Adding Multiple Cost Constraints to Combinatorial Optimization Problems, with Applications to Multicommodity Flows

David Karger* Serge Plotkin†

Abstract

Minimum cost multicommodity flow is an instance of a simpler problem (multicommodity flow) to which a cost constraint has been added. In this paper we present a general scheme for solving a large class of such "cost-added" problems—even if more than one cost is added. One of the main applications of this method is a new deterministic algorithm for approximately solving the minimum-cost multicommodity flow problem.

Our algorithm finds a $(1+\epsilon)$ approximation to the minimum cost flow in $\tilde{O}(\epsilon^{-3}kmn)$ time, where k is the number of commodities, m is the number of edges, and n is the number vertices in the input problem. This improves the previous best deterministic bounds of $O(\epsilon^{-4}kmn^2)$ [9] and $\tilde{O}(\epsilon^{-2}k^2m^2)$ [15] by factors of n/ϵ and $\epsilon km/n$ respectively. In fact, it even dominates the best randomized bound of $\tilde{O}(\epsilon^{-2}km^2)$ [15].

The algorithm presented in this paper efficiently solves several other interesting generalizations of min-cost flow problems, such as one in which each commodity can have its own distinct shipping cost per edge, or one in which there is more than one cost measure on the flows and all costs must be kept small simultaneously. Our approach is based on an extension of the approximate packing techniques in [15] and a generalization of the round-robin approach of [16] to multicommodity flow without costs.

1 Introduction

1.1 The Problem

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 commodities flowing 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 communication networks, VLSI layout, scheduling, and transportation, and hence has been studied extensively [7, 10, 14, 17, 12, 13, 18, 2, 16].

Since multicommodity flow algorithms based on general interior-point methods for linear programming are slow [10, 19, 8], recent emphasis was on designing fast combinatorial algorithms that relied on problem structure. One successful approach has been to develop *approximation algorithms.* If there exists a flow of cost B that satisfies all the demands, the goal of a $(1+\epsilon)$-approximation algorithm is to find a flow of cost at most $(1+\epsilon)B$ that satisfies a $(1-\epsilon)$ fraction of each demand.

The addition of a cost function to the unweighted (no-cost) multicommodity flow problem has until now strongly impacted the performance of approximation

*MIT Laboratory for Computer Science, Cambridge, MA 02138. http://theory.lcs.mit.edu/~karger, karger@lcs.mit.edu. Research performed at AT&T Bell Laboratories.

†Department of Computer Science, Stanford University. http://theory.stanford.edu/people/plotkin/plotkin.html, plotkin@cs.stanford.edu. Research supported by NSF Grant CCR-9304971, and by Terman Fellowship.

algorithms. The minimum-cost multicommodity flow algorithm given by Plotkin, Shmoys, and Tardos [15], runs in $\tilde{O}(\epsilon^{-2}km^2)$ expected time [15]; their deterministic version of this algorithm is slower by a factor of k, running in $\tilde{O}(\epsilon^{-2}k^2m^2)$ time. The deterministic bound was improved for dense graphs by Kamath, Palmon, and Plotkin, [9] who gave an $\tilde{O}(\epsilon^{-4}kmn^2)$ algorithm, where n is the number of nodes. This is more than n times slower than Radzik's deterministic algorithm [16] for the *no cost* version of the problem. Even better running times were achieved for special cases of the no-cost problem [12]. It is interesting to note that adding costs does not significantly affect the running time of the interior-point based algorithms [8, 19].

The main contribution of this paper is a deterministic minimum-cost multicommodity flow algorithm that runs in $\tilde{O}(\epsilon^{-3}kmn)$ time, essentially matching the bound for the unweighted case. Ignoring the ϵ factors, this seems like a natural time bound since it matches the best known bound for computing flows for the k commodities separately.

1.2 Adding Constraints

The min-cost multicommodity flow problem consists of an easier problem (no-cost multicommodity flow) to which a single additional linear constraint (the cost function) has been added. Similarly, the no-cost multicommodity flow problem can be seen in the following way: we take a relatively easy to solve problem P ("find an independently feasible flow for each commodity satisfying the demands for that commodity") and add to it some constraints A that make it harder ("make sure the sum of the flows doesn't violate capacity constraints"). More precisely, this is a special case of the following *packing* problem: given a convex set P and a constraint matrix A, where $Ax \geq 0 \ \forall x \in P$, find $x \in P$ such that $Ax \leq 1$.

A general approach to approximately solving such problems was studied by Plotkin, Shmoys and Tardos [15] and Grigoriadis and Khachiyan [6]. They assumed that there was an *oracle* that, given a linear cost function over P, could find a point in P of minimum cost. They then assigned to each point in P a *potential* based on how much that solution violated the added constraints A. The problem of finding a solution satisfying $Ax \leq 1$ then reduces to the problem of finding a minimum-potential point in P. The potential function is highly non-linear and thus cannot be optimized directly by the oracle. However, the *gradient* of this function is linear; thus, the oracle can be used to determine a good direction to move the point so as to decrease its potential. For multicommodity flow the points in P are sums of flows, so the problem of minimizing a linear potential function is simply the problem of computing several single-commodity min-cost flows—a problem which can be approximately solved in $\tilde{O}(mn)$ time per commodity [5].

The running time of the algorithm in [15] depends on the *width* of the convex set P relative to A, defined as

$$\rho = \max_{x \in P} \max_i a_i x.$$

That is, the width measures the extent by which any constraint in A can be "overflowed" by a point in P. If P consists of k flows that individually obey the capacity constraints, then the sum of those flows can violate the capacity constraints by a factor of at most k, meaning that the width of P is only k. This is essentially the main reason that let [13] and then [3] solve the no-cost multicommodity flow problem in expected $\tilde{O}(\epsilon^{-2}kmn)$ time. Radzik [16] showed that randomization step in these algorithms can be removed, leading to a deterministic $\tilde{O}(\epsilon^{-2}kmn)$ algorithm.

The same approach does not seem to work directly in the min-cost case. We can try solving the problem by looking for the minimum "budget" B such that we can find a multicommodity flow of total cost less than B. This suggests adding a new constraint requiring "total cost less than B" to the constraint matrix A. But under an arbitrary cost function, a given flow can be arbitrarily more expensive than the minimum cost flow. In other words, the added constraint can *blow up the width* of the problem and therefore increase the running time significantly. An alternative scheme is to add the budget constraint to P: we can require that P be restricted such that each flow individually costs no more than the budget B. This reduces the width of the P to k, the number of commodities, but introduces a new problem. The optimization problem over P now becomes: find a flow of minimum cost under one cost metric without exceeding a given budget in some entirely different cost metric—a sort of "two cost" min-cost flow problem, for which no fast algorithm was previously known.

The solution proposed in [15] was to move more of the complexity of the problem into A, and use a sophisticated width-reduction technique that results in a polytope P whose width with respect to A is m and whose optimization oracle involves $\tilde{O}(k)$ shortest path computations. This led to an expected running time of $\tilde{O}(km^2)$.

The main contribution of this paper is development of a new technique especially geared towards solving "packing with budget" problems. Roughly speaking, the technique allows us to take an $\{Ax \leq 1, x \in P\}$ packing problem in which A has width ρ, add q additional packing (budget) constraints of *unbounded width* to the matrix A, and solve the resulting problem as if it had width $\rho + q/\epsilon$, even if the added constraints are actually much wider. The original approach would have treated the resulting problem as one of unbounded width and thus yielded a very slow algorithm.

For example, to find a minimum-cost multicommodity flow, we add the additional budget-constraint row to the matrix A, and then use our technique to get a randomized algorithm with an $\tilde{O}(\epsilon^{-3}kmn)$ expected running time. Replacing randomization by the round-robin technique of [16] allows us to achieve the same time bound deterministically, thus matching the natural bound of "$\tilde{O}(mn)$ per commodity". In other words, we show that approximately computing a k-commodity min-cost flow is not much harder than approximately computing k single-commodity no-cost flows.

Another interesting simple application of our technique is to the "two-cost" single commodity flow problem, where the goal is to find a flow that has approximately minimum cost with respect to one metric while its cost is smaller than some given budget with respect to another, unrelated metric. We give an $\tilde{O}(\epsilon^{-3}mn)$-time $(1+\epsilon)$-approximation algorithm for this problem. Using this algorithm as an oracle for the multicommodity flow polytope (with a per-flow budget constraint) gives us yet another $\tilde{O}(kmn)$-time approximation algorithm for the min-cost multicommodity flow problem for constant ϵ. We can also use this oracle to solve a generalization of multicommodity flow in which the cost of shipping each commodity can be different from the cost of shipping the others—in other words, where there are k different cost vectors, one for each commodity. As discussed in [1, Reference notes to Chapter 17], this generalization has many applications in practice, such as multivehicle tanker scheduling, racial balancing of schools, routing of multiple commodities, and warehousing of seasonal products. We approximately solve this problem in the same $\tilde{O}(kmn)$ time bound for constant ϵ.

Like all previous approximation algorithms for these types of problems, ours uses a potential function that is minimized at feasible points, together with a variant of the gradient-descent method to find that function's minimum. We develop a new approach that prevents the gradient descent from considering points that violate the added budget constraints by a large factor. This lets us pretend that our problem actually has small width. Our algorithm is based on a new, not-quite-exponential potential function whose gradients behave better than those of the purely exponential potential.

2 Fractional Packing With Budgets

2.1 Definitions and notation

The *fractional packing with budget problem (PWB)* is defined as follows:

$$\min(\lambda : Ax \leq \lambda, \beta x \leq \lambda, \text{ and } x \in P), \tag{1}$$

where A is an $(m-1) \times n$ matrix, β is a *budget* vector, and P is a convex set in $\mathbb{R}^n$ such that $Ax \geq 0$ and $\beta x \geq 0$ for each $x \in P$. Our techniques easily extend to the case where we have several additional budget rows β_i; for simplicity, we will concentrate on a single-budget case in this section.

Let A_β be the matrix constructed by concatenating β as an additional row to A. We shall use a_i to denote the ith row of A_β; thus $a_m = \beta$. We shall assume that we have a fast subroutine to solve the following optimization problem for the given convex set P:

Given an n-dimensional vector c, find $\tilde{x} \in P$ such that:

$$c\tilde{x} = \min(cx : x \in P), \tag{2}$$

Let λ^* denote the optimum solution to the PWB problem. For each $x \in P$, there is a corresponding minimum value λ such that $A_\beta x \leq \lambda$ (in each coordinate). We shall use the notation (x, λ) to denote that λ is the minimum value corresponding to x, and may also say that *x has width λ*. A solution (x, λ) is *ϵ-optimal* if $x \in P$ and $\lambda \leq (1+\epsilon)\lambda^*$. If (x, λ) is an ϵ-optimal solution with $\lambda > 1 + \epsilon$, then we can conclude that $\lambda^* > 1$.

To simplify the discussion, we will assume that $\lambda^* = 1$ and look for a solution (x, λ) with $\lambda \leq (1+\epsilon)$. We will also assume that we have a starting point x_0 with corresponding $\lambda_0 \leq 2$. The reduction to this situation is relatively straightforward and changes the running time of our algorithm by at most a polylogarithmic factor [15].

As in [15], the running time of our approximation algorithm will depend on the *width* of the polytope P with respect to the matrix A, defined as

$$\rho = \max_{i<m} \max_{x \in P} a_i x. \tag{3}$$

The key difference between this definition and the one in [15] is that our width ρ is independent of β. The algorithm of [15] uses the width of P with respect to A_β, which might be significantly larger than its width with respect to A. Thus our algorithm will be much faster than that of [15] when the width of P with respect to A is relatively small, while its width with respect to β is very large.

2.2 Algorithm

Denote $u_i(x) = a_i x$, $u(x) = (u_1(x), \ldots u_m(x))$, and let $f(z) = e^{\alpha z}/e^{\alpha}$, where $\alpha = (1/\epsilon)\ln 3m$. To guide the algorithm, we will use the following potential function:

$$\phi(x) = \sum_i f(u_i(x))$$

(note this sum includes a term for $a_m = \beta$).

Recall that we have assumed that there exists a solution to PWB problem with $\lambda = 1$. Thus, there exists x^* with corresponding u^* such that each $u^*(x) \leq 1$ and thus $\phi(x^*) \leq m$. The following simple lemma will be used as a stopping criterion.

Lemma 2.1 If $\phi(x) \leq 3m$, then the corresponding $\lambda \leq 1 + \epsilon$.

Our minimization algorithm starts with some point (x_0, λ_0) and proceeds in iterations, where the point considered "current" is updated at each iteration. As we will show below, each iteration will cause a significant decrease in the potential function ϕ. The algorithm terminates when the potential has become smaller than $3m$, as per Lemma 2.1.

To find the next point, given a current point $x \in P$, consider the linear approximation to ϕ given by the first order Taylor expansion:

$$l_x(\tilde{x}) = \phi(x) + (\nabla_x \phi) \cdot (\tilde{x} - x).$$

Since ϕ is smooth, we know that $\phi(\tilde{x}) \approx l_x(\tilde{x})$ "near" x. Therefore, if we found a point $\tilde{x}$ such that $l_x(\tilde{x}) \ll \phi(x)$, we might also expect that $\phi(\tilde{x}) \ll \phi(x)$, i.e. that we had found a point of much better potential. We can minimize $l_x(\tilde{x})$ over $x \in P$, since $l_x(\tilde{x})$ has the form $c\tilde{x} + d$ and we can use the oracle to minimize $c\tilde{x}$.

Unfortunately, the resulting point $\tilde{x}$ may be so "far" from x that the approximation of ϕ by l_x fails. However, if we only move a small step towards $\tilde{x}$, to the point $x + \sigma(\tilde{x} - x)$, then we know that for sufficiently small σ, we will stay in a neighborhood of x for which the Taylor approximation holds. Since l_x is linear, we know that moving in this direction will improve l_x. Unfortunately, since we only go a σ-fraction of the way towards $\tilde{x}$, we only earn a σ-fraction of the perceived improvement in l_x, and thus in the improvement to ϕ. Therefore, we would like σ to be as large is possible. It turns out that we can make this approach work if we take σ proportional to the inverse of the width ρ of the problem. In [15], this width is the width of P with respect to A_β, which could be unbounded in our case. Our improvement is to replace this parameter with the width of P with respect to A, which we assume to be relatively small.

The intuition behind our improvement is as follows. Note that since ϕ is convex, $l_x(\tilde{x})$ is always *less* than $\phi(\tilde{x})$. If we simply minimize $l_x(\tilde{x})$, we get a point such that $\beta\tilde{x}$ could be arbitrarily large. Suppose that we instead minimize $l_x(\tilde{x}) + m\beta\tilde{x}$. We know that there exists a feasible point x^* with $l_x(x^*) \leq \phi(x^*) \leq m$ and $\beta x^* \leq 1$, so whatever minimizing point $\tilde{x}$ we find satisfies $l_x(\tilde{x}) + m\beta\tilde{x} \leq 2m$. Since $\beta\tilde{x} \geq 0$, we know $l_x(\tilde{x}) \leq 2m$, meaning that we still find a point with a very small potential. At the same time, if $l_x(\tilde{x}) > 0$, then $m\beta\tilde{x} \leq 2m$, meaning that our point $\tilde{x}$ does not violate the constraint β by much. If we only encounter points x with $\beta x \leq 2$, we can pretend that we are in fact in a polytope with $\beta x \leq 2$ everywhere. This means in effect that β no longer induces unbounded width in the polytope.

Unfortunately, our assumption that $l_x \geq 0$ need not be true. However, we can put together a slightly more complicated function that serves the same purpose. Given the current point (x, λ), an iteration of our algorithm starts by finding $\tilde{x}$ that minimizes the following expression over P:

$$\psi_x(\tilde{x}) = \nabla_x \phi(x) \cdot \tilde{x} + (\nabla_x \phi(x) \cdot x)\beta\tilde{x}/(9\alpha) \tag{4}$$

(recall $\alpha = (1/\epsilon \ln 3m)$).

Now we modify x to be

$$\hat{x} \leftarrow (1 - \sigma)x + \sigma\tilde{x} \tag{5}$$

where $\sigma = 1/(20\alpha^2(\rho + \alpha))$

2.3 Analysis

To prove that each iteration results in some progress, we first show that the point $\tilde{x}$ minimizing ψ_x has several useful properties. First note the following: for any (x, λ),

$$
\begin{aligned}
(6) \qquad (\nabla_x \phi(x)) \cdot x &= \sum f'(u_i(x)) u_i(x) \\
&= \sum \alpha f(u_i(x)) u_i(x) \\
&\leq \alpha \phi(x) \lambda,
\end{aligned}
$$

and thus $\psi_x(\tilde{x}) \leq \nabla_x \phi(x) \cdot \tilde{x} + \lambda \phi(x) \beta \tilde{x}/9$ for any $\tilde{x}$. Also, $\nabla_x \phi(x) \cdot x \geq 0$. Now we prove that minimizing ψ_x gives us a point whose potential function seems very small according to the linear approximation l_x, and that at the same time does not violate the constraint βx by much.

Lemma 2.2 *Let $\tilde{x}$ be the minimizer of ψ_x. If the current x has a corresponding $\lambda \leq 3$, then*

$$
\begin{aligned}
(7) \qquad l_x(\tilde{x}) &\leq \frac{2}{3}\phi(x) \\
(8) \qquad \beta \tilde{x} &\leq 10\alpha
\end{aligned}
$$

Proof: Consider a feasible point x^* for the packing problem, and note that (by convexity) $\phi(x^*) \geq l_x(x^*) = \phi(x) + \nabla_x \phi(x) \cdot (x^* - x)$. Thus since $\phi(x^*) < \phi(x)$, we know that $\nabla_x \phi(x) \cdot x^* < \nabla_x \phi(x) \cdot x$.

By definition of $\tilde{x}$, $\psi_x(\tilde{x}) < \psi_x(x^*)$. Each of the two terms defining ψ_x is positive. Thus, each of the two terms in $\psi_x(\tilde{x})$ is no greater than

$$
\begin{aligned}
\psi_x(x^*) &= \nabla_x \phi(x) \cdot x^* + (\nabla_x \phi(x) \cdot x)/(9\alpha) \\
&\leq \nabla_x \phi(x) \cdot x^* + \phi(x)/3.
\end{aligned}
$$

In particular, $(\nabla_x \phi(x) \cdot x)\beta\tilde{x}/(9\alpha) \leq \nabla_x \phi(x) \cdot x^* + (\nabla_x \phi(x) \cdot x)/(9\alpha)$. Solving for $\beta\tilde{x}$ and observing that $\nabla_x \phi(x) \cdot x^* \leq \nabla_x \phi(x) \cdot x$ implies Equation (8).

Proving (7) takes some added work. We have observed that

$$
(9) \quad \nabla_x \phi(x) \cdot \tilde{x} \leq \psi_x(\tilde{x}) \leq \nabla_x \phi(x) \cdot x^* + \phi(x)/3.
$$

Now observe that

$$
\begin{aligned}
l_x(\tilde{x}) &= \phi(x) + \nabla_x \phi(x) \cdot (\tilde{x} - x) \\
&\leq \phi(x) + \nabla_x \phi(x) \cdot (x^* - x) + \phi(x)/3 \\
&= l_x(x^*) + \phi(x)/3
\end{aligned}
$$

But since ϕ is a convex function, $l_x(x^*) \leq \phi(x^*) \leq m \leq \phi(x)/3$. The result follows. ∎

It follows by induction that at all times, $\beta x \leq 10\alpha$. This lets us pretend in our analysis that the constraint βx has width at most 10α.

We do not claim that each iteration reduces λ. In fact, λ might increase as a result of a single iteration. It will be useful to observe that this increase is very small. This justifies the assumption of Lemma 2.2.

Lemma 2.3 *If we start with a point (x_0, λ_0) where $\lambda_0 \leq 2$ and $\phi(x)$ is non-increasing, then λ never exceeds 3.*

Now we show that as long as λ is large, each iteration of our algorithm reduces ϕ. Recall that at a current point x, one iteration determines a point $\tilde{x}$ by minimizing ψ_x over P, and then updates x to a new point $\hat{x} = x + \sigma(\tilde{x} - x)$.

Lemma 2.4 *If an iteration of the algorithm starts with a point x with $\phi(x) \geq 3m$, and $\sigma = 1/(20\alpha^2(\rho + \alpha))$, then $\phi(\hat{x}) = \phi(x)(1 - \Omega(\frac{1}{\alpha^2(\rho+\alpha)}))$.*

Proof: Linearity of l_x and Lemma 2.2 imply that $l_x(\hat{x}) = l_x(x) + \sigma(l_x(\tilde{x}) - l_x(x)) \leq (1 - \sigma/3)\phi(x)$, where we use the fact that $l_x(x) = \phi(x)$. Using the second-order Taylor approximation, we proceed to bound the error in $l_x(\hat{x})$. To simplify the expressions, recall $u_i(x) = a_i x$. According to the second-order Taylor Theorem we have that

$$
\phi(\hat{x}) = l_x(\hat{x}) + \sum \sigma^2 (u_i(\tilde{x}) - u_i(x))^2 f''(r_i),
$$

where the vector r is a convex combination of $u(x)$ and $u(x) + \sigma(u(\tilde{x}) - u(x))$. We now show that the second-order term is negligible for the given choice of σ. Recall that we have chosen $\sigma = 1/(20\alpha^2(\rho + \alpha))$. Since the width of P with respect to A is bounded by ρ, and because $\beta\tilde{x}$ is small by Lemma 2.2, we have $f''(r_i) < 2f''(u_i(x))$. Thus,

$$
\phi(\hat{x}) \leq \phi(x) - \sigma\phi(x)/3 + 2\sum_i \sigma^2 (u_i(\tilde{x}) - u_i(x))^2 f''(u_i).
$$

We now proceed to bound the second order term in the above expression. Inductively, using Lemma 2.3, we can assume that the λ corresponding to the current x is bounded by 3, i.e. $\forall i \leq m, u_i(x) \leq 3$. Recall that by definition of the problem, $u_i(x) \geq 0$ for all i and for all

$x \in P$. Let $\rho' = \rho + \alpha$. Then

$$\begin{aligned}
&\sum_i (u_i(\tilde{x}) - u_i(x))^2 f''(u_i(x)) \\
&\leq \sum_i u_i(\tilde{x})^2 f''(u_i(x)) + \sum u_i(x)^2 f''(u_i(x)) \\
&\leq \rho' \sum_i u_i(\tilde{x}) f''(u_i(x)) + 9 \sum_i f''(u_i(x)) \\
&\leq \rho' \sum_i (u_i(\tilde{x}) \alpha f'(u_i(x)) + 9 \sum_i \alpha^2 f(u_i(x)) \\
&\leq \rho' \alpha \nabla_x \phi(x) \cdot \tilde{x} + 9\alpha^2 \phi(x)
\end{aligned}$$

Using Lemma 2.2 to bound $\nabla_x \phi(x) \cdot \tilde{x} = l_x(\tilde{x}) - \phi(x) + \nabla_x \phi(x) \cdot x \leq \nabla_x \phi(x) \cdot x$, we see that the above expression is bounded by:

$$\begin{aligned}
\rho' \alpha \nabla_x \phi(x) \cdot x + 9\alpha^2 \phi(x) &\leq 3\rho' \alpha^2 \phi(x) + 9\alpha^2 \phi(x) \\
&= O(\rho' \alpha^2 \phi(x))
\end{aligned}$$

where the inequality is implied by (6). The chosen value of σ ensures that this second order term is at most a constant fraction of the first order term, which implies that the reduction in ϕ is at least $\Omega(\sigma\phi(x)) = \Omega(\frac{1}{\alpha^2(\rho+\alpha)}\phi(x))$. ∎

To simplify the running times, we will use a somewhat weaker claim than the above theorem, namely that the reduction in ϕ is $\Omega(1/(\alpha^3\rho))$.

If we start from $\lambda_0 \leq 2$, we need to reduce ϕ by a factor of at most $me^{2\alpha}/m$. Lemma 2.4 implies that this will take $O(\alpha^4\rho) = \tilde{O}(\epsilon^{-4}\rho)$ iterations. Observe that if $1+\epsilon \leq \lambda_0 \leq 1+2\epsilon$, then

$$m \leq \phi(x) \leq me^{(1+2\epsilon)\alpha}/e^{\alpha}$$

and hence reducing λ to below $1+\epsilon$ requires at most $O(\epsilon\alpha^4\rho)$ iterations. This suggests the following approach: solve for some sufficiently small constant ϵ, and then repeatedly set $\epsilon \leftarrow \epsilon/2$, and solve for new ϵ, until the desired precision is reached. This gives the following theorem:

Theorem 2.5 The PWB problem can be solved in $\tilde{O}(\epsilon^{-3}\rho)$ iterations, where each iteration involves a single call to the oracle given by (2).

3 Applications

3.1 Min-cost multicommodity flow

In this section we will sketch two approaches using our algorithm for the PWB problem to solve the min-cost multicommodity flow problem. Consider a min-cost multicommodity flow problem, where the polytope P corresponds to the *conservation constraints, demand constraints*, and *individual capacity constraints* of individual commodities, the matrix A defines the *joint capacity constraints*, and β is the cost function. In other words, $P = P^1 \times P^2 \cdots \times P^k$, where P^i corresponds to feasible flows of commodity i disregarding the rest of the commodities. Each edge has an associated cost β_e and the goal is to find a flow that satisfies capacity constraints up to a $(1+\epsilon)$-factor while costing less than $(1+\epsilon)B$, where B is the given budget. A bisection search on B can be used if, instead of being given a budget, we are told to find an approximately minimum cost solution.

Observe that this statement of the min-cost multicommodity flow problem directly corresponds to the PWB problem considered in the previous section. It is easy to check that the width ρ of P with respect to A is bounded by k, the number of commodities. However, the width of P with respect to A_β may be unbounded, and thus it is not possible to apply the packing algorithm of [15] to this representation of the problem.

The desired oracle (2) corresponds to approximately solving k independent single commodity minimum cost flow problems. This can be done by adapting the Goldberg-Tarjan cost-scaling min-cost flow algorithm [4], as in Leighton et al. [13]. (The adaptation is needed because the costs are exponential.) The resulting oracle runs in $\tilde{O}(kmn)$ time. Applying Theorem 2.5, we get an $\tilde{O}(\epsilon^{-3}k^2mn)$ running time for solving the problem.

To improve the running time we can use an approach that is similar to the one developed in [15]. The idea is to use the fact that the current point x can be represented as $(x^1, x^2, \ldots, x^k) \in P^1 \times P^2 \cdots \times P^k$, where $x^i \in P^i$. At each iteration, we randomly pick $1 \leq i \leq k$, optimize over P^i, and update x^i if it causes a decrease in ϕ. The expression that we minimize over P^i is exactly like (4), where instead of x and $\tilde{x}$, we have x^i and $\tilde{x}^i$, respectively. The value of σ is chosen to be $\sigma = \Theta(1/(\rho^i\alpha^3))$, where ρ^i is the width of P^i with respect to A, which is 1 in our case. Using the fact that $\nabla_x\phi(x) \cdot x = \sum_i \nabla_{x^i}\phi(x^i) \cdot x^i$, it is possible to modify the proof of Lemma 2.4 to show that the expected reduction in ϕ due to a single iteration is $\Omega(\phi(x)/(k\alpha^3))$, which is essentially the same as the reduction due to one iteration of the algorithm that updates all flows simultaneously. Roughly speaking, we loose a factor of k due to the fact that we update only a single flow, but gain a factor of k back due to the fact that we can use a larger σ. But now our oracle runs k times faster for the same improvement.

Using an analysis technique due to Karp [11], this leads to a conclusion that the algorithm will terminate in expected $\tilde{O}(\epsilon^{-3}k)$ iterations, where the running time of each iteration is dominated by the computation of a single commodity minimum-cost flow. Thus, we have the following:

Theorem 3.1 Min-cost multicommodity flow can be $(1+\epsilon)$-approximately solved in $\tilde{O}(\epsilon^{-3}kmn)$ expected time.

The basic idea involved in making the above algorithm deterministic is to use the approach of Radzik [16], who showed how to modify the no-cost multicommodity flow algorithms in [13, 3] by replacing the random sampling with the following *round robin* strategy: instead of picking a random polytope to improve at each iteration, simply go through the polytopes in order, using P^1 in the first iteration, P^2 in the second iteration, and so on, as long as λ does not decrease by a factor of $(1+\epsilon)$. At each iteration, proceed as in the randomized case to try to improve the given $x^i \in P^i$. Radzik proves that this approach "works" for the unweighted multicommodity flow problem, yielding the same running time as the randomized algorithm. We have shown that the same holds for a general product-of-polytopes problem, and in particular for the the min-cost multicommodity flow problem.

Theorem 3.2 Min-cost multicommodity flow can be $(1+\epsilon)$-approximately solved in $\tilde{O}(\epsilon^{-3}kmn)$ time.

3.2 Two-Cost Flows

An intriguing alternative approach is to move our width bounding technique from the algorithm to the oracle. Suppose that, given the budget B, we modify the polytope P by restricting P_i to be the set of feasible flows of commodity i *whose cost does not exceed the budget in isolation.* Just as for the constraints on capacities, we can now deduce that the sum of the flows will have cost at most k times the budget. Therefore, the budget constraint now has width k just like the capacity constraints, so we can apply the original algorithm of [15]. But now the oracle has changed: the oracle for P_i is given a linear potential function and must find a minimum potential feasible flow of commodity i *that does not have cost exceeding the budget.* Observe that the *cost function* determined by the edge costs is completely different from the *potential function* created by the algorithm, but that the oracle must obey limits on both cost and potential. In other words (by reduction to bisection search), it must solve a "two-cost" flow problem: given two cost functions c_1 and c_2, find a flow such that $c_1 f \leq B_1$ and $c_2 f \leq B_2$.

No efficient algorithm other than general linear programming was previously known for solving this problem. However, our PWB model can be extended to solve it approximately. In the PWB model, we considered adding a single unbounded-width constraint to our constraint matrix A. It is easy to generalize this model to add *two* (or any constant number of) constraints, yielding a problem of packing with two budgets (PW2B). Given constraints $\beta_i x \leq 1$, $i \leq t$, let $\beta = \frac{1}{t}\sum \beta_i$ and apply the single-added-constraint approach. Two-cost flow is an instance of PW2B whose polytope consists of feasible flows, whose constraint matrix is empty, and whose two unbounded-width constraints are the cost functions. The oracle needs to minimize a potential function over P—that is, find a min-cost flow. Thus, applying our width bounding techniques, we can solve the two-cost flow problem to within $(1+\epsilon)$ in $\tilde{O}(\epsilon^{-3}mn)$ time.

Using this approximation algorithm as the oracle in the original algorithm of [15] lets us restrict the width to $O(k)$ and thus solve the min-cost multicommodity flow problem using $\tilde{O}(k)$ calls to the oracle, for a total running time of $\tilde{O}(kmn)$ time for constant ϵ. In addition to being a useful oracle for the min-cost multicommodity flow problem, we consider the two-cost flow problem to be a natural problem in its own right, and therefore state:

Lemma 3.3 A $(1+\epsilon)$-approximation to the minimum two-cost flow problem can be computed in $\tilde{O}(\epsilon^{-3}mn)$ time.

3.3 Per-Commodity Costs

Using our two-cost flow algorithm as the oracle in the packing algorithm of [15]), we can solve the following generalized version of min-cost multicommodity flow. Rather than just taking the cost contribution on an edge to be proportional to the *total* flow on that edge, we can make it dependent on *which* commodities are flowing on that edge. In particular, we have a cost vector $c^{(i)}$ of edge costs for commodity i, and if we have flows $x^{(i)}$ for commodity i, then the total flow cost is $\sum c^{(i)}x^{(i)}$. Regular min-cost multicommodity flow is the case where all $c^{(i)}$ are equal. This generalized version of the problem has many practical applications to which the original version does not seem to apply [1, References in Chapter 17]. We have the following:

Theorem 3.4 Multicommodity flow with per-commodity costs can be approximated within a $(1+\epsilon)$-factor in $\tilde{O}(kmn)$ time for constant ϵ.

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows.* Prentice Hall, 1993.

[2] B. Awerbuch and T. Leighton. Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks. In *Proc. 26th Annual ACM Symposium on Theory of Computing*, pages 487–495, 1994.

[3] A. V. Goldberg. A natural randomization strategy for multicommodity flow and related algorithms. *Information Processing Let.*, 42:249–256, 1992.

[4] A. V. Goldberg and R. E. Tarjan. Solving minimum-cost flow problems by successive approximation. In *Proc. 19th Annual ACM Symposium on Theory of Computing*, pages 7–18, 1987.

[5] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by successive approximation. *Math. of Oper. Res.*, 15:430–466, 1990.

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

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

[8] A. Kamath and O. Palmon. Improved interior-point algorithms for exact and approximate solutions of multicommodity flow problems. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms*, 1995.

[9] A. Kamath, O. Palmon, and S. Plotkin. Fast approximation algorithm for min-cost multicommodity flow. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms*, 1995.

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

[11] R.M. Karp. Probabilistic recurrence relations. In *Proc. 23rd Annual ACM Symposium on Theory of Computing*, pages 190–197, 1991.

[12] P. Klein, S. Plotkin, C. Stein, and É. Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. *SIAM Journal on Computing*, June 1994.

[13] T. Leighton, F. Makedon, S. Plotkin, C. Stein, É. Tardos, and S. Tragoudas. Fast approximation algorithms for multicommodity flow problem. *J. Comp. and Syst. Sci.*, 1992.

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

[15] S. Plotkin, D. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Math of Oper. Research*, 1994. To appear.

[16] T. Radzik. Fast deterministic approximation for the multicommodity flow problem. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms*, 1995.

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

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

[19] P. M. Vaidya. Speeding up linear programming using fast matrix multiplication. In *Proc. 30th IEEE Annual Symposium on Foundations of Computer Science*, 1989.