IMPROVED APPROXIMATION ALGORITHM
FOR CONCURRENT MULTI-COMMODITY FLOWS

By

Éva Tardos

\[1\] The work was done in part while the author was visiting the Department of Mathematics at M.I.T. Research partially supported by Air Force contract AFOSR-89-0271 and DARPA contract N00014-89-J-1988.
1 Abstract

We give an $O(nm^2k\epsilon^{-1}(k + \epsilon^{-1}))$ time algorithm to find a feasible solution to the concurrent flow problem in both directed and undirected graphs with uniform capacities that is within a $(1 - \epsilon)$ factor to being optimal. Here $n$, $m$ and $k$ denote the number of nodes, edges or arcs and commodities. The algorithm is a simplified and improved version of an algorithm proposed by Shahrokhi and Matula [3].

Using linear programming one can obtain algorithms whose running time is also polynomial in $-\log \epsilon$. In the undirected case one has to use the ellipsoid method. In the directed case one can use the linear programming algorithm of Vaidya’s [4] to obtain a $O((mk)^{2.5}\log(\epsilon^{-1}))$ time algorithm. This algorithm uses a combination of general linear programming techniques and fast matrix multiplication. It takes advantage of the network structure only in speeding up matrix inversions. The new method is better for a wide range of the parameters even in the directed case.

2 Definitions and Preliminaries

In this section we define the concurrent flow problem and review the fundamental facts about it. We shall deal with two versions of the problem, the directed concurrent multi-commodity flow problem, and the undirected multi-commodity flow problem. A network is a directed graph $G = (V, E)$ with a nonnegative capacity function $u : E \to \mathbb{R}$. We shall use $n$ and $m$ to denote the number of nodes and the number of arcs in this graph, respectively. To simplify the bounds we assume that $m \geq n$. For notational convenience, we assume that the undirected graph underlying $G$ has no parallel edges. If there is an arc from a node $v$ to a node $w$, this arc is unique by assumption, and we denote it by $(v, w)$.

In the directed version a flow $f_{i,j}$ from node $i$ to node $j$ is a nonnegative function on the arcs of $G$, that satisfies the flow conservation constraints at every node other than $i$ and $j$. It’s value is the net flow into $j$. In the case of the undirected problem the notion of the flow and its value is defined analogously except we do not require that the flow to be nonnegative on the arcs.

The input to both the directed and the undirected concurrent multi-commodity flow problem consists of a network $G = (V, E)$ with a nonnegative capacity function $u$, a demand graph $D = (V, K)$, and a demand function $d : K \to \mathbb{R}^+$. We shall use $k = |K|$ to denote the number of commodities.

Given a collection of flows $f_{i,j}$ for $(i, j) \in K$ we shall use $f(v, w)$ to denote $\sum_{i,j} |f_{i,j}(v, w)|$. A feasible multi-commodity flow is a collection of flows $f_{i,j}$ of value $d(i, j)$ such that $f(v, w) \leq u(v, w)$ for every arc $(v, w) \in E$. The concurrent flow problem is to find the maximum value $z$ such that there exists a feasible multi-commodity flow satisfying demands $zd(i, j)$.

The directed version of the concurrent multi-commodity flow problem can be written as a linear program with $O(km)$ nonnegative variables and $O(nk+m)$ constraints. For the undirected version one needs exponentially many $(m2^k)$ inequalities to express the capacity constraints. Hence the ellipsoid method has to be applied to find an optimal solution in polynomial time.
Both versions of the problem can also be solved in strongly polynomial time as shown in [2].

Linear programming duality provides a characterization for the optimality of a solution. Let a length function \( l \) be a non-negative function \( l : E \rightarrow \mathbb{R} \). In the case of the directed problem we shall use \( \text{dist}_l(i,j) \) to denote the length of the shortest path in \( G \) from \( i \) to \( j \) according to the length function \( l \). In the case of the undirected problem \( \text{dist}_l(i,j) \) denotes the length of the shortest path from \( i \) to \( j \) in the undirected graph underlying \( G \) where \( l \) is the length of the edges.

**Theorem 2.1** For both the directed and the undirected concurrent multi-commodity flow problem, for any length function \( l \) the value

\[
\frac{\sum_{(v,w) \in E} l(v,w)u(v,w)}{\sum_{(i,j) \in K} d(i,j)\text{dist}_l(i,j)}
\]

is a an upper bound on the optimal value \( z \). A multi-commodity flow \( f \) is optimal if and only if there exists a length function \( l \) that provides a lower bound equal to \( z \).

The complementary slackness conditions provide a way of checking whether the multi-commodity flow \( f \) and the length function \( l \) are optimal.

**Theorem 2.2** A multi-commodity flow \( f \) is optimal if there exists a length function \( l \) such that

1. \( l(v,w) > 0 \) implies that \( f(v,w) = u(v,w) \),

2. For every commodity \( (i,j) \in K \) the flow \( f(i,j) \) can be decomposed into flows along paths and cycles, such that the cycles have length 0 and the paths have length \( \text{dist}_l(i,j) \).

The **approximation problem** for the concurrent flow problem is to find a feasible concurrent flow \( f \) satisfying the demands \( zd(i,j) \) such that \( z \) is within a factor \((1 - \epsilon)\) to optimal for a given error parameter \( \epsilon > 0 \).

The concurrent multi-commodity flow problem with uniform capacity is a problem where all capacities are the same. We will work with an equivalent version of this problem. We define the **cost** of a multi-commodity flow \( f \) that satisfies the demands \( d(i,j) \) to be \( f^* = \max_{(v,w) \in E} f(v,w) \). The equivalent problem is to find a multi-commodity flow \( f \) of minimum cost satisfying all the demands. The minimal \( f^* \) and the maximal \( z \) relate by \( zf^* = u \) where \( u \) denotes the uniform capacity.

Since scaling does not affect the problem we shall assume, without loss of generality, that \( \sum_{(i,j) \in K} d(i,j) = 1 \). Throughout this paper we shall use \( l^* \) to denote \( \sum_{(v,w) \in E} l(v,w) \) for a length function \( l \).

**Theorem 2.3** Let \( f \) be a multi-commodity flow satisfying the demands \( d(i,j) \) in a uniform concurrent multi-commodity flow problem, such that \( f^* \) is at most \( \epsilon \) more than the minimum. A solution to the concurrent flow problem whose value \( z \) is within a \((1 - \epsilon)\) factor of being optimal can be found in linear time.
Proof: It is easy to find a feasible multi-commodity flow \( f \) satisfying the demands \( d(i, j) \) with \( f^* \leq 1 \) due to the assumption that \( \sum_{(i, j) \in K} d(i, j) = 1 \). Therefore, it is no loss of generality to assume that \( f^* \leq 1 \). Consider the flow \( \frac{\mu}{z} f \). This is a feasible solution to the concurrent multi-commodity flow problem with demands \( \frac{\mu}{z} d(i, j) \). To see that \( \frac{\mu}{z} f \) is within the claimed factor to being optimal let \( z \) be the optimal value. The minimum value of a multi-commodity flow satisfying the demands is \( \frac{\mu}{z} \). By the assumption \( f^* \geq \frac{\mu}{z} + \epsilon \). Rearranging this inequality we get \( \frac{\mu}{z} \leq z(1 - \frac{1}{z}) \leq z(1 - \epsilon) \).

3 Relaxed optimality conditions

A key notion is that of approximate optimality, obtained by relaxing the complementary slackness constraints in Theorem 2.2. These conditions are analogous to the approximate optimality used by Goldberg and Tarjan in the context of the minimum-cost flow problem [1]. For a constant \( \epsilon \geq 0 \), a multi-commodity flow \( f \) and a length function \( l \) is \( \epsilon \)-optimal if the following conditions are satisfied. Let \( l^* \) denote the sum of the lengths of the arcs, i.e., \( l^* = \sum_{(v, w) \in E} l(v, w) \).

1. For every commodity \( (i, j) \in K \) \( f_{i,j} \) is a flow from \( i \) to \( j \) of value at most \( d(i, j) \) and at least \( d(i, j) - \frac{\epsilon}{mk} \).

2. For every arc \( (v, w) \in E \), we have \( l(v, w)(f^* - f(v, w)) \leq \frac{\epsilon}{m} l^* \).

3. For every commodity \( (i, j) \in K \) the flow \( f_{i,j} \) can be decomposed into flows along paths from \( i \) to \( j \) such that all the path have length at most \( dist_l(i, j) + \epsilon l^* \).

The most important property of this notion is that if a multi-commodity flow \( f \) is \( \epsilon \)-optimal then it is close to being optimal also in a different sense. Suppose \( f \) and \( l \) is a pair of \( \epsilon \)-optimal multi-commodity flow and length function. Modify \( f \) to be a multi-commodity flow satisfying the demands \( d(i, j) \) by increasing the flow \( f_{i,j} \) for every \( (i, j) \in K \) along an \( (i, j) \)-path of length \( dist_l(i, j) \) by the required amount. Let \( \hat{f} \) denote the resulting flow.

**Theorem 3.1** Suppose \( f \) and \( l \) is a pair of \( \epsilon \)-optimal multi-commodity flow and length function and \( \hat{f} \) is the multi-commodity flow constructed above. The cost \( \hat{f}^* \) of the multi-commodity flow \( \hat{f} \) is at most \( 3\epsilon \) more than the optimum cost.

**Proof:** By Theorem 2.1 the cost

\[
\frac{\sum_{(i, j) \in K} dist_l(i, j)d(i, j)}{l^*},
\]

where \( l^* = \sum_{(v, w) \in E} l(v, w) \), is a lower bound on the optimum value.

The first condition of \( \epsilon \)-optimality implies that each flow \( f_{i,j} \) is increased by at most \( \frac{\epsilon}{mk} \) by the construction of \( \hat{f} \). Therefore \( \hat{f}^* - f^* \leq \frac{\epsilon}{m} \). This and the second condition imply that for every arc \( (v, w) \in E \) the flow \( \hat{f} \) satisfies \( l(v, w)(\hat{f}^* - \hat{f}(v, w)) \leq \frac{2\epsilon}{m} l^* \). Summing these inequalities over all arcs we get that
\[ \hat{f}^* l^* - \sum_{(i,j) \in K} f_{i,j}(v, w)l(v, w) \leq 2\epsilon l^*. \]  

(2)

The flow \( \hat{f} \) also satisfies condition (3) of \( \epsilon \)-optimality if the shortest path with the added flow is considered as an extra path in the decomposition. Therefore, for every commodity \( (i, j) \in K \)

\[ \sum_{(v,w) \in E} f_{i,j}(v, w)l(v, w) \leq d(i, j)(\text{dist}_t(i, j) + \epsilon l^*). \]

Summing this for all commodities gives

\[ \sum_{(i,j) \in K} \sum_{(v,w) \in E} f_{i,j}(v, w)l(v, w) - \sum_{(i,j) \in K} d(i, j)\text{dist}_t(i, j) \leq \sum_{(i,j) \in K} d(i, j)\epsilon l^* = \epsilon l^*. \]  

(3)

Inequalities (1), (2) and (3) together imply that \( \hat{f}^* \) is at most \( 3\epsilon \) above the optimum.

4 The algorithm

The main idea of the algorithm is that it maintains a multi-commodity flow \( f \) and a length function \( l \) that satisfies conditions (1) and (2) of \( \epsilon \)-optimality and will gradually enforce condition (3) by redirecting the flow onto shorter paths. The algorithm maintains that the \( f_{i,j}(v, w) \) is an integer multiples of a parameter \( \sigma \) (to be chosen later) for all arcs \( (v, w) \) and all commodities \( (i, j) \). To make this possible the original demand \( d(i, j) \) for a commodity \( (i, j) \in K \) has to be replaced by the rounded amount

\[ d'(i, j) = \sigma \left\lfloor \frac{d(i,j)}{\sigma} \right\rfloor. \]

Lemma 4.1 If \( \sigma \leq \frac{\epsilon}{mK} \) and \( f \) is a multi-commodity flow satisfying the demands \( d' \) then condition (1) of \( \epsilon \)-optimality is satisfied.

The algorithm will use the same length function as Shahrokhi and Matula [3]. The length of an arc is a sole function of the flow on the arc, and it exponentially increase as the flow increases. The idea is to make paths that contain arcs with high flow values long. Let \( f \) be a multi-commodity flow. We define the length function \( l \) associated to the multi-commodity flow \( f \) to be

\[ l(v, w) = \exp(\alpha f(v, w)) \]

(4)

for \( (v, w) \in E \), where \( \alpha \) is a parameter that will be defined later.

Lemma 4.2 If \( \alpha \geq \frac{\epsilon}{m} \) then for every multi-commodity flow \( f \) and the associated length function \( l \) satisfy condition (2) of \( \epsilon \)-optimality.
We assume that:

For every \((i, j) \in K\) we have a flow \(f_{i,j}\) of value \(d'(i, j)\) from \(i\) to \(j\) that is decomposed into flows along at most \(m\) paths.

Define the length \(l(v, w)\) of an arc \((v, w)\) by equation 4.

if No path in the decompositions violates (3) of \(\varepsilon\)-optimality
then return (The flow \(f\) and the length function \(l\) are \(\varepsilon\)-optimal.);
else begin
Let \(P\) be a path in the decomposition of flow \(f_{i,j}\) that violates (3) of \(\varepsilon\)-optimality.
Update the flow \(f_{i,j}\) by decreasing it by \(\sigma\) along \(P\), and increasing by \(\sigma\) along a shortest \((i, j)\)-path.
Find a decomposition of the new flow \(f_{i,j}\) into flows along at most \(m\) paths and cycles. Decrease the flow along the cycles and delete the cycles from the decomposition.
end;

Figure 1: An iteration of the algorithm.

Proof: By the definition of \(l\) we have that \(l^* \geq \exp(\alpha f^*)\). From elementary calculus \(\exp(-x) \geq \frac{1}{x}\) for any positive real \(x\). Using these two inequalities we have that for an arc \((v, w)\)

\[
l(v, w) = \exp(\alpha f(v, w)) \leq \exp(-\alpha(f^* - f(v, w)))l^* \leq \frac{1}{\alpha(f^* - f(v, w))}l^*.
\]

Therefore, \(l(v, w)(f^* - f(v, w)) \leq \frac{1}{\alpha}l^* \leq \frac{\varepsilon}{m}l^*\) as required.

Throughout the algorithm \(\alpha\) and \(\sigma\) will satisfy the inequalities in the above two lemmas. The algorithm will start with initializing \(f_{i,j}\) to be a fairly arbitrary flow of value \(d'(i, j)\) from \(i\) to \(j\) (a flow of this value on a simple path). It maintains that \(f_{i,j}(v, w)\) is an integer multiple of \(\sigma\) for every commodity \((i, j) \in K\) and every arc \((v, w) \in E\); and each flow \(f_{i,j}\) can be decomposed into a sum of flows along at most \(m\) simple paths.

In each iteration the algorithm computes the length function associated with the current flow. If the flow and this length function satisfy condition (3) of \(\varepsilon\)-optimality then the algorithm terminates. Otherwise it chooses a commodity \((i, j) \in K\) and an \((i, j)\)-path \(P\) in the decomposition of \(f_{i,j}\) that violates the condition. The iteration decreases the flow \(f_{i,j}\) along \(P\) by \(\sigma\) and increase it along an \((i, j)\)-path of length \(dist_l(i, j)\) by the same amount. Next a decomposition of \(f_{i,j}\) into flows along paths and cycles is computed, and the flow is decreased along the cycles to cancel the cycles from the decomposition. (See also Figure 1.)

Remark: Decreasing the flow along a "long" path and increasing it along a "shorter" path is equivalent to augmenting the flow along some negative cycles in the residual graph. However, working with an explicit decomposition seems to be advantageous, it gives a limit on how much flow is moved by \(f_{i,j}\).

We end this section by bounding the time required for an iteration. The number of iterations will be bounded in the next section.
Lemma 4.3 An iteration of the algorithm can be implemented using $O(nmk)$ arithmetic operations.

Proof: The main parts of the computation involved in an iteration is to compute the length of the paths in the decomposition, and to find a new decomposition for the flow $f_{i,j}$ after the change. The first can be accomplished in $O(n)$ time per paths. There are at most $m$ paths per commodity. This amounts to $O(nmk)$ time. Finding a new decomposition and cancelling the flow along cycles takes $O(nm)$ time.  

5 The number of iterations

The progress of the algorithm will be measured in the decrease in $l^*$ for the associated length function $l$.

Lemma 5.1 Every iteration decreases $l^*$ by a factor of $(1 - \frac{1}{2} \epsilon \alpha \sigma + 2 \alpha^2 \sigma^2)$.

Proof: The algorithm changes $f$ and in two steps in every iteration. First by redirecting $\sigma$ flow from $P$ to a shortest $(i,j)$-path, and second when canceling the flow on the cycles in the new decomposition. We shall prove that the effect of the first change is to decrease $l^*$ by at least the claimed factor. Cancelling the flow along cycles decreases it on some arcs. The length on an arc is a monotone function of the flow value. Hence this change cannot increase $l^*$.

For the rest of this proof let $A$ denote the edges (arcs in the directed case) in the path $P$ that are not in the shortest path, and $B$ the set of edges (or arcs) in the shortest path that are not in $P$. The algorithm will decrease the flow on edges in $A$ and increase the flow on edges in $B$. Let $f$ and $l$ denote the flow and the length function at the beginning of the iteration. $P$ does not satisfy condition (3) of $\epsilon$-optimality, therefore $l(A) - l(B) \geq \epsilon l^*$. For an arc $(v,w)$ in $B$ the first flow defined in this iteration will have $f(v,w) + \sigma$, and the corresponding length is $\exp(\alpha(f(v,w) + \sigma)) = \exp(\alpha \sigma)(l(v,w))$. Similarly the new length of an arc $(v,w)$ in $A$ will be $\exp(-\alpha \sigma)(l(v,w))$. The length of the arcs not in $A \cup B$ does not change. Therefore the decrease in $l^*$ is

$$l(B) + l(A) - \exp(\alpha \sigma)l(B) - \exp(-\alpha \sigma)l(A)$$
$$= (1 - \exp(-\alpha \sigma))(l(A) - l(B)) - (1 - \exp(-\alpha \sigma))(\exp(\alpha \sigma) - 1)l(B)$$
$$\geq \frac{1}{2} \alpha \sigma l^* - 2 \alpha^2 \sigma^2 l^*. \quad (5)$$

Here the inequality was obtained using the estimates $l(A) - l(B) \geq \epsilon l^*$, $l(B) \leq l^*$; $\exp(x) \geq 1 + x$; and the two inequalities $\exp(x) \leq 1 + 2x$ and $\exp(-x) \leq 1 - \frac{1}{2}x$ valid if $0 \leq x \leq 1$.

Using $(1 - x) \leq \exp(-x)$ we get the following corollary.

Corollary 5.1 Every $(\epsilon \alpha \sigma - 4 \alpha^2 \sigma^2)^{-1}$ iterations decrease $l^*$ by a constant factor.

Next we establish the range that $l^*$ might have to decrease.

Lemma 5.2 A multi-commodity flow $f$ that satisfies the properties maintained by the algorithm and has $l^* \leq m \exp(\alpha)$ can be found in linear time. Every multi-commodity flow satisfies $l^* \geq m$. 

7
Proof: An appropriate initial flow can be defined by letting \( f(i,j) \) have value \( d'(i,j) \) on a simple \((i,j)\)-path. This implies that the maximum flow on an arc is at most \( \sum_{(i,j) \in K} d'(i,j) \) and, by assumption, this is at most 1. Therefore \( l(v,w) \leq \exp(\alpha) \) for every arc, and \( l^* \leq m \exp(\alpha) \) as required.

Any flow \( f \) satisfies \( f(v,w) \geq 0 \) for every \((v,w) \in E\) therefore \( l(v,w) \geq 1 \) and \( l^* \geq m \).

If the staring solution is \( \epsilon \)-optimal for some \( \epsilon \) such that \( f^* \geq 3\epsilon \) then a stronger version of the above lemma is true.

**Lemma 5.3** Suppose the multi-commodity flow \( f \) and the corresponding length function \( l \) is \( \epsilon \)-optimal. Then \( l^* \leq m \exp(\alpha f^*) \), and throughout the algorithm \( l^* \geq \exp(\alpha(f^* - 3\epsilon)) \).

Proof: By Theorem 3.1 \( f^* \geq f^* - 3\epsilon \) for every multi-commodity flow \( f \) satisfying the demands \( d'(i,j) \). Therefore the corresponding length function \( l \) has \( l^* \geq \exp(\alpha(f^* - 3\epsilon)) \).

**Theorem 5.4** If \( \alpha \geq \frac{m}{\epsilon}, \sigma \leq \frac{\epsilon}{mk} \) and \( \epsilon \sigma - 4\alpha \sigma^2 > 0 \) then the above algorithm will find an \( \epsilon \)-optimal flow and length function in \( O((\epsilon \sigma - 4\alpha \sigma^2)^{-1}) \) iterations. If the initial solution is \( O(\epsilon) \)-optimal then an \( \epsilon \)-optimal solution is found in \( O((\sigma - 4\alpha \sigma^2)^{-1}) \) iterations.

Proof: The first two inequalities guarantee that conditions of (1) and (2) of \( \epsilon \)-optimality are satisfied throughout the algorithm and if the algorithm terminates the current flow and length function is \( \epsilon \)-optimal as claimed. By the previous lemma \( l^* \) decreases by a constant factor every \( (\epsilon \sigma - 4\alpha \sigma^2)^{-1} \) iterations. Hence, by Lemma 5.2 and Lemma 5.3, the number of iterations is at most \( O((\epsilon \sigma - 4\alpha \sigma^2)^{-1}) \) and \( O((\sigma - 4\alpha \sigma^2)^{-1}) \) in the two cases, as claimed.

We shall analyze two algorithms. The first one is a simplified version of the original algorithm of Shahrokhi and Matula [3]. However, even this version is substantially faster. In this version the parameters \( \alpha \) and \( \sigma \) are chosen at the beginning of the algorithm and the algorithm simply consists of repeating the steps on Figure 1 until an \( \epsilon \)-optimal solution is found. To analyze this algorithm it remains to choose the parameters \( \alpha \) and \( \sigma \). The second version is a scaling algorithm. It is divided into phases. The \( \mu \)-scaling phase starts with a \( 2\mu \)-optimal solution and ends with a \( \mu \)-optimal solution. Then \( \mu \) is divided by two and a new scaling phase is started. The parameters \( \sigma \) and \( \alpha \) will vary from phase to phase.

**Theorem 5.5** With the right choice of \( \sigma \) and \( \alpha \) the algorithm on Figure 1 finds an \( \epsilon \)-optimal multi-commodity flow and length function in \( O(nm^2k\epsilon^{-2}(k + \epsilon^{-1})) \) time.

Proof: The right choice of parameters depends on the relation of \( k \) and \( \epsilon^{-1} \).

If \( k \leq 8\epsilon^{-1} \) then \( \alpha = \frac{m}{\epsilon} \) and \( \sigma = \frac{\epsilon^2}{8m} \) satisfies the required inequalities. The resulting number of iterations is \( O(me^{-3}) \) according to the first part of Theorem 5.4.

If \( k \geq 8\epsilon^{-1} \) then \( \alpha = \frac{mk}{8} \) and \( \sigma = \frac{\epsilon}{mk} \) satisfies the required inequalities. The resulting number of iterations is \( O(mk\epsilon^{-2}) \) according to the first part of Theorem 5.4.

**Theorem 5.6** With the right choice of \( \sigma \) and \( \alpha \) the version of the algorithm that uses scaling runs in \( O(nm^2k\epsilon^{-1}(k + \epsilon^{-1})) \) time.
Proof: The algorithm consists of scaling phases. It starts with the zero flow and \( \mu \geq mk \). This flow and the corresponding length function is \( \mu \)-optimal. The current scaling parameter \( \mu \) will be decreased by a factor of 2 from phase to phase until it reaches \( \epsilon \). During the \( \mu \)-phase we shall use the parameters \( \alpha \) and \( \sigma \) defined in the previous proof depending on the relation of \( 8\mu \) and \( k \). For the continuity of the scaling phases we choose the initial value of \( \mu \) so that \( \frac{k}{\mu} \) is a power of 2. At the end of the \( 2\mu \)-phase we have a \( 2\mu \) optimal multi-commodity flow \( f \) satisfying the demands \( \sigma'[\frac{d(i,j)}{\sigma}] \). The flow is an integer multiples of \( \sigma' \) on every edge, where \( \sigma' \) is the value of \( \sigma \) during the \( 2\mu \) phase. \( \sigma' \) is either \( 2\sigma \) or \( 4\sigma \) depending on the relation of \( 8\mu \) and \( k \). At the beginning of the \( \mu \)-phase each flow is augmented along the current shortest path to the value \( \sigma[\frac{d(i,j)}{\sigma}] \). Each flow has to be augmented by at most \( O(\sigma) \) additional flow. The choice of \( \mu \) implies that \( mk\sigma \leq \mu \). Therefore the resulting flow is \( O(\mu + km\sigma) = O(\mu) \)-optimal. By the second half of Theorem 5.4 a \( \mu \)-optimal solution is found in \( O(m\mu^{-2}) \) iterations.

The overall number of iterations is bounded by a constant multiple of

\[
\sum_{i = -\log_2(mk)}^{\log_2 k + 1} mk2^i + \sum_{i = \log_2(4k + 3)}^{-\log_\epsilon} m2^{2i} = O(mk\epsilon^{-1} + m\epsilon^{-2}).
\]

Now using Lemma 4.3 we obtain the claimed running time. □

Acknowledgment

I would like to thank Andrew Goldberg, Serge Plotkin and David Shmoys for many valuable discussions on the multi-commodity flow problem and the algorithm presented in this paper.

References


