Fast Approximation Algorithms for Multicommodity Flow Problems*

Tom Leighton† Fillia Makedon‡ Serge Plotkin§

Clifford Stein¶ Éva Tardos|| Spyros Tragoudas**

April 19, 1993

*A preliminary version of this paper appeared in the *Proceedings of the 23rd Annual Symposium on Theory of Computing*, 1991, pp. 101-111.

†Department of Mathematics and Laboratory for Computer Science, MIT, Cambridge, MA. Research supported by DARPA under Contracts N00014-87-K-825 and N00014-86-K-0593, the Air Force under Contract OSR-89-0271, and the Army under Contract DAAL-03-86-K-0171.

‡Mathematics and Computer Science Department, Bradley Hall, Dartmouth College, Hanover, NH 03755.

§Department of Computer Science, Stanford University, Stanford CA. Research supported by NSF Research Initiation Award CCR-900-8226, by U.S. Army Research Office Grant DAAL-03-91-G-0102, by ONR Contract N00014–88–K–0166, and by a grant from Mitsubishi Electric Laboratories.

¶Laboratory for Computer Science, MIT, Cambridge, MA 02139. Support provided by NSF PYI Award CCR-89-96272 with matching support from UPS and Sun and by an AT&T Bell Laboratories Graduate Fellowship.

||School of Operations Research, Cornell University, Ithaca NY. Research supported in part by a David and Lucile Packard Fellowship and by the National Science Foundation, the Air Force Office of Scientific Research, and the Office of Naval Research, through NSF grant DMS-8920550.

**Department of Computer Science, Southern Illinois University, Carbondale, Illinois 62901. Supported by an ACM-SIGDA Design Automation Award and the CLEAR Center at UTD, UTD proposal #870049.

Proposed running head: Approximating multicommodity flow

Contact Author
Clifford Stein
545 Technology Square
MIT
Cambridge, MA 02139
(617) 253-6097
`cliff@theory.lcs.mit.edu`

After 8/15
Department of Mathematics and Computer Science
Bradley Hall
Dartmouth College
Hanover, NH 03755
`Clifford.Stein@dartmouth.edu`

Abstract

All previously known algorithms for solving the multicommodity flow problem with capacities are based on linear programming. The best of these algorithms [15] uses a fast matrix multiplication algorithm and takes $O(k^{3.5}n^3m^{.5}\log(nDU))$ time for the multicommodity flow problem with integer demands and at least $O(k^{2.5}n^2m^{.5}\log(n\epsilon^{-1}DU))$ time to find an approximate solution, where k is the number of commodities, n and m denote the number of nodes and edges in the network, D is the largest demand, and U is the largest edge capacity. Substantially more time is needed to find an exact solution. As a consequence, even multicommodity flow problems with just a few commodities are believed to be much harder than single-commodity maximum-flow or minimum-cost flow problems.

In this paper, we describe the first polynomial-time combinatorial algorithms for approximately solving the multicommodity flow problem. The running time of our randomized algorithm is (up to log factors) the same as the time needed to solve k single-commodity flow problems, thus giving the surprising result that approximately computing a k-commodity maximum-flow is not much harder than computing about k single-commodity maximum-flows in isolation.

In fact, we prove that a (simple) k-commodity flow problem can be approximately solved by approximately solving $O(k\log^2 n)$ single-commodity minimum-cost flow problems. Our k-commodity algorithm runs in $O(knm\log^4 n)$ time with high probability. We also describe a deterministic algorithm that uses an $O(k)$-factor more time. Given any multicommodity flow problem as input, both algorithms are guaranteed to provide a feasible solution to a modified flow problem in which all capacities are increased by a $(1+\epsilon)$-factor, or to provide a proof that there is no feasible solution to the original problem.

We also describe faster approximation algorithms for multicommodity flow problems with a special structure, such as those that arise in the "sparsest cut" problems studied in [8, 10, 9], and the uniform concurrent flow problems studied in [12, 9] if $k \leq \sqrt{m}$.

1 Introduction

The multicommodity flow problem involves simultaneously shipping several different commodities from their respective sources to their sinks in a single network so that the total amount of flow going through each edge is no more than its capacity. Associated with each commodity is a demand, which is the amount of that commodity that we wish to ship. Given a multicommodity flow problem, one often wants to know if there is a *feasible* flow, i.e. if it is possible to find a flow which satisfies the demands and obeys the capacity constraints. More generally, we might wish to know the maximum percentage z such that at least z percent of each demand can be shipped without violating the capacity constraints. The latter problem is known as the *concurrent flow problem*, and is equivalent to the problem of determining the minimum ratio by which the capacities must be increased in order to ship 100% of each demands.

In this paper, we describe the first combinatorial approximation algorithms for the concurrent flow problem. Given any positive ϵ, the algorithms find a feasible flow that ships at least $(1-\epsilon)z$ percent of each demand, where z is the maximum percentage obtainable. The running times of the algorithms depend polynomially on ϵ^{-1}, and are significantly better than those of previous algorithms when ϵ is a constant. More specifically, we prove the following result. (Throughout, we use n, m and k to denote the number of nodes, edges and commodities, we assume that the demands and the capacities are integral, and use D and U to denote the largest demands and capacities, respectively.)

Theorem 1.1 For any fixed $\epsilon > 0$, a $(1-\epsilon)$-factor approximation to the simple[1] concurrent flow problem can be found by a randomized algorithm in $O(kmn \log k \log^3 n)$ time and a deterministic algorithm in $O(k^2 mn \log k \log^3 n)$ time, where the constant depends on ϵ.

Our expected running time is the same (up to polylog factors) as the time needed to compute k maximum-flows, thus giving the surprising result that approximately computing a k-commodity concurrent flow is about as difficult as computing k single commodity maximum-flows. In fact, we formally prove that a k-commodity flow problem can be approximately solved by approximately solving $O(k \log k \log n)$ min-cost flow problems.

The running times in the above theorem can be improved when k is large. Let k^* denote the number of different sources. In both the randomized and the deterministic algorithm we can replace k in the running time by k^* at the expense of having to replace one of the $\log n$ terms by a $\log(nU)$. Notice that k^* is at most n for all multicommodity flow problems.

As a consequence of our approximation algorithm for the concurrent flow problem, we obtain a *relaxed decision procedure* for multicommodity flow feasibility. In particular, given a multicommodity flow problem, we can either prove that it is infeasible, or give a feasible flow for the problem in which every capacity is increased by a factor of $1+\epsilon$. Since in practice, the input to a multicommodity flow problem may have some measurement error, by making ϵ small enough, we can obtain a procedure for determining feasibility up to the precision of the input data.

The only previous algorithms for solving (or approximately solving) the general concurrent flow problem rely on general purpose linear programming algorithms. The concurrent flow problem can be formulated as a linear program in $O(mk)$ variables and $O(nk+m)$ constraints. Linear program-

[1] By *simple*, we mean that each commodity has a single source and a single sink.

ming can be used to solve the problem optimally in polynomial time. Kapoor and Vaidya [7] gave a method to speed up the matrix inversions involved in Karmarkar-type algorithms for multicommodity flow problems; combining their technique with Vaidya's new linear programming algorithm that uses fast matrix multiplication [15] yields a time bound of $O(k^{3.5}n^3m^{.5}\log(nDU))$ for the concurrent flow problem with integer demands and at least an $O(k^{2.5}n^2m^{.5}\log(n\epsilon^{-1}DU))$ time bound for the approximation problem. When ϵ is not too small (e.g. if ϵ is constant), then the running time of our algorithm is much faster than that of the previous algorithms for most multicommodity flow problems. In addition, the fact that our algorithm consists of only $O(k\log k\log n)$ minimum-cost flow computations means that it might be more suitable for implementation in practice. (Minimum-cost flow problems are efficiently handled by the network simplex algorithm in practice.)

The only previous combinatorial polynomial approximation algorithms for concurrent flow problems handle only the special case when all the capacities are 1. For this special case, Shahrokhi and Matula [12] gave an algorithm that ran in $O(nm^7)$ time. A faster algorithm was later given by Klein, Plotkin, Stein, and Tardos [9] which runs in expected $O((k+m)(m+n\log n)\log n)$ time. Our new algorithm can be applied to this special case and gives improved bounds when $k < \sqrt{m}/(\log n\log k)$.

Our algorithm is similar in spirit to those of [12] and [9] in that we start with a flow that satisfies the demands but not the capacity constraints, and then we iteratively reroute parts of the flow so as to produce a flow that is closer to optimal. However, our algorithm can handle networks with arbitrary capacities. Our approach differs from that in previous work in that we are able to reroute an entire commodity during each iteration instead of only a single path of flow. To do this, we compute a minimum-cost flow in an auxiliary graph and reroute a portion of the flow accordingly. As a consequence, we are able to make much greater progress during each iteration. Of course, the time to run each iteration goes up, but the tradeoff proves to be worthwhile since the improvement obtained in each iteration is large enough so that we need to solve only $O(k\log k\log n)$ minimum-cost flow problems in order to get an approximately optimal solution.

The running times of the presented algorithms depend polynomially on ϵ^{-1}. The deterministic algorithm runs in time proportional to ϵ^{-2} and the randomized one runs in time proportional to ϵ^{-3}. Goldberg [4] and Grigoriadis and Khachiyan [6] have shown how to improve the dependence on ϵ of the randomized algorithm to ϵ^{-2}.

Leighton and Rao [10] have shown how to use an approximately optimal solution to a concurrent flow problem to find an approximately sparsest cut in a graph. The sparsity of a cut is defined to be the ratio of the number of edges crossing the cut to the product number of nodes on the two sides of the cut. As a consequence, they also showed how to approximately solve a wide variety of NP-hard graph problems, including minimum feedback arc set, minimum cut linear arrangement, and minimum area layout. This result has recently been generalized by Klein, Agrawal, Ravi and Rao [8] to find an approximately most congested cut in a general concurrent flow problem. The congestion of a cut is the ratio between the capacity of the cut and the demand crossing the cut. Given a solution to the concurrent flow problem, they showed how to approximately solve a variety of NP-hard problems, including minimum deletion of clauses of a 2-*CNF*≡ formula, via minimization, minimum chordalization of a graph and register sufficiency.

The previously known concurrent flow algorithms [12, 9] cannot be used in these approximation algorithms except in the special case of problems without capacities, or edge weights. The only algorithms previously known for the capacitated case used linear programming. Using the results in this paper, we can now efficiently approximate all the problems considered above for arbitrary edge-weighted and node-weighted graphs (when appropriate). For example, we can prove the following result.

Theorem 1.2 The most congested cut in a graph with integral demands and capacities can be approximated within a factor of $O(\log n \log kD)$ in expected $O(k^* nm \log^3 n \log nU)$ time, where k^* is the number of different sources among the commodities.

The concurrent flow algorithm can also be used to give an $O(n^2 m \log^3 n \log nU)$ expected time algorithm for finding a cut in an edge (and node) weighted graph that is sparsest up to a factor of $O(\log n)$. For the important special case of regular graphs and unit node weights, we can further improve this bound to $O(m^2 \log^3 n)$.

Our model of computation is the RAM. We shall use the elementary arithmetic operations (addition, subtraction, comparison, multiplication, and integer division), and count each of these as a single step. All numbers occurring throughout the computation will have at most $O(\log(nU))$ bits. For ease of exposition, in Section 4 we shall use a model of computation that allows exact arithmetic on real numbers and we shall assume that exponentiation is a single step. In Section 5 we show how to convert the results to the usual RAM model.

2 Preliminaries and Definitions

An instance of the *simple multicommodity flow problem* consists of an undirected graph $G = (V, E)$, a non-negative capacity $u(vw)$ for every edge $vw \in E$, and a specification of k commodities, numbered 1 through k, where the specification for commodity i consists of a source-sink pair $s_i, t_i \in V$ and a non-negative demand d_i. We will denote the number of different sources by k^*, the number of nodes by n, and the number of edges by m. For notational convenience we assume that $m \geq n$, and that the graph G is connected and has no parallel edges. Also, for notational convenience, we arbitrarily direct each edge. If there is an edge directed from v to w, this edge is unique by assumption, and we denote it by vw. We assume that the capacities and the demands are integral, and denote the largest capacity by U and the largest demand by D.

A multicommodity flow f consists of a function $f_i(vw)$ on the edges of G for every commodity i, which represents the *flow* of commodity i on edge vw. If the flow of commodity i on edge vw is oriented in the same direction as edge vw, then $f_i(vw)$ will be positive, otherwise it will be negative. The signs only serve to indicate the direction of the flows. For every commodity i we require the *conservation constraints*:

$$(1) \quad \sum_{wv \in E} f_i(wv) - \sum_{vw \in E} f_i(vw) = 0 \quad \text{for every node } v \notin \{s_i, t_i\}.$$

We require also that $\sum_{vw \in E} f_i(vw) = d_i$ for $v = s_i$. We define the value of the total flow on edge vw to be $f(vw) = \sum_i |f_i(vw)|$, and say that a multicommodity flow f in G is *feasible* if $f(vw) \leq u(vw)$ for all edges vw. (Note that $f(vw)$ is always non-negative.)

We consider the optimization version of this problem, called the *simple concurrent flow problem*, first defined by Shahrokhi and Matula [12]. In this problem the objective is to compute the maximum possible value z such that there is a feasible multicommodity flow with demands $z \cdot d_i$ for $i = 1, \ldots, k$. We call z the *throughput* of the multicommodity flow. An equivalent formulation of the concurrent flow problem is to compute the minimum $\lambda = 1/z$ such that there is a feasible flow with demands d_i and capacities $\lambda \cdot u(vw)$. We shall use the notation $\lambda(vw)$ to denote the *congestion* $f(vw)/u(vw)$ of an edge $vw \in E$, $\lambda = \max_{vw \in E} \lambda(vw)$, and λ^* to denote the optimal (minimum) value of λ.

A multicommodity flow f satisfying the demands d_i is *ϵ-optimal* if λ is at most a factor $(1+\epsilon)$ more than the minimum possible value. The *approximation problem* associated with the concurrent flow problem is to find an ϵ-optimal multicommodity flow f. We shall assume implicitly throughout that ϵ is at least inverse polynomial in n and is at most $1/9$. If ϵ is bigger than $1/9$, we can run the algorithm for $\epsilon = 1/9$. If ϵ^{-1} is greater than any polynomial in n, our algorithms will still yield a correct solution. However, in this case, the running times of our algorithms will be somewhat greater and will be dominated by the time to solve the problem exactly.

We can extend all the results in the paper to the case where the input graph is directed. In this case we require that all flows are non-negative and oriented in the same direction as the input graph. It is easy to verify that all the results in this paper carry through to this case. Henceforth, we focus only on the undirected case.

The *general multicommodity flow problem* is a natural extension of the simple problem when each commodity has more than one source and sink. For every commodity i we are given a demand vector $\hat{d}_i(v)$. (A negative demand denotes a supply.) We require that $\sum_v \hat{d}_i(v) = 0$ and we shall use D_i to denote $\max_v\{|\hat{d}_i(v)|\}$. The conservation constraints of equation (1) are replaced by the more general conservation constraints:

$$(2) \qquad \sum_{wv \in E} f_i(wv) - \sum_{vw \in E} f_i(vw) = \hat{d}_i(v) \quad \text{for every commodity } i \text{ and every node } v.$$

Our algorithm can be applied to solve both simple and general multicommodity flow problems. As we will see below, our algorithm is slightly faster when applied to a simple multicommodity flow problem; the main point in introducing the general problem is to reduce the number of commodities. Every simple concurrent flow problem is equivalent to a general concurrent flow problem with at most n commodities. Because of this fact and because the running times of our algorithms are proportional to the number of commodities, we will assume that the number of commodities is polynomial in n; this will simplify the expressions for the running times.

We can convert a simple concurrent flow problem to a general concurrent flow problem with k^* commodities by combining those commodities which share a source. For each source s we define a demand vector $\hat{d}_s(v)$ as follows: for each commodity i with $s_i = s$ we set $\hat{d}_s(t_i) = d_i$; we set $\hat{d}_s(s) = -\sum\{d_i : s_i = s\}$; all other demands are set to zero.

Lemma 2.1 Consider a simple k commodity concurrent flow problem and the corresponding k^*-commodity problem defined above. Any feasible solution to one can be converted to a solution to the other with the same value of λ. The conversion of a solution for the k^*-commodity problem to one for the k commodity problem can be done in $O(k^*nm)$ time, or in $O(k^*m \log n)$ time using the dynamic tree data structure.

Proof: The conversion of a solution of the simple concurrent flow with k commodities into a solution of the k^*-commodity problem is straightforward. Assume that we are given a solution to the general concurrent flow problem with k^* commodities. Decompose the flow of each commodity into paths and cycles and combine the flows on paths that have the same source and sink nodes, disregarding the cycles. The running time of this procedure is dominated by the time it takes to decompose flows into paths and cycles, which yields the claim of the lemma. ∎

Notice that the sources and sinks play a symmetric role in the (undirected) problem, and hence k^* in the lemma could have been defined as the number of nodes in a subset that contains an endpoint of each commodity. While finding a minimum such node set is NP-complete, we mention this formulation because in some cases it leads to an efficiently computable k^* which is smaller than the one defined above.

Lemma 2.1 implies that one can replace most of the bounds that are dependent on k by ones that are dependent on k^*. Throughout this paper, unless we explicitly state that a bound is for the *simple* concurrent flow problem, k can be replaced by k^* when applied to a simple concurrent flow problem.

The main subroutine of our algorithm is a minimum-cost flow computation (of a single commodity). We will use the following, slightly unconventional definition. Given a cost vector $c \in \mathbf{R}^E$, the cost of a flow f_i is $\sum_{vw \in E} c(vw)|f_i(vw)|$. Given a demand vector $\hat{d}_i(v)$, and capacities u', the *minimum-cost flow problem* is the problem of finding a flow of minimum cost which satisfies the conservation constraints (2) and has $|f_i(vw)| \leq u'(vw)$ for every edge vw. The residual graph of a flow f_i is the directed graph consisting of the set of edges for which $f_i(vw) < u'(vw)$ and the reversal of the set of edges for which $f_i(vw) > -u'(vw)$. In Section 6.2, we will need to work with the linear-programming dual of a minimum-cost flow. The dual variables on the nodes are commonly referred to as *prices*, and will be denoted by p. A *price function* is a vector $p \in \mathbf{R}^V$. The *reduced cost* of an edge $vw \in E$ is $c(vw) + p(v) - p(w)$, and the negative of this for reverse edges. Linear programming duality implies that a flow f_i is of minimum-cost if and only if there exists a price function p, such that the reduced cost of the edges in the residual graph of f_i are nonnegative (complementary slackness conditions).

Linear programming duality can also be used to give a characterization of the optimum solution for the concurrent flow problem. Let $\ell : E \to \mathbf{R}$ be a nonnegative *length* function. For nodes $v, w \in V$ let $dist_\ell(v, w)$ denote the length of the shortest path from v to w in G with respect to the length function ℓ. It is easy to see that for a simple multicommodity flow f satisfying the demands d_i and capacities $\lambda \cdot u(vw)$, and any length function ℓ,

$$(3) \quad \lambda \sum_{vw \in E} \ell(vw)u(vw) \geq \sum_{vw \in E} f(vw)\ell(vw) = \sum_{i=1}^{k} \sum_{vw \in E} \ell(vw)|f_i(vw)| \geq \sum_{i=1}^{k} dist_\ell(s_i, t_i)d_i.$$

The following theorem is a special case of the linear programming duality theorem.

Theorem 2.2 A simple multicommodity flow f minimizes λ if and only if there exists a nonzero length function ℓ for which all terms in (3) are equal.

3 Relaxed Optimality Conditions

Theorem 2.2 is a characterization of optimality that relates the value of λ to the lengths of the shortest path for each commodity. To drive our algorithm, we will use a slightly different characterization, one which relates the value of λ to the costs of minimum cost flows in appropriately derived graphs. While these characterizations can be proven to be equivalent, by measuring optimality in terms of minimum-cost flows, we are able to develop faster algorithms.

Let ℓ be a nonnegative length function on the edges, f a multicommodity flow, and $\lambda = \max_{vw \in E} \lambda(vw)$. Let C_i be the cost of the current flow for commodity i, using ℓ as the cost function. For a commodity i, let $C_i^*(\lambda)$ be the value of a minimum-cost flow f_i^* satisfying the demands of commodity i, subject to costs ℓ and capacities $\lambda \cdot u(vw)$, *i.e.* let f_i^* be a flow that satisfies $|f_i^*(vw)| \leq \lambda \cdot u(vw)$ and minimizes the cost $C_i^*(\lambda) = \sum_{vw} |f_i^*(vw)| \ell(vw)$. For brevity we shall sometimes use C_i^* to abbreviate $C_i^*(\lambda)$.

Theorem 3.1 For a (general) multicommodity flow f satisfying capacities $\lambda \cdot u(vw)$, and a length function ℓ,

$$(4) \quad \lambda \sum_{vw \in E} \ell(vw) u(vw) \geq \sum_{i=1}^{k} \sum_{vw \in E} |f_i(vw)| \ell(vw) = \sum_{i=1}^{k} C_i \geq \sum_{i=1}^{k} C_i^*(\lambda).$$

A multicommodity flow f minimizes λ if and only if there exists a nonzero length function ℓ for which all the above terms are equal.

We would like to be able to say that the ratio of the last term and the multiplier of λ in the first term gives a lower bound on the optimal value λ^*. The analogous statement for the inequality (3) is obvious, because neither of the two terms depend on λ. In Theorem 3.1 the last term, $\sum_i C_i^*(\lambda)$, depends on λ. Observe, however, that the minimum cost of a flow subject to capacity constraints $\lambda \cdot u(vw)$ cannot increase if λ increases.

Lemma 3.2 Suppose that there exists a multicommodity flow satisfying capacities $\lambda \cdot u(vw)$. Then for any length function ℓ the value $\sum_{i=1}^{k} C_i^*(\lambda) / (\sum_{vw \in E} \ell(vw) u(vw))$ is a lower bound on λ^*.

The goal of our algorithms is to find a multicommodity flow f and a length function ℓ such that this lower bound is within a $(1+\epsilon)$ factor of optimal, i.e. $\lambda \leq (1+\epsilon) \sum_{i=1}^{k} C_i^*(\lambda) / (\sum_{vw \in E} \ell(vw) u(vw))$. In this case, we say that f and ℓ are *ϵ-optimal.* Note that we are using ϵ-optimal to refer both to a flow and a flow and length function pair. Recall, that a flow is ϵ-optimal if it has congestion at most $(1+\epsilon)$ times the minimum possible congestion. If f and ℓ are ϵ-optimal then Lemma 3.2 implies that f is ϵ-optimal. In order to recognize that a flow is ϵ-optimal, we will need to have the "right" length function.

The complementary slackness conditions given by linear programming can be reformulated in terms of conditions on edges and individual commodities. A multicommodity flow f has minimum λ if and only if there exists a nonnegative and non-zero length function ℓ such that:

1. for every edge $vw \in E$, either $\ell(vw) = 0$ or $f(vw) = \lambda \cdot u(vw)$,
2. for every commodity i, $C_i = C_i^*(\lambda)$.

These two conditions characterize when f and ℓ are *optimal*; we shall give two conditions on a multicommodity flow f and a length function ℓ such that together they imply that f and ℓ are

ϵ-*optimal.* These conditions will be relaxed versions of the complementary slackness conditions above. Similar relaxed versions of Theorem 2.2 were used in [9].

Let $\epsilon > 0$ be an error parameter, f a multicommodity flow satisfying capacities $\lambda \cdot u(vw)$, and ℓ a length function. We say that a commodity i is ϵ-*good* if

$$C_i - C_i^*(\lambda) \quad \leq \quad \epsilon C_i + \epsilon \frac{\lambda}{k} \sum_{vw \in E} u(vw)\ell(vw).$$

Otherwise, we say that the commodity is ϵ-*bad.* Intuitively, a commodity is ϵ-good if it is almost as cheap as the minimum cost possible for that commodity or it is at most a small fraction of $\lambda \sum_{vw \in E} u(vw)\ell(vw)$, the total cost of the network. We use this notion in defining the following *relaxed optimality conditions* (with respect to a multicommodity flow f that satisfies capacity constraints $\lambda \cdot u(vw)$, a length function ℓ and an error parameter ϵ):

(R1) For every edge $vw \in E$ either $(1+\epsilon)f(vw) \geq \lambda \cdot u(vw)$
or $u(vw)\ell(vw) \leq \frac{\epsilon}{m} \sum_{v'w' \in E} u(v'w')\ell(v'w')$.

(R2) $$\sum_{i\ \epsilon\text{-bad}} C_i \leq \epsilon \sum_{i=1}^{k} C_i.$$

By a proof similar to that of Theorem 3.2 of [9], we can show that if we can satisfy the relaxed optimality conditions then we actually have an $O(\epsilon)$-optimal flow. A complete version of the proof appears in [13].

Theorem 3.3 Suppose f, ℓ, and ϵ satisfy the relaxed optimality conditions and $\epsilon < 1/9$. Then f is $O(\epsilon)$-optimal, *i.e.* λ is at most a factor $(1+9\epsilon)$ more than the minimum possible value.

As we shall see in the next section, the relaxed optimality conditions will guide our algorithm.

4 Solving Concurrent Flows

In this section, we give approximation algorithms for the concurrent flow problem. As the basic step of our algorithm is finding a minimum-cost flow, we bound the time needed to find a concurrent flow in terms of a number of minimum-cost flow computations. In Section 4.1, we will show how to find a "good" initial solution to the given concurrent flow problem. In Section 4.2, we will describe procedure DECONGEST, which takes a flow with congestion λ and produces a new flow that is either 9ϵ-optimal or has congestion at most $\lambda/2$. Finally, in Section 4.3 we will use these results to give bounds for how long it takes to solve a concurrent flow problem in terms of the number of minimum cost flow computations for two cases – a case of ϵ being a fixed constant, and a more involved case in which ϵ is $o(1)$.

For simplicity of presentation, throughout this section we shall use a model of computation that allows the use of exact arithmetic on real numbers and provides exponentiation as a single step. In Section 5 we will show how to modify our algorithms to work in the standard RAM model. The question of which minimum-cost flow algorithm to use is also deferred to Section 5, where we show that the cost-scaling algorithm of Goldberg and Tarjan [5] is a good choice in most instances.

4.1 Finding an Initial Solution

To find an initial solution, we separately route each commodity i. For commodity i, we find λ_i and a flow f_i, such that f_i satisfies the demands of this commodity and obeys capacity constraints $\lambda_i \cdot u(vw)$. Let λ_i^* denote the minimum possible λ_i. For each commodity i we have $\lambda_i^* \leq \lambda^*$. If the commodity has a single sink and a single source with demand d_i, then the value of the maximum-flow in the graph with capacities $u(vw)$ is d_i/λ_i^*. If the commodity has more than one sink or source, then a $\lambda_i \leq 2\lambda_i^*$ can be found by binary search. Recall that D_i denotes the magnitude of the maximum demand for commodity i. Since λ_i^* must be between $D_i/(nU)$ and nD_i, we need to try only $O(\log nU)$ values. Therefore, we have the following lemma.

Lemma 4.1 An initial multicommodity flow satisfying demands such that $\lambda \leq k\lambda^*$ can be found by k maximum-flow computations for the case where each commodity has a single source and a single sink. In the case of multiple sources and multiple sinks a flow such that $\lambda \leq 2k\lambda^*$ can be found by $O(k \log nU)$ maximum-flow computations.

4.2 Rerouting Flow

Now, we show how, given a flow, we can iteratively reroute commodities in order to produce a new flow that is closer to optimality. We give a procedure DECONGEST which takes a flow f with congestion λ_0 and produces a new flow f' that is either 9ϵ-optimal or has congestion $\lambda' \leq \lambda_0/2$.

The basic idea is that the procedure reroutes an appropriately chosen fraction of the flow of an ϵ-bad commodity onto the edges of a minimum-cost flow associated with this commodity (as described below), in order to reduce congestion. We use a length function $\ell(vw) = e^{\alpha\lambda(vw)}/u(vw)$, where the value of α will be chosen later. This length function has the property that the length of an edge vw is a function of the congestion, *i.e.* the fraction (possibly greater than 1) of the capacity of that edge which is being used. Intuitively, by using lengths as costs in the computation of the minimum-cost flow, we are penalizing edges with high congestion.

At the beginning of procedure DECONGEST, α is chosen so that Relaxed Optimality Condition $R1$ is always satisfied. The act of rerouting flow gradually enforces Relaxed Optimality Condition $R2$. When both conditions are satisfied, then Theorem 3.3 can be used to infer that f is $O(\epsilon)$-optimal. Alternatively, DECONGEST terminates if λ decreases by more than a factor of 2.

More formally, procedure DECONGEST (see Figure 1) takes as input a multicommodity flow f with congestion λ_0, where f satisfies the demands, and an error parameter ϵ. In each iteration, we first choose an ϵ-bad commodity i, and formulate an auxiliary minimum-cost flow problem. The demand of each node v in the auxiliary problem is equal to $\hat{d}_i(v)$, and the desired flow $f_i^*(vw)$ is constrained to be between $-\lambda \cdot u(vw)$ and $\lambda \cdot u(vw)$, where λ is the current congestion. The objective is to minimize $C_i^*(\lambda) = \sum_{vw} |f_i^*(vw)|\ell(vw)$. Given an optimal solution to this problem, we reroute a $\sigma = \frac{\epsilon}{8\alpha\lambda}$ fraction of the flow f_i onto the edges of f_i^* by setting $f_i(vw) \leftarrow (1-\sigma)f_i(vw) + \sigma f_i^*(vw)$, recompute the length function, and repeat. Upon termination, DECONGEST returns an improved flow f which is either 9ϵ-optimal or has maximum congestion $\lambda \leq \lambda_0/2$.

We now show that we can always choose α so that Relaxed Optimality Condition $R1$ is satisfied.

Lemma 4.2 If f is a multicommodity flow which satisfies demands and $\alpha \geq (1+\epsilon)\lambda^{-1}\epsilon^{-1}\ln(m\epsilon^{-1})$ then f and length function $\ell(vw) = e^{\alpha\lambda(vw)}/u(vw)$ satisfy Relaxed Optimality Condition $R1$.

DECONGEST(f,ϵ)
$\lambda_0 \leftarrow \lambda$; $\alpha \leftarrow 2(1+\epsilon){\lambda_0}^{-1}\epsilon^{-1}\ln(m\epsilon^{-1})$.
While $\lambda \geq \lambda_0/2$ **and** f and ℓ are not 9ϵ-optimal
  $\sigma \leftarrow \frac{\epsilon}{8\alpha\lambda}$.
  For each edge vw, $\ell(vw) \leftarrow e^{\alpha\lambda(vw)}/u(vw)$.
  Find an ϵ-bad commodity i.
  Formulate an auxiliary minimum-cost flow problem on G, where each node v has demand $\hat{d}_i(v)$, and the flow on edge vw is constrained to be between $-\lambda \cdot u(vw)$ and $\lambda \cdot u(vw)$.
  Compute flow f_i^* that minimizes $C_i^*(\lambda) = \sum_{vw} |f_i^*(vw)|\ell(vw)$ in the auxiliary problem.
  For all $vw \in E$, $f_i(vw) \leftarrow (1-\sigma)f_i(vw) + \sigma f_i^*(vw)$.
Return f

Figure 1: Procedure DECONGEST

Proof: We show that if an edge $v'w'$ violates the first part of Relaxed Optimality Condition $R1$ then it must satisfy the second part. For this edge $\lambda \cdot u(v'w') > (1+\epsilon)f(v'w')$. Let v^*w^* be the edge such that $\lambda(v^*w^*) = \lambda$. Then

$$\frac{\sum_{vw} u(vw)\ell(vw)}{u(v'w')\ell(v'w')} \geq \frac{u(v^*w^*)\ell(v^*w^*)}{e^{\alpha\lambda/(1+\epsilon)}} \geq \frac{e^{\alpha\lambda}}{e^{\alpha\lambda/(1+\epsilon)}} \geq \frac{m}{\epsilon}.$$

∎

In the beginning of procedure DECONGEST, α is set equal to $2(1+\epsilon){\lambda_0}^{-1}\epsilon^{-1}\ln(m\epsilon^{-1})$ and throughout DECONGEST $\lambda > \lambda_0/2$. Therefore $\alpha \geq (1+\epsilon)\lambda^{-1}\epsilon^{-1}\ln(m\epsilon^{-1})$ throughout.

To measure progress of our algorithm, we introduce a potential function $\Phi = \sum_{vw} u(vw)\ell(vw)$. We now show that rerouting the right amount of flow results in a significant decrease in Φ.

Lemma 4.3 Let i be an ϵ-bad commodity, $\epsilon \leq 1$, and let f_i^* be a minimum-cost flow for this commodity, as described above and let $\frac{\epsilon}{16\alpha\lambda} \leq \sigma \leq \frac{\epsilon}{8\alpha\lambda}$. Let the new flow for commodity i be defined by $f_i(vw) \leftarrow (1-\sigma)f_i(vw) + \sigma f_i^*(vw)$. Then $\Phi - \Phi' = \Omega(\frac{\epsilon^2}{k}\Phi)$, where Φ' is the value of the potential function associated with the new flow.

Proof: Denote by $\ell(vw)$ and $\ell'(vw)$ the length of edge vw before and after rerouting, respectively. Let $\delta(vw)$ denote the increase in flow on vw due to rerouting. Recall that, after rerouting, the flow of the rerouted commodity i on vw is $|(1-\sigma)f_i(vw) + \sigma f_i^*(vw)|$, and hence $|\delta(vw)| = \sigma|f_i^*(vw) - f_i(vw)| \leq \sigma(|f_i(vw)| + |f_i^*(vw)|)$. Moreover, since both f_i and f_i^* have congestion at most λ, $|\delta(vw)| \leq 2\sigma\lambda u(vw)$.

By definition of the length function, $\ell'(vw) = e^{\alpha(f(vw)+\delta(vw))/u(vw)}/u(vw) = e^{\alpha f(vw)/u(vw)+\eta}/u(vw)$, where $\eta = \alpha\delta(vw)/u(vw)$. Observe that $|\eta| \leq 2\alpha\sigma\lambda \leq \epsilon/4 \leq 1/4$. Using the Taylor series, we see that $|\eta| \leq \epsilon/4 \leq 1/4$ implies that for all x, $e^{x+\eta} \leq e^x + \eta e^x + \frac{\epsilon}{2}|\eta|e^x$. Therefore, we have:

$$\begin{aligned}
\ell'(vw) - \ell(vw) &\leq \eta\ell(vw) + \frac{\epsilon}{2}|\eta|\ell(vw) \\
&\leq \frac{\alpha\sigma(|f_i^*(vw)| - |f_i(vw)|)}{u(vw)}\ell(vw) + \frac{\epsilon\alpha\sigma(|f_i(vw)| + |f_i^*(vw)|)}{2u(vw)}\ell(vw).
\end{aligned}$$

We use this bound to estimate the decrease in the potential function.

$$\begin{aligned}\Phi - \Phi' &= \sum_{vw \in E} (\ell(vw) - \ell'(vw))u(vw) \\ &\geq \alpha\sigma \sum_{vw} (|f_i(vw)| - |f_i^*(vw)|)\ell(vw) - \alpha\sigma\frac{\epsilon}{2}\sum_{vw}(|f_i(vw)| + |f_i^*(vw)|)\ell(vw).\end{aligned}$$

Using that $\sum_{vw}(|f_i(vw)| + |f_i^*(vw)|)\ell(vw) \leq C_i + C_i^*(\lambda) \leq 2C_i$ and the fact that commodity i is ϵ-bad we get

$$\Phi - \Phi' \geq \alpha\sigma(C_i - C_i^*(\lambda)) - \alpha\sigma\epsilon C_i \geq \alpha\sigma\left(\epsilon C_i + \epsilon\lambda \frac{\sum_{vw}\ell(vw)u(vw)}{k}\right) - \alpha\sigma\epsilon C_i = \frac{\alpha\sigma\epsilon\lambda}{k}\Phi. \tag{5}$$

Plugging in the value of σ from the statement of the lemma, we get that the decrease is $\Omega(\frac{\epsilon^2}{k}\Phi)$. ∎

Theorem 4.4 Procedure DECONGEST terminates in $O(\epsilon^{-3}k\log n)$ iterations. If the initial congestion λ_0 is $O(\epsilon)$-optimal then DECONGEST terminates in $O(\epsilon^{-2}k\log n)$ iterations.

Proof: Theorem 3.3 implies that if f and ℓ satisfy both of the relaxed optimality conditions than they are 9ϵ-optimal. By Lemma 4.2, the Relaxed Optimality Condition $R1$ is maintained throughout all iterations. If f is not yet 9ϵ-optimal then Relaxed Optimality Condition $R2$ is not satisfied. Hence there exists an ϵ-bad commodity. But every rerouting of flow from an ϵ-bad commodity to the corresponding minimum-cost flow results in a reduction in Φ of at least $\Omega(\frac{\epsilon^2}{k}\Phi)$. Since $1 - x < e^{-x}$, it follows that every $O(k\epsilon^{-2})$ iterations reduce Φ by at least a constant factor.

Next we bound the number of times Φ can be reduced by a constant factor. Let λ_0 be the congestion of the initial flow. For each edge vw, $\lambda(vw) \leq \lambda_0$, so initially, $\Phi \leq me^{\alpha\lambda_0}$. We know that in the beginning of the last iteration at least one edge has congestion at least $\lambda_0/2$, so $\Phi \geq e^{\alpha\lambda_0/2}$. Thus the number of times Φ can decrease by a constant factor is $O(\alpha\lambda_0 + \log m) = O(\alpha\lambda_0)$. Combining this with the number of iterations needed to reduce Φ by a constant factor and plugging in the value of α, we get that the total number of iterations is $O(\epsilon^{-3}k\log(n\epsilon^{-1}))$. We have assumed that ϵ is at least inverse polynomial in n, so this is in fact $O(\epsilon^{-3}k\log n)$.

If the initial flow is $O(\epsilon)$-optimal then we know that throughout DECONGEST , λ will never go below $\lambda_0/(1+O(\epsilon))$. Thus, we have the tighter bound of $e^{\alpha(1+O(\epsilon))^{-1}\lambda_0} \leq \Phi \leq me^{\alpha\lambda_0}$. Combining this with the number of iterations needed to reduce Φ by a constant factor and plugging in the value of α, we get that the total number of iterations is $O(\epsilon^{-2}k\log n)$. ∎

The only computation-intensive part of DECONGEST is finding an ϵ-bad commodity and computing minimum-cost flows. All the rest can be done in $O(m)$ time. The simplest way to find an ϵ-bad commodity is to compute the costs $C_i = \sum_{vw\in E}|f_i(vw)|\ell(vw)$ and the costs of the minimum-cost flows and compare them. In the worst case we need to check all k commodities. Hence, an iteration can be implemented in the time it takes to perform k minimum-cost flow computations.

As in [9], we can perform this computation more efficiently by using a simple randomized strategy. If we compute the cost C_i of each commodity and then randomly choose a commodity with probability proportional to its cost, then with probability of at least ϵ, we have chosen an

ϵ-bad commodity. By computing a single minimum-cost flow we can check whether the commodity is indeed ϵ-bad. We expect to perform this computation ϵ^{-1} times, and hence an iteration can be implemented in expected time equal to $O(mk)$ plus ϵ^{-1} times the time to perform a minimum-cost flow calculation.

Observe that if $k \leq n$ (this will be the case when Lemma 2.1 is applied) the time to compute the cost of all current flows is dominated by the time to compute a minimum-cost flow. If the time required to compute the costs of the k commodities is not dominated, we can use a strategy similar to that of [9] in which we pick an edge with probability proportional to the cost of flow through this edge, and then a commodity with probability proportional to the cost of flow of this commodity through this edge, and reduce the time for random selection from $O(km)$ to the minimum of $O(m+k)$ and $O(m \log k)$.

After every k iterations we can compute minimum-cost flows associated with all the flows and determine whether the current flow is 9ϵ-optimal. Therefore, we can implement DECONGEST as a Las-Vegas algorithm. Note that this results in at most a factor of 2 increase in the number of minimum-cost flows computed during the execution of DECONGEST. Combining Theorem 4.4 and the above discussion we get the following theorem.

Theorem 4.5 Procedure DECONGEST can be implemented randomly using an expected $O(\epsilon^{-4} k \log n)$ minimum-cost flow computations and $O(\epsilon^{-4} mk \log n \log k)$ additional time, or deterministically using $O(\epsilon^{-3} k^2 \log n)$ minimum-cost flow computations, assuming that exponentiation can be implemented in $O(1)$ time. If the initial congestion λ_0 is $O(\epsilon)$ optimal, then both the randomized and deterministic versions of DECONGEST can be implemented in $\Omega(\epsilon^{-1})$ less time.

4.3 Putting It Together

We consider two cases for solving a concurrent flow problem. We first consider the case when ϵ is a fixed constant less than $1/9$. In this case, we first find an initial solution by solving $O(k \log(nU))$ maximum flow problems, as is discussed in Section 4.1. This gives us a flow with $\lambda \leq 2k\lambda^*$. We then call DECONGEST $O(\log k)$ times in order to produce a flow such that $\lambda \leq (1+9\epsilon)\lambda^*$. Applying the first part of Theorem 4.5 we get the following result:

Theorem 4.6 For a constant ϵ, an ϵ-optimal solution for the concurrent flow problem can be found after initialization (Lemma 4.1) by a randomized algorithm that uses an expected number of $O(k \log n \log k)$ minimum-cost flow computations and $O(km \log n \log^2 k)$ additional time, or deterministically using $O(k^2 \log n \log k)$ minimum-cost flow computations, if exponentiation can be implemented in $O(1)$ time.

Observe that for most known algorithms the time to perform $O(k \log n \log k)$ minimum-cost flow computations dominates the time to perform $O(k \log(nU))$ maximum flow computations needed for the initialization stage when we solve an instance of the general multicommodity flow problem.

When ϵ is $o(1)$ we use ϵ-scaling. First we find an ϵ-optimal multicommodity flow with $\epsilon = 1/9$ using the above procedure. The rest of the computation is divided into scaling phases. We start each phase by dividing ϵ by 2. Thus our current flow is 18ϵ-optimal with respect to the new ϵ. The second part of Theorem 4.5 implies that the expected number of minimum-cost flow computations needed to convert this flow into an 9ϵ-optimal one is bounded by $O(\epsilon^{-3} k \log n)$. The time spent on the ϵ-scaling phase is proportional to ϵ^{-3}, and therefore the last scaling iteration dominates the time spent on all the scaling iterations.

Theorem 4.7 For $\epsilon > 0$, an ϵ-optimal solution for the concurrent flow problem can be found after initialization (Lemma 4.1) by a randomized algorithm that uses an expected number of $O(k(\log k + \epsilon^{-3})\log n)$ minimum-cost flow computations, and $O(km(\log k + \epsilon^{-3})\log k \log n)$ additional time, or deterministically using $O(k^2(\log k + \epsilon^{-2})\log n)$ minimum-cost flow computations, if exponentiation can be implemented in $O(1)$ time.

Goldberg [4] and Grigoriadis and Khachiyan [6] have shown how to reduce the running time of our randomized algorithms by an ϵ^{-1} factor. Goldberg gives a somewhat simplified version of our proof that leads to a randomized selection strategy which avoids having to search for an ϵ-bad commodity. Grigoriadis and Khachiyan generalize our algorithm to solve certain types of convex programming problems. Their algorithm, when specialized to the case of solving multicommodity flows, also avoids searching for an ϵ-bad commodity.

5 Implementing One Iteration of DECONGEST

In this section we shall address the issue of how to implement an iteration of the procedure DECONGEST. In the previous section, we assumed a non-standard model of computation that allows exponentiation to be implemented in $O(1)$ time. In this section, we show how to implement an iteration in the standard RAM model of computation, achieving the same time bounds. We then derive bounds on the time to find a minimum-cost flow.

More specifically, in Section 5.1, we will first show that a flow that satisfies a relaxed set of minimum-cost flow constraints will suffice. We then show that a flow satisfying a second set of relaxed constraints can be modified in $O(m)$ time to satisfy the first set of relaxed constraints while having the additional property that the resulting flow can be represented in $O(\log(nU))$ bits per commodity/edge pair. We then give an approximate length function that uses $O(\log(nU))$ bits per edge which can be used in a minimum-cost flow algorithm to produce a flow that satisfies the second set of relaxed constraints.

In Section 5.2, we discuss which minimum-cost flow algorithm to use. We will use different minimum cost flow algorithms in different situations. For general concurrent flow problems, the best choice seems to be either the algorithm of Goldberg and Tarjan [5] or that of Ahuja, Goldberg, Orlin and Tarjan [1]. For concurrent flow with uniform capacity, we use Gabow and Tarjan's [3] algorithm for the assignment problem. When both the demands and capacities are uniform, we use the algorithm that iteratively computes shortest paths in the residual graph with nonnegative costs discovered independently by Ford and Fulkerson [2] and Yakovleva [16].

5.1 Rounding the Flows and Lengths

Procedure DECONGEST, as described in the previous section, iteratively computes f_i^*, which is a flow that satisfies the demands of commodity i subject to capacity constraints $\lambda u(vw)$ on each edge vw, and minimizes $C_i^* = \sum_{vw \in E} |f_i^*(vw)|\ell(vw)$. Instead, we will compute an approximation $\bar{f}_i^*$ to f_i^*. The flow $\bar{f}_i^*$ can have cost somewhat more than the cost of f_i^*, and it may satisfy slightly relaxed capacity constraints. The key to showing that this flow can be used in the algorithm instead of f_i^* is to prove a relaxed version of Lemma 4.3.

Theorem 5.1 Let C_i denote the cost of the current flow of commodity i with respect to the current length function, and let $\bar{f}_i^*$ be a flow that satisfies demands of commodity i and the constraints

$$(6) \quad \begin{cases} \forall vw \in E: & \bar{f}_i^*(vw) \leq 2\lambda u(vw) \\ \sum_{vw\in E} |\bar{f}_i^*(vw)|\ell(vw) & \leq C_i^* + \frac{1}{2}(\epsilon C_i + \epsilon\frac{\lambda\Phi}{k}). \end{cases}$$

Then, if we use $\bar{f}_i^*$ instead of f_i^* in the concurrent flow algorithm with $\frac{\epsilon}{32\alpha\lambda} \leq \sigma \leq \frac{\epsilon}{16\alpha\lambda}$, we get a bound on the decrease of the potential function by $\Omega(\frac{\epsilon^2}{k}\Phi)$.

Proof: The difference between this proof and that of Lemma 4.3 is as follows. Here we can conclude that $|\delta(vw)| \leq 3\sigma\lambda u(vw)$, and $|\eta| \leq 3\alpha\sigma\lambda \leq 3\epsilon/16$. We use that $|\eta| \leq 3\epsilon/16 \leq 1/4$ implies $e^{x+\eta} \leq e^x + \eta e^x + \frac{3\epsilon}{16}|\eta|e^x$. We modify equation (5) appropriately, and conclude that $\Phi - \Phi' \geq \frac{\alpha\sigma\epsilon\lambda\Phi}{4k} = \Omega(\frac{\epsilon^2}{k})$. ∎

In fact, we won't find such a flow directly. What we will do is to compute a flow that satisfies the somewhat tighter constraints,

$$(7) \quad \begin{cases} \forall vw \in E: & \tilde{f}_i^*(vw) \leq \frac{3}{2}\lambda u(vw) \\ \sum_{vw\in E} |\tilde{f}_i^*(vw)|\ell(vw) & \leq C_i^* + \frac{1}{4}(\epsilon C_i + \epsilon\frac{\lambda\Phi}{k}). \end{cases}$$

We will then modify this flow slightly so that it satisfies conditions (6) and the new flow can be represented in $O(\log(nU))$ bits per edge.

Theorem 5.2 Let $\tilde{f}_i^*$ be a flow that satisfies conditions (7). Then, in $O(m)$ time, we can convert it into a flow $\bar{f}_i^*$ that satisfies (6) such that $(1-\sigma)f_i(vw) + \sigma\bar{f}_i^*(vw)$ can be represented in $O(\log(nU))$ bits for every edge vw.

Proof: Given the flow $\tilde{f}_i^*$, we first compute the flow $(1-\sigma)f_i + \sigma\tilde{f}_i^*$ where σ is chosen as in Theorem 5.1. In order to allow this flow to be represented in $O(\log(nU))$ bits, we will round the flow on edge vw to an integer multiple of $\nu = \epsilon^2/(128m^2k\alpha)$. Observe that if we just rounded the flow on every edge vw to the nearest integer multiple of ν, we would have no guarantee that the flow conservation constraints of equation (2) are still satisfied. Thus, we must round more carefully. Let T be a spanning tree in the graph. We round the flow on all the non-tree edges to the nearest multiple of ν. This rounded flow will not necessarily satisfy the conservation constraints, so we use the tree edges to correct for the violations we may have introduced. It is easy to see that by computing the flow values on the edges of T in postorder we can carry out this step in $O(m)$ time. Observe that the amount of flow we had to add to any non-tree edge is at most ν and the amount that we had to add to any tree edge is at most $m\nu$, as the flow on a tree edge may have to correct for the violation across the cut defined by that edge and the tree.

The resulting rounded flow implicitly defines a $\bar{f}_i^*$ as it can be written as $(1-\sigma)f_i + \sigma\bar{f}_i^*$ for an appropriately chosen $\bar{f}_i^*$. The flow $\bar{f}_i^*$ on edge vw is $\tilde{f}_i^*(vw)$ plus σ^{-1} times the rounding error on the edge. We now show that it satisfies the conditions (6). The rounding error on any edge is at most $m\nu$, therefore for every edge, $\bar{f}_i^*(vw) \leq \tilde{f}_i^*(vw) + \sigma^{-1}m\nu$. Plugging in the bounds on

$\bar{f}_i^*(vw)$ from (7) and the values of σ and ν we get an upper bound of $\frac{3}{2}\lambda u(vw) + \frac{1}{2}\lambda$. Since $u(vw)$ is integral, we conclude that $\bar{f}_i^*(vw) \leq 2\lambda u(vw)$. We bound the cost of $\bar{f}_i^*$ as follows.

$$\begin{array}{rcll} \sum_{vw} |\bar{f}_i^*(vw)|\ell(vw) & \leq & \sum_{vw}(|\tilde{f}_i^*(vw)| + \sigma^{-1}m\nu)\ell(vw) & \\ & \leq & \sum_{vw} |\tilde{f}_i^*(vw)|\ell(vw) + m\sigma^{-1}m\nu e^{\alpha\lambda} & \\ & \leq & C_i^* + \frac{1}{4}\left(\epsilon C_i + \frac{\epsilon\lambda e^{\alpha\lambda}}{k}\right) + \frac{\epsilon\lambda e^{\alpha\lambda}}{4k} & \text{(by (7) and the definitions of } \sigma \text{ and } \nu) \\ & \leq & C_i^* + \frac{1}{2}\left(\epsilon C_i + \frac{\epsilon\lambda\Phi}{k}\right) & \text{(using } \Phi \geq e^{\alpha\lambda}) \end{array}$$

Therefore we have satisfied the conditions of the theorem. ∎

Combining the previous two theorems we get the following corollary:

Corollary 5.3 A flow $\tilde{f}_i^*$ satisfying equations (7) suffices to get a bound on the decrease in the potential function by $\Omega(\frac{\epsilon^2}{k}\Phi)$ while maintaining flows represented by $O(\log(nU))$ bits per edge.

Now we will show how to compute a flow that satisfies (7). Clearly we could do so by finding a minimum-cost flow with respect to the exact length function ℓ.

Unfortunately, this length function is exponential in the size of the input and computing it exactly might take too long. Instead, we will describe how to compute an approximate length function $\tilde{\ell}$, such that the flow that has minimum cost with respect to $\tilde{\ell}$ will have cost at most $C_i^* + \epsilon\lambda\Phi/(8k)$ with respect to ℓ. By Corollary 5.3, such flow can be used in order to implement the rerouting step in our algorithm.

The new length function $\tilde{\ell}$ will be integral, it will consist of $O(\log(nU))$ bits per edge, will be approximately related to ℓ by the scalar multiplier $\gamma = \epsilon e^{\alpha\lambda}/(16Umk)$, and will satisfy $\gamma\tilde{\ell}(vw) \leq \ell(vw)$ on every edge vw. It will take $O(\log n)$ time to compute $\tilde{\ell}(vw)$ on each edge vw. In the following we will use $\tilde{C}_i$ and $\tilde{C}_i^*$ to denote the current cost and the minimum cost of commodity i with respect to length $\tilde{\ell}$, respectively.

For each edge, first we compute $e^{\alpha(f(vw)/u(vw)-\lambda)}$ approximately to have at most $\zeta = \epsilon/(16km)$ additive error, then we multiply the result by $\zeta^{-1}U$, divide by $u(vw)$, take the integer part, and set $\tilde{\ell}(vw)$ to be this value. Using the Taylor series we can compute one bit in an e^x in $O(1)$ time. Since $e^{\alpha(f(vw)/u(vw)-\lambda)}$ is at most 1 on every edge, it is sufficient to compute $O(\log(1/\zeta))$ bits to achieve the desired approximation. Computing the approximate length function takes $O(\log(1/\zeta)) = O(\log n)$ time for each edge, and $O(m\log n)$ time in total.

Because of the approximation and the integer rounding, a flow $\tilde{f}_i^*$, which has minimum cost with respect to $\tilde{\ell}$, is not necessarily the minimum-cost flow with respect to ℓ. However, we will show that a flow that is minimum-cost with respect to $\tilde{\ell}$ will satisfy conditions (7).

Lemma 5.4 Let $\tilde{f}_i^*$ be a flow that is minimum cost with respect to the costs $\tilde{\ell}$ defined above. Then $\tilde{f}_i^*$ has cost (with respect to ℓ) at most $\epsilon\lambda\Phi/(8k)$ more than the minimum.

Proof: Recall that $\gamma = e^{\alpha\lambda}\zeta/U$, and $\zeta = \epsilon/(16mk)$. We bound the difference between ℓ and $\gamma\tilde{\ell}$, a scaled up version of the approximate length function. In computing $\gamma\tilde{\ell}$, we introduce errors in two places. First, when computing $e^{\alpha(\lambda(vw)-\lambda)}$ to a precision of ζ, we introduce an error of ζ. This error gets scaled up by $\zeta^{-1}U/u(vw)$ when we scale up and gets increased by 1 when we round $\tilde{\ell}$

down to an integer. Finally, if we scale $\tilde{\ell}$ back to be compatible with ℓ, the whole error gets scaled by γ. Thus,

$$(8) \quad \ell(vw) - \gamma\tilde{\ell}(vw) \leq \gamma\left(\zeta\left(\frac{\zeta^{-1}U}{u(vw)}\right) + 1\right) = \gamma\left(\frac{U}{u(vw)} + 1\right).$$

We defined $\tilde{\ell}$ so that $\gamma\tilde{\ell}(vw) \leq \ell(vw)$ on every edge, hence we have that

$$(9) \quad \gamma\tilde{C}_i^* \leq C_i^*.$$

Using these two equations and the fact that $\Phi \geq e^{\alpha\lambda}$ we get that:

$$(10) \quad \begin{array}{rcll} \sum_{vw} |\tilde{f}_i^*|\ell(vw) - C_i^* & \leq & \sum_{vw} |\tilde{f}_i^*|\ell(vw) - \gamma\tilde{C}_i^* & \text{(by (9))} \\ & \leq & \sum_{vw} |\tilde{f}_i^*| \left(\ell(vw) - \gamma\tilde{\ell}(vw)\right) & \\ & \leq & \sum_{vw} |\tilde{f}_i^*|\gamma\left(\frac{U}{u(vw)} + 1\right) & \text{(by (8))} \\ & \leq & 2\sum_{vw} \lambda u(vw)\frac{e^{\alpha\lambda}\zeta}{U}\left(\frac{2U}{u(vw)}\right) & \\ & \leq & \frac{2m\lambda\Phi\epsilon}{16km} = \frac{\epsilon\lambda\Phi}{8k} & \end{array}$$

∎

Notice that this flow actually satisfies slightly stronger conditions than (7). We will use this stronger condition in the next subsection.

In the randomized implementation we used the cost of the current flow C_i for the selection of a bad commodity i. We will use the rounded cost $\tilde{C}_i$ instead. The rounding error is small relative to $\sum_i \tilde{C}_i^*$, therefore using $\tilde{C}_i$ does not significantly decrease the probability that a bad commodity will be selected.

To summarize, we have just described how to implement DECONGEST in the RAM model of computation. We first compute approximation $\tilde{\ell}$ to the length function ℓ. Then we compute the approximate cost of each commodity and choose a commodity to reroute, either randomly or deterministically. Next we compute an approximate minimum-cost flow for that commodity with respect to the costs $\tilde{\ell}$. This gives us an approximate minimum-cost flow that satisfies equations (7). We then update the flows for commodity i. Finally, we modify the updated flow as described in Theorem 5.2, represent it in $O(\log(nU))$ bits per edge, and start the next iteration. As the above discussion shows, the time to do this is $O(m \log n)$ plus the time to compute a minimum-cost flow.

Theorem 5.5 For $\epsilon > 0$, an ϵ-optimal solution for the concurrent flow problem can be found after initialization (Lemma 4.1) by a randomized algorithm that uses an expected number of $O(k(\log k + \epsilon^{-3}) \log n)$ minimum-cost flow computations and $O(km(\log k + \epsilon^{-3}) \log^2 n)$ additional time, or deterministically using $O(k^2(\log k + \epsilon^{-2}) \log n)$ minimum-cost flow computations, and $O(km(\log k + \epsilon^{-2}) \log^2 n)$ additional time.

5.2 Choosing a Minimum-Cost Flow Algorithm

In this subsection we consider the problem of choosing the appropriate minimum-cost flow routine to use for finding a minimum-cost flow subject to the costs $\tilde{\ell}(vw)$. In some cases we will only compute an approximate minimum-cost flow subject to cost $\tilde{\ell}$ by further rounding the costs before the minimum-cost flow computation. However, in all cases we will find a flow that satisfies (7).

First, we consider the general concurrent flow problem.

Lemma 5.6 For a commodity i, a minimum-cost flow with respect to $\tilde{\ell}$ can be found in $O(nm\log(nU)\log(n^2/m))$ time.

Proof: The Goldberg-Tarjan minimum-cost flow algorithm runs in $O(nm\log(n^2/m)\log(nC))$ time, where C is the maximum value of the cost of an edge assuming that the costs are integral. For the rounding described in Lemma 5.4, it is easy to verify that the maximum edge cost is at most $\frac{16kmU}{\epsilon}$. Plugging this value in for C and recalling that m, k and ϵ^{-1} are all assumed to be polynomial in n completes the proof. ∎

The above bound can be improved if the capacities are small relative to n^2/m. In this case we will round the demands and solve this rounded problem using the double scaling algorithm of Ahuja, Goldberg, Orlin, and Tarjan [1]. We will then satisfy the remaining flow on arbitrary paths. This flow will still satisfy (7) and the rounding will allow us to use a faster algorithm. More precisely, we will prove the following lemma:

Lemma 5.7 For a commodity i, a flow satisfying (7) can be found in $O(nm\log(nU)\log\log(nU))$ time.

Proof: Assume without loss of generality that ϵ^{-1} is an integer and define $\mu = \lambda\epsilon/(16nk)$. We round the demands for commodity i to integer multiples of μ such that the absolute value of each demand does not increase, the rounded demands still sum to zero, and the total decrease in the absolute values of the demands is at most $2n\mu$. (Recall that each node may have a positive or a negative demand.) Since the absolute value of the demand for commodity i has not increased at any node, there must exist a flow satisfying these demands with cost at most $\tilde{C}_i^*$, subject to costs $\tilde{\ell}$.

Both the demands and the capacities are integral multiples of μ. If we divide both the demands and the capacities by μ, we get a problem where the maximum capacity of an edge is $\lambda U/\mu = 16Unk\epsilon^{-1}$. We can then use the double scaling algorithm of Ahuja, Goldberg, Orlin and Tarjan [1] for solving the minimum-cost problem with rounded demands. By Lemma 5.4, this gives a flow that satisfies the capacity constraints $\lambda u(vw)$ and has cost at most $\epsilon\lambda\Phi/(8k)$ more than the minimum cost but does not satisfy all the demands. We then satisfy the remaining demands by arbitrary paths from nodes with excess to nodes with deficit. The last step increases the flow on an edge by no more than $2n\mu = \epsilon\lambda/(8k) \le \epsilon\lambda u(vw)/(8k)$, and adds a total of no more than $2n\mu\sum_{vw\in E}\ell(vw) \le \lambda\epsilon\Phi/(8k)$ to the cost of the flow subject to costs ℓ.

Combining the minimum-cost flow with the flows on the additional paths, we get a flow that satisfies (7) and proves the lemma. ∎

In the case of the simple concurrent flow problem we can make the time required for solving the minimum-cost flow problem independent of U.

Lemma 5.8 For the simple concurrent flow problem, a flow of a commodity i satisfying (7) can be found in the minimum of $O(nm\log n\log(n^2/m))$ and $O(nm\log n\log\log n)$ time.

Proof: We reduce d_i by a factor of $(1-\epsilon/8)$. We then find a flow f_i' which satisfies the reduced demand $d_i' = (1-\epsilon/8)d_i$ and for which the cost with respect to $\tilde{\ell}$ is no more than $\epsilon\lambda \sum_{vw} \tilde{\ell}(vw)u(vw)/(16k)$ above the minimum cost. Then we multiply the flow on every edge by $(1-\epsilon/8)^{-1}$. This gives a flow that satisfies demands, obeys the slightly increased capacity constraints $(1-\epsilon/8)^{-1}\lambda \cdot u(vw)$, and has cost (subject to ℓ) at most $\epsilon C_i/4 + \epsilon\lambda\Phi/(4k)$ above C_i^*, where Φ is the current potential function value. By Theorem 5.2, we can use this flow and still get the same asymptotic improvement in the potential function.

Define $\mu' = \epsilon d_i/(8m)$, and round the capacities $\lambda u(vw)$ used for the min-cost flow problem, down to multiples of μ'. It is easy to show that the minimum-cost flow with respect to $\tilde{\ell}$ that satisfies the decreased demand d_i' and rounded capacity, is no more than $\tilde{C}_i^*$.

For getting the approximate minimum-cost flow we can work with a further rounded length function. We take $\bar{\ell}(vw)$ to be the integer part of $d_i\tilde{\ell}(vw)/(\lambda U)$. Since after the capacity rounding we consider only edges with $\lambda u(vw) \geq \mu'$, we have

$$\bar{\ell}(vw) \leq \frac{16\epsilon^{-1}kmU}{\mu'/\lambda} \cdot \frac{d_i}{\lambda U} = \frac{16\epsilon^{-1}kmd_i}{\mu'} = O(\epsilon^{-2}km^2).$$

Therefore the Goldberg-Tarjan minimum-cost flow algorithm runs in $O(nm\log(n^2/m)\log n)$ time on this problem.

Now we show that the resulting flow, after multiplication by $(1-\epsilon/8)^{-1}$, satisfies (7). The minimum-cost flow has a single source and a single sink and non-negative costs, therefore no edge will carry more than d_i' units of flow. Let $\bar{f}_i^*$ be a minimum-cost flow with respect to $\bar{\ell}$. By an argument similar to the proof of Lemma 5.4 we get that the cost of this flow with respect to ℓ is at most $md_i \cdot \lambda U/d_i \cdot \epsilon e^{\alpha\lambda}/(16kmU) \leq \epsilon\lambda\Phi/(16k)$ larger than the cost of $\tilde{f}_i'$ with respect to ℓ, where $\tilde{f}_i'$ is the minimum-cost flow with respect to $\tilde{\ell}$ that satisfies the reduced demand d_i'. Now Lemma 5.4 implies that (7) is satisfied.

For all but very dense graphs the double scaling algorithm of Ahuja, Goldberg, Orlin and Tarjan [1] gives a better bound. As we observed no edge will carry more than d_i' units of flow in the optimal flow of commodity i. Thus we can also limit capacities to be no more than d_i', i.e. we can set $u'(vw) = \min\{\lfloor\frac{\lambda u(vw)}{\mu'}\rfloor\mu', d_i'\}$. With this modification, the largest capacity is at most $d_i' = O(m\epsilon^{-1}\mu')$. The demand and the capacities are multiples of μ'. Dividing through by the scale factor μ' we get a problem with integral capacities using $O(\log n)$ bits. ∎

Combining Theorem 5.5 and Lemmas 4.1, 5.6, 5.7 and 5.8 we get the following theorem:

Theorem 5.9 For $\epsilon > 0$, an ϵ-optimal solution for the simple concurrent flow problem can be found either in expected $O(mnk(\epsilon^{-3} + \log k)\min\{\log(n^2/m), \log\log n\}\log n)$ time or $O(mnk^*(\epsilon^{-3} + \log k^*)\min\{\log(n^2/m), \log\log nU\}\log nU)$ time; and deterministically by a factor of k and k^* more time with the power of epsilon modified to be (-2).

If the capacities in the concurrent flow problem are uniform then the capacities in the minimum-cost flow problem are all equal to λ. In this case, there are more efficient minimum-cost flow algorithms than the ones mentioned above.

Lemma 5.10 For the simple concurrent flow problem with uniform capacities, a flow for a commodity i satisfying (7) can be found in $O(m^{3/2}\log n)$ time.

Proof: A minimum-cost flow problem with demand $\lambda\lfloor d_i/\lambda\rfloor$ and capacities λ can be reduced to an assignment problem with $O(m)$ edges and $O(m)$ nodes. We shall use the assignment algorithm of Gabow and Tarjan [3] to solve this rounded problem. The remaining flow can be routed by a shortest path computation in the residual graph. We compute a minimum cost flow using the rounded length function. The bounds follow from Lemma 5.4. ∎

Combining the above lemma with Theorem 5.5 and Lemma 4.1 leads to the following theorem. The resulting time bound for the concurrent flow algorithm with uniform capacities improves the previous best bound [9] if $k < \sqrt{m}/(\log n \log k)$.

Theorem 5.11 For $\epsilon > 0$, an ϵ-optimal solution for the simple concurrent flow problem with uniform capacities can be found in expected $O(km^{3/2}\log^2 n(\epsilon^{-3} + \log k))$ time and in $O(k^2 m^{3/2}\log^2 n(\epsilon^{-2} + \log k))$ time deterministically.

When both the capacities and demands are uniform and k is relatively large, we can obtain better performance by using the minimum-cost flow algorithms of [2] and [16] that repeatedly augment the flow along the shortest path in the residual graph. The resulting time bound is the same up to log factors as those obtained in [9].

6 The Minimum-Ratio Cut Problem

As an application of our concurrent flow algorithms we give fast implementations of the minimum-ratio cut approximation algorithms of Leighton and Rao [10], its extension to hypergraphs by Makedon and Tragoudas [11], its extension to node weighted graphs, and the approximation algorithm of Klein, Agrawal, Ravi, and Rao [8]. The computational bottleneck of these algorithms is solving a concurrent flow problem and its linear programming dual. First, we will summarize the minimum-ratio cut approximation results. Then we will show how our concurrent flow algorithm can be used to find an approximately optimal dual solution to the corresponding concurrent flow problems in addition to finding a near optimal flow. Finally, we shall give even faster running times for the special case of the Leighton–Rao problem where the input graph G has low maximum degree.

6.1 Cut Approximation Results

Let G be an undirected graph with capacities on its edges. For a subset of the nodes A, we use $\bar{A}$ to denote the complement of A, the associated *cut* is the set of edges $\Gamma(A)$ leaving the set A. Let $u(\Gamma(A))$ denote the sum of the capacities of the edges in the cut. Leighton and Rao [10] gave an $O(\log n)$-approximation algorithm for the problem of minimizing the ratio $u(\Gamma(A))/(|A||\bar{A}|)$ over all cuts. By applying this approximation algorithm they obtained polylog-times-optimal approximation algorithms for a wide variety of NP-complete graph problems, including minimum flux, minimum feedback arc set, minimum cut linear arrangement, and minimum area layout. Makedon and Tragoudas [11] extended this result to hypergraphs.

Consider the concurrent flow problem on G with one unit of demand between every pair of nodes. Clearly λ^* must satisfy $\lambda^* \cdot u(\Gamma(A)) \geq d(A,\bar{A}) = |A||\bar{A}|$ for every cut $\Gamma(A)$, where $d(A,\bar{A})$ denotes the sum of all demands across the cut. Therefore, $\min u(\Gamma(A))/(|A||\bar{A}|)$ over all cuts $\Gamma(A)$

gives an upper bound on $1/\lambda^*$. Leighton and Rao show that this minimum is within an $O(\log n)$ factor of the value $1/\lambda^*$.

The computational bottleneck of the Leighton and Rao algorithm is computing a nearly optimal λ and the corresponding near optimal linear programming dual solution for the concurrent flow problem on G with one unit of demand between every pair of nodes. The dual solution is a non-negative length function ℓ that maximizes the ratio $\sum_{v,w} dist_\ell(v,w)/(\sum_{vw\in E} u(vw)\ell(vw))$ (see Theorem 2.2). Linear programming duality implies this maximum is equal to λ^*. Leighton and Rao use a linear programming algorithm to find the length function.

A natural extension is the problem where we are given nonnegative node weights $\nu(v)$ for $v \in V$ in addition to the capacities on the edges. For a subset X of V let $\nu(X)$ denote the sum of the weights on the nodes in X. Consider the extension of the minimum-cut problem to minimizing $u(\Gamma(A))/(\nu(A)\nu(\bar{A}))$ over all cuts. The Leighton and Rao algorithm can be extended to give an $O(\log n)$ approximation algorithm for this problem. The corresponding concurrent flow problem has demand between every pair of nodes, where the demand $d(s,t)$ between nodes s and t equals to $\nu(s)\nu(t)$. (If the weights are scaled so that the total node-weight is n, then the main change to the Leighton–Rao algorithm is to select the node s for starting a tree with $\nu(s)$ maximum.)

Klein, Agrawal, Ravi, and Rao [8] extended the Leighton and Rao results to the case of simple concurrent flow problems with integral capacities and arbitrary integral demands. For a source-sink pair (s,t), let $d(s,t)$ denote the corresponding demand. The minimum ratio cut problem is to minimize the ratio $u(\Gamma(A))/d(A,\bar{A})$ over all cuts.

The minimum value is an upper bound on $1/\lambda^*$ for the concurrent flow problem. Klein, Agrawal, Ravi, and Rao [8] proved that this upper bound is at most a factor of $O(\log nU \log kD)$ above $1/\lambda^*$ in general and gave an $O(\log nU \log kD)$ approximation algorithm for the minimum cut problem, where U is the maximum capacity and D is the maximum demand. Tragoudas [14] has observed that their algorithm can be modified to give the $O(\log n \log kD)$ factor instead.

Using this result they give approximation algorithms for chordalization of a graph and for register sufficiency. Similar to the Leighton-Rao algorithm, the computational bottleneck of their algorithm is solving the dual of the concurrent flow problem, i.e., finding a length function ℓ such that the ratio $\sum_{s,t\in V} d(s,t)dist_\ell(s,t)/\sum_{vw\in E} u(vw)\ell(vw)$ is close to maximum.

6.2 Finding Good Dual Solutions

In order to be able to replace linear programming in the minimum-ratio cut algorithms by our more efficient algorithm, we need to compute a length function $\hat{\ell}$, such that for some constant $\epsilon > 0$, this function satisfies

$$(11) \quad R(\hat{\ell}) = \frac{\sum_{s,t\in V} d(s,t)\mathrm{dist}_{\hat{\ell}}(s,t)}{\sum_{vw\in E} \hat{\ell}(vw)u(vw)} \geq \frac{\lambda^*}{1+\epsilon}.$$

In other words we wish to find a length function, $\hat{\ell}$, for which the ratio in (3) between the last term and the first term with the λ removed is at least $\lambda^*/(1+\epsilon)$. In order to do so, we will use the concurrent flow algorithm to find a length function $\tilde{\ell}$. We show that with respect to this length function the ratio in (4) of the last term and the first term with the λ removed is close to λ^*. We then show how to modify this length function so that if satisfies (11) above.

First we consider the concurrent flow problem that directly corresponds to the given minimum-ratio cut problem, and combine all the commodities that share a source into a single commodity as suggested Lemma 2.1. This decreases the number of commodities to $k^* \leq n$. We shall index the resulting commodities by their sources. Given a target ϵ, if our concurrent flow algorithm used the exact length function ℓ, it would compute a flow satisfying capacities $\lambda \cdot u(vw)$ such that:

$$Q = \frac{\sum_s C_s^*(\lambda)}{\sum_{vw \in E} \ell(vw) u(vw)} \geq \frac{\lambda^*}{1+\epsilon}.$$

But we actually compute flows with respect to an approximate length function $\tilde{\ell}$, described in the proof of Lemma 5.4. Let $\tilde{Q}$ denote the corresponding ratio with ℓ replaced by $\tilde{\ell}$ and C_i^* replaced by $\tilde{C}_i^*$. First we show that $\tilde{Q}$ is almost as close to λ^* as Q.

Lemma 6.1 Let f be the flow and $\tilde{\ell}$ be the length function returned by our algorithm. Then $\tilde{Q} \geq \frac{\lambda^*}{1+2\epsilon}$.

Proof: Let $\gamma = \epsilon e^{\alpha\lambda}/(16mkU)$. Recall that this is the factor that approximately relates the real lengths to the approximate lengths. By the way the approximate lengths were computed, $\gamma\tilde{\ell}(vw) \leq \ell(vw)$ for every edge vw. Also, by arguments similar to those used to derive (10) we have that

$$C_i^* - \gamma \tilde{C}_i^* \leq \epsilon\lambda\Phi/(8k) \leq \epsilon\lambda^*\Phi/(4k).$$

Using these two facts, we have the following bound on $\tilde{Q}$:

$$\begin{aligned}
\tilde{Q} &= \frac{\gamma \sum_s \tilde{C}_i^*(\lambda)}{\sum_{vw \in E} \gamma\tilde{\ell}(vw)u(vw)} \\
&\geq \frac{\gamma \sum_s \tilde{C}_i^*(\lambda)}{\sum_{vw \in E} \ell(vw)u(vw)} \\
&\geq \frac{\sum_s C_i^*(\lambda) - \epsilon\lambda^* \sum_{vw \in E} \ell(vw)u(vw)/4}{\sum_{vw \in E} \ell(vw)u(vw)} \\
&= \frac{\sum_s C_i^*(\lambda)}{\sum_{vw \in E} \ell(vw)u(vw)} - \epsilon\lambda^*/4 \\
&\geq \frac{\lambda^*}{1+\epsilon} - \frac{\epsilon\lambda^*}{4} \geq \frac{\lambda^*}{1+2\epsilon}. \quad \blacksquare
\end{aligned}$$

Now we describe how to modify this length function to produce one that satisfies (11) above. Observe that setting $\hat{\ell} = \tilde{\ell}$ does not necessary work, since $\sum_{s,t \in V} d(s,t)\mathrm{dist}_{\tilde{\ell}}(s,t)$ might be significantly smaller than $\sum_s \tilde{C}_s^*(\lambda)$. Instead of using $\tilde{\ell}$ directly, we will compute a new length function $\hat{\ell}$. The idea is to compute a minimum-cost flow with respect to costs $\tilde{\ell}$ and capacities $\lambda \cdot u(vw)$ for each commodity and then use the optimal price function $\tilde{p}_s$ to change $\tilde{\ell}$ by adding to it the sum of the absolute values of reduced costs for edges with negative reduced costs.

Let $\tilde{f}_s^*$ denote the minimum-cost flow for commodity s with respect to $\tilde{\ell}$, and $\tilde{p}_s$ the optimal price function. Let us denote $\ell_s(vw) = -\min\{0, \tilde{\ell}(vw) + \tilde{p}_s(v) - \tilde{p}_s(w)\}$; *i.e.* $\ell_s(vw)$ is the absolute value of the reduced cost if it is negative, and zero otherwise. Recall, that the complementary slackness conditions imply that if $\ell_s(vw) > 0$ then $\tilde{f}_s^*(vw) = \lambda u(vw)$. We define the new length function as $\hat{\ell}(vw) = \tilde{\ell}(vw) + \sum_s \ell_s(vw)$. We need the following lemma to estimate the numerator of $R(\hat{\ell})$.

Lemma 6.2 The flow $\tilde{f}_s^*$ is minimum-cost subject to cost $\tilde{\ell} + \ell_s$, its cost is $\sum_t d(s,t)\text{dist}_{\tilde{\ell}+\ell_s}(s,t)$.

Proof: We prove the optimality of $\tilde{f}_s^*$ by showing that $\tilde{f}_s^*$ and the price function $\tilde{p}_s$ satisfy the complementary slackness conditions. By the definition of ℓ_s we have that $\tilde{\ell}(vw) + \ell_s(vw) + \tilde{p}_s(v) - \tilde{p}_s(w)$ is nonnegative and it is positive if and only if $\tilde{\ell}(vw) + \tilde{p}_s(v) - \tilde{p}_s(w)$ is positive. By complementary slackness applied to cost $\tilde{\ell}$, flow $\tilde{f}_s^*$ and prices $\tilde{p}_s$, if this value is positive, then $\tilde{f}_s^*$ is zero.

Now consider the cost of $\tilde{f}_s^*$ subject to the cost function $\tilde{\ell} + \ell_s$. There are no edges with negative reduced cost, therefore the cost of the flow is at least $\sum_t d(s,t)(\tilde{p}_s(t) - \tilde{p}_s(s))$. All edges that carry flow have zero reduced cost. This implies that the cost of the flow is equal to $\sum_t d(s,t)(\tilde{p}_s(t) - \tilde{p}_s(s))$ and $\tilde{p}_s(t) - \tilde{p}_s(s) = \text{dist}_{\tilde{\ell}+\ell_s}(s,t)$. ∎

Theorem 6.3 $R(\hat{\ell}) \geq (1+2\epsilon)^{-1}\lambda^*$.

Proof: We shall estimate the numerator of $R(\hat{\ell})$ using the above lemma. For a source s we have that

$$\sum_t d(s,t)\text{dist}_{\hat{\ell}}(s,t) \geq \sum_t d(s,t)\text{dist}_{\tilde{\ell}+\ell_s}(s,t) = \sum_{vw}(\tilde{\ell}(vw)+\ell_s(vw))\tilde{f}_s^*(vw) = \tilde{C}_s^*(\lambda)+\sum_{vw}\ell_s(vw)\tilde{f}_s^*(vw).$$

By complementary slackness, and the definition of ℓ_s we find that if $\ell_s(vw) \neq 0$ then $\tilde{f}_s^*(vw) = \lambda u(vw)$. Summing over all sources we get that

$$\sum_{s,t\in V} d(s,t)dist_{\hat{\ell}}(s,t) \geq \sum_s \tilde{C}_s^*(\lambda) + \lambda\sum_{vw} u(vw)\sum_s \ell_s(vw).$$

Dividing the two sides of this equation by $\sum_{vw}\hat{\ell}(vw)u(vw)$ we get that

$$R(\hat{\ell}) \geq \frac{\sum_s \tilde{C}_s^*(\lambda) + \lambda\sum_{vw} u(vw)\sum_s \ell_s(vw)}{\sum_{vw} u(vw)\tilde{\ell}(vw) + \sum_{vw} u(vw)\sum_s \ell_s(vw)}.$$

Applying the simple mathematical fact that for positive a, b, x and λ, if $a/b \leq \lambda$ then $(a+\lambda x)/(b+x) \geq a/b$, we see that the left side of the above equation is at least $\tilde{Q}$ which by Lemma 6.1 is at least $\lambda^*/(1+2\epsilon)$. ∎

Corollary 6.4 An ϵ-optimal flow and length function pair (f, ℓ) produced by our concurrent flow algorithm can be translated into a length function $\hat{\ell}$ needed by the minimum-ratio cut algorithms in $O(k^* nm \log(n^2/m)\log(nU))$ time. The dual objective value associated with $\hat{\ell}$ will be within an $(1 - O(\epsilon))$ factor to the optimum.

We can use the approximate minimum-cost flow computation in Lemma 5.7 instead of Lemma 5.6. With an argument similar to the above, but somewhat more involved, we replace the $\log(n^2/m)$ in the theorem by a $\log\log(nU)$. We obtain the following corollary.

Corollary 6.5 An $O(\log n)$-approximation to the node weighted cut problem with general capacities can be found in $O(n^2 m \log nU \log^2 n \min\{\log(n^2/m), \log\log nU\})$ expected time. An $O(\log n \log kD)$ -approximation to the minimum-ratio cut problem with general demands and capacities can be found in $O(k^* nm \log nU \log k \log n \min\{\log(n^2/m), \log\log nU\})$ expected time.

An analogous theorem can be obtained for finding approximately sparsest cuts in hypergraphs using the concurrent flow algorithm in conjunction with the approximation algorithm of Makedon and Tragoudas [11].

6.3 Graphs with Low Maximum Degree

Next we improve the running time given in Corollary 6.5 for low-degree graphs G. The new running time will depend on Δ, the maximum degree of any node in the graph.

We consider the minimum-ratio cut problem for graphs with unit demands, where the graph that has an edge between the source and sink of each commodity is a constant degree expander on V. (We call this graph the *demand graph.*) While the case of the expander demand graph with unit demands seems like an obscure special case, it is in fact an important one. The Leighton and Rao [10] algorithm uses the solution of a concurrent flow problem in which the demand graph is the complete graph. However, one can modify the Leighton and Rao algorithm to use the solution to this new concurrent flow problem and its dual problem to derive an $O(\log n)$ approximation to the minimum-ratio $u(\Gamma(A))/(|A||\bar{A}|)$ over all cuts. To get an idea how the two problems are related consider a cut $\Gamma(A)$ and assume that $|A| \leq |\bar{A}|$. Because the demand graph is a constant degree expander, $c|A| \leq d(A, \bar{A}) \leq \hat{c}|A|$ for some constants c and $\hat{c}$. Therefore, $u(\Gamma(A))/d(A, \bar{A})$ is $\Theta(n)$ times more than $u(\Gamma(A))/(|A||\bar{A}|)$.

The first step in solving this problem is to round all the capacities up to integer multiples of a parameter μ in such a way that the ratio $u(\Gamma(A))/(|A||\bar{A}|)$ is not changed by more than a factor of two. Notice that $|\Gamma(A)| \leq \Delta|A|$. We shall use r to denote the maximum of $|\Gamma(A)|/d(A, \bar{A})$ over all cuts $\Gamma(A)$. Notice that $r \leq \Delta/c$, where c is the expansion parameter of the demand graph.

Theorem 6.6 Let λ^* be the optimum value of the concurrent flow problem, and let $\mu \leq (r\lambda^*)^{-1}$. If we round each capacity $u(e)$ up to $\hat{u}(e)$, the next integer multiple of μ, then the minimum ratio $\hat{u}(\Gamma(A))/(|A||\bar{A}|)$ of a cut $\Gamma(A)$ with capacity $\hat{u}$ is at most twice of the minimum ratio with u.

Proof: For all cuts $\Gamma(A)$, it must be that $\lambda^* u(\Gamma(A)) \geq d(A, \bar{A})$. The rounding error $\hat{u}(\Gamma(A)) - u(\Gamma(A))$ is at most $\mu|\Gamma(A)| \leq |\Gamma(A)|(2r\lambda^*)^{-1} \leq d(A, \bar{A})|/\lambda^* \leq u(\Gamma(A))$. This implies that for every cut $\hat{u}(\Gamma(A)|/(|A||\bar{A}|) \leq 2u(\Gamma(A))/(|A||\bar{A}|)$, i.e. the new ratio is at most twice the old ratio. ∎

Rounding to integer multiples of μ preserves the minimum-ratio cut up to a factor of two. If we want to preserve λ^* up to a constant factor we have to do a somewhat finer rounding.

Theorem 6.7 Let λ^* be the optimum value of the concurrent flow problem and let $\mu \leq \epsilon(20r\lambda^* \log mU \log n)^{-1}$. If we round each capacity $u(e)$ up to $\hat{u}(e)$, the next integer multiple of μ, then the minimum congestion $\hat{\lambda}^*$ subject to capacities $\hat{u}(e)$ is at most a factor of $1+\epsilon$ less than the minimum congestion λ^* with u.

Proof: The idea is to use the $O(\log n \log kD)$ approximation result of Klein, Agrawal, Ravi, and Rao [8] as improved by Tragoudas [14]. Consider the following auxiliary concurrent flow problem. The graph is G with capacities u. For every edge $vw \in E$ there is a demand of value $d(v, w) = \hat{u}(vw) - u(vw)$ from v to w. Observe that the demands in the auxiliary problem are integral and at most μ, and $\log \mu$ is at most $\log(\epsilon mU/(20r \log nU \log n)) \leq 2\log(mU)$. Using the same estimates as in the previous proof we can conclude that the minimum of $u(\Gamma(A))/d(A, \bar{A})$ over all cuts $\Gamma(A)$

is at most $\epsilon/(20 \log mU \log n)$. By the above approximation result the minimum congestion λ^* for this problem is at most ϵ. That is, the added capacities can be routed in an ϵ-fraction of the original capacities u.

Now consider an optimal flow $\hat{f}$ of congestion $\hat{\lambda}^*$ in the rounded problem. To get a solution in the original problem we route the part of flow $\hat{f}$ that uses the added capacity in the way this demand is routed in the optimal solution to the auxiliary problem. This does not increase the congestion by more than a factor of $1+\epsilon$. ∎

Next consider the question of how long it takes to solve a rounded concurrent flow problem. For simplicity we shall restrict our attention to the case when ϵ is a constant. The number of commodities is $O(n)$. The capacities in the minimum-cost flow problem are integer multiples of $\lambda\mu$. We shall use the minimum-cost flow algorithm due to Ford–Fulkerson [2] and Yakovleva [16], that repeatedly augments the flow along the shortest path in the residual graph, to solve these problems. Given a concurrent flow with congestion λ, the number of shortest path computations in a minimum-cost flow subroutine is at most $\mu^{-1}\lambda^{-1}+1$, the upper integer part of the demand, which is 1, divided by the unit of the capacity, which is $\lambda\mu$.

We use these ideas to solve the minimum-ratio cut and the concurrent flow problem. The $O((\lambda^{-1}\mu^{-1}+1)(m+n\log n))$ time required for solving the minimum-cost flow problem might not dominate the $O(m \log n)$ needed to compute the approximate length function. To simplify the bounds we shall count each minimum-cost flow computation as $O((\lambda^{-1}\mu^{-1}+1)m\log n))$ time. These bounds can be further improved by using the data structures described in [9].

Notice that here we do not have time to find an initial flow using k maximum-flow computations suggested in Lemma 4.1. The capacities of this problem are not rounded, therefore we have to use a general maximum-flow algorithm, and all such algorithms take $\Omega(mn)$ time. However, an initial flow that is optimal up to a factor of $O(mk)$ can be computed by routing each demand on the path with maximum bottleneck capacity from its source to its sink.

An iteration of the algorithm will use Theorem 6.6 or 6.7 with μ defined by $c(\Delta\lambda_0)^{-1}$ (respectively $\epsilon c(20\Delta\lambda_0 \log mU \log n)^{-1}$). We terminate the iteration if λ decreases below $\lambda_0/2$. At that point we divide λ_0 by two, and start the next iteration. We use the flow obtained in the previous iteration as our initial flow.

Theorem 6.8 An $O(\log n)$ approximation to the minimum ratio $u(\Gamma(A))/(|A||\bar{A}|)$ over all cuts $\Gamma(A)$ in a graph with capacities u and maximum degree Δ can be computed in $O(nm\Delta\log^3 n)$ expected time.

Theorem 6.9 For any constant ϵ, an ϵ approximation to a unit demand concurrent flow problem in a graph with maximum degree Δ with a constant degree expander demand-graph can be computed in $O(nm\Delta \log^4 n \log nU)$ expected time.

In regular graphs $n\Delta = m$, therefore the running times of the above two algorithms are roughly (up to a polylogarithmic factor) $O(m^2)$.

Acknowledgments

We are grateful to Andrew Goldberg, Tishya Leong, Jim Orlin, Rina Rotshild, David Shmoys and Peter Shor for helpful discussions.

References

[1] R. K. Ahuja, A.V. Goldberg, J. B. Orlin, and R.E. Tarjan. Finding minimum cost flows by double scaling. Sloan Working Paper 2047-88, MIT, Cambridge, MA, 1988.

[2] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canad. J. Math.*, 8:399–404, 1956.

[3] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18:1013–1036, 1989.

[4] A. V. Goldberg, Personal communication. Jan., 1991.

[5] A. V. Goldberg and R. E. Tarjan. Solving minimum-cost flow problems by successive approximation. *Mathematics of Operations Research*, 15(3):430–466, 1990.

[6] M. D. Grigoriadis and L. G. Khachiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. Technical Report DCS-TR-273, Department of Computer Science, Rutgers University, New Brunswick, NJ, March 1991.

[7] S. Kapoor and P. M. Vaidya. Fast algorithms for convex quadratic programming and multicommodity flows. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 147–159, 1986.

[8] P. Klein, A. Agrawal, R. Ravi, and S. Rao. Approximation through multicommodity flow. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 726–737, 1990.

[9] P. Klein, S. A. Plotkin, C. Stein, and É. Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. Technical Report 961, School of Operations Research and Industrial Engineering, Cornell University, 1991. A preliminary version of this paper appeared in *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 310–321, 1990. To appear in SIAM J. Computing.

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

[11] F. Makedon and S. Tragoudas. Approximating the minimum net expansion: Near optimal solutions to circuit partitioning problems. In *Procedings of the 1990 Workshop on Graph Theoretic Concepts in Computer Science*, June 1990.

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

[13] C. Stein. *Approximation algorithms for multicommodity flow and shop scheduling problems.* PhD thesis, MIT, Cambridge, MA, August 1992. Also appears as MIT/LCS/TR-550.

[14] S. Tragoudas. *VLSI partitioning approximation algorithms based on multicommodity flow and other techniques.* PhD thesis, University of Texax at Dallas, 1991.

[15] P. M. Vaidya. Speeding up linear programming using fast matrix multiplication. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 332–337, 1989.

[16] M.A. Yakovleva. A problem on minimum transportation cost. In V.S. Nemchinov, editor, *Applications of Mathematics in Economic Research*, pages 390–399. Izdat. Social'no-Ekon. Lit., Moscow, 1959.