm so that it matches previously known algorithms. We describe a different procedure that requires k maximum bottleneck path computations which can be performed in $\mathcal{O}(m \log m)$ time each. Since the total time spent on this new procedure no longer dominates the time to solve the scaled problem with objective function value ≥ 1, the resulting algorithm solves the maximum concurrent flow problem in $\mathcal{O}^*(\epsilon^{-2} m(m+k))$ time.

For the cost bounded problem, Garg and Könemann [10] use a minimum cost flow subroutine. We use a cost bounded maximum bottleneck path subroutine, 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 [10]. 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 bottleneck 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: $D(l) := \sum_e u(e)l(e) \geq 1$. See Figure 3.1 for a summary of the algorithm. Garg and Könemann [10] prove the following sequence of lemmas, for $\delta = (\frac{m}{1-\epsilon})^{-1/\epsilon}$. Here, β is the optimal objective function value.

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.*

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 relies 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

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

   Initialize $l(e) = \delta/u(e)\ \forall e,\ x \equiv 0$.
   while $D(l) < 1$
      for $j = 1$ to $k$ do
         $d'_j \leftarrow d_j$
         while $D(l) < 1$ and $d'_j > 0$
            $P \leftarrow$ shortest path in $\mathcal{P}_j$ using $l$
            $u \leftarrow \min\{d'_j, \min_{e \in P} u(e)\}$
            $d'_j \leftarrow d'_j - u$
            $x(P) \leftarrow x(P) + u$
            $\forall e \in P,\ l(e) \leftarrow l(e)(1 + \frac{\epsilon u}{u(e)})$
         end while
   end while
   Return $(x, l)$.
```

FIG. 3.1. *FPTAS for max concurrent flow*

maximum 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 [23] that is also used in [10]. 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 $\log k$ 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. In fact, by grouping commodities by their common source node, we can compute all such paths in $\mathcal{O}(\min\{k, n\} m \log m)$ 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 [10] and follows a similar analysis as for the maximum concurrent flow algorithm.

Again, we improve on the run time in [10] 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.*

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

REFERENCES

[1] *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1995.
[2] C. Albrecht. Provably good global routing by a new approximation algorithm for multicommodity flow. In *Proceedings of the International Conference on Physical Design (ISPD)*, pages 19–25, San Diego, CA, 2000. ACM.

[3] Y. Aumann and Y. Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM J. Comput.*, 27(1):291–301, February 1998.

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

[5] D. Bienstock. An implementation of the exponential potential reduction method for general linear programs. Working paper, 1999.

[6] D. Bienstock. Approximation algorithms for linear programming: Theory and practice. Survey in preparation., July 2000.

[7] L. K. Fleischer. Approximating fractional multicommodity flows independent of the number of commodities. In *40th Annual IEEE Symposium on Foundations of Computer Science*, pages 24–31, 1999.

[8] L. Fratta, M. Gerla, and L. Kleinrock. The flow deviation method: an approach to store-and-forward communication network design. *Networks*, 3:97–133, 1973.

[9] N. Garg, January 1999. Personal communication.

[10] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *39th Annual IEEE Symposium on Foundations of Computer Science*, pages 300–309, 1998.

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

[12] 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.

[13] 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.

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

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

[16] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.

[17] 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 18–25, 1995.

[18] 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.

[19] 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.

[20] 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.

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

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

[23] 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.

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

[25] M. Sato. Efficient implementation of an approximation algorithm for multicommodity flows. Master's thesis, Division of Systems Science, Graduate School of Engineering Science, Osaka University, February 2000.

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

[27] 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.

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

This research was sponsored in part by National Science Foundation (NSF) grant no. CCR-0122581.