Fast Approximation Algorithm for Minimum Cost Multicommodity Flow

Anil Kamath*  Omri Palmon†  Serge Plotkin‡

Abstract

Minimum-cost multicommodity flow problem is one of the classical optimization problems that arises in a variety of contexts. Applications range from finding optimal ways to route information through communication networks to VLSI layout.

In this paper, we describe an efficient deterministic approximation algorithm, which given that there exists a multicommodity flow of cost $B$ that satisfies all the demands, produces a flow of cost at most $(1 + \delta)B$ that satisfies $(1 - \epsilon)$-fraction of each demand. For constant $\delta$ and $\epsilon$, our algorithm runs in $O^*(kmn^2)$ time, which is an improvement over the previously fastest (deterministic) approximation algorithm for this problem due to Plotkin, Shmoys, and Tardos, that runs in $O^*(k^2m^2)$ time.

The presented algorithm is inherently parallel and can be implemented to run in $O^*(mn)$ time on PRAM with linear number of processors, instead of $O^*(kmn)$ time with $O(n^3)$ processors of the previously known approximation algorithm.

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. In the min-cost multicommodity flow problem, each edge has an associated cost and the goal is to find a flow of minimum cost that satisfies all the demands. 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 [1,2,3,4,5].

*Department of Computer Science, Stanford University. Research supported by U.S. Army Research Office Grant DAAL-03-01-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-03-01-G-0102, by a grant from Mitsubishi Electric Laboratories, by NSF grant number CCR-9304071 and by Terman Fellowship.
Since multicommodity flow approaches based on interior point methods for linear programming lead to high asymptotic bounds on running time, recent emphasis was on designing fast combinatorial approximation algorithms. If there exists a flow of cost $B$ that satisfies all the demands, the goal of an $(\epsilon, \delta)$-approximation algorithm is to find a flow of cost at most $(1 + \delta)B$ that satisfies $(1 - \epsilon)$ fraction of each demand.

The main result of this paper is an $O^*(kmn^2 \log(CU))$ deterministic approximation algorithm where $n$, $m$, and $k$ denote the number of nodes, edges, and commodities respectively; the constant in the big-$O$ depends on $\epsilon^{-1}$ and $\delta^{-1}$. The fastest previously known deterministic approximation algorithm for min-cost multicommodity flow problem due to Plotkin, Shmoys, and Tardos [12] runs in $O^*(k^2 m^2 \log(CU))$ time for constant $\epsilon$ and $\delta$. Our algorithm represents a significant improvement in the running time when $km \gg n^2$ which is the case for majority of multicommodity flow problems arising in the context of routing in high-speed networks and VLSI layout. For comparison, the best approximation algorithm based on interior point techniques is due to Kamath and Palmon [6] and is slower by a factor of $\Omega(k^{1.5} \sqrt{m/n})$ for constant $\epsilon$.

Combinatorial approximation algorithms for various variants of multicommodity flow can be divided according to whether they are based on relaxing the capacity [14, 9, 10, 12] or conservation constraints [11, 2]. In particular, the algorithm in [12] is based on relaxing the capacity and budget constraints. In other words, the algorithm starts with a flow that satisfies the demands but does not satisfy either the capacity or budget constraints. It repeatedly reroutes commodities in order to keep demands satisfied while reducing the amount by which the current flow overflows the capacities or overspends the budget.

Recently, Awerbuch and Leighton [11, 2] proposed several algorithms for multicommodity flow without costs that are based on relaxing the conservation constraints. More precisely, these algorithms do not try to maintain a flow that satisfies all of the demands. Instead, they maintain a preflow like in the single commodity flow algorithms of Goldberg and Tarjan [3, 4]. These algorithms repeatedly adjust the preflow on an edge-by-edge basis with the goal of maintaining capacity constraints while making preflow closer to a flow. One of the main advantages of this approach is that it leads to algorithms that can be naturally implemented in a distributed system.

The main problem of extending the algorithms in [11, 2] to the min-cost case is the fact that these algorithms are based on minimizing (at each step) a function that defines the penalties for not satisfying the conservation constraints. The reason these algorithms are fast is that this function is separable, i.e., it is a sum of non-linear terms where each term depends on a single edge and thus this function can be minimized on an edge-by-edge basis. A natural extension of this approach to the min-cost case is to introduce an additional penalty term that depends on the cost of the current preflow. Unfortunately, the resulting function is non-separable since this term has to be non-linear for the technique to work. This in turn leads to an algorithm that has to perform a very slow global optimization at each step. In fact, this optimization does not seem to be much easier than solving the original problem.

---

1. We say that $f(n) = O^*(g(n))$ if $f(n) = O(g(n) \log^k n)$ for constant $k$.
2. Randomized version of this algorithm runs faster by a factor of $k$. 

2
In this paper we propose a different approach that allows us to deal with the min-cost multicommodity flow case. Our algorithm builds on the feasibility multicommodity algorithm of Awerbuch and Leighton [2]. As in their algorithm we relax the conservation constraints. In addition we relax the budget constraint. Our algorithm maintains a preflow that always satisfies capacity constraints; the preflow is gradually adjusted through local perturbations in order to make it as cheap and as close to flow as possible. The main innovation lies in the pair of two functions – one that is separable (can be minimized on an edge-by-edge basis) and one that is not separable that are used to guide the flow adjustment at each step. Essentially the flow adjustment is guided by the separable function that is redefined at each step while the progress is measured by the other function which is non-separable.

Our algorithm can be viewed as a generalization of the algorithm in [2] to the min-cost case and as such preserves some of the advantages of that algorithm. In particular the edge-by-edge adjustment of the flow done by our algorithm is inherently parallel. The min-cost multicommodity flow algorithm in [12] is based on repeated computation of shortest paths. Although this computation can be done in NC it requires \( n^3 \) processors making it impractical. For the case where a linear number of processors is available (one for each variable i.e. commodity-edge pair) the algorithm in [12] runs in \( O^*(kmn \log CU) \) time for constant \( \epsilon \) and \( \delta \). In contrast our algorithm runs in \( O^*(mn) \) time. We believe that the inherent parallelism in our algorithm combined with locality of reference will lead to efficient implementation on modern superscalar computers.

Recently after the publication of the preliminary version of this paper Karger and Plotkin have discovered a new min-cost multicommodity flow algorithm [8]. Their algorithm is based on capacity relaxation and outperforms the algorithm described here in a sequential setting.

Section 3 presents the continuous version of the algorithm and proves fast convergence assuming that at each iteration the algorithm computes an exact minimization of a quadratic function for each edge. In Section 4 we show how to reduce the amortized time per iteration by using a rounding scheme that allows us to compute approximate minimization at each iteration. Section 5 presents a proof that this rounding does not increase the total number of iterations.

## 2 Preliminaries and Definitions

An instance of the multicommodity flow problem consists of a directed graph \( G = (V, E) \) a non-negative capacity \( u(vw) \) for every edge \( vw \in E \) and a specification of \( k \) commodities \( \Gamma \) 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 \( m \geq n \Gamma \) and that the graph \( G \) is connected and has no parallel edges.

A multicommodity flow \( f \) consists of a non-negative function \( f_i(vw) \) on the edges of \( G \) for every commodity \( i \) which represents the flow of commodity \( i \) on edge \( vw \). Given flow \( f_i(vw) \) the excess \( Ex_i(f, v) \) of commodity \( i \) at node \( v \) is defined by:
\( \text{(1)} \quad Ex_i(f, v) = d_i(v) - \sum_{w:w \in E} f_i(vw) + \sum_{w:w \in E} f_i(wv), \)

where \(d_i(s_i) = d_i(t_i) = -d_i\) and \(d_i(v) = 0\) otherwise. Since the algorithm will update the demands \(d_i\), we will use \(d_i'\) to denote the original demands.

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.

Multicommodity flow with costs problem is to find a feasible multicommodity flow whose cost with respect to a given nonnegative cost vector \(c\) is below a given budget \(B\):

\[ \sum_{vw} f(vw) c(vw) \leq B. \]

During the description of our algorithm it will be convenient to work with preflow\(\Gamma\) which is a flow that satisfies capacity constraints but does not necessarily satisfy the conservation constraints. An \((\epsilon, \delta)\)-approximation to the multicommodity flow problem is a feasible flow that satisfies demands \((1 - \epsilon)d_i\) and whose cost does not exceed \((1 + \delta)B\). An \((\epsilon, \delta)\)-preflow is a preflow of cost at most \((1 + \delta)B\) where the sum of the absolute value of excesses of each commodity \(i\) does not exceed \(cd_i\). Note that such preflow can be transformed into an \((\epsilon, \delta)\)-optimal solution in \(O(kmn)\) time.

### 3 Continuous algorithm

In this section we present the (slow) “continuous” version of the min-cost multicommodity flow algorithm. The improvement in running time due to rounding is addressed in the next section. If a feasible solution of cost \(B/(1 + \delta)\) satisfying all the demands exists\(\Gamma\) the algorithm produces an \((O(\epsilon), O(\delta))\)-preflow. We present here the algorithm for directed case; the undirected case can be treated similarly. We assume without loss of generality that the minimum node degree is \(\frac{m}{2n}\).

#### 3.1 The Algorithm

The algorithm starts with zero flow\(\Gamma\) which is a valid preflow. For every node pair \(v, w\) such that \(vw \in E\) or \(wv \in E\) and commodity \(i\) it maintains an excess \(Ex_i(f, vw)\).\(^3\) The excesses \(Ex_i(f, vw)\) are maintained such that for every commodity \(i\) at node \(v\) the excess \(Ex_i(f, v)\) as defined by (1) is equal to the sum of \(Ex_i(f, vw)\) for all \(w\) such that \(vw \in E\) or \(wv \in E\). Given a flow \(f\) we define the following potential function:

\[ \Phi = \Phi_1 + \gamma \Phi_2 \Phi_1 \]

\(^3\)Note that, in general, \(Ex_i(f, vw) \neq Ex_i(f, wv)\).
where

$$\Phi_1 = \sum_{i,vw \in E \text{ or } vw \in E} \exp \left( \frac{\alpha E_{x}(f, vw)}{d_i^*} \right),$$

$$\Phi_2 = \frac{1}{B} \sum_{e \in E} c(e) f(e).$$

We use $d_i^*$ to denote the original demand of commodity $i$. $\Phi_1$ is the current approximation to the current value of $\Phi_1$ and should satisfy:

$$\frac{1}{1+\delta} \Phi_1 \leq \tilde{\Phi}_1 \leq (1+\delta) \Phi_1.$$  

As we show below, the best performance is achieved for

$$\alpha = 8e^{-1} m \log(2mk),$$

$$\gamma = \alpha \epsilon / (2m).$$

Roughly speaking, $\Phi_1$ measures how close the current preflow is to a flow and $\Phi_2$ measures how close the cost of the current flow is to the budget $B$. Unfortunately, it is not clear how to use $\Phi$ to directly measure the progress of the algorithm since it is redefined each time $\Phi_1$ changes by more than a constant factor. Instead, we define another potential function:

$$\Psi = \log \frac{\Phi_1}{2mk} + \frac{\gamma}{1+\delta} \Phi_2.$$  

The algorithm starts from the zero flow and proceeds in phases. In the beginning of a phase we choose $\sigma = \frac{2m}{1+\alpha \gamma - \Psi}$, where $\Psi$ is the value of potential function $\Psi$ in the beginning of the phase. Each phase proceeds in iterations where each iteration consists of the following steps:

**Algorithm 1: (single iteration)**

1. Add capacity $\sigma u(vw)$ to each edge $vw \in E$. Change demand of each commodity $i$: $d_i = d_i + \sigma d_i^* \Gamma$ where $d_i^*$ is the original demand of commodity $i$. Update the value of $Ex$ accordingly.

2. Find the increment $\Delta f$ in the flow vector $f$ that minimizes $\Phi$ under the constraint $0 \leq \Delta f_i(vw) \leq \sigma d_i^*$ and $\sum \Delta f_i(vw)$ is less than the available capacity. Update $f = f + \Delta f$.

3. Recompute new excess $Ex_i(f, v)$ at each node and distribute 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 $v$.

4. Update the current value of $\tilde{\Phi}_1 \Gamma$ if necessary.

A phase is terminated either after $1/\sigma$ iterations or when the value of $\Psi$ falls below $\Psi_0 / 2$. In the end of a phase we scale down all the flows, the demands $\Gamma$ and the capacities (and hence all
the excesses by a factor of \(1 + r\sigma\Gamma\) where \(r\) is the number of iterations in this phase. Note that in the end of each phase the capacities and the demands are equal to the original capacities and the demands. Moreover, since we scale flows and capacities by the same factor \(\Gamma\) the preflow in the end of each phase satisfies the original capacity constraints.

The second step is the heart of the algorithm. Since function \(\Phi\) is separable, \(i.e.,\) can be written as a sum of functions where each one depends on the flow on a single edge, the algorithm computes change in flow \(\Delta f\) by minimizing the following function independently for each edge \(vw \in E:\)

\[
\sum_{i=1}^{k} \left[ \exp \left( \frac{\alpha E_x(f, vw) - \Delta f_i(vw)}{d_i^e} \right) + \exp \left( \frac{\alpha E_x(f, vw) + \Delta f_i(vw)}{d_i^e} \right) \right] + \frac{\gamma \Phi_1 c(vw) \Delta f_i(vw)}{B}.
\]

### 3.2 Bounding number of phases

In this section we show a bound on the number of phases of Algorithm 1. Initially, we assume that each phase is executed exactly as stated above, \(i.e.,\) the minimization in step 2 and the excess rebalancing at step 3 are computed exactly. First we show that sufficiently small value of \(\Psi\) implies that the current flow is close to optimum.

**Lemma 3.1** If \(\Psi \leq (1 + \delta)\gamma\), then the current preflow is \((2\epsilon, 3\delta)\)-optimum.

**Proof:** Assume that the current preflow is not \((2\epsilon, 3\delta)\)-optimum. There can be two cases: either there exists an edge \(vw\) with excess \(E_x(f, vw) > \frac{d_i^e}{\epsilon}\) or the current cost of the flow exceeds \((1 + 3\delta)B\). In the second case, \(\Phi_1 > (1 + \delta)\Gamma\) and hence the fact that \(\Phi_1 \geq 2mk\) implies that \(\Psi > (1 + \delta)\gamma\). If there exists an edge \(vw\) with excess \(E_x(f, vw) > \frac{d_i^e}{\epsilon}\) for

\[
\Phi_1 > \exp(\alpha \epsilon/m) \geq (2m)^8k^8.
\]

Thus:

\[
\log \frac{\Phi_1}{2mk} \geq 7 \log(2mk) = 1.75 \alpha \epsilon/(2m) > (1 + \delta)\gamma
\]

when \(\delta < 0.75\). \(\blacksquare\)

Observe that \(\Psi\) might increase during an iteration. On the other hand, the rescaling of the demands, flows and capacities at the end of each phase halves \(\Psi\). The main point of the following lemma is to show that this reduction is significantly larger than the total sum of increases in a single phase.

**Lemma 3.2** The increase in \(\Phi\) during a single iteration is bounded from above by:

\[
4\sigma^2\alpha^2 \frac{m}{\epsilon} \Phi_1 + \frac{\gamma}{1 + \delta} \sigma \Phi_1.
\]
We defer the proof of this lemma. Instead, we prove that it implies convergence in a small number of phases. Lemma 3.2 together with the definition of $\Psi$ gives a bound on the increase in $\Psi$:

**Lemma 3.3** The increase $\Delta \Psi$ in $\Psi$ during a single iteration is bounded by $4\sigma^2 \alpha^2 \frac{n}{m} + \gamma \sigma$.

**Proof**: Let $\Delta \Phi_1$ and $\Delta \Phi_2$ denote the change in $\Phi_1$ and $\Phi_2$ during one iteration, respectively. Denote the total increase in $\Phi$ during one iteration by $\Delta \Phi = \Delta \Phi_1 + \gamma \Phi_1 \Delta \Phi_2$. Lemma 3.2 implies that:

$$\Delta \Psi \leq \log \frac{\Phi_1 + \Delta \Phi_1}{\Phi_1} + \frac{\gamma}{1 + \delta} \Delta \Phi_2 \leq \frac{1}{\Phi_1} (\Delta \Phi_1 + \gamma \Delta \Phi_2 \frac{\Phi_1}{1 + \delta})$$

$$\leq \frac{1}{\Phi_1} (\Delta \Phi_1 + \gamma \Phi_1 \Delta \Phi_2) = \frac{\Delta \Phi}{\Phi_1}$$

$$\leq 4\sigma^2 \alpha^2 \frac{n}{m} + \sigma \frac{\Phi_1}{\Phi_1} \frac{\gamma}{1 + \delta} \leq 4\sigma^2 \alpha^2 \frac{n}{m} + \sigma \gamma$$

Note that we used the bounds on $\Phi_1$ given by (2).

**Lemma 3.4** Let $\Psi_0$ and $\Psi_s$ be the value of the potential function $\Psi$ at the beginning of a phase and at the end of this phase (after scaling of the demands and capacities), respectively. If $\Psi_0 \geq \gamma (1 + \delta)$ then $\Psi_s \leq (1 - \delta/4) \Psi_0$.

**Proof**: Observe that scaling down of demands and flows at the end of a phase can not increase $\Psi$. Hence, the claim holds for the case when the phase was terminated because $\Psi$ fell below $\Psi_0/2$. Otherwise, demands and flows are halved at the end of the iteration causing halving of the excesses. This corresponds to taking a square root of each one out of $2mk$ terms in $\Phi_1 \Gamma$ which causes $\log(\Phi_1 / (2mk))$ to go down by at least a factor of 2. Since $\Phi_2$ is linear in the value of the flow $\Phi_1$, we have:

$$\Psi_s \leq \frac{1}{2} \left( \Psi_0 + (4\sigma \alpha^2 \frac{n}{m} + \gamma) \right)$$

$$= \Psi_0 - \frac{1}{2} \left( \Psi_0 - (4\sigma \alpha^2 \frac{n}{m} + \gamma) \right)$$

$$\leq \Psi_0 - \frac{1}{2} \left( \delta \Psi_0 - 4\sigma \alpha^2 \frac{n}{m} \right)$$

where the last equality is implied by the assumption that $\Psi_0 > (1 + \delta) \gamma$. The desired result can be obtained by choosing

$$\sigma = \frac{\delta}{16\alpha^2 \frac{n}{m}} \Psi_0.$$
Theorem 3.5 The algorithm terminates in \(O(\delta^{-1} \log(m \epsilon^{-1}))\) phases, where each phase consists of \(O(\sigma^{-1}) = O(\epsilon^{-1} \epsilon^2 mn)\) iterations.

Proof of Lemma 3.2: Consider a min-cost multicommodity flow \(f^*\) that satisfies the original demands \(d^*_i\). Note that \(\sigma f^*_i(vw)\) is one of the possibilities for \(\Delta f^*_i(vw)\) computed in the second step of each iteration. Hence the total increase in \(\Phi\) in a single iteration is not worse than the increase in \(\Phi\) if we set

\[ \forall i : \Delta f^*_i(vw) = \sigma f^*_i(vw). \]

In order to bound increase in \(\Phi\) we will use the fact that \(e^{e^* + t} \leq e^* + te^* + t^2 e^*\) for \(|t| \leq 1/2\). Observe that our choice of \(\alpha\) and \(\sigma\) implies that \(\alpha \sigma d^*_i \leq d^*_i / 2\) for all \(i\) and hence we can use the above second order approximation. Consider the contribution to \(\Phi\) of the change in excesses due to increase in demand of every commodity \(i\) by \(\sigma d^*_i \) and due to flow change \(\Delta f = \sigma f^*\). Observe that for every \(i\) this operation does not change excesses of commodity \(i\) at its source \(s_i\) and its sink \(t_i\). Recall that we have assumed that there exists a solution of cost \(B/ (1 + \delta)\). Thus the cost of \(f^*\) is bounded by \(B/ (1 + \delta)\). To simplify notation denote

\[ \phi_i(f, v, w) = \exp \left( \frac{Ex_i(f, vw)}{d^*_i} \right). \]

Hence the increase in \(\Phi\) during a single iteration can be bounded by:

\[
\begin{align*}
\sum_{i, v \notin \{s_i, t_i\}} \alpha \sigma \sum_{w \in V \in E} (\phi_i(f, w, v) - \phi_i(f, v, w)) \frac{f^*_i(vw)}{d^*_i} \\
+ \sum_{i, v \in V \in E} \alpha^2 \sigma^2 (\phi_i(f, w, v) + \phi_i(f, v, w)) \left( \frac{f^*_i(vw)}{d^*_i} \right)^2 \\
+ \sigma \frac{\gamma}{1 + \delta} \hat{\Phi}_i.
\end{align*}
\]

We will bound each term in equation (3) separately. In order to bound the first term we decompose \(f^*\) into a collection of flow paths from sources to their respective sinks. Consider flow path \(P\) of commodity \(i\) from \(s_i\) to \(t_i\) and denote its value by \(\sigma f^*_i(P)\). The contribution of this flow path to the first term is bounded by:

\[
\sum_{vw \in P} \alpha \sigma (\phi_i(f, w, v) - \phi_i(f, v, w)) \frac{f^*_i(P)}{d^*_i}
\]

Since the excesses associated by node \(v\) with all its incident edges are the same (by Step 3 of ALGORITHM-1) the above contribution is zero. The fact that \(f^*_i(vw) \leq d^*_i\) implies that the second term can be bounded by:
\[ \sum_{i, v \in E} \alpha^2 \sigma^2 \left( \phi_i(f, w, v) + \phi_i(f, v, w) \right) \frac{f_i^2(vw)}{d_i^2} \]

Since we have assumed that the minimum degree is \( \frac{m}{2n} \), the contribution to this term of a flow path \( P \) of value \( f_i^*(P) \) is bounded by:

\[ \sum_{\substack{w, \in P \ v, w \in P}} \alpha^2 \sigma^2 \left( \phi_i(f, w, v) + \phi_i(f, v, w') \right) \frac{f_i^2(P)}{d_i^2} \leq 4\alpha^2 \sigma^2 \frac{n}{m} f_i^*(P) \phi_i. \]

The claim of the lemma follows from the fact that the sum of flow values \( f_i(P) \) over all flow paths of commodity \( i \) in the flow \( f^* \) is exactly \( d_i^* \).

Theorem 3.5 implies that for constant \( \epsilon \) and \( \delta \) we can find an \((\epsilon, \delta)\)-approximate preflow in \( O^*(mn) \) iterations where each iteration consists of minimizing convex function (3) for each edge. In fact, it is easy to see that the proof of Lemma 3.2 remains unchanged if we will optimize over a second-order approximation to (3). Thus, each iteration can be implemented in \( O(mk \log k) \) time which implies the following theorem.

**Theorem 3.6** Algorithm-1 can be implemented to run in \( O^*(\delta^{-2} \epsilon^{-2} km^2 n) \) time.

Observe that since each iteration can be implemented in \( O(\log k) \) time using \( km \) processors on PRAM, the algorithm can be implemented to run in \( O^*(\delta^{-2} \epsilon^{-2} mn) \) time in parallel as mentioned in the introduction.

### 4 Improving The Running Time

Each iteration of Algorithm-1 involves optimizing a quadratic function for each edge which can take up to \( O(mk \log k) \) time per iteration. Intuitively, the problem lies in the fact that a large fraction of the flow updates made by the continuous algorithm do not lead to a sufficiently large progress toward the solution. In this section, we will present a variation of Algorithm-1 which not only simplifies the algorithm but ensures that each flow update leads to a substantial progress. The improvement is based on the “packetizing” technique introduced in [2].

In Section 5 we will prove that this algorithm converges to a solution in asymptotically the same number of phases as Algorithm-1.

The first modification is to restrict Step 2 of the algorithm so that on any edge it considers
for optimization only those commodity/edge pairs \((i, vw \in E)\) that satisfy:

\begin{align*}
\text{(6)} \quad Ex_i(f, vw) &\geq Ex_i(f, vw) + 4\sigma d_i^* \\
Ex_i(f, vw) &\geq -\beta d_i
\end{align*}

where \(\beta = \log(mk\delta^{-1}v^{-1})/\alpha\). All the flow and excess vectors considered henceforth will be restricted to the subspace determined by this restriction. Also, the algorithm uses a new set of excesses \(\tilde{Ex}_i\) defined as follows:

\[
\tilde{Ex}_i(f, vw) = \begin{cases} 
Ex_i(f, vw) - \sigma d_i & vw \in E \\
Ex_i(f, vw) + \sigma d_i & vw \in E
\end{cases}
\]

Let \(\Phi'_{(i,vw)}(Ex)\) be the negated gradient of the potential function w.r.t. the flow vector for commodity \(i\) in edge \(vw \in E\) defined as:

\[
\Phi'_{(i,vw)}(Ex) = -\frac{\gamma}{B} c(vw) \tilde{\Phi}_1 + \frac{\alpha}{d_i} \left( \exp(\alpha \frac{Ex_i(f, vw)}{d_i^*}) - \exp(\alpha \frac{Ex_i(f, vw)}{d_i^*}) \right).
\]

Note that since the edges and commodities obey condition (6) the gradient vector \(\Phi'(\tilde{Ex})\) is non-negative. The updated algorithm is as follows:

**Algorithm-2 (single iteration)**

1. Add capacity \(\sigma u(vw)\) to each edge \(vw \in E\). Change demand of each commodity \(i\):
   
   \(d_i = d_i + \sigma d_i^*\) where \(d_i^*\) is the original demand of commodity \(i\). Update the value of excesses accordingly.

2. Find an unsaturated edge and a commodity \(i\) satisfying condition (6) with the largest gradient \(\Phi'_{(i,vw)}(\tilde{Ex})\) and move a unit \(\sigma d_i^*\) or an amount that is sufficient to saturate that edge (whichever is less). Update the flow and excesses. Observe that this is equivalent to finding the change \(\Delta f\) in the flow vector \(f\) that maximizes \(\sum \Phi'_{(i,vw)}(\tilde{Ex}) \Delta f_i(vw)\) under the constraints that \(0 \leq \Delta f_i(vw) \leq \sigma d_i^*\) and \(\sum \Delta f_i(vw)\) does not exceed available capacity.

3. Recompute new excesses \(Ex_i(f, vw)\) at each node \(v\). Rebalance excesses inside each node to within an additive error of \(\sigma d_i^*\) by moving excess in increments of at least \(\sigma d_i^*\).

4. Update the current value of \(\tilde{\Phi}_1\) if necessary.

As in Algorithm-1, a phase is terminated either after \(1/\sigma\) iterations or when the value of \(\Psi\) falls below \(\Psi_0/2\). In the end of a phase we scale down all the flows, the demands \(\Gamma\) and the capacities (and hence all the excesses) by a factor of \(1 + r\sigma\) where \(r\) is the number of iterations in this phase.
Using a heap on every edge-node pair, it is straightforward to implement Algorithm-2 to run in $O(\log k)$ time per excess update. Excess updates can be divided into two types depending on whether or not the update was large (exceeded $\sigma d_i^*$) or small. Observe that if flow of commodity $i$ on edge $vw$ was updated by less than $\sigma d_i^*$, the only reason this flow might not be updated again in the next iteration is if for some commodity $j$ the excess $E_j(f,vw)$ or $E_j(f,wv)$ was changed by a multiple of $\sigma d_j^*$ during rebalancing in Step 3 of the current iteration. If on the other hand, the excess updates add-up to $\sigma d_i^*$ over several iterations, then we can precompute the number of iterations and do one single excess update of $\sigma d_i^*$ at the end. Hence, the work associated with each update can be amortized over the large updates, that require $O(\log k)$ work each. Thus, it is sufficient to bound the number of large excess updates.

**Theorem 4.1** The total number of large updates to excesses in a single phase is bounded by $O^*(\epsilon^{-3} \delta^{-3} km n^2)$.

*Proof:* Let $\beta$ be defined as in condition (6) and consider a potential function

$$\Gamma = \sum_{i, v \in E} \max \left\{ -\beta - 2\sigma, \frac{E_i(f,vw)}{d_i} \right\}^2$$

Let $\Psi_0$ denote the value of $\Psi$ at the start of the phase. The fact that at the start of the phase $\log[\Phi_i/(2mk)] \leq \Psi_0$ together with the fact that $\Psi_0/\alpha \geq 2\sigma$ implies that the initial value of $\Gamma$ is bounded by $O(mk\Psi_0^2/\alpha^2 + mk\beta^2)$. Increase in $\Gamma$ due to increase in demands by $\sigma d_i^*$ during an iteration is bounded by

$$O(k\sigma(\Psi_0/\alpha + \beta + \sigma)) = O(k\sigma(\Psi_0/\alpha + \beta)).$$

Thus, the total increase in $\Gamma$ throughout a phase is bounded by $O(k(\Psi_0/\alpha + \beta))$. Note that each time we move $\sigma d_i^*$ amount of excess across an edge or within a node, the decrease in $\Gamma$ is $\Omega(\sigma^2)$ and hence the total number of updates can be bounded by

$$\frac{1}{\sigma^2} O(mk\Psi_0^2/\alpha^2 + mk\beta^2 + k(\Psi_0/\alpha + \beta))$$

and the claim follows from the observation that

$$\Omega\left(\frac{\epsilon}{m}\right) \leq \Psi_0 \leq O(\alpha).$$

Thus, we have the following Theorem:

**Theorem 4.2** Single phase of Algorithm-2 can be implemented to run in $O^*(\epsilon^{-3} \delta^{-3} km n^2)$ time.
5 Bounding the number of phases of Algorithm-2

In this section we analyze the behavior of Algorithm-2 during a single phase and show a bound on the number of phases needed for obtaining an $(\epsilon, \delta)$-approximate preflow. There are several important differences between Algorithm-1 and Algorithm-2. First, instead of exact rebalancing of excesses inside each node, we execute an approximate rebalancing. Second, we ignore commodities on edges with small potential differences. Third, we do not compute flow increment that leads to the largest reduction in the potential function.

First, we will show that the approximate rebalancing and ignoring commodity/edge pairs with small potential difference does not increase the number of phases by more than a constant factor.

Lemma 5.1 If redistribution of excesses in Step 3 of Algorithm-1 is done within an additive error of $2\sigma d^*_i$ and if we limit Step 2 to include only those edges and commodities that satisfy condition (6) then the increase in $\Phi$ during a single iteration is bounded from above by:

$$O \left( \sigma^2 \frac{n}{m} \right) \Phi_1 + \frac{\gamma}{1 + \delta} \sigma \Phi_1.$$ 

Proof: The main ideas in the proof are that relatively small imperfections during rebalancing in Step 3 add-up to only a minor increase in the potential function and that we do not lose much by ignoring edge-commodity pairs that do not give large reduction in the potential function.

Step 3 of Algorithm-2 redistributes excesses inside nodes within an additive error of $\sigma d^*_i$. That is:

$$\forall i, v \in V, w, w' \text{ adjacent to } v \text{ either:}$$

$$|Ex_i(f, vw) - Ex_i(f, vw')| \leq 2\sigma d^*_i$$

or $$-\beta d^*_i \geq \max \{Ex_i(f, vw), Ex_i(f, vw')\}.$$ 

Since for $x \leq 1$ we have $e^x - 1 \leq 3x\Gamma$ the contribution to the first term in (3) of a flow path $P$ of value $f^*_i(P)$ can be bounded by:

$$O \left( \sum_{w_1 \in P \land v \in P} \alpha^2 \sigma^2 \min(\phi_i(f, w, v), \phi_i(f, v, w')) \frac{f^*_i(P)}{d^*_i} \right) + O \left( \sum_{w_1 \in P \land v \in P} \alpha \sigma \exp(-\alpha \beta) \frac{f^*_i(P)}{d^*_i} \right).$$ 

If we consider the error due to disregarding commodity/edge pairs that have very small excesses i.e.

$$Ex_i(f, vw) \leq -\beta d_i$$
then the associated error can be bound by

\[
O\left( \sum_{uv \in P} \alpha \sigma \exp\left(-\alpha \beta \frac{f^*_i(P)}{d_i^*}\right) \right).
\]

Now consider the error introduced by disregarding commodity/edge pairs that do not satisfy

\[
Ex_i(f, vw) \geq Ex_i(f, wv) + 4\sigma d_i^*.
\]

Again using the fact that for \( x \leq 1 \Gamma \) we have \( e^x - 1 \leq 3x \Gamma \) the error associated with edge \( vw \) commodity \( i \) and path \( P \) can not exceed:

\[
\alpha \sigma \left( \exp\left( \frac{\alpha Ex_i(f, vw)}{d_i^*} \right) - \exp\left( \frac{\alpha Ex_i(f, wv)}{d_i^*} \right) \right) \frac{f^*_i(P)}{d_i^*} \leq \alpha \sigma \exp\left( \frac{\alpha Ex_i(f, wv)}{d_i^*} \right) (\exp (4\alpha \sigma) - 1) \frac{f^*_i(P)}{d_i^*} \leq O\left( \alpha^2 \sigma \exp\left( \frac{\alpha Ex_i(f, wv)}{d_i^*} \right) \frac{f^*_i(P)}{d_i^*} \right).
\]

By summing up over all the flow paths and over all the commodities we substituting the value of \( \beta \Gamma \) and using the fact that \( \Phi_1 \geq 2mk \Gamma \) we see that each of the three different types of error can be bounded by

\[
O(\alpha^2 \sigma \frac{n}{m} \Phi_1).
\]

Thus increase in \( \Phi \) during a single iteration remains bounded by

\[
O\left( \sigma^2 \alpha^2 \frac{n}{m} \right) \Phi_1 + \frac{\gamma}{1 + \delta} \sigma \Phi_1.
\]

We shall use the notation \( Ex + \Delta f \) to indicate the excesses that result after updating the flow by amount \( \Delta f \). The increase in potential function due to incrementing the demand depends only on the excesses at the beginning of the iteration and is independent of the technique used for incrementing the flow. Hence we need to bound only the change in potential function due to the flow updates. We first claim that using \( E \bar{x} \) instead of \( E x \) does not adversely affect the potential function reduction. The proof of the following Lemma is essentially identical to the proof of Lemma 5.1.2.

**Lemma 5.2** If \( f^* \) is the optimal solution then

\[
\Phi(E \bar{x} + \sigma f^*) - \Phi(E \bar{x}) \leq O\left( \sigma^2 \alpha^2 \frac{n}{m} \right) \Phi_1 + \frac{\gamma}{1 + \delta} \sigma \Phi_1.
\]

**Lemma 5.3** The change in potential function due to the flow increment \( \Delta f \) used by Algorithm-2 can be bounded by

\[
\Phi(Ex + \Delta f) - \Phi(Ex) \leq \Phi(E \bar{x} + \sigma f^*) - \Phi(E \bar{x}).
\]
\textit{Proof:} Since for all $i$ and $vw \in E$ the flow $\Delta f$ satisfies $0 \leq \Delta f_i(vw) \leq \sigma d_i$ and the excesses satisfy condition (6) we know that $\Phi'(Ex + \Delta f) \geq \Phi'(Ex)$. Using first-order Taylor series expansion we get
\[
\Phi(Ex + \Delta f) - \Phi(Ex) \leq -\Delta f^T \Phi'(Ex + \Delta f) \leq -\Delta f^T \Phi'(Ex).
\]
Furthermore since $\Delta f$ is the optimal solution in the space (that includes $\sigma f^*$) that maximizes the linear objective $\Delta f^T \Phi'(Ex)$ we have
\[
-\Delta f^T \Phi'(Ex) \leq -\sigma f^*T \Phi'(Ex) \leq \Phi(Ex + \sigma f^*) - \Phi(Ex).
\]
Hence the result follows. \qed

Combining Lemmas 5.2 and 5.3 we conclude that the increase in the potential function for the modified algorithm is a constant multiple of the bound obtained for Algorithm-1. Hence we have the following theorem.

\textbf{Theorem 5.4} \textit{Algorithm-2 terminates in $O(\delta^{-1} \log(m\epsilon^{-1}))$ phases.}

Using the above theorem together with Theorem 4.2 we get the following claim:

\textbf{Theorem 5.5} An $(\epsilon, \delta)$-approximation to the minimum-cost multicommodity flow problem can be obtained in $O^*(\epsilon^{-3} \delta^{-3} kmn^2)$ time.

\section*{Acknowledgements}

We would like to thank Baruch Awerbuch, Andrew Goldberg, Tom Leighton and Éva Tardos for helpful discussions.

\section*{References}


