Simple and Fast Distributed Multicommodity Flow Algorithm

Anil Kamath* **Omri Palmon**† **Serge Plotkin**‡

Abstract

In this paper, we present a simple and fast distributed algorithm for approximately solving the minimum-cost multicommodity flow problem. If there exists a flow that satisfies all of the demands, our algorithm runs in $O(m^2\epsilon^{-2}\log n)$ communication rounds and produces a flow that satisfies $(1-\epsilon)$-fraction of each demand, where m denotes the number of edges in the graph. If there exists a feasible flow of cost B, then our algorithm terminates in $O(m^2\epsilon^{-2}\delta^{-2}\log(nCU)\log^2(n\delta^{-1}))$ rounds and finds a flow that satisfies $(1-\epsilon)$ fraction of each demand and whose cost is below $(1+\delta)B$, where C and U are maximum cost and maximum edge capacity, respectively.

The algorithm is *local* in the sense that in one round each processor makes a local computation and exchanges information with each one of its neighbors. The only previously known local multicommodity flow approximation algorithm, due to Awerbuch and Leighton [2], can not deal with the min-cost problem, and runs in $O(m^2k^{1.5}n\epsilon^{-3})$ communication rounds, which can be as much as $\Omega(n^4\epsilon^{-1})$ factor slower than our algorithm.

Perhaps the most important property of our algorithm is its resilience to failures. For example, in the case where there are no costs, failure of an edge can delay the convergence of the algorithm by at most $O(m^2\epsilon^{-2}\log n)$ rounds, provided that this failure does not make the problem infeasible.

The performance of our distributed algorithm should be contrasted with the performance one can achieve by taking the best sequential approximation algorithm for the min-cost multicommodity flow problem [8] and implementing it in a distributed setting. This results in a distributed algorithm that runs in $O^*(kmn\epsilon^{-2})$ communication rounds, where k is the number of commodities and n is the number of nodes in the graph. This algorithm is not resilient to edge failures and is slower than the algorithm presented here for constant ϵ and δ and large k.

---

*Department of Computer Science, Stanford University. Research supported by U.S. Army Research Office Grant DAAL-03-91-G-0102.

†Department of Computer Science, Stanford University. Research supported by a grant from Stanford Office of Technology Licensing.

‡Department of Computer Science, Stanford University. Research supported by U.S. Army Research Office Grant DAAL-03-91-G-0102 and by a grant from Mitsubishi Electric Laboratories.

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 present a simple and fast distributed approximation algorithm for a variant of the *minimum-cost concurrent multicommodity flow.* If there exists a feasible flow, *i.e.* flow that satisfies all the demands, then in $O^*(m^2\epsilon^{-2})$ communication rounds, our algorithm finds a flow that satisfies at least $(1-\epsilon)$-fraction of each demand, where m is the number of edges in the network.[1] If there exists a feasible flow of cost B, then our algorithm finds a flow that satisfies $(1-\epsilon)$ fraction of the demands and whose cost is below $(1+\delta)B$ in $O^*(m^2\epsilon^{-2}\delta^{-2}\log(CU))$ rounds, where C and U are maximum edge cost and maximum edge capacity, respectively.

A communication round consists of each processor performing a computation on its local data and sending the results to all of its neighbors. Our algorithm is *local* in a sense that it does not depend on any global computations like shortest paths or min-cost flow, that were the main subroutines of the previously known efficient combinatorial multicommodity flow algorithms [5, 6, 8]. The only previously known local approximate concurrent flow algorithm, due to Awerbuch and Leighton [2], does not address the min-cost problem and runs in $O^*(m^2k^{1.5}n\epsilon^{-3})$ rounds, where k is the number of commodities and n is the number of nodes. This can be as much as an $\Omega(n^4\epsilon^{-1})$ factor slower than our algorithm.

The main advantages of a "local" algorithm is ease of implementation in a distributed system where centralized control is either infeasible or very expensive. Moreover, our algorithm is resilient to failures: an edge failure (or, as a matter of fact, and change in the network topology) that does not make the problem infeasible, can delay the convergence of the algorithm by at most the number of rounds stated above. The only two nodes needed to be notified about the failure are the nodes adjacent to the failed edge. This makes it unnecessary to run the complicated and costly "global reset" algorithms [1] after every failure/reappearance of an edge. Another interesting property of our algorithm is that it can be modified to run forever, maintaining an approximate solution to the flow, automatically adjusting it after every edge failure or demand change.

Multicommodity flow arises naturally in many contexts, including virtual circuit routing in a communication network, VLSI layout, scheduling, and transportation, and hence was extensively studied [3, 4, 11, 12, 7, 9, 10]. Since it can be written as a linear program, one can use

[1] We say that $f(n) = O^*(g(n)$ if $f(n) = O(g(n)\log^k n)$ for some constant k.

interior point methods to solve it exactly. Unfortunately, the fact that the number of variables is large causes these methods to generate rather slow algorithms. In particular, the fastest interior-point min-cost multicommodity flow algorithm [4, 11, 12] relies on fast matrix multiplication subroutine 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 D is the largest demand, and U is the largest edge capacity. Since this algorithm involves repeated matrix multiplications, it does not seem appropriate for efficient implementation in the distributed setting.

Significantly better running times were achieved by approximation algorithms. In particular, if there exists a feasible solution, then the algorithm due to Leighton, Makedon, Plotkin, Stein, Tardos, and Tragoudas [6] will produce an ϵ-approximation to the feasible flow in $O^*(k^2mn\epsilon^{-2})$ time. A flow that both satisfies at least $(1-\epsilon)$ fraction of each demand and whose cost is within $(1+\epsilon)$ of the optimum can be obtained by the algorithm due to Plotkin, Shmoys and Tardos [8] in $O^*(k^2m^2\epsilon^{-2}\log C)$ time, where C is the maximum edge cost. The running times of these algorithms can be improved by a factor of k by use of randomization. Both of these algorithms are based on repeated computations of minimum-cost flows and shortest paths, and hence are not very easy to implement in a distributed setting. (Especially in networks with with dynamic topology.) The best performance one can hope to achieve by implementing the algorithm in [8] in a static network is $O^*(kmn\epsilon^{-2}\log C)$ rounds. This significantly exceeds the number of rounds of the algorithm presented in this paper for large k and constant ϵ, δ.

The algorithm presented in this paper is very simple. The idea is to maintain a *preflow*, which is flow that obeys capacity constraints but does not obey flow-conservation constraints. We refer to the difference between the flow into a node and the flow out of the node as the *excess* at this node. The algorithm starts with zero flow, which is a valid preflow. Each iteration consists of two phases. In the first phase, each node distributes its excess among its adjacent edges. Then, independently for each edge, the flow through the edge is updated in a way that reduces a certain non-linear function of the excesses assigned to this edge. Intuitively, this operation "spreads" the excesses throughout the graph, eventually obtaining a solution with very small excesses.

Our algorithm is in some sense similar to the one in [2]. There are two main differences that allowed us to get much better performance. First, we use a different local optimization criteria. Our criteria can be viewed as an adaptation to the excess-minimization context of the exponential "length function", introduced by Shahrokhi and Matula [9]. Second, we use preflow instead of explicitly maintaining queues. Roughly speaking, if one interprets flow as the "amount of flow in unit time", then the fact that a node has excess implies that each time unit there is more flow coming in than going out. The algorithm in [2] explicitly maintains the number of flow units queued at each node and tries to minimize the size of these queues. Instead, our algorithm minimizes the excess, which corresponds to the *rate of growth* of the queues.

2 Preliminaries and Definitions

An instance of the *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 nodes by n, and the number of edges by m. 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.

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. Given flow $f_i(vw)$, the *excess* $\mathrm{Ex}_i(f, v)$ of commodity i at node v is defined by:

$$\mathrm{Ex}_i(f, v) \stackrel{\mathrm{def}}{=} d_i(v) - \sum_{w:vw \in E} f_i(vw),$$

where $d_i(s_i) = d_i$, $d_i(t_i) = -t_i$ and $d_i(v) = 0$ otherwise.

A flow where all excesses are zero is said to satisfy the *conservation constraints*. A flow satisfies *capacity constraints* if $\sum_i |f_i(vw)| \leq u(vw)$ for all edges vw. A flow that satisfies both the capacity and the conservation constraints is *feasible*. During the description of our algorithm it will be convenient to work with *preflow*, which is a flow that satisfies capacity constraints but does not necessarily satisfy the conservation constraints. An ϵ-approximation to the multicommodity flow feasibility problem is a feasible flow that satisfies demands $(1-\epsilon)d_i$.

3 The Flow Without Costs

In this section we present the distributed multicommodity flow algorithm. If a feasible solution satisfying all the demands exists, the algorithm will produce a flow that satisfies at least $(1-\epsilon)$ fraction of each demand.

The Algorithm The algorithm starts with zero flow, which is a valid preflow. The algorithm maintains for every node pair v, w such that $vw \in E$ or $wv \in E$, commodity i, and edge $vw \in E$ an excess $Ex_i(f, vw)$.[2] Excess of commodity i at node v is the sum of $Ex_i(f, vw)$ for all w. Each iteration consists of two phases. In the first phase, each node recomputes $Ex_i(f, vw)$, effectively distributing its excess among its edges. Then, independently for each edge, the flow through the edge is updated in a way that reduces the potential function defined below. Intuitively, this operation "spreads" the excesses throughout the graph, eventually obtaining a solution with very small remaining excesses.

[2]Note that, in general, $\mathrm{Ex}_i(f, vw) \neq \mathrm{Ex}_i(f, wv)$.

Given a flow f, we define the following potential function:

$$\Phi(f) \stackrel{\text{def}}{=} \sum_{vw \in E} \sum_{i=1}^{k} \exp\left(\alpha \frac{Ex_i(f, vw)}{d_i}\right) + \exp\left(\alpha \frac{Ex_i(f, wv)}{d_i}\right), \tag{1}$$

where

$$\alpha \stackrel{\text{def}}{=} 4\epsilon^{-1} m \log(mk).$$

The algorithm starts from the zero flow and proceeds in iterations, where each iteration consists of two phases. The first phase computes the excess $Ex_i(f, v)$ at each node and distributes it equally among the edges incident to this node by setting $Ex_i(f, vw) = Ex_i(f, v)/\delta(v)$, where $\delta(v)$ is the degree of v. The second phase calculates the flow update $\Delta f_i(vw)$ on each edge vw for each commodity i where the goal is to minimize the potential function $\Phi(f)$. In fact, since this function can be written as a sum of functions where each one depends on the flow on a single edge, the algorithm minimizes the following function independently for each edge vw:

$$\sum_{i=1}^{k} \exp\left(\alpha \frac{Ex_i(f, vw) - \Delta f_i(vw)}{d_i}\right) + \exp\left(\alpha \frac{Ex_i(f, wv) + \Delta f_i(vw)}{d_i}\right). \tag{2}$$

In order to ensure convergence to feasible flow, the minimization is done under the following constraints:

$$\begin{aligned} \sum_{i=1}^{k} |f_i(vw) + \Delta f_i(vw)| &\leq u(vw) \\ |f_i(vw) + \Delta f_i(vw)| &\leq d_i \end{aligned}$$

The new flow is computed by setting $f_i(vw) = f_i(vw) + \Delta f_i(vw)$. In the rest of this section we will assume that the minimization is executed as described above. In fact, one can minimize a much simpler expression, thus improving the complexity of each iteration. We will address this issue in Section 4.

Analysis of the algorithm We will show that existence of a feasible solution that satisfies all of the demands implies that each iteration reduces the potential function. We will concentrate on proving the reduction due to the second phase in each iteration, since the first phase can not increase the potential function. The following theorem shows that sufficiently small value of $\Phi(f)$ implies that the sum of the excesses of any commodity does not exceed an ϵ-fraction of the demand of this commodity. Note that such a preflow can be easily converted into a flow that satisfies $(1-\epsilon)$-fraction of each demand.

Theorem 3.1 If the value of the potential function $\Phi(f) \leq \exp(\alpha \frac{\epsilon}{m})$ then for each commodity i, we have $\sum_v |Ex_i(f, v)| \leq \epsilon d_i$.

Theorem 3.2 For each iteration $\Phi(f+\Delta f) \leq \Phi(f)$. Furthermore, if there exists a feasible solution to the problem then in each iteration

$$\Phi(f+\Delta f) \leq \Phi(f)\left(1-\frac{\log^2 \frac{\Phi(f)}{mk}}{16\alpha^2}\right).$$

Proof: As we have noted above, the first phase of each iteration can not increase the potential function. Obviously, the second phase can not increase the potential function value. It remains to prove that the decrease due to the second phase is large.

Let f^* denote a feasible solution that satisfies all of the demands such that $|f^*(vw)| \leq d_i$ for every commodity i and edge vw. Let $\tilde{f} \stackrel{\text{def}}{=} f^* - f$. Observe that flow update defined by $\sigma\tilde{f}$ for any $\sigma \in [0, \alpha^{-1}]$ satisfies the optimization constraints. Hence the reduction in the potential function can be bounded by computing the effect of updating the current flow by $\sigma\tilde{f}$.

We will use the fact that $e^{x+t} \leq e^x + te^x + t^2 e^x$ for $|t| \leq 1$. Thus, if $\left|\alpha\sigma\tilde{f}_i(vw)\right| \leq d_i$ for all i and vw, we have:

$$\Phi\left(f+\sigma\tilde{f}\right) \leq \sum_{i=1}^{k}\sum_{vw\in E} \exp\left(\alpha\frac{Ex_i(f,vw)-\sigma\tilde{f}_i(vw)}{d_i}\right)+\exp\left(\alpha\frac{Ex_i(f,wv)+\sigma\tilde{f}_i(vw)}{d_i}\right) \tag{3}$$

$$\begin{aligned}
\leq\ & \Phi(f) \\
& -\sum_{i=1}^{k}\sum_{vw\in E}\sigma\alpha\frac{\tilde{f}_i(vw)}{d_i}\left[\exp\left(\alpha\frac{Ex_i(f,vw)}{d_i}\right)-\exp\left(\alpha\frac{Ex_i(f,wv)}{d_i}\right)\right] && (4)\\
& +\sum_{i=1}^{k}\sum_{vw\in E}\sigma^2\alpha^2\left(\frac{\tilde{f}_i(vw)}{d_i}\right)^2\left[\exp\left(\alpha\frac{Ex_i(f,vw)}{d_i}\right)+\exp\left(\alpha\frac{Ex_i(f,wv)}{d_i}\right)\right]. && (5)
\end{aligned}$$

We will evaluate each one of the terms in the above expression separately. To evaluate (4) we decompose $\tilde{f}$ into cycles and paths from positive to negative excesses. Notice that the cycles do not affect the value of (4). Moreover, each flow path of value F that starts with edge vv' and ends with $u'u$, contributes

$$\sigma\alpha F\cdot\left(\exp\left(\alpha\frac{Ex_i(f,vv')}{d_i}\right)-\exp\left(\alpha\frac{Ex_i(f,uu')}{d_i}\right)\right).$$

Hence, (4) is equal to:

$$\sigma\sum_i\sum_{vw\in E}\alpha\frac{Ex_i(f,vw)}{d_i}\exp\left(\alpha\frac{Ex_i(f,vw)}{d_i}\right) \geq \sigma\Phi(f)\log\left(\frac{\Phi(f)}{mk}\right),$$

where the inequality follows from the fact that the sum has mk terms and $\sum y_i \log y_i$ is convex. To evaluate expression (5) we use the fact that $\left|\tilde{f}_i(vw)\right| \leq 2d_i$ and thus

$$\sum_i\sum_{vw\in E}\sigma^2\alpha^2\left(\frac{\tilde{f}_i(vw)}{d_i}\right)^2\exp\left(\alpha\frac{Ex_i(f,vw)}{d_i}\right) \leq 4\sigma^2\alpha^2\Phi(f).$$

Combining our bounds on (4) and (5) we get from (3) that for $\sigma \leq \alpha^{-1}$:

$$\Phi\left(f+\sigma\tilde{f}\right) \leq \Phi(f) - \sigma\Phi(f)\log\frac{\Phi(f)}{mk} + 4\sigma^2\alpha^2\Phi(f).$$

Thus taking

$$\sigma = \frac{\log\frac{\Phi(f)}{km}}{8\alpha^2}$$

we get

$$\Phi\left(f+\sigma\tilde{f}\right) \leq \Phi(f)\left[1 - \frac{\log^2\frac{\Phi(f)}{mk}}{16\alpha^2}\right].$$

This completes the proof of the theorem. ∎

Remark: Note that the way we proved the decrease in the potential function is by expressing a lower bound on this decrease as a sum of quadratic functions in Δf and then showing that there exists a flow that achieves a large decrease. We will use this fact in Section 5, where we present an efficient implementation of each iteration.

Theorem 3.3 After $O(m^2\epsilon^{-2}\log mk)$ iterations, we have $\sum_v |Ex_i(f,v)| \leq \epsilon d_i$ for each commodity i.

Proof: Theorem 3.2 implies that as long as $\Phi(f) \geq 2m^2k^2$, the value of the potential function is halved in $O(\alpha^2/\log^2\Phi(f))$. Thus, the number of iterations until the potential function reaches $2m^2k^2$ is bounded by $O\left(\alpha^2/\log(2m^2k^2)\right) = O(m^2\epsilon^{-2}\log mk)$. The claim follows by substituting the value of α into the claim of Theorem 3.1. ∎

4 Flow With Costs

In this section we show how to modify the algorithm described in Section 3 to approximately solve the minimum cost multicommodity flow problem. More precisely, we consider the problem where every edge vw has an associated cost $c(vw)$, and the goal is to find a flow f that satisfies at least a $(1-\epsilon)$ fraction of each demand and whose cost $\sum_{i,vw}|f_i(vw)|c(vw) \leq (1+\delta)B$, where B is a given budget and $\delta > 0$ is some given parameter.

In this section we will assume that B is given and that there exists a feasible flow of cost at most B that satisfies all the demands. Note that by doing bisection search, one can easily extend our approximation algorithm to the case where B is unknown in advance. Details will appear in the full paper.

In order to take the costs into account, we will use the following potential function, which is a modification of the function used in the previous section.

$$\Phi(f) \stackrel{\text{def}}{=} \sum_{vw\in E}\sum_{i=1}^{k}\left[\exp\left(\alpha\frac{Ex_i(f,vw)}{d_i}\right) + \exp\left(\alpha\frac{Ex_i(f,wv)}{d_i}\right)\right]$$

$$+\frac{\gamma}{(1+\delta)B}\sum_i\sum_{vw\in E}c_i(vw)|f_i(vw)|$$

where $\gamma = m^2k^2e^{4/\delta}/\delta$ and $\alpha = \frac{m}{\epsilon}\log\gamma$. To simplify notation, we will use Φ_1 to denote the first term and Φ_2 to denote the second. Also, we will write Φ instead of $\Phi(f)$ where it will not cause confusion.

Observe that we can express $\Phi(f)$ as a sum of terms such that each term depends only on the flow on a single edge and is independent of the flows on the other edges. Thus, the distributed algorithm presented in Section 3 can be based on $\Phi(f)$ instead of the potential function given by (1). In other words, the only difference will be the minimization step, which will update the flow on each edge in order to minimize the corresponding term in $\Phi(f)$ instead of the corresponding term in (1).

As before, it is easy to see that the algorithm ensures that Φ does not increase. Roughly speaking, our analysis consists of two parts. First we show that a sufficiently small value of $\Phi(f)$ implies that we have good approximation to the MFCB problem. Then we prove that $\Phi(f)$ decreases sufficiently fast.

Theorem 4.1 **If preflow f obeys capacity constraints and $\Phi(f) \leq \gamma(1+\delta)$, then the cost of f is at most $(1+3\delta)B$ and the total excess of commodity i is bounded by ϵd_i.**

Proof: Omitted. ▮

Observe that using the standard technique of decomposing preflow into paths from sources to sinks, from positive to negative excesses, and into cycles, a preflow that satisfies the condition of the above theorem can be easily transformed into a flow of cost at most $(1+\delta)B$ that satisfies $(1-\epsilon)$ fraction of each demand. This transformation is done differently in the sequential and distributed setting and we defer the details to the full paper.

Theorem 4.2 **The algorithm that minimizes the potential function $\Phi(f)$ solves the MCFB problem in $O(\frac{m^2}{\epsilon^2\delta^2}\log^2(mk\delta^{-1})\log(mCU))$ iterations where C is the maximum cost and U is the maximum capacity associated with any edge in the graph.**

Proof: Let f^* denote the optimum flow of cost at most B that satisfies all the demands. Like in the proof of Theorem 3.2, we will use the fact that the decrease in the total potential function at each iteration is at least as large as the decrease obtained by setting the new flow to be $(1-\sigma)f + \sigma f^*$. Let $\Delta f = \sigma(f^* - f)$. Then, for $\sigma \leq 1/\alpha$, the decrease in the total potential function is at least:

$$\Phi(f) - \Phi(f+\Delta f) \geq \sigma\Phi_1(f)\log\frac{\Phi_1}{mk} - 4\sigma^2\alpha^2\Phi_1(f) + \sigma\Phi_2(f) - \sigma\frac{\gamma}{(1+\delta)}. \tag{6}$$

We will distinguish three cases, according to the relative values of Φ_1 and Φ_2. First consider the case where $\Phi_1 \geq \gamma\delta/2$ and $\Phi_2 \leq \Phi_1$. Using the fact that $\log \frac{\Phi_1}{mk} \geq 4/\delta$, we get that for

$$\sigma = \frac{\log \frac{\Phi_1}{mk}}{8\alpha^2}$$

the decrease is bounded by:

$$\frac{\log^2 \frac{\Phi_1}{mk}}{32\alpha^2}\Phi.$$

The number of iterations of this type is at least $O(\alpha^2/\log(mk))$.

The second type of iteration occurs when $\Phi_1 \geq \gamma\delta/2$ and $\Phi_2 \geq \Phi_1$. Using the same value of σ as above, the decrease per iteration can be bounded in this case by $\Omega(\Phi/\alpha^2)$. Since in this case $\gamma \leq \Phi \leq O(\gamma mCU)$, the number of such iterations is bounded by $O(\alpha^2 \log(mCU))$.

The third type of iteration occurs when $\Phi_1 \leq \gamma\delta/2$. We will bound the number of such iterations for $\Phi \in [(1+\eta)\gamma, (1+2\eta)\gamma]$ for $\eta \geq \delta$. When Φ is in this range, then $\Phi_2 \geq (1+\eta/2)\gamma$. Observe that the first term in expression (6) is always nonnegative. For $\eta \leq 1$, since

$$\Phi_2 - \gamma/(1+\delta) \geq \eta\Phi_2/3$$

by choosing

$$\sigma = \frac{\eta}{8\delta\alpha^2}$$

we see that the decrease in the potential function is at least $\Omega(\eta^2/(\alpha^2\delta)\Phi)$. Hence, the number of iterations of this type for $\Phi \in [(1+\eta)\gamma, (1+2\eta)\gamma]$ is bounded by $O(\alpha^2\delta/\eta)$. Thus, the total number of such iterations for the range $\Phi \in [(1+\delta)\gamma, 2\gamma]$ is bounded by $O(\alpha^2)$. For the case where $\Phi_1 \leq \gamma\delta/2$ and $\Phi \geq 2\gamma$, choice of $\sigma = 1/(8\alpha^2\delta)$ implies that the decrease is at least $\Omega(\alpha^2\delta)$, and hence the total number of such iterations is bounded by $O(\alpha^2\delta\log(mCU))$.

Observe that if we are not in any of the types of iterations addressed above, then $\Phi \leq (1+\delta)\gamma$. Since Φ monotonically decreases, the claim of the theorem follows by summing up the bound on the number of iterations of each type and applying Theorem 4.1. ∎

5 Implementation of a single iteration

The previous sections described the algorithms in terms of iterations, where the main part of each iteration is minimization of a set of independent functions of flow, one for each edge. In this section we will address the question of how to perform this minimization efficiently.

The main idea is to minimize a quadratic approximation to the potential function instead of the function itself. The proofs of Theorems 3.2 and 4.2 rely on the fact that the decrease in Φ as a function of the flow update Δf, can be bounded by a sum of independent quadratic functions in Δf (one per edge), as long as the following constraints are satisfied for every commodity i

and edge vw:

$$\begin{aligned} |\Delta f_i(vw)| &\leq d_i/\alpha \\ |f_i(vw) + \Delta f_i(vw)| &\leq d_i \\ \sum_i |f_i(vw) + \Delta f_i(vw)| &\leq u(vw). \end{aligned} \tag{7}$$

Moreover, these proofs show that there exists a Δf that satisfies the above constraints and for which the value of the sum of these quadratic functions is sufficiently large. Hence, instead of minimizing the potential function itself, we can achieve at least as large a decrease by maximizing these quadratic functions over the region defined by (7).

The advantage of working with the quadratic approximation is that it is relatively easier to optimize than the original potential function that includes exponentials. Observe that the quadratic functions that we need to maximize are concave. The following theorem proves that the optimization can be done in $O(k \log k)$ time per edge.

Theorem 5.1 Maximizing $\sum_{i=1}^{k} a_i x_i + b_i x_i^2$ where $b_i \leq 0$, subject to the constraints $\sum_{i=1}^{k} |x_i + y_i| \leq u$ and $e_i \leq x_i \leq f_i$ can be done in $O(k \log k)$ time.

Proof: Omitted.

References

[1] Y. Afek, B. Awerbuch, and E. Gafni. Applying Static Network Protocols to Dynamic Networks. In *Proc. 28th IEEE Annual Symposium on Foundations of Computer Science*, pages 358–370, 1987.

[2] B. Awerbuch and T. Leighton. A simple local-control approximation algorithm for multicommodity flow. In *Proc. 34rd IEEE Annual Symposium on Foundations of Computer Science*, pages 459–468, 1993.

[3] T. C. Hu. Multi-Commodity Network Flows. *J. ORSA*, 11:344–360, 1963.

[4] S. Kapoor and P. M. Vaidya. Fast Algorithms for Convex Quadratic Programming and Multicommodity Flows. In *Proc. 18th Annual ACM Symposium on Theory of Computing*, pages 147–159, 1986.

[5] 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*, 1992. Accepted for Publication.

[6] T. Leighton, F. Makedon, S. Plotkin, C. Stein, É. Tardos, and S. Tragoudas. Fast approximation algorithms for multicommodity flow problem. In *Proc. 23th ACM Symposium on the Theory of Computing*, pages 101–111, May 1991.

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

[8] S. A. Plotkin, D. Shmoys, and É. Tardos. Fast Approximation Algorithms for Fractional Packing and Covering. In *Proc. 32nd IEEE Annual Symposium on Foundations of Computer Science*, 1991.

[9] F. Shahrokhi and D. W. Matula. The maximum concurrent flow problem. Technical Report CSR-183, Department of Computer Science, New Mexico Tech., 1988.

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

[11] P. M. Vaidya. Speeding up Linear Programming Using Fast Matrix Multiplication. In *Proc. 30th IEEE Annual Symposium on Foundations of Computer Science*, 1989.

[12] P.M. Vaidya. A new algorithm for minimizing convex functions over convex sets. In *Proc. 30th IEEE Annual Symposium on Foundations of Computer Science*, pages 338–343, 1989.