Faster Approximation Schemes for Fractional Multicommodity Flow Problems*

George Karakostas†

Abstract

We present fully polynomial approximation schemes for concurrent multicommodity flow problems that run in time independent of the number of commodities k. We show that by modifying the algorithms by Garg & Könemann [5] and Fleischer [3] we can reduce their running time to a logarithmic dependence on k, and essentially match the running time of [3] for the maximum multicommodity flow problem.

1 Introduction

Flow problems are at the core of many optimization problems. Usually the input to such a problem consists of a directed network $G = (V, E)$, an edge utilization (or capacity) function $u : E \rightarrow \mathbb{R}^+$ and a source-sink pair (s, t) (single commodity flow) or a set of k source-sink pairs (s_i, t_i), $1 \leq i \leq k$ (multicommodity flow). In general one would like to calculate flows f_i from s_i to t_i that would optimize an objective function, subject to the constraint that the sum of flows through an edge cannot exceed the capacity of the edge, possibly together with other constraints. The simplest multicommodity flow problem is the *maximum multicommodity flow* problem: the objective is the maximization of the total flow (throughput) $\sum_{i=1}^{k} f_i$ subject to the capacity constraints. A more complex variation called the *maximum concurrent flow* problem is the following: besides the above, we are also given a set of k *demands* d_i, one for each commodity pair (s_i, t_i), and we are asked to satisfy the maximum fraction λ of all demands, i.e. $\max \lambda$, subject to $f_i \geq \lambda d_i$, $\forall i$. The maximum concurrent flow problem and some other problems related to it are the focus of this work.

The maximum concurrent flow problem has recently attracted the attention of researchers working on approximation algorithms. The maximum concurrent flow problem is the dual of the LP relaxation of the *sparsest cut* problem. Finding the sparsest cut of a graph with weighted edges means finding a cut S that minimizes the ratio of the weight of the edges cut to the total demand that is separated by the cut. There are many approximation algorithms for the sparsest cut based on rounding the solution of its LP relaxation (most recently the algorithms of [11] and [2]). Thus any algorithm for solving the maximum concurrent flow problem provides also a solution to this LP and is the starting point for these approximation algorithms (more on the sparsest cut problem and other problems related to it and to the maximum concurrent flow can be found in the survey by Shmoys [15]).

Although the problems considered in this paper can be solved optimally (since they can be formulated as linear programs), in many applications the need to solve them quickly is greater than the need for optimality. Hence for the last decade there has been a series of papers on the design of *fully polynomial time approximation schemes (FPTAS)* for multicommodity flow problems. An approximation scheme is a family of algorithms that compute a solution within a factor $(1 - \varepsilon)$ of the optimal for *any constant* ε. If the running time of an approximation scheme is *poly(input size, 1/ε)* then the scheme is *fully polynomial-time*.

A line of research (cf. [14] [9] [10] [6] [12] [13] [8] [7]) based on Langrangian relaxation and linear programming decomposition techniques has led to ever decreasing running times for multicommodity flow problems. All these algorithms

*©2001 Telcordia Technologies Inc. All Rights Reserved.

†Applied Research, Telcordia Technologies Inc., 445 South St., Morristown, NJ 07960. E-mail: gk@research.telcordia.com

compute an initial flow, and then try to redistribute flow from more congested paths to less congested paths. In a departure from this general theme, Young [16] described a randomized algorithm which builds a solution from scratch, and does not reroute flow. At every step Young's algorithm solves a shortest path problem (on the dual LP), instead of a maximum single commodity flow problem in earlier algorithms.

Garg and Könemann [5] followed a similar approach. However, the simple and elegant analysis of their algorithm allowed them to make a greater advance at each step: while Young's procedure pushes a unit flow at each step along a (shortest) path, [5] pushes enough flow so as to saturate the minimum capacity edge of the path. Their results generalize to *fractional packing problems*, and yield an algorithm whose running time does not depend on the *width* of the problem (i.e. the maximum extent to which any constraint can be violated). Fleischer [3] noticed that for the maximum multicommodity flow problem, one can use *approximate* shortest paths, in order to avoid recalculations of shortest paths for commodities with a common source. By making the greatest possible advance for every *source* (instead for every *commodity*), [3] reduces the dependence of the running time on k to a mere logarithmic factor. Unfortunately, the techniques of [3] for the maximum multicommodity flow do not apply to the maximum concurrent flow. Nevertheless, using different techniques (which we also use and are described in what follows) Fleischer[3] reduces the dependence of the algorithm in [5] on k to a linear factor by replacing a maximum single commodity flow computation needed by [5] with a shortest path computation.

In this paper, we build on the work of [5] and [3] to virtually eliminate the dependence of the maximum concurrent flow on k, thus matching the running time of [3] for the maximum multicommodity flow. We notice that the analysis of [5] still holds if at every step we try to push flow from a single source along the whole *tree* produced by Dijkstra's algorithm. This has the effect of the techniques of [3] on the maximum multicommodity flow, and combined with the other improvements in [3] produces better algorithms for the maximum concurrent flow and relevant problems. Our results are summarized in Table 1. Our improvement in the running time becomes apparent in many applications (e.g. routing over a telephone network) where the network topology is itself very sparse (i.e. $m = O(n)$) but there are demands for almost every pair of nodes (i.e. $k = \Omega(n^2)$).

The definition of a lossy network and the relevant results appear in Section 5.

2 Notation

In what follows, G will represent the underlying network graph for the flow computations, n is its number of nodes and m is its number of edges. We will assume that G is connected, so $m \geq n$. We use the notation $\tilde{O}(f)$ to denote a quantity that is $O(f \log^{O(1)} m)$.

3 Maximum concurrent flow

We are given a graph $G = (V, E)$ with edge capacities $u : E \to \mathbb{R}^+$ and k commodities with (s_i, t_i) being the *(source,sink)* pair for commodity i. For each commodity i we are also given a demand $d(i) > 0$. We want to find the largest λ such that there is a multicommodity flow which routes at least $\lambda d(i)$ units of commodity i for all commodities i. Since we are interested in fractional flows, this λ can be expressed as the solution of a linear program: Let $\mathcal{P}_i$ be the set of paths between s_i and t_i in G, and let $\mathcal{P} := \cup_i \mathcal{P}_i$. Variable $x(P)$ denotes the amount of flow sent along path P, for every $P \in \mathcal{P}$. Then the LP path formulation of the maximum concurrent multicommodity flow problem is the following:

$$\begin{array}{rcll} \text{maximize} & \lambda & \text{s.t.} & \\ \sum_{P: e \in P} x(P) & \leq & u(e) & \forall e \in E \\ \sum_{P \in \mathcal{P}_i} x(P) & \geq & \lambda d(i) & \forall i \\ x(P) & \geq & 0 & \forall P \end{array}$$

The dual LP has a variable $l(e)$ for each capacity constraint of the primal and a variable $z(i)$ for every commodity demand constraint:

Problem	Previous best	This work
max concurrent flow	$\tilde{O}(\varepsilon^{-2}m(m+k))$ [3]	$\tilde{O}(\varepsilon^{-2}m^2)$
min cost concurrent flow	$\tilde{O}(\varepsilon^{-2}m(m+k)\log M)$ [3]	$\tilde{O}(\varepsilon^{-2}m^2\log M)$
max concurrent flow for lossy networks	$\tilde{O}(\varepsilon^{-2}m(k+m))$ [4]	$\tilde{O}(\varepsilon^{-2}m^2)$
min cost concurrent flow for lossy networks	$\tilde{O}(m(km+\varepsilon^{-2}(k+m)\log M))$ [4]	$\tilde{O}(m^2(k+\varepsilon^{-2}\log M))$

Table 1: Comparison of previous algorithms to our results. $\tilde{O}(f)$ denotes a quantity that is $O(f\log^{O(1)} m)$, M is the largest integer that is used to specify capacities, costs or demands.

$$
\begin{array}{rcll}
\text{minimize} & & \sum_{e\in E} u(e)l(e) & \text{s.t.} \\
\sum_{e\in P} l(e) & \geq & z(i) & \forall i, \forall P\in\mathcal{P}_i \\
\sum_{i=1}^{k} d(i)z(i) & \geq & 1 & \\
l(e) & \geq & 0 & \forall e \\
z(i) & \geq & 0 & \forall i
\end{array}
$$

Let $D(l) := \sum_e u(e)l(e)$ be the quantity minimized by the dual. Define $\mathrm{dist}_i(l)$ as the distance of the shortest path from s_i to t_i in G under the length function l. Let $\alpha(l) := \sum_i d(i)\mathrm{dist}_i(l)$. As observed in [5], minimizing $D(l)$ under the dual constraints is equivalent to computing lengths $l(e)$ for the edges that minimize $D(l)/\alpha(l)$. Let $\beta := \min_l D(l)/\alpha(l)$.

The algorithm by Garg and Könemann [5] runs in time $\tilde{O}(\varepsilon^{-2}m(m+k))$, provided $\beta \geq 1$. If $\beta < 1$, Fleischer [3] describes a procedure for scaling the problem so that $\beta \geq 1$, using k maximum bottleneck path computations[1] that run in time $O(\max\{n,k\}m)$. We use this scaling procedure as well, but we improve the running time for the case $\beta \geq 1$ to match the running time of the scaling, for a total running time of $\tilde{O}(\varepsilon^{-2}m^2)$. For the moment we will assume that $\beta \geq 1$. Later we will describe Fleischer's scaling procedure that removes this assumption.

[1]The scaling procedure used in [5] solves k maximum flow problems. This turns out to be the bottleneck of their algorithm, causing a running time that matches previously known concurrent flow algorithms. This bottleneck was improved by [3]. We improve the algorithm of [5] itself.

3.1 The case $\beta \geq 1$

We modify the algorithm in [5] (also used in [3]). The algorithm proceeds in phases and each phase is composed of $|\mathcal{S}|$ iterations, with $\mathcal{S}$ being the set of *sources* (notice that this is *not* necessarily the number of *commodities* k). In iteration j of the i^{th} phase we consider all the commodities with the same source s_j. Let $c_1, c_2, \ldots, c_r$ be these commodities. We route $d(c_q), q = 1, \ldots, r$ units of commodity c_q in a series of steps. Let $l_{i,j,s}$ be the length function at the end of the s^{th} step. At every step we compute the shortest paths $P^{c_q}_{i,j,s}$ from s_j to the sink of commodity c_q, i.e. $P^{c_q}_{i,j,s}$ has length $\mathrm{dist}_{c_q}(l_{i,j,s-1})$. It is crucial to our improvement that the shortest paths for all commodities with a common source s_j can be computed with only *one* call to Dijkstra's algorithm. Let $d^{c_q}_{i,j,s} > 0$ be the amount of commodity c_q that has not been routed yet at step s (obviously $d^{c_q}_{i,j,0} = d(c_q)$). Notice that we consider only commodities with *strictly* positive remaining demand, and we ignore commodities whose demand has already been completely routed. Then at step s we route $f^{c_q}_{i,j,s} = \min\{c, d^{c_q}_{i,j,s-1}\}$ units of each commodity c_q along path $P^{c_q}_{i,j,s}$, where c is the minimum capacity of the edges on *all* these paths. After this we set $d^{c_q}_{i,j,s} := d^{c_q}_{i,j,s-1} - f^{c_q}_{i,j,s}$, and for every edge e on these paths we set

$$
\begin{aligned}
l_{i,j,s}(e) &:= l_{i,j,s-1}(e)\left(1+\varepsilon\cdot\frac{\text{total new flow through } e}{u(e)}\right) \\
&= l_{i,j,s-1}(e)\left(1+\varepsilon\cdot\frac{\sum_{c_q: e\in P^{c_q}_{i,j,s}} f^{c_q}_{i,j,s}}{u(e)}\right).
\end{aligned}
$$

Note that for the minimum capacity edge $l()$ increases by a factor of at least $1+\varepsilon$, while for any edge $l()$ increases by a factor of at most $1+k\varepsilon$. After the last step s $d^{c_q}_{i,j,s} = 0$ for all commodities c_q. The whole procedure ends as soon as $D(l_{i,j,s}) \geq 1$, and produces a length function l and a corresponding multicommodity flow x which is infeasible for the primal LP. However, by applying the analysis of [5] we prove that scaling down x will produce a close to optimal *feasible* solution.

Initially $l_{1,0,0}(e) = \delta/u(e)$ for all edges e, where δ is a parameter to be defined later. The algorithm is described in Figure 1. For brevity we use P_{c_q} to denote $P^{c_q}_{i,j,s}$, the shortest path for commodity c_q in the s^{th} step of the j^{th} iteration in the i^{th} phase.

LEMMA 3.1. *If $\beta \geq 1$ the algorithm terminates after at most $t := \lceil \beta \log_{1+\varepsilon} \frac{1+k\varepsilon}{\delta} \rceil$ phases.*

Proof. After step s of the j^{th} iteration of the i^{th} phase

$$\begin{aligned}
&D(l_{i,j,s}) = \\
&= D(l_{i,j,s-1}) + \varepsilon \cdot \sum_{e \in P_{c_1} \cup \ldots \cup P_{c_r}} l_{i,j,s-1}(e) \sum_{q: e \in P_{c_q}} f^{c_q}_{i,j,s} \\
&= D(l_{i,j,s-1}) + \varepsilon \cdot \sum_{e \in P_{c_1} \cup \ldots \cup P_{c_r}} \sum_{q: e \in P_{c_q}} l_{i,j,s-1}(e) \cdot f^{c_q}_{i,j,s} \\
&= D(l_{i,j,s-1}) + \varepsilon \cdot \sum_{q=1}^{r} f^{c_q}_{i,j,s} \cdot \sum_{e \in P_{c_q}} l_{i,j,s-1}(e) \\
&= D(l_{i,j,s-1}) + \varepsilon \cdot \sum_{q=1}^{r} f^{c_q}_{i,j,s} \cdot \mathrm{dist}_{c_q}(l_{i,j,s-1})
\end{aligned}$$

where the third equality is a simple rearrangement of terms, and the fourth equality holds because P_{c_q} is by definition the shortest path for commodity c_q under the length function of the previous step. By taking into account all the steps of the iteration we get

$$\begin{aligned}
&D(l_{i,j,s_{last}}) = \\
&= D(l_{i,j,0}) + \varepsilon \cdot \sum_{s=1}^{s_{last}} \sum_{q=1}^{r} f^{c_q}_{i,j,s} \cdot \mathrm{dist}_{c_q}(l_{i,j,s-1}) \\
&\leq D(l_{i,j,0}) + \varepsilon \cdot \sum_{q=1}^{r} \mathrm{dist}_{c_q}(l_{i,j,s_{last}}) \sum_{s=1}^{s_{last}} f^{c_q}_{i,j,s} \\
&= D(l_{i,j,0}) + \varepsilon \cdot \sum_{q=1}^{r} \mathrm{dist}_{c_q}(l_{i,j,s_{last}}) d(c_q)
\end{aligned}$$

where the second inequality holds because l is a monotonically increasing function. For brevity we drop the step subscript (using the facts $D(l_{i,j,s_{last}}) = D(l_{i,j+1,0})$ and $l_{i,j,s_{last}} = l_{i,j+1,0}$), so we just proved

$$D(l_{i,j}) \leq D(l_{i,j-1}) + \varepsilon \cdot \sum_{q=1}^{r} \mathrm{dist}_{c_q}(l_{i,j}) d(c_q)$$

Using the same arguments for the iterations, we can show

$$D(l_{i,0}) \leq D(l_{i-1,0}) + \varepsilon \cdot \sum_{q=1}^{k} \mathrm{dist}_q(l_{i,0}) d(q)$$

Again we drop the iteration subscript for brevity, and by observing that $\alpha(l_i) = \sum_{q=1}^{k} \mathrm{dist}_q(l_i) d(q)$ by definition we have

$$D(l_i) \leq D(l_{i-1}) + \varepsilon \cdot \alpha(l_i) \tag{3.1}$$

The rest of the analysis now proceeds exactly as in [5]. From the observation above, $\beta = \min_l \frac{D(l)}{\alpha(l)}$, so from (3.1)

$$D(l_i) \leq \frac{D(l_{i-1})}{1 - \varepsilon/\beta}$$

and from the fact that $D(l_0) = m\delta$ we have

$$\begin{aligned}
D(l_i) &\leq \frac{m\delta}{(1-\varepsilon/\beta)^i} \\
&\leq \frac{m\delta}{1-\varepsilon/\beta} e^{\frac{\varepsilon(i-1)}{\beta-\varepsilon}} \\
&\leq \frac{m\delta}{1-\varepsilon} e^{\frac{\varepsilon(i-1)}{\beta(1-\varepsilon)}}
\end{aligned}$$

Input : Graph $G = (V, E)$, capacities $u(e)$, pairs (s_i, t_i) with demands $d(i), 1 \le i \le k$, accuracy ε.

Output: Primal solution x, dual length function l

Initialize $l(e) := \delta/u(e), \forall e, \quad x(P) := 0, \forall P$.
while $D(l) < 1$ **do**
  for $i = 1$ **to** $|\mathcal{S}|$ **do**
    $d'(c_q) := d(c_q), q = 1, \ldots, r$
    while $D(l) < 1$ and $d'(c_q) > 0$ for some q **do**

$$\begin{array}{lcl} P_{c_q} & := & \text{shortest path in } \mathcal{P}_{c_q} \text{ using } l,\ q = 1, \ldots, r \text{ with } d'(c_q) > 0 \\ c & := & \min_{e \in P_{c_1} \cup \ldots \cup P_{c_r}} u(e) \end{array}$$

$$\left.\begin{array}{lcl} f_{c_q} & := & \min\{d'(c_q), c\} \\ d'(c_q) & := & d'(c_q) - f_{c_q} \\ x(P_{c_q}) & := & x(P_{c_q}) + f_{c_q} \end{array}\right\} q = 1, \ldots, r \text{ with } d'(c_q) > 0$$

$$l(e) := l(e)\left(1 + \varepsilon \cdot \frac{\sum_{c_q : e \in P_{c_q}} f_{c_q}}{u(e)}\right),\ \forall e \in P_{c_1} \cup \ldots \cup P_{c_r}$$

    end while
  end for
end while
$x(P) := x(P)/\log_{1+\varepsilon} \frac{1+\varepsilon k}{\delta},\ \forall P$
$\lambda := \min_i \frac{\sum_{P \in \mathcal{P}_i} x(P)}{d(i)}$
Output λ

Figure 1: Maximum concurrent flow FPTAS for $\beta \ge 1$.

where in the last inequality we use the hypothesis $\beta \ge 1$. The procedure stops after $D(l_t) \ge 1$ for some phase t. Hence

(3.2)
$$1 \le D(l_t) \le \frac{m\delta}{1-\varepsilon} e^{\frac{\varepsilon(t-1)}{\beta(1-\varepsilon)}} \implies \beta \le \frac{\varepsilon(t-1)}{(1-\varepsilon)\ln \frac{1-\varepsilon}{m\delta}}$$

In order to bound the primal solution λ produced by the algorithm, we prove the following claim:

CLAIM 1. $\lambda > \frac{t-1}{\log_{1+\varepsilon} \frac{1+k\varepsilon}{\delta}}$.

Proof. Since after each phase we route $d(i)$ units of commodity i, after $t-1$ phases we have routed at least $(t-1)d(i)$ units. This flow may be infeasible (it may violate capacity constraints), hence we scale it down to make it feasible. For every $u(e)$ units of flow routed through edge e we increase the length $l(e)$ by a factor of at least $1+\varepsilon$. At the beginning $l_0(e) = \delta/u(e)$ and by the end of phase $t-1$ $l_{t-1}(e) < 1/u(e)$ (since $D(l_{t-1}) < 1$). Hence at the end of the algorithm $l_t(e) < (1+k\varepsilon)/u(e)$ (each step of the last phase increases $l(e)$ by at most a factor of $1+k\varepsilon$, and the algorithm stops as soon as $D(l_{t-1}) \ge 1$). Therefore the total amount of flow through e in all t phases is strictly less than $\log_{1+\varepsilon} \frac{(1+k\varepsilon)/u(e)}{\delta/u(e)} = \log_{1+\varepsilon} \frac{1+k\varepsilon}{\delta}$ times its capacity. So by scaling the flow by $\log_{1+\varepsilon} \frac{1+k\varepsilon}{\delta}$ we get the claimed *feasible* primal solution.

This implies that

(3.3)
$$1 \le \frac{\beta}{\lambda} < \frac{\beta}{t-1} \log_{1+\varepsilon} \frac{1+k\varepsilon}{\delta}$$

which, in turn, implies the lemma.

From (3.3) and (3.2) we have

$$\frac{\beta}{\lambda} < \frac{\varepsilon \log_{1+\varepsilon} \frac{1+k\varepsilon}{\delta}}{(1-\varepsilon)\ln \frac{1-\varepsilon}{m\delta}} = \frac{\varepsilon}{(1-\varepsilon)\ln(1+\varepsilon)} \cdot \frac{\ln \frac{1+k\varepsilon}{\delta}}{\ln \frac{1-\varepsilon}{m\delta}}$$

By setting

$$\delta := \frac{1}{(1+k\varepsilon)^{\frac{1-\varepsilon}{\varepsilon}}} \cdot \left(\frac{1-\varepsilon}{m}\right)^{\frac{1}{\varepsilon}} \tag{3.4}$$

the dual-primal solution ratio becomes less than $(1-\varepsilon)^{-3}$ and we can pick ε so that this ratio is less than $1+w$ for any $w > 0$. Hence we have proven the following

LEMMA 3.2. *The proposed algorithm for the concurrent multicommodity flow problem is an approximation scheme, provided $\beta \geq 1$.*

Next we deal with the case $\beta < 1$ and prove that this approximation scheme runs in polynomial time.

3.2 The case $\beta < 1$

For δ as in (3.4) the number of phases cannot be more than $t = \lceil \frac{\beta}{\varepsilon} \log_{1+\varepsilon} \frac{(1+k\varepsilon)m}{1-\varepsilon} \rceil$, provided that $\beta \geq 1$. Garg and Könemann in [5] observe that if $\zeta(i)$ is the maximum flow of commodity i that can be routed in G when no other commodity is routed, the value $\zeta = \min_i\{\zeta(i)/d(i)\}$ is an upper bound of the optimal solution (at best, all maximum *single* commodity flows for all commodities can be routed simultaneously). The solution that routes a fraction of $1/k$ of each flow $\zeta(i)$ is a feasible solution, so ζ/k is a lower bound of the optimal solution. Hence $\zeta/k \leq \beta \leq \zeta$, so by multiplying the initial demands by ζ/k we ensure that $\beta \geq 1$. But this computation requires k maximum single commodity flow computations, and becomes the bottleneck of the algorithm in [5]. Fleischer [3] reduced this time by observing that an m-factor approximation of the $\zeta(i)$'s is enough for our purposes. Since any flow can be decomposed into at most m path flows, the flow along a maximum capacity path is an m-approximation to $\zeta(i)$. Such a path can be computed easily in $O(n \log n + m)$ time by using a modified version of Dijkstra's algorithm that updates its node labels using the maximum capacities computed so far (instead of edge costs). In fact *all* these paths/estimates can be computed in $O(min\{n,k\}m \log n)$ time, if we group the commodities by their common source node.

The procedure above can make β as big as km, thus increasing the running time of the algorithm (recall that the number of phases t depends on β). [5] proposes the following solution to reducing the dependence on β: if the algorithm does not stop after $T := 2\lceil \frac{1}{\varepsilon} \log_{1+\varepsilon} \frac{(1+k\varepsilon)m}{1-\varepsilon} \rceil$ phases, we know that $\beta \geq 2$. We double the demands of all commodities, so that β is halved and still $\beta \geq 1$, and continue running the algorithm for another T phases. If the algorithm doesn't stop we double again the demands and continue doing this until the algorithm ends. Since every time β is halved, this doubling can happen at most $\log(km)$ times, the total number of phases is at most $T \log(km)$. This number can be further reduced by using a standard technique of [12] that is used in [5], [3]. First we can compute a 2-approximate solution $\hat{\beta}$ using the scheme above. This takes $O(\log k \log m)$ phases. Since $\beta \leq \hat{\beta} \leq 2\beta$, if we multiply all the demands by $\hat{\beta}/2$, only at most T additional phases are needed to get an ε-approximation because $1 \leq \beta \leq 2$.

3.3 The running time

The number of iterations in each phase is bounded by the number of sources, which is at most $\min\{n,k\}$. Since we need $O(\log m \log k)$ phases to reduce β to less than 2, and at most $T = O(\varepsilon^{-2} \log m)$ additional phases to get an ε-approximation, the total number of iterations is $O(\min\{n,k\} \log m(\log k + \varepsilon^{-2}))$. In order to compute the total running time, we need to calculate the number of steps in the algorithm. At every step, the length of an edge increases by a factor of at least $1+\varepsilon$. At the beginning each edge has length $\delta/u(e)$ and at the end its legth is at most $(1+k\varepsilon)/u(e)$. So the number of steps exceeds the number of iterations by at most $m \log_{1+\varepsilon} \frac{1+k\varepsilon}{\delta} = O(\varepsilon^{-2} m(\log m + \log k))$. Hence the total number of steps is at most $\tilde{O}(\varepsilon^{-2}(m + \min\{n,k\})) = \tilde{O}(\varepsilon^{-2} m)$. Each step involves *one* run of Dijkstra's algorithm that takes $O(n \log n + m)$ time, and other computations that take at most $O(m)$ time, for a total of $O(m \log n)$ time per step. Putting all these together with the scaling procedure we get an algorithm that runs in time $\tilde{O}(\varepsilon^{-2}m^2)$.

THEOREM 3.1. *The algorithm described above is a FPTAS for the maximum concurrent multicommodity flow problem, that runs in $\tilde{O}(\varepsilon^{-2}m^2)$ time.*

4 Minimum cost concurrent flow

A cost version of the maximum concurrent flow problem is the following: in addition to demands and capacities, we are also given a budget B and a cost function $c : E \to \mathbb{R}^+$ for a unit of flow that passes through an edge, and we are asked to maximize the concurrent flow without violating the budget constraint. It is clear that if we have a PTAS for this problem, we have a PTAS for the minimum cost concurrent flow, by just using binary search to compute the minimum budget that satisfies all the demands (i.e. $\lambda \geq 1$). This search incurs a factor of at most $\log M$, where M is the biggest number used to specify capacities, demands or costs. Note that we can have different costs for different commodities, since there is a different primal variable for every commodity-path pair.

The analysis is almost identical to the analysis for the maximum concurrent flow above and in [5]. The dual has one extra variable ϕ for the budget constraint. The algorithm is the same as in the maximum concurrent flow case, but instead of l the length function we use is $l + c \cdot \phi$. Initially $\phi = \delta/B$. The flow send through at every step is the flow calculated above but scaled down (if necessary) so that its total cost is not more than B. Lengths l are updated as before, and the new ϕ becomes $\phi := \phi(1+\varepsilon \cdot \frac{\text{cost of flow sent}}{B})$. For the analysis one needs only observe that at every step either the length l of an edge or ϕ increases by a factor of at least $1+\varepsilon$ and at most $1+k\varepsilon$. The analysis above together with the analysis in [5] shows the following:

THEOREM 4.1. *There is a FPTAS that computes the minimum cost concurrent flow in time $\tilde{O}(\varepsilon^{-2}m^2 \log M)$.*

5 Concurrent flow for lossy networks

A generalization of the flow problems discussed so far involves the introduction of one more network parameter, the *gain factor* $\gamma : E \to \mathbb{R}^+$. An edge e with gain factor $\gamma(e) > 0$ allows a fraction $\gamma(e)$ of the flow that enters e to exit. Hence, in case $\gamma(e) > 1$, $\gamma(e) = 1$ or $\gamma(e) < 1$, the edge increases, conserves or 'leaks' the flow that passes through it. A network with a gain factor γ is a *generalized network* and the obvious extensions of flow problems to such networks are *generalized flow* problems. In the special case of $\gamma(e) \leq 1$ for all edges, the network is called *lossy*.

Our analysis applies directly to lossy networks, with the appropriate (more general) definition of a shortest path. [4] describes how a modification of Dijkstra's algorithm can be used to solve the generalized shortest path problem in the case of lossy networks. By adapting our algorithm to this framework very much in the same way the results of [3] were adapted to generalized networks in [4], we get the following:

THEOREM 5.1. *There is a FPTAS that computes the maximum concurrent flow for lossy networks in time $\tilde{O}(\varepsilon^{-2}m^2 \log M)$.*

THEOREM 5.2. *There is a FPTAS that computes the minimum cost concurrent flow for lossy networks in time $\tilde{O}(m^2(k + \varepsilon^{-2} \log M))$.*

Details will appear in the full version of this paper.

The running time of our algorithm in the case of the minimum cost concurrent flow is *not* independent of k. It would be interesting to remove this dependence, and to extend our results to any generalized network (not only lossy ones).

6 Acknowledgements

The author thanks Tami Carpenter for initiating this research and for many helpful discussions.

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows: theory, algorithms, and applications.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] Y. Aumann and Y. Rabani. An $O(logk)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM Journal on Computing*, **27**:1, pp. 291-301, 1998.

[3] L. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics*, **13**, 2000.

[4] L. Fleischer and K.D. Wayne. Fast and simple approximation schemes for generalized flow. In *Proceedings 10th ACM/SIAM SODA*, pp. 981-982, 1999.

[5] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings 39th IEEE FOCS*, pp. 300-309, November 1998.

[6] 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.

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

[8] D. Karger and S. Plotkin. Adding multiple cost constraints to combinatorial optimization problems, with applications to multicommodity flows. In *Proceedings of 27th ACM STOC*, pp. 18-25, 1995.

[9] 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.

[10] T. Leighton, F. Makedon, S. Plotkin, C. Stein, É. Tardos, and S. Tragoudas. Fast approximation schemes for multicommodity flow problems. *JCSS* **50**:228-243, 1995.

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

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

[13] T. Radzik. Fast deterministic approximation for the multicommodity flow problem. In *Proceedings of the 6th ACM/SIAM SODA*, pp. 486-492, 1995.

[14] F. Shahrokhi and D.W. Matula. The maximum concurrent flow problem. *JACM*, **37**:318-334, 1990.

[15] D. Shmoys. *Cut problems and their application to divide-and-conquer*, Chapter 5 in D.S. Hockbaum (Ed.) *Approximation algorithms for NP-hard problems*, PWS Publishing Company, Boston, 1997.

[16] N. Young. Randomized routing without solving the linear program. In *Proceedings of the 6th ACM/SIAM SODA*, pp. 170-178, 1995.