Distributed Algorithms for Multicommodity Flow Problems via Approximate Steepest Descent Framework

Baruch Awerbuch [*] Rohit Khandekar[†] Satish Rao[‡]

Abstract
We consider solutions for distributed multicommodity flow problems, which are solved by multiple agents operating in a cooperative but uncoordinated manner. We show first distributed solutions that allow $1 + \epsilon$ approximation and whose convergence time is essentially linear in the maximal path length, and is independent of the number of commodities and the size of the graph.

Our algorithms use a very natural approximate steepest descent framework, combined with a blocking flow technique to speed up the convergence in distributed and parallel environment.

Previously known solutions that achieved comparable convergence time and approximation ratio required *exponential* computational and space overhead per agent.

1 Introduction

1.1 Motivation Consider a collection of *uncoordinated* agents trying to optimize a certain *global* objective. The agents must follow a specific protocol (namely, local program of the agent); our goal is to design such a local program in order to achieve fast convergence to the near-optimum solution. Notice that the fact that the agents are uncoordinated does not preclude them from executing the same local program, as long as everybody is honest. Certainly, in some scenarios agents may act dishonestly and deviate from the protocol, either in order to increase their own revenue at the expense of others (selfish behavior), or simply in order to sabotage the system (malicious behavior). This paper only deals with honest case, and does not deal with the issue of designing either incentives to prevent selfish behavior or defenses against malicious behavior.

A canonical distributed optimization problem is solving a Linear Program in a distributed environment. From the computational point of view, the following case is interesting: we have exponential number of variables, yet polynomial dimensionality. One classical example is multicommodity flow: there is an exponential number of paths, yet the dimensionality of the solution is certainly bounded by the number of edges.

Informally, the above network bandwidth management problems can be modeled as multicommodity flow problems in a directed capacitated graph, with a collection of *commodities*, each characterized by the following: source (where the flow is originated), sink (where the flow ends up), benefit (the monetary value of this flow), and demand (the amount of flow available). The normalized load of each edge is the ratio between flow on this edges and its capacity. The flow for each commodity must satisfy Kirchoff laws of flow conservation at intermediate points (except for source and destination). The collection of all the flows must satisfy capacity constraints, namely total flow on each edge cannot exceed its capacity.

While multicommodity flow is a classical combinatorial optimization problem, it also directly addresses a number of practically important issues of congestion and bandwidth management in connection-oriented network architectures, as stated below[1]:

Pure Routing: decide how to route *all the demand* to minimize maximal edge load.

Pure Flow Control: decide which fraction of a demand commodity is *admitted* (rest is *rejected*), assuming infinite demand, and assuming routing is predetermined to go over a single path.

Combined routing & flow control: combination of the two decisions above: whom do admit, and how to route.

There are a number of variations of classical multicommodity flow and Linear Programming problems that describe above network problems.

Maximum concurrent flow (MCF): maximizes the minimal ratio between the flow of a commodity and its demand (disregarding their benefit), i.e., minimized maximum edge load while meeting all flow demand. This is exactly the *pure routing* problem in networks.

Maximum-benefit flow (MBF): maximizes the total benefit of all the flows (disregarding their demand); it simultaneously optimizes routing and admis-

[*] Johns Hopkins University. baruch@cs.jhu.edu. Partially supported by NSF grants CCF 0515080, ANIR-0240551, and CCR-0311795, and CNS-0617883.

[†] University of Waterloo. rkhandekar@gmail.com.

[‡] University of California, Berkeley. satishr@eecs.berkeley.edu.

[1] Current Internet routing architecture is not properly addressing these issues, and thus is not supporting quality of service guarantees.

sion control to maximize overall network throughput. This is exactly the combined routing and flow control problem in networks.

Positive Linear Programming (LP): (also called pure packing problem) corresponds to solving *pure flow control* problem in network; each variable corresponds to flow on the unique fixed path for the commodity. Note that polynomial representations of MCF and MBF problems involve negative coefficients, e.g., to capture Kirchoff's flow preservation laws and thus do not fit into this framework.

One can introduce separate variables for flows of each commodity on each path, rather than for edges, and thus represent multicommodity flows as positive LP's. This formulation allows solving MCF and MBF problems in this framework. The computational overhead of such representation is *linear* in the number of paths and *super-polynomial* (in the worst case, exponential) in the size of the network, even for relatively short (logarithmic) length paths. In a distributed setting, positive LP has been widely studied recently [GY02, LN93, BBR97, You01] and can be considered solved in a satisfactory manner. The resulting distributed solutions for flow problems suffer from exponential representation issue. In this paper, we focus on solutions using polynomial representations that do not require exponential overhead.

1.2 Distributed Computation Model

We assume a collection of agents, each agent representing a different flow, interacting with a shared "billboard" in a synchronous manner. The billboard maintains the current state of total network flows, namely keeps record of total flow on each edge of the network, without distinguishing which commodity is this flow coming from. At each time step, each agent may "read" the values of flow on various network edges from this billboard, perform local computation, and then "write" its own flows on this billboard, namely re-routing its flow based on the current state of the network flows.

At each time step, each agent decides *where* to send its flow (this is the *routing* decision) and *how much* of its flow will be sent overall (this is the *admission control* decision). These decisions are made, at each time step, by all the agents in parallel and without coordination with other agents, except for the fact that the agents have access to the common clock, they see the congestion of the network edges being used by the flows of their commodity, and they all execute the same program.

In order to make our model more realistic, we make additional restriction on the use of the billboard, namely that an agent responsible for a certain commodity can only read the part of the billboard which is currently used by its flow. The reason for this restriction is that in reality, the flow over a certain path can measure congestion over the edges used in this path, but cannot see congestion over the other edges used by other flows, since the flows are uncoordinated.

For example, if we consider a set of clients, each client being able to send a job to one of the "near-by" servers, then the problem becomes load-balancing in a bi-partite graph [ANR92], where each client can see the load of near-by servers and decide how to re-balance its flow based on this limited feedback.

Also, since this model does not allow any explicit communication between agents, it is impossible to solve the problem by a centralized algorithm as in, say, [PST95], [GK98]. In fact it is not even a-priori obvious that approximately optimal solutions are even *computable.*

Complexity measures. As the time goes on, the quality of the flows should improve in that the flow decisions should approach the optimum flow. The *approximation ratio* of the flow is the ratio between the performance of this flow and optimal performance.

The *distributed convergence time* (for certain approximation ratio $1+\epsilon$) is the number of parallel rounds it takes for the distributed algorithms to start meeting such bounds. The *computational complexity* of each agent is the local computational overhead imposed by local program of each agent, in terms of number of local computational steps.

The distributed convergence time is an information-theoretic measure that is independent of the model of computation, while computational complexity is model-dependent and thus is not particularly robust. It is thus reasonable to draw distinction between polynomial time and super-polynomial time in this model.

1.3 Our results

We will denote by $\tau(MCF)$ and $\tau(MBF)$ the distributed convergence times of computation of maximum concurrent flow and maximum-benefit flow, and we denote by $\mu(MCF)$ and $\mu(MBF)$ the total amount of computation by each agent. Let k denote the number of commodities, and let m, n denote the number of edges and nodes in the network. Let an integer L, $1 \leq L < n$, be the maximal path length in a near-optimal flow solution (this is essentially a constant in many important applications); note that $L = 1$ in a bi-partite case. Denote by $\mathrm{P} < n^L$ the total number of paths. Our main results are stated in Theorem 1.1 and 1.2, respectively and Figure 1 compares them with the existing results.

Reference	Approximation	Distributed Time	Sequential work
[AA94]	$O(\log n)$	$\tilde{O}(\log \mathrm{P}) = \tilde{O}(L)$	$\tilde{O}(\mathrm{P}) = \tilde{O}\left(n^L\right)$
[LN93, BBR97, You01]	$1+\epsilon$	$\tilde{O}(\log \mathrm{P}) = \tilde{O}(L)$	$\tilde{O}(\mathrm{P}) = \tilde{O}\left(n^L\right)$
This paper	$1+\epsilon$	$\tilde{O}(\log \mathrm{P}) = \tilde{O}(L)$	$\tilde{O}\left(m^2 \cdot \log \mathrm{P}\right) = \tilde{O}\left(m^2 \cdot L\right)$

Figure 1: Comparison of our results with existing work ($\tilde{O}(\cdot)$ absorbs polylog factors).

THEOREM 1.1. *For the MCF algorithm in §2.2, the convergence time and computation cost are:*

$$\begin{aligned}\tau(MCF) &\leq \frac{L}{\epsilon^6}\log^3 m \log\frac{k}{\epsilon}\\ &= \tilde{O}(\log \mathrm{P}) = \tilde{O}(L) \text{ and}\\ \mu(MCF) &= O\left(\frac{L\cdot m^2}{\epsilon^6}\log^3 m\log\frac{k}{\epsilon}\right)\\ &= \tilde{O}\left(m^2\cdot\log \mathrm{P}\right) = \tilde{O}\left(m^2 L\right).\end{aligned}$$

THEOREM 1.2. *For the MBF algorithm in §3.2, the convergence time and computation cost are:*

$$\begin{aligned}\tau(MBF) &\leq \frac{L}{\epsilon^4}\log^2 m \log\frac{k}{\epsilon}\\ &= \tilde{O}(\log \mathrm{P}) = \tilde{O}(L) \text{ and}\\ \mu(MBF) &= O\left(\frac{L\cdot m^2}{\epsilon^4}\log^2 m\log\frac{k}{\epsilon}\right)\\ &= \tilde{O}\left(m^2\cdot\log \mathrm{P}\right) = \tilde{O}\left(m^2 L\right).\end{aligned}$$

1.4 Comparison with existing work The only existing parallel methods with comparable convergence times and approximation ratios apply only to positive LP, namely pure "packing" or pure "covering" or a combination of pure packing and covering. These results were achieved by Luby and Nisan [LN93], Awerbuch and Azar [AA94], Bartal-Byers-Raz [BBR97], and Young [You01]. As mentioned before, multicommodity flow problem do not fit naturally into this framework. That is, in order to fit this framework, and exponential computational overhead is needed with explicit representation of flow paths. Assuming flow paths with length bounded by L, the resulting algorithms can converge in $\tilde{O}(\log \mathrm{P}) = \tilde{O}(L)$ rounds at the expense of exponential computational overhead, $\mu(MCF), \mu(MBF)$ for which only *exponential* upper bounds existed $\tilde{O}(\mathrm{P}) = \Omega(n^L)$ (Figure 1).

The essence of our improvement over prior work [LN93], [AA94], [BBR97], [You01] is that, without compromising on the convergence time we accomplish, for the first time in the literature, a solution with *polynomial* computation overhead of $\tilde{O}(\log P) = \tilde{O}(L)$.

1.5 Intuitive explanation of the algorithm The basic idea of our algorithm is properly exploiting the concepts of Lagrangian relaxation and steepest descent. The idea of Lagrangian relaxation is to introduce an exponential potential function, that sums up exponents of congestion over all network edges. Suppose that initially all the flows are zero and we increase the flows slowly to meet the demand. While we do this, the potential function grows and our goal is to keep the potential function under control.

Alternatively (for MCF problem) we can introduce a fictitious edge with low capacity connecting the source and the sink for each commodity, and send initially all the flow for this commodity over this fictitious edge. In the future, we will re-route the flow from such a fictitious edge to real network edges, thus creating impression that the demand routed through the real network is only growing.

For the MBF problem, introduce a fictitious commodity with almost zero benefit and a fictitious edge between its source and sink, and route all benefit that needs to be accrued through this edge, using only fictitious commodity.

Notice that if we can eventually satisfy the demand or benefit requirements while keeping the potential function increase to be at most polynomially higher than the potential increase using optimal flows, then this is equivalent to achieving $1+\epsilon$ approximation.

In order to achieve this goal we proceed in a *steepest descent* manner: we reroute portion of the flow from the expensive paths to the cheapest paths, just like in [PST95] or [GK98]. The problem is how to efficiently implement this in a distributed concurrent environment. The difficulty here is quite similar to what has been reported, for example in [AA94] and to some extent in [EDM05, FRV06], namely, a concurrent attempt to reroute on a shortest path causes this path to stop being the shortest, causing unpredictable oscillations.

We suggest a new algorithmic technique to handle such oscillations, which achieves the same effect as in exponential overhead approaches of Luby and Nisan [LN93], Bartal et al. [BBR97], and Young [You01] without exponential overhead.

Essentially, this involves extracting the sufficient conditions for approximate steepest descent framework to be efficient, and finding a polynomial algorithm that meets these conditions. Specifically, a sufficient condition is that the flow of each commodity on each edge increases at most multiplicatively (or by a polynomially small additive amount). The efficient algorithm that we propose finds a maximal collection of such flow augmentations using the blocking flow subroutine.

1.6 Outline of the rest of the paper The technical contents of the paper consists of the following. We start with a description of the basic steepest descent framework for MCF problem in §2.1 and the distributed MCF algorithm in §2.2. We present the corresponding steepest descent framework for MBF problem in §3.1 and the distributed MBF algorithm in §3.2.

2 Maximum Concurrent Multicommodity Flow Problem

The maximum concurrent multicommodity flow problem (MCF) is defined as follows. Let $G = (V, E)$ be a directed graph with edge-capacities $c_e \geq 0$. There are k *commodities* $i = 1, \ldots, k$. The commodity i is associated with a source $s_i \in V$, a sink $t_i \in V$, and flow demand $d_i \geq 0$. The objective is to route, for each commodity i, a flow of value d_i from s_i to t_i possibly split along several paths such that the maximum ratio of the total flow routed along an edge to its capacity is minimized. Let f_e^i be the flow of commodity i routed along edge e. The edge e is said to have a congestion of $\mathrm{cong}_e = (\sum_i f_e^i)/c_e$. Thus we want to minimize $\max_e \mathrm{cong}_e$.

Here we describe an efficient distributed algorithm that computes a $(1+\epsilon)$-approximate solution to the above problem for any given $\epsilon > 0$.

2.1 An *approximate steepest descent* framework for MCF algorithms Our algorithm is motivated by the algorithm of Garg and Könemann [GK98] and we begin by describing a framework for MCF algorithms inspired by their algorithm. Let us assume for simplicity that the demands are scaled so that the optimum value of $\max_e \mathrm{cong}_e = 1$. The algorithm will employ *approximate steepest descent* in order to minimize the convex potential

$$\Phi = \sum_e \left(m^{1/\epsilon}\right)^{\mathrm{cong}_e}$$

where m is the number of edges. Note that an $m^{O(1)}$-approximation of Φ yields a $(1+O(\epsilon))$-approximation for $\max_e \mathrm{cong}_e$. This follows from a simple fact that

$$\begin{aligned}
\max_e \mathrm{cong}_e &\leq \frac{\epsilon}{\log m}\log\Phi \\
&\leq \frac{\epsilon}{\log m}\log(m^{O(1)}\Phi^*) \\
&\leq O(\epsilon) + \frac{\epsilon}{\log m}\log(m\cdot(m^{1/\epsilon})^{\max_e \mathrm{cong}_e^*}) \\
&\leq 1 + O(\epsilon).
\end{aligned}$$

Here Φ^* and cong_e^* denote the potential and the congestion on edge e under the optimum routing. We show that the algorithm described here indeed computes an $m^{O(1)}$-approximation of the minimum Φ.

Initially all the flows f_e^i are zero. The algorithm goes in $T = (\log m)/\epsilon^2$ phases and routes a flow of value d_i/T for each commodity i in each phase, thereby constructing a feasible solution in the end. A phase, in turn, is divided into several steps. In each step, each commodity augments its flow along certain paths *simultaneously.* Let $\{f_e^i\}$ be the current flow values. Suppose commodity i augments a flow of Δf_e^i along an edge e. The overall increase in Φ due to this augmentation along e is at most

$$(2.1) \qquad \sum_i \Delta f_e^i \cdot \frac{\log m}{\epsilon c_e}\left(m^{1/\epsilon}\right)^{\mathrm{cong}'_e}$$

where cong'_e is the congestion on edge e after this step. This follows from the inequality $e^\delta - 1 \leq \delta e^\delta$ where $\delta = (\sum_i \Delta f_e^i \log m)/(\epsilon c_e)$. Let us associate a length $l_e = \frac{1}{c_e}(m^{1/\epsilon})^{\mathrm{cong}_e}$ with an edge e. In order to minimize the increase in Φ, the algorithm augments the flow of commodity i along $(1+\epsilon)$ approximate shortest paths from s_i to t_i under this length function. The amount of flow augmented is subject to the *step-size constraint* that for any edge e, its length increases by at most an ϵ fraction. That is, the total flow $\Delta f_e = \sum_i \Delta f_e^i$ sent through e in a step should satisfy $l_e(\Delta f_e \log m)/(\epsilon c_e) \leq \epsilon \cdot l_e$, i.e., $\Delta f_e \leq \epsilon^2 c_e/\log m$. In this framework, we intentionally do not specify a particular way of augmenting the flow. We rather prove that any augmentation that satisfies the *routing-along-short-paths* and the *step-size* constraints yields a good approximation.

Our assumption that the optimum value of $\max_e \mathrm{cong}_e = 1$ implies that the optimum value of Φ is $\Phi^* \geq m^{1/\epsilon}$. The following lemma proves that at the end of the algorithm, the value of Φ is an $m^{O(1)}$-approximation of the optimum, thereby establishing that the algorithm computes an $(1+O(\epsilon))$ approximation to the maximum congestion. Let $\Phi(t)$ denote the value of the potential Φ after phase t.

LEMMA 2.1. $\Phi(T) \leq m^{O(1)} \cdot m^{1/\epsilon}$.

Proof. Fix a phase t and let us estimate the change in the potential, $\Phi(t) - \Phi(t-1)$, in this phase. To this end, we first analyze the change in Φ in a single step in this phase. Let $l^{(1)}$ and $l^{(2)}$ be the length functions in the beginning and at the end of a step respectively. The increase in Φ in this step is at most $\frac{\log m}{\epsilon} \sum_i \Delta f_i \cdot (1+\epsilon) l_i^{(2)}$ where Δf_i is the flow sent for commodity i in this step. This follows from the expression (2.1). The factor of $(1+\epsilon)$ is due to the fact that the flow is routed along $(1+\epsilon)$ approximate shortest paths which have length at most $(1+\epsilon)l_i^{(2)}$. Since the shortest path length $l_i(t)$ between the ith pair at the end of phase t is at least $l_i^{(2)}$ and since we route a total flow of $\epsilon^2 d_i/\log m$ for each commodity i in each phase, we have,

$$\begin{aligned}\Phi(t) - \Phi(t-1) &\leq \frac{\log m}{\epsilon} \sum_i \frac{\epsilon^2 d_i}{\log m} \cdot (1+\epsilon) l_i(t) \\ &= \epsilon(1+\epsilon) \sum_i d_i l_i(t). \qquad (2.2)\end{aligned}$$

PROPOSITION 2.1. $\Phi(t) \geq \sum_i d_i l_i(t)$.

Proof. We prove this in two different ways. The first proof is based on linear programming duality. The LP dual of the MCF problem can be viewed as an assignment of the lengths $l_e \geq 0$ to the edges such that $(\sum_e c_e l_e)/(\sum_i d_i l_i)$ is minimized where l_i is the shortest path length from s_i to t_i under the length function l. Since the optimum value of $\max_e \text{cong}_e$ is 1, it follows from weak duality that for any length function l, we have $(\sum_e c_e l_e)/(\sum_i d_i l_i) \geq 1$. Since $\Phi(t) = \sum_e c_e l_e(t)$, we conclude $\Phi(t) \geq \sum_i d_i l_i(t)$.

Our second proof unfolds the LP duality and is self contained. Let $\{g_e^i\}$ be the flow in the optimum solution to the MCF problem. Thus we have $\sum_i g_e^i \leq c_e$ for each edge e. Furthermore since the solution $\{g_e^i\}$ routes d_i units of flow from s_i to t_i and since $l_i(t)$ is the shortest path length from s_i to t_i, we have $\sum_i \sum_e g_e^i l_e(t) \geq \sum_i d_i l_i(t)$. Thus we have

$$\Phi(t) = \sum_e c_e l_e(t) \geq \sum_i \sum_e g_e^i l_e(t) \geq \sum_i d_i l_i(t).$$

Combining the above proposition with inequality (2.2), we get

$$\Phi(t) \leq \frac{\Phi(t-1)}{1-\epsilon(1+\epsilon)}.$$

Since $\Phi(0) = m$ and $T = (\log m)/\epsilon^2$, we have $\Phi(T) \leq m(1+O(\epsilon))^{(\log m)/\epsilon^2} \leq m^{O(1)} \cdot m^{1/\epsilon}$ as desired.

We remark that the Garg-Könemann algorithm [GK98] for MCF is an instance of the above framework. In each phase of their algorithm, flows of different commodities are routed one after another along short paths. A single commodity is routed in any step, and hence it is easy to ensure the step-size constraint. They route the flows along the shortest path and saturate the minimum capacity edge on this path.

2.2 Our distributed MCF algorithm We are now ready to present our distributed algorithm which is also an instance of the above framework. We are seeking for a distributed algorithm where there is a minimal co-ordination between different agents. Only global information accessible to the agents is a common clock and the congestion on the edges accessible by them. We would like to route the flows of all commodities in parallel. We therefore use a special way of ensuring the step-size constraint. We initially route a tiny amount of flow of all commodities on all edges and later increase this flow *multiplicatively.* The initial flow may not even satisfy the flow conservation constraints. However the total capacity used in this initial pre-flow is ϵ fraction of the edge-capacities, thus this affects the optimality only to an extent ϵ.

Since the algorithm is an instance of the framework described in the previous section, we need to specify the details of how a phase and a step is implemented. The complete description of the algorithm is given in Figure 2. The algorithm has $T = (\log m)/\epsilon^2$ phases and each phase has $T_p = O(L(\log^2 m \log(k/\epsilon))/\epsilon^4)$ steps, where L denotes the maximum number of edges on any path between a source-sink pair. Let f_e^i be the flow of commodity i on edge e at some stage in the algorithm. In the following step, we allow the additional amount, Δf_e^i, of commodity i to be routed on this edge, to be at most $\epsilon^2 f_e^i/\log m$. Since we maintain the feasibility invariant $\sum_i f_e^i \leq c_e$, the total additional flow allowed is at most $\epsilon^2 c_e/\log m$. Thus the step-size constraints are satisfied. Unlike the Garg-Könemann algorithm, however, we now augment flows for each commodity by computing *blocking flows* under these step-size constraints and the constraint that the flow needs to be routed along short paths. Due to this, we end up saturating at least one edge on each (approximately) shortest path. Using this, we are able to show that we need only $T_p = O(L\cdot\text{polylog}(mk))$ steps in order to send $\epsilon^2/\log m$ fraction of the entire demand.

Now we prove that the above algorithm indeed routes d_i/T demand of each commodity i in each phase. This would then imply that since it adheres to the framework, it computes a $(1+O(\epsilon))$ approximation at the end of $T = (\log m)/\epsilon^2$ phases.

- **Initialize**: Set $f_e^i \leftarrow \epsilon c_e/k$ for each edge e and commodity i.
- For $T = O((\log m)/\epsilon^2)$ phases do:

  Phase: For $T_p = O(L(\log^2 m \log(k/\epsilon))/\epsilon^4)$ steps do:

  Step: Let $\{f_e^i\}$ be the current flow values. For each commodity i such that we have not yet routed d_i/T flow between its source-sink pair in this phase do in *parallel*:

  1. Define the capacities $c_e^i = \epsilon^2 f_e^i / \log m$.
  2. Compute a blocking flow under capacities c_e^i from s_i to t_i that routes flows along $(1+\epsilon)$ approximate shortest paths under the current length function, that is, no $(1+\epsilon)$ approximate shortest path has a residual capacity.
  3. Route the above flow subject to the constraint that the flow of commodity i routed in this phase does not exceed d_i/T.

Figure 2: A distributed algorithm for the MCF problem

LEMMA 2.2. *The number of steps, $T_p = O(L(\log^2 m \log(k/\epsilon))/\epsilon^4)$, in a phase is sufficient to route d_i/T flow for each commodity i.*

Proof. Imagine that a phase is run until each commodity i has sent its desired flow. We argue that this phase needs at most T_p steps. Consider a commodity i. Each blocking flow (except perhaps the last one) for this commodity saturates at least one edge on every $(1+\epsilon)$ shortest path. The total flow f_e^i of this commodity on such a saturated edge e increases by a $(1+\epsilon^2/\log m)$ factor. Since the initial flow on this edge was $\epsilon c_e/k$ and it never exceeds $(1+O(\epsilon))c_e$, the maximum number of times an edge can be saturated is $O((\log m \log(k/\epsilon))/\epsilon^2)$. Because any path has at most L edges, after $O(L(\log m \log(k/\epsilon))/\epsilon^2)$ steps, the shortest-path length increases by a factor of at least $(1+\epsilon)$. Now from Lemma 2.1, during the entire course of the algorithm, the shortest path length increases by a factor of at most $m^{O(1)+1/\epsilon}$. Therefore after $O(L(\log^2 m \log(k/\epsilon))/\epsilon^4)$ blocking flow computations, the commodity i must have sent at least d_i/T flow.

LEMMA 2.3. *The algorithm can be implemented in $O(L(\log^3 m \log(k/\epsilon))/\epsilon^6 \cdot T_{bf})$ parallel running time (with k processors) and $O(Lk(\log^2 m \log(k/\epsilon))/\epsilon^4 \cdot T_{bf})$ sequential running time, where $T_{bf} = \tilde{O}(m^2)$ denotes the time needed for a single commodity blocking flow computation along $(1+\epsilon)$ approximate shortest paths.*

Proof. The parallel running time follows directly from the bounds on the number of phases and number of steps in each phase. Each step, recall, involves computing at most k blocking flows, one for each of the k commodities.

For the sequential time, we argue that each of the k commodities needs at most $O(L(\log^2 m \log(k/\epsilon))/\epsilon^4)$ blocking flow computations. This, in turn, follows from the argument in the proof of Lemma 2.2 that after $O(L(\log m \log(k/\epsilon))/\epsilon^2)$ steps, the shortest path length increases by a factor of at least $(1+\epsilon)$. Now note that during the algorithm, since the length of any edge increases by a factor of at most $m^{O(1)+1/\epsilon}$, the shortest path length increases by at most the same factor. Therefore, the total number of blocking flow computations for a commodity is at most $O(L(\log^2 m \log(k/\epsilon))/\epsilon^4)$.

The desired blocking flow can be computed in $\tilde{O}(m^2)$ time simply by doing $O(m)$ successive shortest path computations and saturating at least one edge after each computation.

3 Maximum Benefit Multicommodity Flow Problem

The maximum benefit multicommodity flow problem (MBF) is defined as follows. Similar to that in the MCF problem, let $G = (V, E)$ be a directed graph with edge-capacities $c_e \geq 0$. There are k *commodities* $i = 1, \ldots, k$. This time, the commodity i is associated with a source $s_i \in V$, a sink $t_i \in V$, and a per-unit-flow benefit $b_i \geq 0$. The objective is to route a flow f_i from s_i to t_i possibly split along several paths such that the entire flow can be routed without violating the capacity constraints, i.e., the total flow routed through edge e is at most c_e, and the total benefit $\sum_i b_i f_i$ is maximized.

Here we describe a distributed algorithm that achieves a $(1+\epsilon)$ approximation.

3.1 An *approximate steepest descent* framework for MBF algorithms We now describe a framework for MBF problem that is motivated by the algorithms of Garg and Könemann [GK98] and Fleischer [Fle00] and is similar to the one for the MCF problem. We again consider the same convex potential

$$\Phi = \sum_e \left(m^{1/\epsilon}\right)^{\text{cong}_e}$$

where $\text{cong}_e = \sum_i f_e^i/c_e$ is the congestion on edge e due to the flows f_e^i of commodities i on edge e. As before, we associate a length $l_e = \frac{1}{c_e}(m^{1/\epsilon})^{\text{cong}_e}$ with an edge e.

Initially all the flows f_e^i are zero. The algorithm goes in several steps. In each step, each commodity augments its flow along certain paths *simultaneously.* These augmentations are subject to two constraints:

1. (*routing-along-beneficial-paths*) the commodity i is allowed to augment its flow along a path of length l under the current length function only if l/b_i is at most $(1+\epsilon)\min_j l_j/b_j$ where l_j is the shortest path length from s_j to t_j under the current length function,
2. (*step-size*) the total flow Δf_e of all commodities together sent along e in a step should be at most $\epsilon^2 c_e/\log m$.

The algorithm stops when the potential Φ crosses $m^{1/\epsilon}/e^\epsilon$ for the first time. Here, again, we prove that no matter how the commodities augment their flows, as long as they satisfy the above *routing-along-beneficial-paths* and *step-size* constraints, the algorithm yields a good approximation.

Lemma 3.1. *The algorithm computes a solution that satisfies* $\max_e \text{cong}_e \le 1$ *and* $B \ge B^*(1-O(\epsilon))$ *where* B *and* B^* *are the benefits accrued by our and the optimum algorithm respectively.*

Proof. Note that the length of any edge increases by a factor of at most $e^\epsilon \approx (1+\epsilon)$ in any step. Therefore the final potential Φ is at most $m^{1/\epsilon}$. Therefore in the end,

$$\max_e \text{cong}_e \le \frac{\epsilon}{\log m}\log m^{1/\epsilon} \le 1.$$

Now fix a step t and observe that the change in potential, $\Phi(t)-\Phi(t-1)$, in this step is at most

$$\frac{\log m}{\epsilon}(1+\epsilon)\sum_i \Delta f^i l_i(t)$$

where Δf^i denotes the flow of commodity i routed in this step, $l(t)$ denotes the length function at the end of step t, and $l_i(t)$ denotes the shortest path length from s_i to t_i under this length function. This follows from an argument similar to the one in Lemma 2.1. Let $\alpha(t) = \min_i l_i(t)/b_i$ be the minimum cost to benefit ratio at the end of step t. Since the flow was subject to the routing-along-beneficial-paths constraint, we have

$$\Delta f^i l_i(t) \le (1+\epsilon)^2\alpha(t)\cdot b_i\Delta f^i.$$

One factor of $(1+\epsilon)$ is due to the fact that we route flows along $(1+\epsilon)$ approximate beneficial paths. The second factor of $(1+\epsilon)$ is due to the fact that the lengths $l_e(t)$ are at most $(1+\epsilon)$ times the lengths during this step. The above inequality, in turn, implies that

$$\sum_i \Delta f^i l_i(t) \le (1+\epsilon)^2\alpha(t)\cdot B(t) \tag{3.3}$$

where $B(t)$ denotes the total benefit accrued by the algorithm in step t.

Proposition 3.1. $\Phi(t)/\alpha(t) \ge B^*$.

Proof. We prove this in two different ways. The first proof is based on LP duality. The LP dual of the MBF problem can be viewed as an assignment of the lengths $l_e \ge 0$ to the edges such that $(\sum_e c_e l_e)/(\min_i l_i/b_i)$ is minimized where l_i denotes the shortest path length from s_i to t_i under the length function l. Since the optimum benefit is B^*, it follows from weak duality that for any length function l, we have $(\sum_e c_e l_e)/(\min_i l_i/b_i) \ge B^*$. Applying this observation to the length function $l(t)$ proves the proposition.

Our second proof unfolds the LP duality and is self contained. Let $\{g_e^i\}$ be the flow in the optimum solution to the MBF problem. Thus this flow accrues a total benefit of B^* and satisfies $\sum_i g_e^i \le c_e$ for each edge e. Now we have

$$\begin{aligned}\Phi(t) &= \sum_e c_e l_e(t)\\ &\ge \sum_i\sum_e g_e^i l_e(t)\\ &\ge B^*\cdot\min_i l_i(t)/b_i\\ &= B^*\cdot\alpha(t).\end{aligned}$$

The second inequality follows from the fact that the flow g_e^i is not more beneficial than the most beneficial paths.

Combining the above proposition with inequality (3.3) and relating it to the change in the potential, we conclude

$$\frac{\Phi(t)-\Phi(t-1)}{\Phi(t)} \le (1+\epsilon)^3\frac{\log m}{\epsilon}\cdot\frac{B(t)}{B^*}.$$

- **Initialize**: Set $\alpha \leftarrow 1/\max_i b_i$. Set $f_e^i \leftarrow \epsilon c_e/k$ for each edge e and commodity i.
- For $T = O((\log m)/\epsilon^2)$ phases do:

  Phase: For $T_p = O(L(\log m \log(k/\epsilon))/\epsilon^2)$ steps do:

  Step: Let $\{f_e^i\}$ be the current flow values. For each commodity i such that $l_i/b_i < \alpha(1+\epsilon)$ do in *parallel*:

  1. Define the capacities $c_e^i = \epsilon^2 f_e^i/\log m$.
  2. Route a blocking flow under capacities c_e^i from s_i to t_i along $(1+\epsilon)$ approximate shortest paths under the current length function, that is, no $(1+\epsilon)$ approximate shortest path has a residual capacity.

  End of phase: Set $\alpha \leftarrow \alpha(1+\epsilon)$.

Figure 3: A distributed algorithm for the MBF problem

Since the left-hand-side is at most $e^\epsilon - 1 \approx \epsilon$, using the approximation $(1-\epsilon)\log(\Phi(t)/\Phi(t-1)) \le (\Phi(t)-\Phi(t-1))/\Phi(t)$, we get

$$(1-\epsilon)\frac{\epsilon}{\log m}\log\frac{\Phi(t)}{\Phi(t-1)} \le (1+\epsilon)^3\frac{B(t)}{B^*}.$$

Since the initial potential is $\Phi(0) = m$, the final potential is $\Phi(T) \le m^{1/\epsilon}$, by telescoping sum we conclude that $B \ge B^*(1-O(\epsilon))$.

3.2 Our distributed MBF algorithm We now describe our distributed algorithm for the MBF problem. During the algorithm we maintain a value α that satisfies $\alpha \le \min_i l_i/b_i < \alpha(1+\epsilon)$ where l_i is the shortest path length from s_i to t_i under the current length function. Initially we set $\alpha = 1/\max_i b_i$ and increase it by a factor of $(1+\epsilon)$ as soon as there is no commodity i such that $l_i/b_i < \alpha(1+\epsilon)$. We call the duration corresponding to a fixed value of α a phase. Since the length of any edge increases by a factor of at most $m^{1/\epsilon}$, the value of α also increases by at most the same factor during the course of the algorithm. Thus the total number of phases is $O((\log m)/\epsilon^2)$. This idea of keeping track of the current length-to-benefit ratio was first used by Fleischer [Fle00].

We now describe the initialization and how to implement a phase of the algorithm. See Figure 3. We initialize the flows as before to occupy an ϵ fraction of the edge-capacities. This affects the feasibility only to an extent ϵ. Each of the $O((\log m)/\epsilon^2)$ phases is further divided into $T_p = O(L(\log m \log(k/\epsilon))/\epsilon^2)$ steps. In each step, each commodity i that has beneficial paths, i.e., paths of length l such that $l/b_i \le \alpha(1+\epsilon)$, routes a blocking flow along approximate shortest paths. In the end of a phase, we increase the value of α by $(1+\epsilon)$. In Lemma 3.2, we prove that T_p steps are sufficient to increase the value of $\min_i l_i/b_i$ by a factor of at least $(1+\epsilon)$, thus we maintain our α-invariant correctly. Given this, it is then clear from Lemma 3.1, that the above algorithm computes an $(1+O(\epsilon))$ approximate solution to the MBF problem.

LEMMA 3.2. *In any phase, after at most $T_p = O(L(\log m \log(k/\epsilon))/\epsilon^2)$ steps, the value of $\min_i l_i/b_i$ increases by at least $(1+\epsilon)$.*

Proof. Imagine that a phase is run till the value of $\min_i l_i/b_i$ increases by a factor of $(1+\epsilon)$. We prove that it takes at most T_p steps. Consider a commodity. Each blocking flow of this commodity saturates at least one edge on any $(1+\epsilon)$ approximate shortest path. Every time an edge saturates, the flow of that commodity on it increases by a factor of $(1+\epsilon^2/\log m)$. Since the flow of any commodity on any edge e goes from $\epsilon c_e/k$ to at most c_e, an edge can be saturated at most $O((\log m \log(k/\epsilon))/\epsilon^2)$ times. Since there are at most L edges on any paths, after $O(L(\log m \log(k/\epsilon))/\epsilon^2)$ steps, the length of the shortest path for this commodity increases by a factor of at least $(1+\epsilon)$. Thus the proof is complete.

The following lemma is now evident.

LEMMA 3.3. *The algorithm has at most $O(L(\log^2 m \log(k/\epsilon))/\epsilon^4)$ steps where each step can be implemented in T_{bf} parallel running time (with k processors) where $T_{bf} = \tilde{O}(m^2)$ denotes the time needed for a single commodity blocking flow computation along $(1+\epsilon)$ approximate shortest paths.*

4 Conclusions and Open Questions

One obvious open question is eliminating the dependency on L and get a poly-logarithmic convergence time for any value of L. Another open question is extending other results in [GK98], e.g., min-cost flows to efficient distributed solutions. Also, the sequential time bounds that we provide in this paper are surely not very tight; a better bound on the cost blocking flow should lead to reducing the computational overhead.

References

[AA94] B. Awerbuch and Y. Azar. Local optimization of global objectives: Competitive distributed deadlock resolution and resource allocation. In *Proc. 35th IEEE Symp. on Found. of Comp. Science*, pages 240–249, 1994.

[ANR92] Y. Azar, J. Naor, and R. Rom. The competitiveness of on-line assignment. In *Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms*, pages 203–210, 1992.

[BBR97] Y. Bartal, J. W. Byers, and D. Raz. Global optimization using local information with applications to flow control. In *Proc. 38th IEEE Symp. on Found. of Comp. Science*, pages 303–312, 1997.

[EDM05] E. Even-Dar and Y. Mansour. Fast convergence of selfish rerouting. In *Proc. 16th ACM-SIAM Symp. on Discrete Algorithms*, pages 772–781, 2005.

[Fle00] L. K. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics*, 13:505–520, 2000.

[FRV06] S. Fischer, H. Räcke, and B. Vöcking. Fast convergence to Wardrop equilibria by adaptive sampling methods. In *Proc. 38th ACM Symp. on Theory of Computing*, pages 653–662, 2006.

[GK98] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proc. 39th IEEE Symp. on Found. of Comp. Science*, pages 300–309, 1998.

[GY02] N. Garg and N. E. Young. On-line end-to-end congestion control. In *Proc. 43rd IEEE Symp. on Found. of Comp. Science*, pages 538–546, 2002.

[LN93] M. Luby and N. Nissan. A parallel approximation algorithm for positive linear programming. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 448–457, 1993.

[PST95] S. Plotkin, D. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Math of Oper. Research*, 20:257–301, 1995.

[You01] N. E. Young. Sequential and parallel algorithms for mixed packing and covering. In *Proc. 42nd IEEE Symp. on Found. of Comp. Science*, pages 538–546, 2001.