Faster and Simpler Algorithms for Multicommodity Flow and other Fractional Packing Problems

Naveen Garg*
Computer Science and Engineering
Indian Institute of Technology, New Delhi, India

Jochen Könemann†
GSIA, Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

This paper considers the problem of designing fast, approximate, combinatorial algorithms for multicommodity flows and other fractional packing problems. We provide a different approach to these problems which yields faster and much simpler algorithms. Our approach also allows us to substitute shortest path computations for min-cost flow computations in computing maximum concurrent flow and min-cost multicommodity flow; this yields much faster algorithms when the number of commodities is large.

1. Introduction

Consider the problem of computing a maximum s-t flow in a graph with unit edge capacities. While there are many different algorithms known for this problem we discuss one which views the problem purely as one of packing s-t paths so that constraints imposed by edge-capacities are not violated. The algorithm associates a length with each edge and at any step it routes a unit flow along the shortest s-t path. It then multiplies the length of every edge on this path by $1+\epsilon$ for a fixed ϵ. Thus the longer an edge is the more is the flow through it. Since we always choose the shortest s-t path to route flow along, we essentially try to balance the flow on all edges in the graph. One can argue that, if, after sufficiently many steps, M is the maximum flow through an edge, then the flow computed is almost M times the maximum s-t flow. Therefore scaling the flow by M gives a feasible flow which is almost maximum.

Note that the length of an edge at any step is exponential in the total flow going through the edge. Such a length function was first proposed by Shahrokhi and Matula [13] who used it to compute the throughput of a given multicommodity flow instance. While this problem (and all other problems considered in this paper) can be formulated as a linear program and solved to optimality using fast matrix multiplication [16], in [13], people were mainly interested in providing fast, possibly approximate, combinatorial algorithms. Their procedure, which applied only to the case of uniform edge capacities, computed a $(1+\omega)$-approximation to the maximum throughput in time polynomial in ω^{-1}. The key idea of their procedure, which was adopted in a lot of subsequent work, was to compute an initial flow by disregarding edge capacities and then to reroute this, iteratively, along short paths so as to reduce the maximum congestion on any edge.

The running time of [13] was improved significantly by Klein *et.al.* [9]. It was then extended and refined to the case of arbitrary edge capacities by Leighton *et.al.* [10], Goldberg [4] and Radzik [12] to obtain better running times; see Table 1 for the current best bound.

Plotkin, Shmoys and Tardos [11] and Grigoriadis and Khachiyan [7] observed that a similar technique could be applied to solve any fractional packing or covering problem. Their approach, for packing problems, starts with an infeasible solution. The amount by which a packing constraint is violated is captured by a variable which is exponential in the extent of this violation. At any step the packing is modified by a *fixed amount* in a direction determined by these variables. Hence, the running time of the procedure depends upon the maximum extent to which any constraint could be violated; this is referred to as the *width* of the problem [11]. The running time of their algorithm for packing problems being only pseudo-polynomial, [11] suggest different ways of reducing the width of the problem.

In a significant departure from this line of research and motivated by ideas from randomized rounding, Young [17] proposed an *oblivious rounding* approach to packing problems. Young's approach has the essential ingredient of previous approaches — a length function which measures, and is exponential in, the extent to which each constraint is vio-

*Supported by the EU ESPRIT LTR Project N. 20244 (ALCOM-IT). Work done while the author was at the Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany.

†Work done while the author was at the Universität des Saarlandes, Im Stadtwald, 66123 Saarbrücken, Germany.

lated by a given solution. However, [17] builds the solution from scratch and at each step adds to the packing a variable which violates only such packing constraints that are not already too violated. In particular, for multicommodity flow, it implies a procedure which does not involve rerouting flow (the flow is only scaled at the end) and which for the case of maximum s-t flow reduces to the algorithm discussed at the beginning of this section.

Our Contributions. In this paper we provide a unified framework for a host of multicommodity flow and packing problems which yields significantly simpler and faster algorithms than previously known. Our approach is similar to Young's approach for packing problems. However, we develop a new and simple combinatorial analysis which has the added flexibility that it allows us to make the greatest possible advance at each step. Thus for the setting of maximum s-t flows with integral edge capacities, Young's procedure routes a unit flow at each step while our procedure would route enough flow so as to saturate the minimum capacity edge on the shortest s-t path. This simple modification is surprisingly powerful and delivers better running times and simpler proofs. In particular, it lets us argue that *the contribution of a constraint to the running time of the procedure cannot exceed a certain bound which is independent of the width.* This yields a new and straightforward strongly-polynomial combinatorial approximation algorithm for the fractional packing problem (Section 3). The earlier algorithm for this problem, due to Grigoriadis and Khachiyan [5] reduced the problem to two resource sharing problems.

Our approach yields a new, very natural, algorithm for maximum concurrent flow (Section 5) which extends in a straightforward manner to min-cost multicommodity flows (Section 6). Both these algorithms use a min-cost flow computation as a subroutine as do all earlier algorithms. Contradicting popular belief that using min-cost flow as a subroutine is better, we provide algorithms for these two problems which use shortest path computations as a subroutine and are faster than previous algorithms by at least a factor $\min\left\{n, \frac{nk\log k}{m}\right\}$ where k, m, n are the number of commodities, edges and vertices respectively.

Table 1 summarizes our results. $T_{\rm sp}$ and $T_{\rm mcf}$ are the times to compute single-source shortest paths and single-commodity min-cost flow in a graph with positive edge lengths and costs while $T_{\rm orc}$ is the time taken for each call to an oracle as in [11]. All our algorithms are deterministic and compute a $(1+\omega)$-approximation to the optimum solution. For brevity we define $C_1 \stackrel{\rm def}{=} \lceil \frac{1}{\epsilon_1} \log_{1+\epsilon_1} m \rceil$ where $(1-\epsilon_1)^{-2} = 1+\omega$ and $C_2 \stackrel{\rm def}{=} \lceil \frac{1}{\epsilon_2} \log_{1+\epsilon_2} \frac{m}{1-\epsilon_2} \rceil$ where $(1-\epsilon_2)^{-3} = 1+\omega$.

Note that in the running time of our algorithms for concurrent flow problems we can replace $C_2 \log k$ by $O(\log n(\omega^{-2} + \log k))$ using a trick from earlier papers; we remark on this in Section 5.

2. Maximum multicommodity flow

Given a graph $G = (V, E)$ with edge capacities $c : E \rightarrow \mathbf{R}^+$ and k pairs of terminals (s_i, t_i), with one commodity associated with each pair, we want to find a multicommodity flow such that the sum of the flows of all commodities is maximized. The dual of the maximum multicommodity flow problem is an assignment of lengths $l : E \rightarrow \mathbf{R}^+$ to the edges such that $D(l) \stackrel{\rm def}{=} \sum_e l(e)c(e)$ is minimized. This is subject to the constraint that the shortest path between any pair s_i, t_i under the length function l, which we denote by $\mathtt{dist}_i(l)$, is at least one. Let $\alpha(l) \stackrel{\rm def}{=} \min_i \mathtt{dist}_i(l)$ be the minimum length path between any pair of terminals. Then the dual problem is equivalent to finding a length function $l : E \rightarrow \mathbf{R}^+$ such that $\frac{D(l)}{\alpha(l)}$ is minimized. Let $\beta \stackrel{\rm def}{=} \min_l D(l)/\alpha(l)$.

The algorithm proceeds in iterations. Let l_{i-1} be the length function at the beginning of the $i^{\rm th}$ iteration and f_{i-1} be the total flow routed in iterations $1 \ldots i-1$. Let P be a path of length $\alpha(l_{i-1})$ between a pair of terminals and let c be the capacity of the minimum capacity edge on P. In the $i^{\rm th}$ iteration we route c units of flow along P. Thus $f_i = f_{i-1} + c$. The function l_i differs from l_{i-1} only in the lengths of the edges along P; these are modified as $l_i(e) = l_{i-1}(e)(1 + \epsilon c/c(e))$, where ϵ is a constant to be chosen later.

Initially every edge e has length δ, *ie.*, $l_0(e) = \delta$ for some constant δ to be chosen later. For brevity we denote $\alpha(l_i), D(l_i)$ by $\alpha(i), D(i)$ respectively. The procedure stops after t iterations where t is the smallest number such that $\alpha(t) \geq 1$.

2.1. Analysis

For every iteration $i \geq 1$

$$\begin{aligned} D(i) &= \sum_e l_i(e)c(e) \\ &= \sum_e l_{i-1}(e)c(e) + \epsilon \sum_{e \in P} l_{i-1}(e)c \\ &= D(i-1) + \epsilon(f_i - f_{i-1})\alpha(i-1) \end{aligned}$$

which implies that

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

Problem	Previous Best	Our running time	Improvement
Max. multicomm. flow	O($\omega^{-3}m^2\log m$) [14]	mC_1kT_{sp}	ω^{-1}
Fractional Packing	[5]	mC_1T_{orc}	
Spreading metrics	$O(\omega^{-3}nm\log nT_{\mathrm{sp}})$ [3]	$mC_1(nT_{\mathrm{sp}})$	ω^{-1}
Maximum concurrent flow	$O(k(\omega^{-2}+\log k)\log nT_{\mathrm{mcf}})$ [12, 10]	$(2k\log k)C_2T_{\mathrm{mcf}}$	In constants
		$(2k\log k+m)C_2T_{\mathrm{sp}}$	For $k\geq m/n$
Max. cost-bounded concurrent flow	$O(k(\omega^{-2}+\log k)\log n\log(\omega^{-1}k)$ $T_{\mathrm{mcf}})$ [6]	$(2k\log k+1)C_2T_{\mathrm{mcf}}$	$\log(\omega^{-1}k)$
		$(2k\log k+m+1)C_2T_{\mathrm{sp}}$	For $k\geq m/n$

Table 1. A summary of our results

Consider the length function l_i-l_0. Note that $D(l_i-l_0)=D(i)-D(0)$ and $\alpha(l_i-l_0)\geq\alpha(i)-\delta L$ where L is the maximum number of edges on path along which flow is routed. 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) we get

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

Note that for any i, $\alpha(i)$ is maximum when for all $j, 0\leq j\leq i-1, \alpha(j)$ is the largest possible. Hence

$$\begin{aligned}\alpha(i) &\leq \alpha(i-1)(1+\epsilon(f_i-f_{i-1})/\beta)\\ &\leq \alpha(i-1)e^{\epsilon(f_i-f_{i-1})/\beta}\end{aligned}$$

Since $\alpha(0)\leq\delta L$ this implies

$$\alpha(i)\leq\delta Le^{\epsilon f_i/\beta}$$

By our stopping condition

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

and hence

$$\frac{\beta}{f_t}\leq\frac{\epsilon}{\ln(\delta L)^{-1}}\qquad(2)$$

Claim 2.1 *There is a feasible flow of value* $\frac{f_t}{\log_{1+\epsilon}\frac{1+\epsilon}{\delta}}$

Proof: Consider an edge e. For every $c(e)$ units of flow routed through e the length of e increases by a factor of at least $1+\epsilon$. The last time its length was increased, e was on a path of length strictly less than 1. Since every increase in edge-length is by a factor of at most $1+\epsilon$, $l_t(e)<1+\epsilon$. Since $l_0(e)=\delta$ it follows that the total flow through e is at most $c(e)\log_{1+\epsilon}\frac{1+\epsilon}{\delta}$. Scaling the flow, f_t, by $\log_{1+\epsilon}\frac{1+\epsilon}{\delta}$ then gives a feasible flow of claimed value. ■

Thus the ratio of the values of the optimum dual and the primal solutions, γ, is $\frac{\beta}{f_t}\log_{1+\epsilon}\frac{1+\epsilon}{\delta}$. By substituting the bound on β/f_t from (2) we obtain

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

The ratio $\frac{\ln(1+\epsilon)\delta^{-1}}{\ln(\delta L)^{-1}}$ equals $(1-\epsilon)^{-1}$ for $\delta=(1+\epsilon)((1+\epsilon)L)^{-1/\epsilon}$. Hence with this choice of δ we have

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

Since this quantity should be no more than our approximation ratio $(1+w)$ we choose ϵ appropriately.

2.2. Running time

In the i^{th} iteration we increase the length of the minimum capacity edge along P by a factor of $1+\epsilon$. Since for any edge $e, l_0(e)=\delta$ and $l_t(e)<1+\epsilon$ the number of iterations in which e is the minimum capacity edge on the path chosen in that iteration is at most $\lceil\frac{1}{\epsilon}\log_{1+\epsilon}L\rceil$. Using the fact that there are m edges we get the following theorem.

Theorem 2.1 *There is an algorithm that computes a $(1-\epsilon)^{-2}$-approximation to the maximum multicommodity flow in time $m\lceil\frac{1}{\epsilon}\log_{1+\epsilon}L\rceil kT_{\mathrm{sp}}$ where L is the maximum number of edges on a path between any source-sink pair and T_{sp} is the time required to compute the shortest s-t path in a graph with non-negative edge-weights.*

3. Packing LP

A packing **LP** is a linear program of the kind $\max\{c^Tx|Ax\leq b, x\geq 0\}$ where A, b and c are $(m\times$

$n), (m \times 1)$ and $(n \times 1)$ matrices all of whose entries are positive. We also assume that for all i, j, the $(i, j)^{\text{th}}$ entry of A, $A(i, j)$, is at most $b(i)$. The dual of this **LP** is $\min \{b^T y | A^T y \geq c, y \geq 0\}$.

We view the rows of A as edges and the columns as paths. $b(i)$ is the capacity of edge i and every unit of flow routed along the j^{th} column consumes $A(i, j)$ units of capacity of edge i while providing a benefit of $c(j)$ units.

The dual variable $y(i)$ corresponds to the length of edge i. Define the *length* of a column j with respect to the dual variables y as $\texttt{length}_y(j) \stackrel{\text{def}}{=} \sum_i A(i, j) y(i)/c(j)$. Finding a shortest path now corresponds to finding a column whose length is minimum; define $\alpha(y) \stackrel{\text{def}}{=} \min_j \texttt{length}_y(j)$. Also define $D(y) \stackrel{\text{def}}{=} b^T y$. Then the dual program is equivalent to finding a variable assignment y such that $D(y)/\alpha(y)$ is minimized.

Once again our procedure will be iterative. Let y_{k-1} be the dual variables and f_{k-1} the value of the primal solution at the beginning of the k^{th} iteration. Let q be the minimum length column of A *ie.*, $\alpha(y_{k-1}) = \texttt{length}_{y_{k-1}}(q)$ — this corresponds to the path along which we route flow in this iteration. The minimum capacity edge is the row for which $b(i)/A(i, q)$ is minimum; let this be row p. Thus in this iteration we will increase the primal variable $x(q)$ by an amount $b(p)/A(p, q)$ so that $f_k = f_{k-1} + c(q) b(p)/A(p, q)$. The dual variables are modified as

$$y_k(i) = y_{k-1}(i)(1 + \epsilon \frac{b(p)/A(p, q)}{b(i)/A(i, q)})$$

where ϵ is a constant to be chosen later.

The initial values of the dual variables are given by $y_0(i) = \delta/b(i)$, for some constant δ to be chosen later. For brevity we denote $\alpha(y_k), D(y_k)$ by $\alpha(k), D(k)$ respectively. Thus $D(0) = m\delta$. The procedure stops at the first iteration t such that $D(t) \geq 1$.

3.1. Analysis

The analysis here proceeds almost exactly as in the case of maximum multicommodity flow. For every iteration $k \geq 1$

$$\begin{aligned} D(k) &= \sum_i b(i) y_k(i) \\ &= \sum_i b(i) y_{k-1}(i) + \epsilon \frac{b(p)}{A(p, q)} \sum_i A(i, q) y_{k-1}(i) \\ &= D(k-1) + \epsilon (f_k - f_{k-1}) \alpha(k-1) \end{aligned}$$

which, as before, implies that

$$D(k) = D(0) + \epsilon \sum_{l=1}^{k} (f_l - f_{l-1}) \alpha(l-1)$$

Let $\beta \stackrel{\text{def}}{=} \min_y D(y)/\alpha(y)$. Then $\beta \leq D(l-1)/\alpha(l-1)$ and so

$$D(k) \leq m\delta + \frac{\epsilon}{\beta} \sum_{l=1}^{k} (f_l - f_{l-1}) D(l-1)$$

which implies that

$$D(k) \leq m\delta e^{\epsilon f_k/\beta}$$

By our stopping condition

$$1 \leq D(t) \leq m\delta e^{\epsilon f_t/\beta}$$

and hence

$$\frac{\beta}{f_t} \leq \frac{\epsilon}{\ln(m\delta)^{-1}}$$

Claim 3.1 *There is a feasible solution to the packing* **LP** *of value* $\frac{f_t}{\log_{1+\epsilon} \frac{1+\epsilon}{\delta}}$

Proof: The primal solution x we constructed has value f_t. However, it may not be feasible since some packing constraint $(\sum_j A(i, j) x(j))/b(i) \leq 1$ may be violated. When we pick column q and increase $x(q)$ by $b(p)/A(p, q)$ we increase the left-hand-side (LHS) of the i^{th} constraint by $\frac{A(i,q) b(p)}{b(i) A(p,q)}$ $(= z$ say). Simultaneously we increase the dual variable $y(i)$ by a multiplicative factor of $1 + \epsilon z$. By our definition of p it follows that $z \leq 1$ and hence increasing the LHS of the i^{th} constraint by 1 causes an increase in $y(i)$ by a multiplicative factor of at least $1 + \epsilon$. Note that $y_{t-1}(i) < 1/b(i)$ and so $y_t(i) < (1 + \epsilon)/b(i)$. Since $y_0(i) = \delta/b(i)$ it follows that the final value of the LHS of the i^{th} constraint is no more than $\log_{1+\epsilon} \frac{1+\epsilon}{\delta}$. Since this is true for every i, scaling the primal solution by $\log_{1+\epsilon} \frac{1+\epsilon}{\delta}$ gives a feasible solution of value as in the claim. ■

The rest of the analysis is exactly the same as in Section 2.1 with m replacing L. Thus $\delta = (1 + \epsilon)((1 + \epsilon) m)^{-1/\epsilon}$.

3.2. Running time

In the k^{th} iteration we increase the dual variable of the "minimum capacity" row by a factor of $(1 + \epsilon)$. Since for any row i, $y_0(i) = \delta/b(i)$ and $y_t(i) < (1 + \epsilon)/b(i)$ and there are m rows in all, the total number of iterations is at most $m \lceil \frac{1}{\epsilon} \log_{1+\epsilon} m \rceil$.

Theorem 3.1 *There is an algorithm that computes a* $(1 - \epsilon)^{-2}$*-approximation to the Packing* **LP** *in time* $m \lceil \frac{1}{\epsilon} \log_{1+\epsilon} m \rceil T_{\text{orc}}$ *where* m *is the number of rows and* T_{orc} *is the time required to compute the minimum length column.*

4. Spreading metrics

Given a graph $G = (V, E)$ with edge costs $c : E \to \mathbf{R}^+$, a spreading metric is an assignment of lengths to the edges, $l : E \to \mathbf{R}^+$, so as to minimize $\sum_e l(e)c(e)$ subject to the constraint that for any set $S \subseteq V$ and vertex $r \in S$, $\sum_{v\in S} \mathtt{dist}_{r,v}(l) \geq f(S)$ where $\mathtt{dist}_{r,v}(l)$ is the distance from r to v under the length function l and $f()$ is a function only of the size of S. For the linear arrangement problem $f(S) = (|S|-1)(|S|-3)/4$ [2] while for the problem of computing a ρ-separator[1] $f(S)$ is defined as $|S| - \rho|V|$ [3].

Since the length function l is positive, the shortest paths from r to the other vertices in S forms a tree — the shortest path tree rooted at r. Thus the above constraints can be equivalently stated as: for any tree T, for any subset S of vertices in T and for any vertex $r \in S$

$$\sum_{v\in S} \mathtt{dist}_{r,v}(l,T) \geq f(S)$$

where $\mathtt{dist}_{r,v}(l,T)$ denotes the distance from r to v in tree T under the length function l.

Let $u_e(T,S,r)$ be the number of vertices of S in the subtree below edge e when T is rooted at r. Then the above constraint can be rewritten again to obtain the **LP**

$$\begin{array}{lrcl} \text{minimize} & \sum_e l(e)c(e) & & \\ \text{subject to} & & & \\ \forall T, \forall S \subseteq T, \forall r \in S & \sum_{e\in T} l(e)u_e(T,S,r) & \geq & f(S) \end{array}$$

The primal program, which is a packing **LP**, has a non-negative variable $x(T,S,r)$ for every tree T, subset $S \subseteq T$ and vertex $r \in S$ and is as follows

$$\begin{array}{lrcl} \text{maximize} & \sum_{T,S,r} x(T,S,r)f(S) & & \\ \text{subject to} & & & \\ \forall e \in E & \sum_{T:e\in T} x(T,S,r)u_e(T,S,r) & \leq & c(e) \end{array}$$

Note that the packing **LP** has exponentially many variables. However, the $(1+w)$-approximation to the optimum fractional solution, in the previous Section, only needed an oracle that returned the "most violated constraint" of the dual **LP**. In this setting, this oracle is a subroutine, which, given a length function l finds a triple (T,S,r) for which $(\sum_{e\in T} l(e)u_e(T,S,r))/f(S)$, or equivalently $(\sum_{v\in S} \mathtt{dist}_{r,v}(l,T))/f(S)$, is minimum.

Our subroutine will try out all n choices for vertex r and for each of these it will determine the best choice of T, S. For a given r and every subset S, the expression $\sum_{v\in S} \mathtt{dist}_{r,v}(l,T)$ is minimized when T is the tree of shortest paths from r and under the length function l. Therefore, for a given r, our choice of T will be the shortest path tree rooted at r. Since $f(S)$ depends only on $|S|$, given that $|S| = k$, the ratio $(\sum_{v\in S} \mathtt{dist}_{r,v}(l,T))/f(S)$ is minimized when S is the set of k nearest vertices to r. Amongst the n different choices for k, and hence for S, we choose the set for which the above ratio is minimum.

[1] a minimum cost set of edges whose removal disconnects the graph into connected components each of which has at most $\rho|V|$ vertices.

The subroutine thus requires n single-source shortest path computations. The running time of the procedure is obtained by noting that the subroutine is invoked once in each of the $m\lceil \frac{1}{\epsilon}\log_{1+\epsilon} m\rceil$ iterations.

Theorem 4.1 *There is an algorithm that computes a $(1-\epsilon)^{-2}$-approximation to Spreading metrics in time $m\lceil \frac{1}{\epsilon}\log_{1+\epsilon} m\rceil nT_{\rm sp}$ where $T_{\rm sp}$ is the time required to compute single-source shortest paths in a graph with non-negative edge-weights.*

5. Maximum concurrent flow

Once again we are given a graph with edge capacities $c : E \to \mathbf{R}^+$ and k commodities with s_i, t_i being the source, sink for commodity i. Now each commodity has a demand $d(i)$ associated with it and we want to find the largest λ such that there is a multicommodity flow which routes $\lambda d(i)$ units of commodity i.

Let $\mathtt{min\_cost}_j(l)$ be the minimum cost of shipping $d(j)$ units of flow from s_j to t_j where $l(e)$ is the cost of shipping one unit of flow along edge e and the total flow through e is at most $c(e)$. Further let $\alpha(l) \stackrel{\text{def}}{=} \sum_{j=1}^k \mathtt{min\_cost}_j(l)$. The dual problem now is an assignment of lengths to the edges, $l : E \to \mathbf{R}^+$, such that $D(l)/\alpha(l)$ is minimized. Let β be this minimum. For now we assume that $\beta \geq 1$ and shall remove this assumption later.

The algorithm now proceeds in phases; each phase is composed of k iterations. Consider the $j^{\rm th}$ iteration of the $i^{\rm th}$ phase and let $l_{i,j-1}$ be the length function before this iteration. In this iteration we route $d(j)$ units of commodity j along the paths given by $\mathtt{min\_cost}_j(l_{i,j-1})$. Let $f_{i,j}(e)$ be the flow through edge e. The length function is modified as $l_{i,j}(e) = l_{i,j-1}(e)(1 + \epsilon f_{i,j}(e)/c(e))$. Then

$$\begin{aligned} D(l_{i,j}) &= \sum_e l_{i,j}(e)c(e) \\ &= D(l_{i,j-1}) + \epsilon\sum_e l_{i,j-1}(e)f_{i,j}(e) \\ &= D(l_{i,j-1}) + \epsilon \cdot \mathtt{min\_cost}_j(l_{i,j-1}) \end{aligned}$$

The lengths at the start of the $(i+1)^{\rm th}$ phase are the same as that at the end of the $i^{\rm th}$ phase, *ie.*, $l_{i+1,0} = l_{i,k}$. Initially, for any edge e, $l_{1,0}(e) = \delta/c(e) = l_{0,k}(e)$.

5.1. The Analysis

We shall be interested in the values of the functions $D(), \alpha()$ only for the length functions $l_{i,k}, i \geq 0$. For brevity we denote $D(l_{i,k}), \alpha(l_{i,k})$ by $D(i), \alpha(i)$ respectively. With these new notations we have for $i \geq 1$

$$D(i) = D(l_{i,k}) = D(l_{i,0}) + \epsilon \sum_{j=1}^{k} \texttt{min\_cost}_j(l_{i,j-1})$$

Since the edge-lengths are monotonically increasing $\texttt{min\_cost}_j(l_{i,j-1}) \leq \texttt{min\_cost}_j(l_{i,k})$ and hence

$$D(i) \leq D(l_{i,0}) + \epsilon \sum_{j=1}^{k} \texttt{min\_cost}_j(l_{i,k}) = D(i-1) + \epsilon\alpha(i)$$

Since $\frac{D(i)}{\alpha(i)} \geq \beta$ we have

$$D(i) \leq \frac{D(i-1)}{1-\epsilon/\beta}$$

Since $D(0) = m\delta$ we have for $i \geq 1$

$$\begin{aligned} D(i) &\leq \frac{m\delta}{(1-\epsilon/\beta)^i} \\ &= \frac{m\delta}{1-\epsilon/\beta}\left(1+\frac{\epsilon}{\beta-\epsilon}\right)^{i-1} \\ &\leq \frac{m\delta}{1-\epsilon/\beta} e^{\frac{\epsilon(i-1)}{\beta-\epsilon}} \\ &\leq \frac{m\delta}{1-\epsilon} e^{\frac{\epsilon(i-1)}{\beta(1-\epsilon)}} \end{aligned}$$

where the last inequality uses our assumption that $\beta \geq 1$.

The procedure stops at the first phase t for which $D(t) \geq 1$. Therefore,

$$1 \leq D(t) \leq \frac{m\delta}{1-\epsilon} e^{\frac{\epsilon(t-1)}{\beta(1-\epsilon)}}$$

which implies

$$\frac{\beta}{t-1} \leq \frac{\epsilon}{(1-\epsilon)\ln\frac{1-\epsilon}{m\delta}} \tag{3}$$

In the first $t-1$ phases, for every commodity j, we have routed $(t-1)d(j)$ units. However, this flow may violate capacity constraints.

Claim 5.1 $\lambda > \frac{t-1}{\log_{1+\epsilon} 1/\delta}$.

Proof: Consider an edge e. For every $c(e)$ units of flow routed through e, we increase its length by at least a factor $1+\epsilon$. Initially, its length is $\delta/c(e)$ and after $t-1$ phases, since $D(t-1) < 1$, the length of e satisfies $l_{t-1,k}(e) < 1/c(e)$. Therefore the total amount of flow through e in the first $t-1$ phases is strictly less than $\log_{1+\epsilon} \frac{1/c(e)}{\delta/c(e)} = \log_{1+\epsilon} 1/\delta$ times its capacity. Scaling the flow by $\log_{1+\epsilon} 1/\delta$ implies the claim. ■

Thus the ratio of the values of the dual and primal solutions, γ, is strictly less than $\frac{\beta}{t-1}\log_{1+\epsilon} 1/\delta$. Substituting the bound on $\beta/(t-1)$ from (3) we get

$$\gamma < \frac{\epsilon \log_{1+\epsilon} 1/\delta}{(1-\epsilon)\ln\frac{1-\epsilon}{m\delta}} = \frac{\epsilon}{(1-\epsilon)\ln(1+\epsilon)} \frac{\ln 1/\delta}{\ln\frac{1-\epsilon}{m\delta}}$$

For $\delta = (m/(1-\epsilon))^{-1/\epsilon}$ the ratio $\frac{\ln 1/\delta}{\ln\frac{1-\epsilon}{m\delta}}$ equals $(1-\epsilon)^{-1}$ and hence

$$\gamma \leq \frac{\epsilon}{(1-\epsilon)^2\ln(1+\epsilon)} \leq \frac{\epsilon}{(1-\epsilon)^2(\epsilon-\epsilon^2/2)} \leq (1-\epsilon)^{-3}$$

Now it remains to choose ϵ suitably so that $(1-\epsilon)^{-3}$ is at most our desired approximation ratio $1+w$.

5.2. Running time

By weak-duality we have

$$1 \leq \gamma < \frac{\beta}{t-1}\log_{1+\epsilon}\frac{1}{\delta}$$

and hence the number of phases in the above procedure, t, is strictly less than $1 + \beta\log_{1+\epsilon} 1/\delta$ which implies that $t = \lceil \frac{\beta}{\epsilon}\log_{1+\epsilon}\frac{m}{1-\epsilon} \rceil$.

The running time of our computation depends on β which can be reduced/increased by multiplying the demands/capacities appropriately. Let z_i be the maximum possible flow of commodity i and let $z \stackrel{\text{def}}{=} \min_i z_i/d(i)$. Then z denotes the maximum fraction of the demands that can be routed independently and hence $z/k \leq \beta \leq z$. We scale the capacities/demands so that $z/k = 1$ thus satisfying our assumption that $\beta \geq 1$. Note however that β could now be as large as k.

If our procedure does not stop within $2\lceil \frac{1}{\epsilon}\log_{1+\epsilon}\frac{m}{1-\epsilon} \rceil$ (= T, say) phases then we know that $\beta \geq 2$. We double the demands of all commodities and continue the procedure. Note that β is now half its value in the previous phase and is at least 1. We run the procedure for an additional T phases and if it does not halt we again double demands. Since we halve the value of β after every T phases, the total number of phases is at most $T\log k$.

Theorem 5.1 *There is an algorithm that computes a $(1-\epsilon)^{-3}$-approximation to the maximum concurrent flow in time $2k\log k\lceil \frac{1}{\epsilon}\log_{1+\epsilon}\frac{m}{1-\epsilon} \rceil T_{\rm mcf}$ where $T_{\rm mcf}$ is the time required to compute a minimum cost s-t flow in a graph with non-negative edge-costs.*

The number of phases can be reduced further using an idea from [11]. We first compute a 2-approximation to β using the procedure outlined above. This requires $O(\log k \log m)$ phases and returns $\hat{\beta}$, $\beta \leq \hat{\beta} \leq 2\beta$. Now create a new instance by multiplying demands by $\hat{\beta}/2$; this instance has $1 \leq \beta \leq 2$. Therefore we need at most an additional T phases to obtain a $(1+w)$-approximation. Thus the number of phases is $O(\log m(\log k + (\epsilon \ln 1+\epsilon)^{-1}))$ which multiplied by k gives the number of single commodity min-cost flow computations required.

6. Minimum cost multicommodity flow

Given an instance of the multicommodity flow problem, as in the previous section, edge costs $b : E \rightarrow \mathbf{R}^+$, where $b(e)$ represents the cost incurred in shipping 1 unit of flow along edge e, and a bound B, we consider the problem of maximizing λ subject to the additional constraint that the cost of the flow is no more than B. The dual of this linear program is an assignment of lengths to the edges, $l : E \rightarrow \mathbf{R}^+$, and a scalar ϕ — which we view as a length associated with a pseudo-edge of capacity B — such that $D(l,\phi) \stackrel{\text{def}}{=} \sum_e l(e)c(e) + \phi B$ is minimized subject to the constraint that $\alpha(l,\phi) \stackrel{\text{def}}{=} \sum_j \texttt{min\_cost}_j(l+\phi b)$ is at least 1. This is equivalent to finding a length function (l,ϕ) such that $D(l,\phi)/\alpha(l,\phi)$ is minimum; let β denote this minimum value. As in the case of maximum concurrent flow we begin by assuming that $\beta \geq 1$.

Once again the algorithm proceeds in phases each of which is composed of k iterations. In the j^{th} iteration of the i^{th} phase we begin with length functions $(l_{i,j-1}, \phi_{i,j-1})$ and route $d(j)$ units of commodity j. As before, for all edges e, define $l_{i+1,0}(e) = l_{i,k}(e)$ and $l_{1,0}(e) = l_{0,k}(e) = \delta/c(e)$. Similarly $\phi_{i+1,0} = \phi_{i,k}$ and $\phi_{1,0} = \delta/B$.

The flow in each iteration is routed in a sequence of steps; in each step we only route so much flow that its cost does not exceed the bound B. Let $(l^{s-1}_{i,j}, \phi^{s-1}_{i,j})$ be the length functions at the start of the s^{th} step (see Fig. 1); the lengths at the start of the first step are given by $l^0_{i,j} = l_{i,j-1}$ and $\phi^0_{i,j} = \phi_{i,j-1}$. Further, let $d^{s-1}_{i,j}$ be the flow of commodity j that remains to be routed in this iteration. We compute $f^s_{i,j} \stackrel{\text{def}}{=} \texttt{min\_cost}_j(l^{s-1}_{i,j} + b\phi^{s-1}_{i,j})$ which routes $d(j)$ units of flow of commodity j. Since we need to route only $d^{s-1}_{i,j}$ units of flow we multiply the flow function $f^s_{i,j}$ by $d^{s-1}_{i,j}/d(j)$. If $B^s_{i,j}$ is the cost of flow $f^s_{i,j}$ then the cost of the scaled flow is $B^s_{i,j} d^{s-1}_{i,j}/d(j)$. If this quantity exceeds B then we multiply the original flow function $f^s_{i,j}$ by $B^s_{i,j}/B$. We reuse notation and denote the final scaled flow and its cost by $f^s_{i,j}, B^s_{i,j}$ respectively. Now $f^s_{i,j}$ routes at most $d^{s-1}_{i,j}$ units of flow at cost $B^s_{i,j} \leq B$.

The length functions are modified in a similar manner as before. Thus $l^s_{i,j} = l^{s-1}_{i,j}(1 + \epsilon f^s_{i,j}(e)/c(e))$ and $\phi^s_{i,j} = \phi^{s-1}_{i,j}(1 + \epsilon B^s_{i,j}/B)$. Further, only $d^s_{i,j} = d^{s-1}_{i,j} - f^s_{i,j}$, more units of commodity j remain to be routed in this iteration. The iteration ends at the step p for which $d^p_{i,j} = 0$. The procedure stops at the first step at which $D()$ exceeds 1; let this happen in the t^{th} phase.

6.1. Analysis

Note that now

$$\begin{aligned} & D(l^s_{i,j}, \phi^s_{i,j}) \\ = \;& D(l^{s-1}_{i,j}, \phi^{s-1}_{i,j}) + \epsilon \cdot \texttt{min\_cost}_j(l^{s-1}_{i,j} + b\phi^{s-1}_{i,j}) f^s_{i,j}/d(j) \\ \leq \;& D(l^{s-1}_{i,j}, \phi^{s-1}_{i,j}) + \epsilon \cdot \texttt{min\_cost}_j(l^{p}_{i,j} + b\phi^{p}_{i,j}) f^s_{i,j}/d(j) \end{aligned}$$

where the last inequality holds because the edge-lengths are monotonically increasing over steps. The total flow routed in the p steps equals the demand of commodity j, *ie.*, $\sum_{s=1}^p f^s_{i,j} = d(j)$. Summing over all p steps we get

$$D(l^p_{i,j}, \phi^p_{i,j}) \leq D(l^0_{i,j}, \phi^0_{i,j}) + \epsilon \cdot \texttt{min\_cost}_j(l^p_{i,j} + b\phi^p_{i,j})$$

The length functions at the start of the $(j+1)^{\text{th}}$ iteration are given by $l_{i,j} = l^p_{i,j}$ and $\phi_{i,j} = \phi^p_{i,j}$. Moving from steps to iterations we have

$$\begin{aligned} & D(l_{i,j}, \phi_{i,j}) \\ \leq \;& D(l_{i,j-1}, \phi_{i,j-1}) + \epsilon \cdot \texttt{min\_cost}_j(l_{i,j} + b\phi_{i,j}) \\ \leq \;& D(l_{i,j-1}, \phi_{i,j-1}) + \epsilon \cdot \texttt{min\_cost}_j(l_{i,k} + b\phi_{i,k}) \end{aligned}$$

where the last inequality uses the fact that the edge-lengths are monotonically increasing over iterations. Summing over all iterations in the i^{th} phase we have

$$\begin{aligned} D(l_{i,k}, \phi_{i,k}) &\leq D(l_{i,0}, \phi_{i,0}) + \epsilon \sum_{j=1}^{k} \texttt{min\_cost}_j(l_{i,k} + b\phi_{i,k}) \\ &= D(l_{i-1,k}, \phi_{i-1,k}) + \epsilon\alpha(l_{i,k}, \phi_{i,k}) \end{aligned}$$

As before we abbreviate $D(l_{i,k}, \phi_{i,k}), \alpha(l_{i,k}, \phi_{i,k})$ to $D(i), \alpha(i)$ respectively to obtain

$$D(i) \leq D(i-1) + \epsilon\alpha(i)$$

The remainder of the analysis is exactly as in Section 5.1. The only modification is in the claim about the throughput of the flow routed. Now we need to argue that the cost of the flow after we scale it by $\log_{1+\epsilon} 1/\delta$ is at most B, or equivalently, that the cost of the flow routed in the first $t-1$ iterations is at most $B \log_{1+\epsilon} 1/\delta$. This follows from the fact that $\phi_{t-1,k} < 1/B$ (since $D(t-1) < 1$), that $\phi_{1,0} = \delta/B$ and that in our procedure every time we route flow whose total cost is B we increase ϕ by at least a factor $1+\epsilon$.

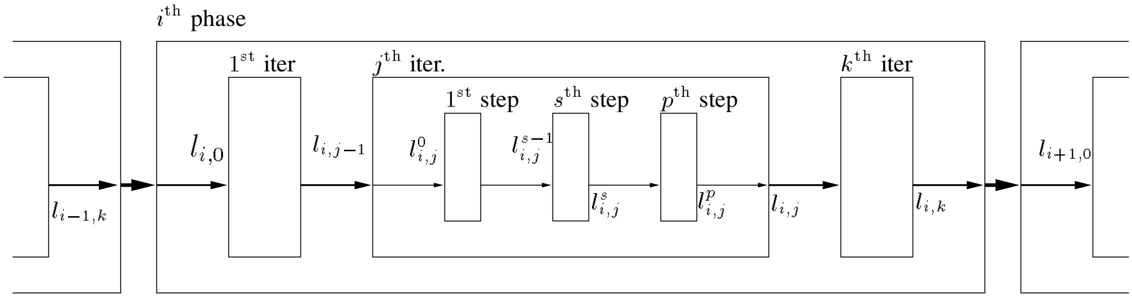

Figure 1. The notation used in Sections 6 and 7. The length functions above the central axis are the lengths *before* the box on the right and the ones below are the lengths *after* the box on the left.

6.2. Running time

Note that except for the last step in each iteration, in all other steps we increase the length function ϕ by a factor $1+\epsilon$. This implies that the total number of steps exceeds the number of iterations by at most $\log_{1+\epsilon} 1/\delta$.

Now define z_i as the maximum possible flow of commodity i of cost no more than B. Again $z \stackrel{\text{def}}{=} \min_i z_i/d(i)$ denotes the maximum fraction of the demands that can be routed if the capacity constraints and the bound B on the cost of the flow applied independently to each commodity. Thus $z/k \leq \beta \leq z$ and we multiply demands suitably so that for the new instance $1 \leq \beta \leq k$. As before we double the demands, thereby halving β, after every T phases. Thus the number of iterations is $kT \log k$ and so our procedure for minimum cost multicommodity flow needs at most $(2k\log k+1)\lceil \frac{1}{\epsilon}\log_{1+\epsilon}\frac{m}{1-\epsilon}\rceil$ single-commodity min-cost flow computations.

Theorem 6.1 *There is an algorithm that computes a $(1-\epsilon)^{-3}$-approximation to the maximum cost-bounded concurrent flow in time $(2k\log k+1)\lceil \frac{1}{\epsilon}\log_{1+\epsilon}\frac{m}{1-\epsilon}\rceil T_{\text{mcf}}$ where T_{mcf} is the time required to compute a minimum cost s-t flow in a graph with non-negative edge-costs.*

7. Avoiding min-cost flow computations

We now use ideas from our algorithm for min-cost multicommodity flow to give algorithms for the maximum concurrent flow and min-cost multicommodity flow problems which use shortest path computations instead of min-cost flow computations and are faster than the algorithms in Section 5 and 6 by at least a factor $\min\left\{n, \frac{nk\log k}{m}\right\}$.

7.1. Maximum concurrent flow revisited

Define $\alpha(l) \stackrel{\text{def}}{=} \sum_j d(j)\texttt{dist}_j(l)$ where $\texttt{dist}_j(l)$ denotes the shortest path distance between s_j and t_j under the length function l. The dual to the maximum concurrent flow problem can also be viewed as an assignment of lengths to edges, $l: E \rightarrow \mathbf{R}^+$, such that $D(l)/\alpha(l)$ is minimized. Let β be this minimum.

The structure of this new algorithm is similar to that in the previous section. Thus the algorithm runs in phases each of which is composed of k iterations. In the j^{th} iteration of the i^{th} phase we route $d(j)$ units of commodity j in a sequence of steps. Let $l^{s-1}_{i,j}$ be the length function before the s^{th} step and let $P^s_{i,j}$ be the shortest path between s_j and t_j, *ie.*, $P^s_{i,j}$ has length $\texttt{dist}_j(l^{s-1}_{i,j})$. In this step we route $f^s_{i,j} = \min\left\{c, d^{s-1}_{i,j}\right\}$ units of flow along $P^s_{i,j}$ where c is the capacity of the minimum capacity edge on this path. We now set $d^s_{i,j}$ to $d^{s-1}_{i,j} - f^s_{i,j}$; the iteration ends after p steps where $d^p_{i,j} = 0$.

Thus at each step we perform a shortest path computation instead of a min-cost flow computation as in Section 6. The length functions are modified in exactly the same manner as before and the analysis is almost exactly the same. Thus after routing all flow of commodity j we have

$$D(l^p_{i,j}) \leq D(l^0_{i,j}) + \epsilon \cdot d(j)\texttt{dist}_j(l^p_{i,j})$$

and after routing all commodities in the i^{th} phase we have

$$D(l_{i,k}) \leq D(l_{i,0}) + \epsilon \sum_{j=1}^{k} d(j)\texttt{dist}_j(l_{i,k})$$

Using the same abbreviations as before we again obtain

$$D(i) \leq D(i-1) + \epsilon\alpha(i)$$

Beyond this point we follow the analysis of Section 5.1 to argue that we have a $(1+\omega)$-approximation for the same choice of ϵ and δ.

For the running time we again note that in each step, except the last one in an iteration, we increase the length of at least one edge by a factor $1+\epsilon$. Since each edge has an initial length of δ and a final length less than $1+\epsilon$, the number of steps exceeds the number of iterations by at most $m\log_{1+\epsilon}\frac{1+\epsilon}{\delta}$. Thus the total number of steps is at most $(2k\log k+m)\lceil\frac{1}{\epsilon}\log_{1+\epsilon}\frac{m}{1-\epsilon}\rceil$ and each of these involves one shortest path computation.

Theorem 7.1 *There is an algorithm that computes a $(1-\epsilon)^{-3}$-approximation to the maximum concurrent flow in time $(2k\log k+m)\lceil\frac{1}{\epsilon}\log_{1+\epsilon}\frac{m}{1-\epsilon}\rceil T_{\mathrm{sp}}$ where T_{sp} is the time required to compute the shortest s-t path in a graph with non-negative edge-weights.*

7.2. Min-cost multicommodity flow revisited

We now define $\alpha(l,\phi)\stackrel{\mathrm{def}}{=}\sum_j d(j)\mathtt{dist}_j(l+b\phi)$. The dual to the min-cost multicommodity flow problem is an assignment of lengths to edges, $l: E\rightarrow\mathbf{R}^+$, and a scalar ϕ such that $D(l)/\alpha(l)$ is minimized. Let β be this minimum.

The algorithm differs from the one developed in Section 6 in that at any step we route flow along only one path, which, if this is the s^{th} step of the j^{th} phase of the i^{th} iteration, is the shortest path between s_j and t_j under the length function $l_{i,j}^{s-1}+b\phi_{i,j}^{s-1}$. If the minimum capacity edge on this path has capacity c then the flow function at this step, $f_{i,j}^s$, corresponds to routing c units of flow along this path. If $c\le d_{i,j}^{s-1}$ and the cost of this flow is less than B we route this flow completely. Else we scale it so that the flow routed in this step has cost no more than B and the total flow routed in this iteration does not exceed $d(j)$.

The analysis of the algorithm proceeds as in Section 6.1 with the only modification that $\mathtt{min\_cost}_j(.)$ is replaced with $d(j)\mathtt{dist}_j(.)$. For the running time we need only observe that in each step, except the last step in an iteration, we increase, either the length of some edge or the value of ϕ by a factor $1+\epsilon$. The lengths of the edges and ϕ can each be increased by a factor $1+\epsilon$ at most $\log_{1+\epsilon}\frac{1+\epsilon}{\delta}$ times. Hence the number of steps exceeds the number of iterations by at most $(m+1)\lceil\frac{1}{\epsilon}\log_{1+\epsilon}\frac{m}{1-\epsilon}\rceil$.

Theorem 7.2 *There is an algorithm that computes a $(1-\epsilon)^{-3}$-approximation to the maximum concurrent flow in time $(2k\log k+m+1)\lceil\frac{1}{\epsilon}\log_{1+\epsilon}\frac{m}{1-\epsilon}\rceil T_{\mathrm{sp}}$ where T_{sp} is the time required to compute the shortest s-t path in a graph with non-negative edge-weights.*

7.3. Integrality

A multicommodity flow has integrality q if the flow of every commodity on every edge is a non-negative integer multiple of q. In this section we show how small modifications to the algorithms discussed in previous sections lead to flows that have small integrality.

Our algorithm for maximum multicommodity flow routes flow along a path P in the i^{th} iteration. If c is the capacity of a minimum capacity edge on P then we require that the flow routed in this iteration be no more than c. However, note that if we route $q<c$ units along P and increase the length of an edge e on P by a factor $(1+\epsilon q/c(e))$ then the algorithm still delivers a $(1-\epsilon)^{-2}$-approximation to the maximum multicommodity flow, albeit with a worse running time. To obtain a feasible flow we scale the flow constructed in this manner by $\log_{1+\epsilon}1/\delta$. Thus if we were routing q units in a certain iteration then only $\frac{q}{\log_{1+\epsilon}1/\delta}$ units "appear" in the feasible solution.

Theorem 7.3 *Let e be the minimum capacity edge in G and $q\le c(e)$. Then one can in polynomial time compute a flow f which is a $(1-\epsilon)^{-2}$-approximation to the maximum multicommodity flow and f has integrality $\frac{q\epsilon}{\log_{1+\epsilon}L}$.*

Corollary 7.4 *If all edges in G have capacity at least $\frac{1}{\epsilon}\log_{1+\epsilon}L$ then there is an integral flow which is a $(1-\epsilon)^{-2}$-approximation to the maximum multicommodity flow.*

For maximum concurrent flow we use the algorithm from Section 7. Recall that in the s^{th} step of the j^{th} iteration in the i^{th} phase we route $f_{i,j}^s=\min\{c,d_{i,j}^{s-1}\}$ units of flow along path $P_{i,j}^s$ where c is the capacity of the minimum capacity edge on this path and $d_{i,j}^{s-1}$ is the residual demand of the j^{th} commodity. As in the case of maximum multicommodity flow we route $q<f_{i,j}^s$ units of flow in this step and increase the length of an edge e on P by a factor $(1+\epsilon q/c(e))$. To ensure that exactly q units of flow can be routed in each step of the j^{th} iteration we require that $d(j)$ be an integral multiple of q. To obtain a feasible flow we scale the flow constructed by $\log_{1+\epsilon}1/\delta$. Hence in the final solution the flow appears in units of $\frac{q\epsilon}{\log_{1+\epsilon}m/(1-\epsilon)}$.

Theorem 7.5 *Let e be the minimum capacity edge in G and $q\le c(e)$. If all demands are integral multiples of q then one can, in polynomial time, compute a flow f which is a $(1-\epsilon)^{-3}$-approximation to the maximum concurrent flow and f has integrality $\frac{q\epsilon}{\log_{1+\epsilon}m/(1-\epsilon)}$.*

Corollary 7.6 *If all edges in G have capacity at least $\frac{1}{\epsilon}\log_{1+\epsilon}\frac{m}{1-\epsilon}$ and all demands are integral multiples of $\frac{1}{\epsilon}\log_{1+\epsilon}\frac{m}{1-\epsilon}$ then there is an integral flow which is a $(1-\epsilon)^{-3}$-approximation to the maximum concurrent flow.*

The above theorem and its corollary also hold for the setting of min-cost multicommodity flows.

Acknowledgements. The first author would like to Philip Klein, Cliff Stein and Neal Young for useful discussions.

References

[1] B. Awerbuch and F.T. Leighton. Improved approximation algorithms for the multicommodity flow problem and local competitive routing in dynamic networks. In *Proceedings, ACM Symposium on Theory of Computing*, pages 487–496, 1994.

[2] G. Even, J. Naor, S. Rao, and B. Schieber. Divide-and-conquer approximation algorithms via spreading metrics. In *Proceedings, IEEE Symposium on Foundations of Computer Science*, pages 62–71, 1995.

[3] G. Even, J. Naor, S. Rao, and B. Schieber. Fast approximate graph partitioning algorithms. In *Proceedings, ACM-SIAM Symposium on Discrete Algorithms*, pages 639–648, 1997.

[4] A.V. Goldberg. A natural randomization strategy for multicommodity flow and related algorithms. *Inform. Process. Lett.*, 42:249–256, 1992.

[5] M. Grigoriadis and L.G. Khachiyan. An exponential function reduction method for block angular convex programs. Technical Report LCSR-TR-211, Department of Computer Science, Rutgers University, New Brunswick, NU, 1993.

[6] M. Grigoriadis and L.G. Khachiyan. Approximate minimum-cost multicommodity flows in $\tilde{O}(\epsilon^{-2}knm)$ time. *Math. Programming*, 75:477–482, 1996.

[7] M.D. Grigoriadis and L.G. Khachiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. *SIAM J. Optimization*, 4(1):86–107, 1994.

[8] D. Karger and S. Plotkin. Adding multiple cost constraints to combinatorial optimization problems, with applications to multicommodity flows. In *Proceedings, ACM Symposium on Theory of Computing*, pages 18–25, 1995.

[9] P. Klein, S. Plotkin, C. Stein, and E. Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. *SIAM J. Comput.*, 23(3):466–487, 1994.

[10] T. Leighton, F. Makedon, S. Plotkin, C. Stein, S. Tragoudas, and E. Tardos. Fast approximation algorithms for multicommodity flow problems. *J. Comput. System Sci.*, 50:228–243, 1995.

[11] S. Plotkin, D. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Math. Oper. Res.*, 20:257–301, 1995.

[12] T. Radzik. Fast deterministic approximation for the multicommodity flow problem. In *Proceedings, ACM-SIAM Symposium on Discrete Algorithms*, pages 486–492, 1995.

[13] F. Shahrokhi and D. Matula. The maximum concurrent flow problem. *J. ACM*, 37(2):318–334, 1990.

[14] David Shmoys. *Approximation algorithms for NP-hard problems*, chapter Cut problems and their application to divide and conquer, pages 192–235. PWS Publishing Company, 1997.

[15] C. Stein. *Approximation algorithms for multicommodity flow and scheduling problems*. PhD thesis, MIT, 1992.

[16] P.M. Vaidya. Speeding up linear programming using fast matrix multiplication. In *Proceedings, IEEE Symposium on Foundations of Computer Science*, pages 332–337, 1989.

[17] N. Young. Randomized rounding without solving the linear program. In *Proceedings, ACM-SIAM Symposium on Discrete Algorithms*, 170-178, 1995.