SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NY 14853-7501

TECHNICAL REPORT NO. 999

February, 1992

Fast Approximation Algorithms for Fractional Packing and Covering Problems

By

Serge A. Plotkin[1], David B. Shmoys[2], Éva Tardos[3]

[1]Department of Computer Science, Stanford University, Stanford, CA. Research supported by NSF Research Initiation Award CCR-900-8226, by U.S. Army Research Office Grant DAAL-03-91-G-0102, by ONR Contract N00014-88-K-0166, and by a grant from Mitsubishi Corporation.

[2]School of Operations Research, Cornell University, Ithaca NY. Research partially supported by an NSF PYI award CCR-89-96272 with matching support from UPS, and Sun Microsystems, and by the National Science Foundation, the Air Force Office of Scientific Research, and the Office of Naval Research, through NSF grant DMS-8920550.

[3]School of Operations Research, Cornell University, Ithaca NY. Research supported in part by a Packard Fellowship, an NSF PYI award, a Sloan Fellowship, and by the National Science Foundation, the Air Force Office of Scientific Research, and the Office of Naval Research, through NSF grant DMS-8920550.

Abstract

This paper presents fast algorithms that find approximate solutions for a general class of problems, which we call fractional packing and covering problems. The only previously known algorithms for solving these problems are based on general linear programming techniques. The techniques developed in this paper greatly outperform the general methods in many applications, and are extensions of a method previously applied to find approximate solutions to multicommodity flow problems. Our algorithm is a Lagrangean relaxation technique; an important aspect of our results is that we obtain a theoretical analysis of the running time of a Lagrangean relaxation-based algorithm.

We give several applications of our algorithms. The new approach yields several orders of magnitude of improvement over the best previously known running times for algorithms for the scheduling of unrelated parallel machines in both the preemptive and the non-preemptive models, for the job shop problem, for the cutting-stock problem, for the network embedding problem, and for the minimum-cost multicommodity flow problem.

1 Introduction

We consider the following general type of problem: given a convex set $P \subseteq \mathbf{R}^n$ and a set of m inequalities $Ax \leq b$, decide if there exists a point $x \in P$ that satisfies $Ax \leq b$. We assume that we have a fast subroutine to minimize a non-negative cost function over P. A *fractional packing problem* is the special case of this problem when P is in the positive orthant and $A \geq 0$. The intuition for this name is that if P is a polytope, then each $x \in P$ can be written as a convex combination of vertices of P; we are attempting to fractionally pack (or combine) vertices of P subject to the "capacity" constraints $Ax \leq b$.

A wide variety of problems can be expressed as fractional packing problems. Consider the following formulation of deciding if the maximum flow from s to t in an m-edge graph G is at least f: let the polytope $P \subseteq \mathbf{R}^m$ be the convex combination of incidence vectors of (s,t)-paths, scaled so that each such vertex is itself a flow of value f; the edge capacity constraints are given by $Ax \leq b$, and so A is the $m \times m$ identity matrix; the required subroutine for P is a shortest-path algorithm. In words, we view the maximum flow problem as packing (s,t)-paths subject to capacity constraints. We can similarly formulate the multicommodity flow problem, by setting $P = P^1 \times \cdots \times P^k$ where P^ℓ is the polytope of all feasible flows of commodity ℓ, and $Ax \leq b$ describes the joint capacity constraints. In Section 6, we shall discuss the following further applications: the Held & Karp lower bound for the traveling salesman problem [11] (packing 1-trees subject to degree-2 and cost constraints); scheduling unrelated parallel machines in both the preemptive and non-preemptive models, as well as scheduling job shops (packing jobs subject to machine load constraints); embedding graphs with small dilation and congestion (packing short paths subject to capacity constraints); and the minimum-cost multicommodity flow problem (packing paths subject to capacity and budget constraints).

Fractional covering problems are another important case of this framework: in this case, P and A are as above, but the constraints are $Ax \geq b$, and we have a maximization routine for P. The cutting-stock problem is an example of a covering problem: paper is produced in wide rolls, called *raws*, and then subdivided into several different widths of narrower ones, called *finals*; there is a specified demand for the number of rolls of each final width, and the aim is to cover that demand using as few raws as possible. Gilmore & Gomory proposed a natural integer programming formulation of this problem and studied methods to solve it as well as its linear relaxation [5, 6]. This linear relaxation is also the key ingredient of the fully polynomial approximation scheme for the bin-packing problem that is due to Karmarkar & Karp [13]. We also consider problems with simultaneous packing and covering constraints.

In this paper we focus on obtaining approximate solutions for these problems. For an error parameter $\epsilon > 0$, a point $x \in P$ is an ϵ-approximate solution for the packing problem if $Ax \leq (1+\epsilon)b$. A point $x \in P$ is an ϵ-approximate solution for the covering problem if $Ax \geq (1-\epsilon)b$. The running time of each of our algorithms depends polynomially on ϵ^{-1}, and the *width* of the set P relative to $Ax \leq b$, as defined by $\rho = \max_i \max_{x \in P} a_i x / b_i$, where $a_i x \leq b_i$ is the ith row of $Ax \leq b$. Significantly, the running time does not depend explicitly on n, and hence it can be applied when n is exponentially large, assuming that there exists a polynomial subroutine to optimize cx over P, where $c = y^t A$ (and t denotes transpose).

In many applications, there will also exist a more efficient randomized variant of our algorithm. When analyzing one of these algorithms, we shall estimate only the expected running time. The primary reason for this is that since the running time of any randomized algorithm exceeds twice its expectation with probability at most $1/2$, if we make k independent trials of the algorithm, allowing twice the expectation for each trial, then the probability that none of these trials completes the computation is at most 2^{-k}. Thus, by increasing the running time by a logarithmic factor, we can always convert the statement about expectation to one that holds with high probability. In fact, for some of our applications, the stronger statement can be made with no degradation in the bound. Furthermore, the improved efficiency of these randomized algorithms can be obtained while maintaining the same worst-case running time as the deterministic version.

All of the problems in our framework are known to be solvable in polynomial time (without relaxing the right-hand-sides). Consider the problem of packing the vertices of a polytope P subject to the constraints $Ax \leq b$. We can apply the ellipsoid method to solve the dual problem, which has a constraint for every vertex of P, since the separation subroutine for the dual problem can be solved with the optimization subroutine for P. The problem can be solved more efficiently by the algorithm of Vaidya [25]; it obtains the optimal value in $O(mL)$ calls to an optimization subroutine for P plus $O(m\mathcal{M}(m)L)$ additional time; L and $\mathcal{M}(m)$ denote the binary size of the problem and the time needed to invert m by m matrices, respectively. Alternatively, one can apply the other linear programming algorithm of Vaidya [25] for problems where the polytope P can be described with few variables. Furthermore, if the problem has an appropriate network structure then the ideas of Kapoor and Vaidya [12] can be used to speed up the matrix inversions involved.

The algorithm in [25] obtains an optimal dual solution to a fractional packing or covering problem, but no primal solution. One can use ideas of Karmarkar & Karp [13] to obtain a primal solution as well. If Vaidya's algorithm uses T calls to the separation subroutine for the dual linear program to find an approximate dual solution, then Karmarkar & Karp obtain an approximate primal solution deterministically by approximately solving $O(T + m^2 \log(T/m))$ linear programs, or by using randomization solving only $O(m \log(T/m))$ linear programs, each with m variables and T inequalities. We can use Vaidya's algorithm [26] to solve these linear programs.

The parameter L in the bound of either linear programming algorithm of Vaidya depends on the quality of the starting solution; for example, in [26], it depends on how close the point is to the central path. In some applications, such as the bipartite matching problem, it is possible to find a starting solution with $L = \log(n\epsilon^{-1})$, which is, roughly speaking, the number of bits of accuracy required [9]. Unfortunately, we do not know of comparable results for any of our applications. For two of our applications, the bin-packing problem and the Held-Karp bound, such results would yield algorithms with running times that clearly improve on the algorithms based on the approach presented here.

Our algorithm outperforms Vaidya's algorithm if ϵ is large (*e.g.*, a constant), ρ is small relative to m, or the optimization subroutine for P is faster than matrix inversion. In many cases, ρ is not sufficiently small (even exponential), so in Section 5 we give techniques that often

Application	Deterministic Time	Speedup	Randomized Time	Speedup
Preemptive scheduling of N jobs on M machines	$O^*(MN^2)$	$\Omega^*(M^{2.5}N^{1.5})$	$O^*(MN)$	$\Omega^*(N^{2.5}M^{2.5})$
Nonpreemptive scheduling of N jobs on M machines	$O^*(M^2N)$	$\Omega^*(N)$	$O^*(MN)$	$\Omega^*(MN)$
Min-cost K-commodity flow in M-edge N-node graph	$O^*(K^2M^2)$	$\Omega^*(M^{.5}NK^{1.5})$	$O^*(KM^2)$	$\Omega^*(M^{.5}NK^{2.5})$
Cutting-stock with M widths of finals	$O^*(M^2)$	$\Omega^*(M^2\mathcal{M}(M))$	$O^*(M^2)$	$\Omega^*(M\mathcal{M}(M))$
Job shop scheduling of N μ-operation jobs on M machines	$O^*((N\mu M)^2+(N\mu)^3)$	$\Omega^*(N^{6.5}\mu^4)$	—	—
Embedding an N-node bounded degree graph into a bounded degree graph with flux α.	$O^*(N^3\frac{1}{\alpha})$	$\Omega^*(N^4\alpha)$	$O^*(N^2\frac{1}{\alpha})$	$\Omega^*(N^5\alpha)$

Figure 1: Summary of the performance of the described algorithms

reduce ρ. Figure 1 summarizes the comparison of our algorithm to Vaidya's for our applications, giving the speedup over his algorithms [25, 26] when we assume that $\epsilon > 0$ is any constant, and ignore polylogarithmic factors. A function $f(n)$ is said to be $\Omega^*(g(n))$ if there exists a constant c such that $f(n)\log^c n \geq \Omega(g(n))$; we define O^* analogously.

Our approach extends a method previously applied to find approximate solutions to multi-commodity flow problems, first by Shahrokhi & Matula [23], and later by Klein, Plotkin, Stein & Tardos [15] and Leighton, Makedon, Plotkin, Stein, Tardos & Tragoudas [18]. Recently, extensions of this method to other applications were found independently by Grigoriadis & Khachiyan [10].

An important theoretical aspect of our results is their connection to Lagrangean relaxation. The main idea of our algorithm is as follows. We maintain a point $x \in P$ that does not satisfy $Ax \leq b$, and repeatedly solve an optimization problem over P to find a direction in which the violation of the inequalities can be decreased. To do this, we define a "penalty" y on the rows of $Ax \leq b$. Rows with $a_i x$ large relative to b_i get a large penalty, other rows get smaller penalties. We relax the $Ax \leq b$ constraints, and instead solve the Lagrangean relaxed problem $\min(yA\tilde{x} : \tilde{x} \in P)$. The idea is that a large penalty tends to imply that the resulting point $\tilde{x}$ improves the corresponding inequality. We then set $x := (1-\sigma)x + \sigma\tilde{x}$, where σ is a suitably small number, and repeat. These ideas are often used to obtain empirically good algorithms for solving linear programs; however, unlike previous methods, we give a rule for adjusting the penalties for which a theoretical analysis proves a very favorable performance in many applications. Lagrangean relaxation has been recognized as an important tool for combinatorial optimization problems since the work of Held & Karp on the traveling salesman problem [11]; in our discussion of this application (in Section 6) we examine the relationship between our algorithm and the traditional approach.

As in [15], our algorithms can also be modified to generate integral approximate solutions and thus yield theorems relating the linear and fractional optima along the lines of Raghavan &

Thompson [22] and give alternative deterministic algorithms to obtain the results of Raghavan [21]. The modified algorithm is, in some cases, more efficient than the original algorithm, due to the fact that it terminates as soon as it can no longer improve the current solution while maintaining integrality. We will discuss this integer version of the packing algorithm at the end of Section 2, and use the algorithm for the job-shop and network embedding problems in Section 6. One could derive analogous integer approximation theorems for both the covering problem and for problems in the more general form considered in Section 4. However, due to the lack of applications we do not include the resulting theorems.

For simplicity of presentation, throughout the paper we shall use a model of computation that allows the use of exact arithmetic on real numbers and provides exponentiation as a single step. In [18], it has been shown that the special case of the algorithm for the multicommodity flow problem can be implemented in the RAM model without increasing the running time. Analogously, we can use approximate exponentiation and a version of the algorithm that relies only on a subroutine to find a nearly optimal solution over the polytope P, and hence avoid the need for exponentiation as a step. However, in order to convert the results to the RAM model, we need to perform further rounding; we must also limit the size of the numbers throughout the computation. It is easy to limit the numbers by a polynomial in the input length, similar to the size of the numbers used in exact linear programming algorithms. However, we do not know how to find an ϵ-approximate solution using polylogarithmic precision for the general case of the problems considered.

2 The Fractional Packing Problem

The *fractional packing problem* is defined as follows:

PACKING: $\exists ? x \in P$ such that $Ax \leq b$, where A is an $m \times n$ nonnegative matrix, $b > 0$, and P is a convex set in the positive orthant of $\mathbf{R}^n$.

We shall use a_i to denote the ith row of A and b_i to denote the ith coordinate of b. We shall assume that we have a fast subroutine to solve the following optimization problem for the given convex set P and matrix A:

Given an m-dimensional vector $y \geq 0$, find $\tilde{x} \in P$ such that

$$c\tilde{x} = \min(cx : x \in P), \text{ where } c = y^t A. \tag{1}$$

For a given error parameter $\epsilon > 0$, a vector $x \in P$ such that $Ax \leq (1+\epsilon)b$ is an *ϵ-approximate solution* to the PACKING problem. In contrast, a vector $x \in P$ such that $Ax \leq b$ is called an *exact solution*. An *ϵ-relaxed decision procedure* either finds an ϵ-approximate solution or concludes that no exact solution exists.

The running time of our relaxed decision procedure depends on the *width* of P relative to $Ax \leq b$, which is defined by

$$\rho = \max_i \max_{x \in P} \frac{a_i x}{b_i}. \tag{2}$$

In general, ρ might be large, even superpolynomial in the size of the problem. We shall discuss techniques to reduce the width in Section 5.

Relaxed Optimality. Consider the following optimization version of the PACKING problem:

$$\min(\lambda : Ax \leq \lambda b \text{ and } x \in P), \tag{3}$$

and let λ^* denote its optimal value. For each $x \in P$, there is a corresponding minimum value λ such that $Ax \leq \lambda b$. We shall use the notation (x, λ) to denote that λ is the minimum value corresponding to x. 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 no exact solution to the PACKING problem exists. On the other hand, if (x, λ) is an ϵ-optimal solution with $\lambda \leq 1+\epsilon$, then x is an ϵ-approximate solution to the PACKING problem.

Linear programming duality gives a characterization of the optimal solution for the optimization version. Let $y \geq 0$, $y \in \mathbf{R}^m$ denote a *dual solution*, and let $C_{\mathcal{P}}(y)$ denote the minimum cost cx for any $x \in P$ where $c = y^t A$. The following inequalities hold for any feasible solution (x, λ) and dual solution y:

$$\lambda y^t b \geq y^t A x \geq C_{\mathcal{P}}(y). \tag{4}$$

Observe that both $y^t b$ and $C_{\mathcal{P}}(y)$ are independent of x and λ. Hence, for any dual solution y, $\lambda^* \geq C_{\mathcal{P}}(y)/y^t b$. The goal of our algorithm is to find either an ϵ-approximate solution x, or an ϵ-optimal solution (x, λ) such that $\lambda > 1+\epsilon$. In the latter case we can conclude that no exact solution to the PACKING problem exists. The ϵ-optimality of a solution (x, λ) will be implied by a dual solution y such that $(1+\epsilon)C_{\mathcal{P}}(y)/y^t b \geq \lambda$. Since $\lambda > 1+\epsilon$, it follows that $C_{\mathcal{P}}(y)/y^t b > 1$; hence, $\lambda^* > 1$ and no exact solution to the PACKING problem exists. Linear programming duality implies that there exists a dual solution y^* for which $C_{\mathcal{P}}(y^*)/y^{*t} b = \lambda^*$. Hence, for optimal (x^*, λ^*) and y^*, all three terms in (4) are equal.

Consider an error parameter $\epsilon > 0$, a point $x \in P$ satisfying $Ax \leq \lambda b$, and a dual solution y. We define the following *relaxed optimality conditions*:

($\mathcal{P}1$) $(1-\epsilon)\lambda y^t b \leq y^t A x$;

($\mathcal{P}2$) $y^t A x - C_{\mathcal{P}}(y) \leq \epsilon(y^t A x + \lambda y^t b)$.

Lemma 2.1 If (x, λ) and y are feasible primal and dual solutions that satisfy the relaxed optimality conditions $\mathcal{P}1$ and $\mathcal{P}2$ and $\epsilon \leq 1/6$, then (x, λ) is 6ϵ-optimal.

Proof: From $\mathcal{P}1$ and $\mathcal{P}2$ we have that

$$C_{\mathcal{P}}(y) \geq (1-\epsilon)y^tAx - \epsilon\lambda y^tb \geq (1-\epsilon)^2\lambda y^tb - \epsilon\lambda y^tb \geq (1-3\epsilon)\lambda y^tb.$$

Hence, $\lambda \leq (1-3\epsilon)^{-1}C_{\mathcal{P}}(y)/(y^tb) \leq (1-3\epsilon)^{-1}\lambda^* \leq (1+6\epsilon)\lambda^*$. ∎

The Algorithm. The core of the algorithm is the procedure IMPROVE-PACKING, which takes as input a point $x \in P$ and an error parameter $\epsilon \geq 0$. Given x, it computes λ_0, the minimum λ such that $Ax \leq \lambda b$ is currently satisfied. IMPROVE-PACKING produces a new feasible solution (x, λ) such that x is 6ϵ-optimal or $\lambda \leq \lambda_0/2$. It uses a dual solution y defined as a function of x, where $y_i = \frac{1}{b_i}e^{\alpha a_i x/b_i}$; we call this choice of y the *dual solution corresponding to* x. We will choose α so that the relaxed optimality condition $\mathcal{P}1$ is satisfied for (x, λ) and its corresponding dual solution. Thus, if the current solutions (x, λ) and y satisfy $\mathcal{P}2$, then λ is sufficiently close to optimal, and IMPROVE-PACKING terminates. Otherwise, we find a point $\tilde{x} \in P$ that attains the minimum $C_{\mathcal{P}}(y)$, and modify x by setting $x \leftarrow (1-\sigma)x + \sigma\tilde{x}$. Although a single update of x might increase its corresponding λ, we will show that a sequence of such updates gradually reduces λ.

Lemma 2.2 If $\alpha \geq 2\lambda^{-1}\epsilon^{-1}\ln(2m\epsilon^{-1})$, then any feasible solution (x, λ) and its corresponding dual solution y satisfy $\mathcal{P}1$.

Proof: For this proof, it is useful to introduce a localized version of $\mathcal{P}1$:

($\hat{\mathcal{P}}1$) For each $i = 1, \ldots, m$, $(1-\epsilon/2)\lambda b_i \leq a_i x$ or $y_i b_i \leq \frac{\epsilon}{2m}y^tb$.

Let $I = \{i : (1-\epsilon/2)\lambda b_i \leq a_i x\}$. Condition $\hat{\mathcal{P}}1$ implies $\mathcal{P}1$, since

$$\lambda y^tb = \lambda\sum_{i\in I} y_ib_i + \lambda\sum_{i\notin I} y_ib_i \leq \frac{1}{1-\epsilon/2}\sum_{i\in I} y_ia_ix + \lambda\sum_{i\notin I}\frac{\epsilon}{2m}y^tb \leq \frac{1}{1-\epsilon/2}y^tAx + \frac{\epsilon}{2}\lambda y^tb,$$

and therefore $(1-\epsilon)\lambda y^tb \leq (1-\epsilon/2)^2\lambda y^tb \leq y^tAx$.

Next we have to show that the hypothesis of the lemma implies that $\hat{\mathcal{P}}1$ is satisfied. Notice that $y^tb = \sum_i e^{\alpha a_i x/b_i}$. By the minimality property of λ we have that $y^tb \geq e^{\alpha\lambda}$. Consider any row i for which $a_ix < (1-\epsilon/2)\lambda b_i$. This implies that $y_ib_i < e^{(1-\epsilon/2)\alpha\lambda}$, and so $y_ib_i/(y^tb) < e^{-\epsilon\alpha\lambda/2} \leq \epsilon/(2m)$. Hence, $\hat{\mathcal{P}}1$ is satisfied. ∎

At the beginning of IMPROVE-PACKING (see Figure 2), α is set to $4\lambda_0^{-1}\epsilon^{-1}\ln(2m\epsilon^{-1})$; hence, the relaxed optimality condition $\mathcal{P}1$ is satisfied throughout the execution of the procedure. The following lemma shows that moving the right amount towards a minimum-cost point $\tilde{x}$ results in a significant decrease in the potential function $\Phi = y^tb = \sum_i e^{\alpha a_i x/b_i}$.

Lemma 2.3 Consider a point $x \in P$ and an error parameter $0 < \epsilon \leq 1$ such that x and its corresponding dual solution y have potential function value Φ and do not satisfy $\mathcal{P}2$. Let $\tilde{x} \in P$

IMPROVE-PACKING(x, ϵ)
$\lambda_0 \leftarrow \max_i a_i x/b_i$; $\alpha \leftarrow 4\lambda_0^{-1}\epsilon^{-1}\ln(2m\epsilon^{-1})$; $\sigma \leftarrow \frac{\epsilon}{4\alpha\rho}$.
While $\max_i a_i x/b_i \geq \lambda_0/2$ **and** x and y do not satisfy $\mathcal{P}2$
    For each $i = 1, \ldots, m$: set $y_i \leftarrow \frac{1}{b_i}e^{\alpha a_i x/b_i}$.
    Find a min-cost point $\tilde{x} \in P$ for costs $c = y^t A$.
    Update $x \leftarrow (1-\sigma)x + \sigma\tilde{x}$.
Return x.

Figure 2: Procedure IMPROVE-PACKING.

attain the minimum $C_{\mathcal{P}}(y)$. Assume that $\sigma \leq \frac{\epsilon}{4\alpha\rho}$. Define a new solution by $\hat{x} = (1-\sigma)x + \sigma\tilde{x}$, and let $\hat{\Phi}$ denote the potential function value for $\hat{x}$ and its corresponding dual solution $\hat{y}$. Then $\Phi - \hat{\Phi} \geq \alpha\sigma\epsilon\lambda\Phi$.

Proof: By the definition of ρ, $Ax \leq \rho b$ and $A\tilde{x} \leq \rho b$. This implies that $\alpha\sigma|a_i x - a_i\tilde{x}|/b_i \leq \epsilon/4 \leq 1/4$. Using the second-order Taylor theorem, we see that if $|\delta| \leq \epsilon/4 \leq 1/4$ then, for all x, $e^{x+\delta} \leq e^x + \delta e^x + \frac{\epsilon}{2}|\delta|e^x$. Setting $\delta = \alpha\sigma(a_i\tilde{x} - a_i x)/b_i$, we see that

$$\begin{aligned}\hat{y}_i &\leq y_i + \frac{1}{b_i}\frac{\alpha\sigma(a_i\tilde{x} - a_i x)}{b_i}e^{\alpha a_i x/b_i} + \frac{1}{b_i}\frac{\epsilon\alpha\sigma|a_i\tilde{x} - a_i x|}{2b_i}e^{\alpha a_i x/b_i} \\ &\leq y_i + \alpha\sigma\frac{1}{b_i}(a_i\tilde{x} - a_i x)y_i + \epsilon\alpha\sigma\frac{1}{2b_i}(a_i\tilde{x} + a_i x)y_i.\end{aligned}$$

Using this inequality to bound the change in the potential function, we get

$$\begin{aligned}\Phi - \hat{\Phi} &= \sum_i (y_i - \hat{y}_i)b_i \geq \alpha\sigma\sum_i (a_i x - a_i\tilde{x})y_i - \alpha\sigma\frac{\epsilon}{2}\sum_i (a_i x + a_i\tilde{x})y_i \\ &= \alpha\sigma(y^t Ax - y^t A\tilde{x}) - \alpha\sigma\frac{\epsilon}{2}(y^t Ax + y^t A\tilde{x}) \geq \alpha\sigma(y^t Ax - C_{\mathcal{P}}(y)) - \alpha\sigma\epsilon y^t Ax.\end{aligned}$$

The fact that $\mathcal{P}2$ is not satisfied implies that the decrease in Φ is at least $\alpha\sigma\epsilon\lambda\Phi$. ∎

During IMPROVE-PACKING, σ is set equal to $\frac{\epsilon}{4\alpha\rho}$, which implies that the decrease in the potential function due to a single iteration is $\Omega(\frac{\epsilon^2\lambda}{\rho}\Phi)$. Observe that throughout the execution of IMPROVE-PACKING we have $e^{\alpha\lambda_0/2} \leq \Phi \leq me^{\alpha\lambda_0}$. If the input solution is $O(\epsilon)$-optimal, then we have the tighter bound, $e^{\alpha(1+O(\epsilon))^{-1}\lambda_0} \leq \Phi \leq me^{\alpha\lambda_0}$. Together with the previous lemma, this can be used to bound the number of iterations in a single call to IMPROVE-PACKING.

Theorem 2.4 The procedure IMPROVE-PACKING terminates after $O(\epsilon^{-3}\lambda_0^{-1}\rho\log(m\epsilon^{-1}))$ iterations. If λ_0 is $O(\epsilon)$-optimal, then IMPROVE-PACKING terminates after $O(\epsilon^{-2}\lambda_0^{-1}\rho\log(m\epsilon^{-1}))$ iterations.

We shall use the procedure IMPROVE-PACKING repeatedly to find an ϵ_0-approximate solution for any given $\epsilon_0 > 0$. We first find a 1-approximate solution, thereby solving the problem for

$\epsilon_0 \geq 1$, and then show how to use this solution to obtain an ϵ_0-approximate solution for any smaller value of ϵ_0. Set $\epsilon = 1/6$, and call IMPROVE-PACKING with an arbitrarily chosen solution $x \in P$; repeatedly call this subroutine with the output of the previous call until the resulting solution (x, λ) is 6ϵ-optimal or $\lambda \leq 1 + 6\epsilon$. If, at termination, $\lambda \leq 1 + 6\epsilon = 2$, then x is a 1-approximate solution. Otherwise, $\lambda > 2$ and x is 1-optimal, and hence no exact solution exists. The first part of Theorem 2.4 implies that the number of iterations during such a call to IMPROVE-PACKING with input (x, λ) is $O(\lambda^{-1}\rho \log m)$. Since this bound is proportional to λ^{-1} and it at least doubles with every call, the number of iterations during the last call dominates the total in all of the calls, and hence $O(\rho \log m)$ iterations suffice overall.

If $\epsilon_0 < 1$, then we continue with the following ϵ-scaling technique. The rest of the computation is divided into scaling phases. In each phase, we set $\epsilon \leftarrow \epsilon/2$, and call IMPROVE-PACKING once, using the previous output as the input. Before continuing to the next phase, the algorithm checks if the current output (x, λ) satisfies certain termination conditions. If x is an ϵ_0-approximate solution, then the algorithm outputs x and stops; otherwise, if $\lambda > 1 + 6\epsilon$, the algorithm claims that no exact solution exists, and stops. First observe that if the algorithm starts a new phase, the previous output (x, λ) has $\lambda \leq 1 + 6\epsilon \leq 2$, and this is the new input. As a result, for each ϵ-scaling phase, the output is an exact solution or is 6ϵ-optimal. Hence, if the output of a phase (x, λ) has $\lambda > 1$, then x is 6ϵ-optimal; if $\lambda > 1 + 6\epsilon$, then the algorithm has proven that no exact solution exists. Furthermore, if no such proof is found by the point when $\epsilon \leq \epsilon_0/6$, the output (x, λ) has $\lambda \leq 1 + 6\epsilon \leq 1 + \epsilon_0$; x is an ϵ_0-approximate solution. Finally, note that for each phase, the input is a 12ϵ-optimal solution with respect to the new value of ϵ. The second part of Theorem 2.4 implies that the number of iterations needed to convert this solution into a 6ϵ-optimal one is bounded by $O(\epsilon^{-2}\rho \log(m\epsilon^{-1}))$. Since the number of iterations during each scaling phase is proportional to the current value of ϵ^{-2} and this value doubles each phase, the bound for the last scaling phase dominates the total for all scaling phases.

An iteration of IMPROVE-PACKING consists of computing the dual vector y and finding the point $\tilde{x} \in P$ that minimizes the cost cx, where $c = y^t A$. Assuming that exponentiation is a single step, the time required to compute y is $O(m)$ plus the time needed to compute (or maintain) Ax for the current point x.

Theorem 2.5 For $0 < \epsilon \leq 1$, repeated calls to IMPROVE-PACKING can be used so that the algorithm either finds an ϵ-approximate solution for the fractional packing problem or else proves that no exact solution exists; the algorithm uses $O(\epsilon^{-2}\rho \log(m\epsilon^{-1}))$ calls to the subroutine (1) for P and A, plus the time to compute Ax for the current iterate x between consecutive calls.

Notice that the running time does not depend explicitly on n, the dimension of P. This makes it possible to apply the algorithm to problems defined with an exponential number of variables, assuming we have a polynomial-time subroutine to compute a point $x \in P$ of cost $C_{\mathcal{P}}(y)$ given any positive y, and that we can compute Ax for the current iterate x in polynomial time.

Randomized Version. In some cases, the bound in Theorem 2.5 can be improved using randomization. This approach was introduced by Klein, Plotkin, Stein, & Tardos [15] in the

context of multicommodity flow; we shall present other applications in Section 6.

Let us assume that the polytope P can be written as a product of polytopes of smaller dimension, *i.e.*, $P = P^1 \times \cdots \times P^k$, where the coordinates of each vector x can be partitioned into $(x^1, \ldots, x^k)$ and $x \in P$ if and only if $x^\ell \in P^\ell$, $\ell = 1, \ldots k$. The inequalities $Ax \leq b$ can then be written as $\sum A^\ell x^\ell \leq b$, and we shall let a_i^ℓ denote the ith row of A^ℓ, $i = 1, \ldots, m$, $\ell = 1, \ldots, k$. Let ρ^ℓ denote the width of P^ℓ relative to $A^\ell x^\ell \leq b$, $\ell = 1, \ldots, k$. Clearly, $\rho = \sum_\ell \rho^\ell$. A subroutine to compute $C_{\mathcal{P}}(y)$ for P consists of k subroutines, where the ℓth subroutine minimizes cx^ℓ subject to $x^\ell \in P^\ell$ for costs of the form $c = y^t A^\ell$. Randomization speeds up the algorithm by roughly a factor of k if $\rho^\ell = \hat{\rho}$ for each ℓ, or the k subroutines have the same time bound.

This assumption is satisfied, for example, in the multicommodity flow problem considered in [18]. One way to define the multicommodity flow problem as a packing problem is to let P^ℓ be the polytope of all feasible flows satisfying the ℓth demand and the capacity constraints $x^\ell \leq u$, and let the matrix $Ax \leq b$ describe the joint capacity constraints $\sum_\ell x^\ell \leq u$. For this problem we get that $\rho^\ell = 1$ for every ℓ. We shall present other applications in Section 6.

The idea of the more efficient algorithm is as follows. To find a minimum-cost point $\tilde{x}$ in P, IMPROVE-PACKING calculates k minimum-cost points $\tilde{x}^\ell \in P^\ell$, $\ell = 1, \ldots, k$. Instead, we will choose ℓ at random with a probability that depends on ρ^ℓ (as described below), compute a single minimum-cost point $\tilde{x}^\ell$, and consider perturbing the current solution using this $\tilde{x}^\ell$. The perturbation is done only if it leads to a decrease in Φ. In order to check if $\mathcal{P}2$ is satisfied by the current solution, it would be necessary to compute $C_{\mathcal{P}}(y)$. This is no longer done each iteration; instead, this condition is checked with probability $1/k$. This particular method of randomizing is an extension of an idea that Goldberg [7] has used for the multicommodity flow problem, and was also independently discovered by Grigoriadis & Khachiyan [10].

The key to the randomized version of our algorithm is the following lemma. The proof of this lemma is analogous to the proof of Lemma 2.3.

Lemma 2.6 Consider a point $(x^1, \ldots, x^k) \in P^1 \times \cdots \times P^k$, with corresponding dual solution y and potential function value Φ, and an error parameter ϵ, $0 < \epsilon \leq 1$. Let $\tilde{x}^s$ be a point in P^s that minimizes the cost $c^s x^s$, where $c^s = y^t A^s$, $s = 1, \ldots, k$, and assume that $\sigma^s \leq \min\{\epsilon/(4\rho^s\alpha), 1\}$. Define a new point by changing only x^s, where $x^s \leftarrow (1 - \sigma^s)x^s + \sigma^s \tilde{x}^s$. If $\hat{\Phi}$ denotes the potential function value of the new solution, then $\Phi - \hat{\Phi} \geq \alpha\sigma^s((1 - \epsilon)y^t A^s x^s - y^t A^s \tilde{x}^s)$.

In this lemma, we have restricted $\sigma^s \leq 1$ to ensure that the new point is in P; to get the maximum improvement, the algorithm uses $\sigma^s = \min\{1, \epsilon/(4\alpha\rho^s)\}$. Since the algorithm changes x only when the update would decrease the potential function, the decrease in the potential function associated with updating x_s is $\alpha\sigma^s\Delta_s$ where $\Delta_s = \max\{((1 - \epsilon)y^t A^s x^s - y^t A^s \tilde{x}^s), 0\}$. Let $S = \{s : 4\alpha\rho^s < \epsilon\}$ and define $\rho' = \sum_{s \notin S} \rho^s$. The probability $\beta(s)$ with which we pick an index s is defined as follows:

$$\beta(s) = \begin{cases} \frac{\rho^s}{2\rho'} & \text{for } s \notin S \\ \frac{1}{2|S|} & \text{for } s \in S \end{cases}$$

Using Lemma 2.6, we get the following theorem:

Theorem 2.7 For $0 < \epsilon \leq 1$, repeated calls to the randomized version of IMPROVE-PACKING can be used so that the algorithm either finds an ϵ-approximate solution for the fractional packing problem defined by a polytope $P = P^1 \times \cdots \times P^k$ and inequalities $\sum_\ell A^\ell x^\ell \leq b$, or else proves that no exact solution exists; the algorithm is expected to take $O(\epsilon^{-2}\rho \log(m\epsilon^{-1}) + k\log(\rho\epsilon^{-1}))$ iterations, each of which makes one call to the subroutine (1) for P^ℓ and A^ℓ, for a single value of ℓ, $\ell \in \{1,\ldots,k\}$, plus the time to compute $\sum_\ell A^\ell x^\ell$ for the current iterate $(x^1,\ldots,x^k)$ between consecutive calls.

Proof: We first analyze a single call to the randomized variant of IMPROVE-PACKING, and show that it is expected to terminate within $O(\epsilon^{-3}\rho\lambda_0^{-1}\log(m\epsilon^{-1}) + \epsilon^{-1}k)$ iterations. There are two types of iterations: those where $\mathcal{P}2$ is satisfied, and those where it isn't. We bound these separately. In the former case, since the algorithm checks, with probability $1/k$, whether $\mathcal{P}2$ is satisfied, and if so, the algorithm terminates, then we expect that $O(k)$ of these iterations will suffice to detect that $\mathcal{P}2$ is satisfied, and terminate. In the latter case, we will show the expected decrease of the potential function Φ during one iteration is at least $\min\{\epsilon^2\lambda/(8\rho), \ln(2m\epsilon^{-1})/k\}\Phi$. Since $\mathcal{P}2$ is not satisfied, we have that $\sum_s \Delta_s \geq \epsilon\lambda\Phi$, where $\alpha\sigma^s\Delta_s$ is the decrease in Φ associated with updating x_s. Using this fact and applying Lemma 2.6, we see that the expected decrease in Φ is

$$\begin{aligned} \sum_s \alpha\sigma^s\Delta_s\beta(s) &= \sum_{s\notin S} \alpha\frac{\epsilon}{4\rho^s\alpha}\cdot\frac{\rho^s}{2\rho'}\Delta_s + \sum_{s\in S}\frac{\alpha}{2|S|}\Delta_s \\ &\geq \min\left\{\frac{\epsilon}{8\rho}, \frac{\alpha}{2k}\right\}\sum_{s=1}^{k}\Delta_s \geq \min\left\{\frac{\epsilon}{8\rho}, \frac{\alpha}{2k}\right\}\epsilon\lambda\Phi. \end{aligned}$$

Since $\alpha \geq 2\lambda^{-1}\epsilon^{-1}\ln(2m\epsilon^{-1})$, the claimed bound on the expected decrease of Φ follows.

We use a result due to Karp [14] to analyze the number of iterations used by the randomized version of IMPROVE-PACKING. Let δ_Φ denote the ratio of upper and lower bounds on the potential function Φ during a single execution of IMPROVE-PACKING. Each iteration of the algorithm when $\mathcal{P}2$ is not satisfied is expected to decrease the potential function to $p\Phi$, where $p = 1 - \min\{\epsilon^2\lambda/(8\rho), \ln(2m\epsilon^{-1})/k\}$; let $b = 1/p$. The potential function never increases. Let the random variable T denote the number of iterations of the algorithm when $\mathcal{P}2$ is not satisfied. Karp proved a general result which implies that

$$Prob(T \geq \log_b \delta_\Phi + \omega + 1) \leq p^{\omega-1}p^{\lfloor\log_b\delta_\Phi\rfloor+1}\delta_\Phi. \tag{5}$$

To bound the expected number of iterations, we estimate $\sum_j Prob(T \geq j)$; since $p < 1$, (5) implies that this expectation is $O(\log_b \delta_\Phi)$. Note that $\delta_\Phi \leq me^{\alpha\lambda_0/2}$, and in the case when the

input is $O(\epsilon)$-optimal, $\delta_\Phi \leq me^{\epsilon\alpha\lambda_0}$. Hence we see that the randomized version of IMPROVE-PACKING is expected to terminate in $O(\epsilon^{-2}\rho\alpha+\epsilon^{-1}k) = O(\epsilon^{-3}\rho\lambda_0^{-1}\log(m\epsilon^{-1})+\epsilon^{-1}k)$ iterations, and is an ϵ^{-1} factor faster if the input is $O(\epsilon)$-optimal.

We use this randomized version of IMPROVE-PACKING repeatedly to obtain an ϵ_0-approximate solution in exactly the same way as in the deterministic case. First we set $\epsilon = 1/6$, and then we use ϵ-scaling. The expected number of iterations in total, is the sum of the expectations over all calls to IMPROVE-PACKING. The expected number of iterations in each call to IMPROVE-PACKING with $\epsilon = 1/6$ is $O(\rho\lambda_0^{-1}\log m + k)$, and each call during ϵ-scaling is expected to have $O(\epsilon^{-2}\rho\log(m\epsilon^{-1}) + k)$ iterations. For both of these bounds, the first term is identical to the bound for the deterministic case. There are at most $\log\rho$ calls to IMPROVE-PACKING with $\epsilon = 1/6$, and $\log\epsilon_0^{-1}$ calls during ϵ-scaling, and hence there are $O(\epsilon_0^{-2}\rho\log(m\epsilon_0^{-1}) + \log(\rho\epsilon_0^{-1})k)$ iterations expected in total.

To complete the proof, we must also observe that the routine to check $\mathcal{P}2$ is expected to be called in only an $O(1/k)$ fraction of the iterations. This implies the theorem. ∎

In fact, it is straightforward to bound the expected number of calls to optimize over each P^ℓ, $\ell = 1,\ldots,k$, by $O(\epsilon^{-2}\rho^\ell\log(m\epsilon^{-1}) + \log(\rho\epsilon^{-1}))$. Let T^ℓ denote the time required for the minimization over P^ℓ. Assume that the minimization over P, used by the deterministic algorithm, takes time $T = \sum_\ell T^\ell$. Notice that if we have, in addition, that $T^\ell = T/k$ for each $\ell = 1,\ldots,k$, or $\rho^\ell = \hat{\rho}$ for each $\ell = 1,\ldots,k$, then the time required for running the subroutines in the randomized version is expected to be a factor of k less than was required in the deterministic version.

If $T = \sum_\ell T^\ell$, then we can combine the deterministic and the randomized algorithms in a natural way. By running one iteration of the deterministic algorithm after every k iterations of the randomized one, we obtain an algorithm that simultaneously has the expected performance of Theorem 2.7 and the worst-case performance of Theorem 2.5, except that it will need to compute Ax (for the current solution x) k times more often than is required by Theorem 2.5. Finally, in the introduction, we mentioned that results about expectation could be converted into results that hold with high probability by repeatedly running the algorithm for twice as long as its expected running time bound. In fact, the structure of our algorithm makes this "restarting" unnecessary, since the final solution obtained by IMPROVE-PACKING is at least as good as the initial solution. Thus, all of our results can be extended to yield running time bounds that hold with high probability, without changing the algorithm.

Relaxed Versions. It is not hard to show that our relaxed decision procedure for the packing problem could also use a subroutine that finds a point in P of cost not much more than the minimum, and this gives a bound on the number of iterations of the same order of magnitude as the original version.

Theorem 2.8 If the optimization subroutine (1) used in each iteration of IMPROVE-PACKING is replaced by an algorithm that finds a point $\tilde{x} \in P$ such that $y^tA\tilde{x} \leq (1+\epsilon/2)C_{\mathcal{P}}(y) + (\epsilon/2)\lambda y^tb$ for any given $y \geq 0$, then the resulting procedure yields a relaxed decision procedure for the packing

problem; furthermore, in either the deterministic or the randomized implementations, the number of iterations can be bounded exactly as in Theorems 2.5 and 2.7, respectively.

Proof: It is easy to prove that the analog of Lemma 2.3 remains valid, using $\sigma \leq \epsilon/(8\alpha\rho)$. The only change in the proof is to use the second-order Taylor theorem with $|\delta| \leq \epsilon/8 \leq 1/8$, in order to bound the second-order error term for $e^{x+\delta}$ by $\epsilon|\delta|e^x/4$; this yields that the improvement in the potential function is at least $(\epsilon/2)\alpha\sigma\lambda\Phi$. Lemma 2.6 can be modified similarly. Since this improvement is of the same order, the rest of the proof follows directly from these lemmas. ∎

In some applications, there is no efficient optimization subroutine known for the particular polytope P, as in the case when this problem is NP-hard. However, Theorem 2.8 shows that it suffices to have a fully polynomial approximation scheme for the problem.

Another use of this approximation is to convert our results to the RAM model of computation. In this paper we have focused on a model that allows exact arithmetic and assumes that exponentiation can be performed in a single step. As was done in [18] for the multicommodity flow problem, we can use approximate exponentiation to compute an approximate dual solution $\tilde{y}$ in $O(m \log(m\rho\epsilon^{-1}))$ time per iteration. This dual has the property that if we use $\tilde{c} = \tilde{y}^t A$ in the optimization routine, then the order of the number of iterations is the same as in the stronger model. This still does not suffice to convert the results to the RAM model, since we must also bound the precision needed for the computation. It is easy to limit the length of the numbers by a polynomial in the input length, similar to the length of the numbers used in exact linear programming algorithms. However, it might be possible to find an ϵ-approximate solution using decreased precision, as was done in [18] for the multicommodity flow problem. We leave this as an open problem.

In the application to the minimum-cost multicommodity flow problem, even approximate optimization over P will be too time consuming, and we will use a further relaxed subroutine. In order to be able to use this relaxed subroutine, we must adapt the algorithm to solve a relaxed version of the packing problem itself. The *relaxed packing problem* is defined as follows:

RELAXED PACKING: Given $\epsilon > 0$, an $m \times n$ nonnegative matrix A, $b > 0$, and convex sets P and $\hat{P}$ in the positive orthant such that $P \subseteq \hat{P}$, find $x \in \hat{P}$ such that $Ax \leq (1+\epsilon)b$, or show that $\nexists x \in P$ such that $Ax \leq b$.

The modified algorithm uses the following subroutine:

Given a dual solution y, find $\tilde{x} \in \hat{P}$ such that

$$y^t A\tilde{x} \leq \min(y^t Ax : x \in P). \tag{6}$$

It is easy to adapt both the algorithms and the proofs for the relaxed problem using this subroutine. For example, it is necessary to change only the second relaxed optimality condition, which becomes:

$(\hat{\mathcal{P}}2)$ $y^t Ax - y^t A\tilde{x} \leq \epsilon(y^t Ax + \lambda y^t b)$,

where (x, λ) and y denote the current solution in $\hat{P}$ and its corresponding dual, and $\tilde{x}$ denotes the solution returned by subroutine (6). Furthermore, σ is determined by $\hat{\rho}$, the width of $\hat{P}$ with respect to $Ax \leq b$. We shall state the resulting theorem for the case when P and $\hat{P}$ are in the product form such that $P = P^1 \times \cdots \times P^k$, $\hat{P} = \hat{P}^1 \times \cdots \times \hat{P}^k$ and $P^\ell \subseteq \hat{P}^\ell$ for $\ell = 1, \ldots, k$.

Theorem 2.9 The relaxed packing problem can be solved by a randomized algorithm that is expected to use a total of $O(\epsilon^{-2}\hat{\rho}\log(m\epsilon^{-1}) + k\log(\hat{\rho}\epsilon^{-1}))$ calls to any of the subroutines (6) for P^ℓ, $\hat{P}^\ell$ and $A^\ell x^\ell \leq b$, $\ell = 1, \ldots, k$, or by a deterministic algorithm that uses $O(\epsilon^{-2}\hat{\rho}\log(m\epsilon^{-1}))$ calls for P^ℓ, $\hat{P}^\ell$ and $A^\ell x^\ell \leq b$, for each $\ell = 1, \ldots, k$, plus the time to compute $\sum_\ell A^\ell x^\ell$, $\ell = 1, \ldots, k$, for the current iterate $(x^1, \ldots, x^k)$ between consecutive calls.

Integer Packing. In some cases, a modified version of the packing algorithm can also be used to find near-optimal solutions to the related integer packing problem. This approach is a generalization of the approach used in [15] to obtain integer solutions to the uniform multicommodity flow problem. In Section 6, we will apply this algorithm to the job-shop scheduling problem and the network embedding problem.

To simplify notation, we outline the modification to IMPROVE-PACKING to find integer solutions for the case when P is not in product form. If the input solution x is given explicitly as a convex combination of points $x^s \in P$ returned by the subroutine, $x = \sum_s \nu^s x^s$, then each iterate produced by the algorithm is maintained as such a convex combination. Furthermore, if the values ν_s for the input are all integer multiples of the value of σ for this call to the algorithm, then this property will also be maintained throughout its execution.

The original version of the packing algorithm updates x by setting it equal to $(1-\sigma)x + \sigma\tilde{x}$, where $\tilde{x} \in P$ is the point returned by the subroutine. Even if both x and $\sigma\tilde{x}$ are integral the new point $(1-\sigma)x + \sigma\tilde{x}$ might not be. The modified algorithm computes $y^t Ax^s$ for every point x^s in the convex combination. It selects the point x^q with maximum $y^t Ax^q$, and updates x to $x + \sigma(\tilde{x} - x^q)$. Since the current iterate is represented as a convex combination where each ν^s is an integer multiple of σ, the updated point is in P; furthermore, the updated point can be similarly represented. To bound the number of iterations, we again use the potential function Φ, and the same calculation as in Lemma 2.3 shows that the decrease in Φ during one iteration is at least $\alpha\sigma(1-\epsilon)y^t Ax^q - C_{\mathcal{P}}(y)$. Since $y^t Ax^q \geq y^t Ax$ by the choice of q, it follows that this decrease is at least $\alpha\sigma\epsilon\lambda\Phi$; thus we get an identical bound for the number of iterations for this modified version of the algorithm. The disadvantage of this version is that we need more time per iteration. In addition to the call to the subroutine, we must find the current solution x^q with maximum $y^t Ax^q$.

We state the resulting theorem for the version of IMPROVE-PACKING for packing problems in the product form $P = P^1 \times \cdots \times P^k$ and inequalities $\sum_\ell A^\ell x^\ell \leq b$. The algorithm maintains each $x^\ell \in P^\ell$ as a convex combination of points in P^ℓ returned by the subroutine with coefficients that are integer multiples of the current value σ^ℓ. We further modify the deterministic version in order to maintain σ as large as possible: in each iteration, we will update only one x^ℓ, deterministically choosing ℓ to maximize $\alpha\sigma^\ell\Delta_\ell$; the analysis of the randomized algorithm is based on the fact that the expected decrease of Φ is the expectation of $\alpha\sigma^\ell\Delta_\ell$, and so by choosing the maximum, we guarantee as least as good an improvement in Φ. Furthermore, in contrast to the randomized version, we still check if $\mathcal{P}2$ holds each iteration, and so there is no need to count iterations when $\mathcal{P}2$ is satisfied, but this is not detected.

Theorem 2.10 For any ϵ, $0 < \epsilon \leq 1$, given an input solution $(x^1, \ldots, x^k) \in P^1 \times \cdots \times P^k$, where each x^ℓ is represented as a convex combination of solutions returned by the subroutine (1) for P^ℓ and A^ℓ, and the coefficients are integer multiples of the current step size σ^ℓ, $\ell = 1, \ldots, k$, the modified version of IMPROVE-PACKING finds a similarly represented solution to the PACKING problem which is 6ϵ-optimal, or else the corresponding value of λ has been reduced by a factor of 2. The randomized version of the algorithm is expected to use a total of $O(\lambda_0^{-1}\epsilon^{-3}\rho\log(m\epsilon^{-1}) + k\epsilon^{-1})$ calls to any of the subroutines (1) for P^ℓ and A^ℓ, $\ell = 1, \ldots, k$; the deterministic version of the algorithm uses $O(\lambda_0^{-1}\epsilon^{-3}\rho\log(m\epsilon^{-1}) + k\epsilon^{-1})$ calls to the subroutine (1) for P^ℓ and A^ℓ, for each $\ell = 1, \ldots, k$. If the initial solution is $O(\epsilon)$-optimal, then the number of calls for both the deterministic and randomized versions is a factor of ϵ^{-1} smaller.

We use this result to obtain an integer packing theorem. For simplicity of notation, we shall state the result in terms of $\bar{\rho} = \max_\ell \rho^\ell$, instead of the individual ρ^ℓ values. We shall assume that there is a parameter d such that each coordinate of any point returned by the subroutine is an integer multiple of d. If each σ^s is an integer multiple of $1/d$, then the current solution is integral. We will set σ^s equal to the minimum of 1 and the maximum value $2^r/d$ that is at most $\frac{\epsilon}{4\alpha\rho^s}$, where r is an integer. The algorithm will work by repeatedly calling the modified version of IMPROVE-PACKING, and will terminate as soon as $\frac{\epsilon}{4\alpha\bar{\rho}} < \frac{1}{d}$. The main outline of the algorithm is the same as above. First set $\epsilon = 1/6$, and repeatedly call the modified version of IMPROVE-PACKING until a 6ϵ-optimal solution has been found. Then we begin the ϵ-scaling phase, and continue until σ^s becomes too small, where s is such that $\rho^s = \bar{\rho}$. Unlike the previous algorithms, this algorithm continues even if it has been shown that there does not exist $x \in P$ such that $Ax \leq b$. This algorithm finds an integer point in P, but it might only satisfy a greatly relaxed version of the packing constraints. The following theorem gives a bound on the quality of the solution delivered by this algorithm. The theorem is an extension of a result in [15] for the multicommodity flow problem with uniform capacities. The existence of an integer solution under similar assumptions has been proven by Raghavan [21]. However, Raghavan constructs the integer solution using linear programming.

Theorem 2.11 Let $\lambda' = \max(\lambda^*, (\bar{\rho}/d)\log m)$. There exists an integral solution to $\sum_\ell A^\ell x^\ell \leq \lambda b$ with $x^\ell \in P^\ell$ and $\lambda \leq \lambda^* + O(\sqrt{\lambda'(\bar{\rho}/d)\log(mkd)})$. Repeated calls to the randomized integer version of IMPROVE-PACKING find such a solution $(\bar{x}, \bar{\lambda})$ with an expected total of $O(d\rho/\bar{\rho} + \rho\log(m)/\bar{\lambda} +$

$k \log(d\rho/\bar{\rho})$) calls to any of the subroutines (1) for P^ℓ and A^ℓ, $\ell = 1, \ldots, k$. A deterministic version of the algorithm uses $O(d\rho/\bar{\rho} + \rho \log(m)/\bar{\lambda} + k \log(d\rho/\bar{\rho}))$ calls to each of the k subroutines.

Proof: We first analyze the number of iterations of the deterministic algorithm given above, using Theorem 2.10 in a way similar to the analysis of the algorithm for the fractional packing problem. Let $(\bar{x}, \bar{\lambda})$ denote the solution output by the algorithm. First we compute the number of iterations when $\epsilon = 1/6$. The first term of the bound in Theorem 2.10 depends on λ^{-1}, which doubles with each call to IMPROVE-PACKING, and so its total contribution can be bounded by its value for the final call with $\epsilon = 1/6$. Since the value of λ changes by at most a factor of 2 during ϵ-scaling, this term contributes a total of $O(\rho \log(m)/\bar{\lambda})$ iterations to the overall bound. The contribution of the second term is k times the number of times that IMPROVE-PACKING is called with $\epsilon = 1/6$; this yields a term of $O(k \log(\rho/\bar{\lambda}))$ in the overall bound. To bound the number of iterations during ϵ-scaling, we first bound the value of ϵ at termination. Focus on the call to IMPROVE-PACKING for which $\sigma^s < 1/d$, where s is such that $\rho^s = \bar{\rho}$. Recall that $\lambda^* \leq \lambda \leq 2\lambda^*$ throughout ϵ-scaling. Since $\sigma^s = \Theta(\epsilon^2 \lambda/(\bar{\rho} \log(m\epsilon^{-1})))$, it suffices to have $\epsilon = \Theta(\sqrt{\bar{\rho} \log(mkd)/(\lambda^* d)})$, in order that $\sigma^s < 1/d$. The total contribution of the first term of the bound in Theorem 2.10 throughout ϵ-scaling can again be bounded by its value for the final call to IMPROVE-PACKING, which is $O(\rho d/\bar{\rho})$. The total contribution of the second term can be bounded by $O(k \log \epsilon^{-1}) = O(k \log(\bar{\lambda} d/\bar{\rho}))$. This yields the claimed bound. The analysis of the randomized version is identical.

Next consider the quality of the solution found. The algorithm can terminate due to reducing σ^s below $1/d$ either while $\epsilon = 1/6$, or during the ϵ-scaling. Suppose that the former option occurs. We know that $\sigma^s \geq \epsilon/(8\alpha\rho^s) = \Omega(\bar{\lambda}/(\bar{\rho} \log m))$. This implies that $\bar{\lambda} = O((\bar{\rho}/d) \log m)$, and this is within the claimed bound. Next assume that $\epsilon < 1/6$ when σ^s becomes less than $1/d$. Since $\lambda^* \leq \bar{\lambda} \leq 2\lambda^*$ and $\sigma^s = \Theta(\epsilon/(\alpha\rho^s))$, we see that $\sigma^s = \Theta(\epsilon^2 \lambda^*/(\bar{\rho} \log(m\epsilon^{-1})))$ in this case. This implies that $\epsilon = \Theta(\sqrt{\bar{\rho} \log(m\epsilon^{-1})/(d\lambda^*)})$, which is $O(\sqrt{\bar{\rho} \log(mkd)/(d\lambda^*)})$. The output, which is obtained at the end of the previous scaling phase, is $O(\epsilon)$-optimal. Therefore, $\bar{\lambda}$ meets the claimed bound. ∎

3 The Fractional Covering Problem

The *fractional covering problem* is defined as follows:

COVERING: $\exists ? x \in P$ such that $Ax \geq b$, where A is a nonnegative $m \times n$ matrix, $b > 0$, and P is a convex set in the positive orthant of $\mathbf{R}^n$.

In this section, we shall describe a relaxed decision procedure for the fractional covering problem whose running time depends on the *width* ρ of P relative to $Ax \geq b$; as in the previous section, the width is defined $\max_i \max_{x \in P} a_i x/b_i$.

We shall assume that we are given a fast subroutine to solve the following optimization

problem for P and A:

Given an m-dimensional vector $y \geq 0$, find $\tilde{x} \in P$ such that

$$c\tilde{x} = \min(cx : x \in P), \text{ where } c = y^t A. \tag{7}$$

For a given error parameter $\epsilon > 0$, an *ϵ-approximate solution* to the COVERING problem is a vector $x \in P$ such that $Ax \geq (1-\epsilon)b$; an *exact solution* is a vector $x \in P$ such that $Ax \geq b$.

Relaxed Optimality. Consider the following optimization version of the COVERING problem:

$$\max(\lambda : Ax \geq \lambda b \text{ and } x \in P), \tag{8}$$

and let λ^* denote its optimal value. For each $x \in P$ there is a corresponding maximum value λ such that $Ax \geq \lambda b$. We shall use the notation (x, λ) to denote that λ is the maximum value corresponding to x. A solution (x, λ) is *ϵ-optimal* if $x \in P$ and $\lambda \geq (1-\epsilon)\lambda^*$.

The method to solve this problem is analogous to the one used for the fractional packing problem. Let $y \geq 0$, $y \in \mathbf{R}^m$ denote a *dual solution*, and let $C_{\mathcal{C}}(y)$ denote the maximum value of cx for any $x \in P$ where $c = y^t A$. Let (x, λ) denote a feasible solution, and consider the following chain of inequalities:

$$\lambda y^t b \leq y^t A x \leq C_{\mathcal{C}}(y). \tag{9}$$

Observe that for any dual solution y, the value $C_{\mathcal{C}}(y)/y^t b$ is an upper bound on λ^*. We will use the following two relaxed optimality conditions.

($\mathcal{C}1$) $(1+\epsilon)\lambda y^t b \geq y^t A x$

($\mathcal{C}2$) $C_{\mathcal{C}}(y) - y^t A x \leq \epsilon(C_{\mathcal{C}}(y) + y^t b)$.

Note that the last term in $\mathcal{P}2$ is λ times the last term in $\mathcal{C}2$. This is done to improve the running times. Due to this difference in the definition we cannot claim that a pair (x, λ) and y satisfying condition $\mathcal{C}1$ and $\mathcal{C}2$ are ϵ-optimal unless λ is close to 1. We have the following lemma instead.

Lemma 3.1 Suppose that (x, λ) and y are feasible primal and dual solutions that satisfy the relaxed optimality conditions $\mathcal{C}1$ and $\mathcal{C}2$. If $\lambda \leq 1 - 3\epsilon$, then there does not exist an exact solution to the fractional covering problem. If $\lambda \geq 1 - 3\epsilon$, then x is 3ϵ-optimal.

Proof: $\mathcal{C}1$ and $\mathcal{C}2$ imply that

$$C_{\mathcal{C}}(y) \leq (1-\epsilon)^{-1}(y^t A x + \epsilon y^t b) \leq (1-\epsilon)^{-1}((1+\epsilon)\lambda + \epsilon) y^t b.$$

IMPROVE-COVER(x, ϵ)
$\lambda_0 \leftarrow \min_i a_i x/b_i$; $\alpha \leftarrow 4\lambda_0^{-1}\epsilon^{-1}\ln(4m\epsilon^{-1})$; $\sigma \leftarrow \frac{\epsilon}{4\alpha\rho}$.
While $\min_i a_i x/b_i \leq 2\lambda_0$ **and** x and y do not satisfy $\mathcal{C}2$
  For each $i = 1, \ldots, m$: set $y_i \leftarrow \frac{1}{b_i} e^{-\alpha a_i x/b_i}$.
  Find a maximum-cost point $\tilde{x} \in P$ for costs $c = y^t A$.
  Update $x \leftarrow (1-\sigma)x + \sigma\tilde{x}$.
Return x.

Figure 3: Procedure IMPROVE-COVER.

Consider the case when $\lambda \geq 1 - 3\epsilon$. For any dual solution y, $\lambda^* \leq C_{\mathcal{C}}(y)/(y^t b)$. This implies that

$$\frac{\lambda^*}{\lambda} \leq \frac{C_{\mathcal{C}}(y)}{\lambda y^t b} \leq \frac{(1+\epsilon)\lambda + \epsilon}{\lambda(1-\epsilon)} \leq \frac{1+\epsilon}{1-\epsilon} + \frac{\epsilon}{(1-\epsilon)(1-3\epsilon)} \leq \frac{1}{1-3\epsilon}.$$

On the other hand, if $\lambda \leq 1 - 3\epsilon$, we have

$$\lambda^* \leq \frac{C_{\mathcal{C}}(y)}{y^t b} \leq \frac{(1+\epsilon)(1-3\epsilon)+\epsilon}{1-\epsilon} < 1.$$

Hence, in this case, there is no exact solution to the fractional covering problem. ∎

The Algorithm. The heart of the covering algorithm is the procedure IMPROVE-COVER (see Figure 3), which is the covering analog of the procedure IMPROVE-PACKING. It uses a dual solution y defined as a function of x, where $y_i = \frac{1}{b_i} e^{-\alpha a_i x/b_i}$ for some parameter α; y is the *dual solution corresponding to* x. Throughout the procedure, the current solution (x, λ) and its corresponding dual solution y will satisfy $\mathcal{C}1$. If $\mathcal{C}2$ is also satisfied, then we can either conclude that no feasible solution exists, or that λ is sufficiently close to optimality, and we can terminate. Otherwise, we find the point $\tilde{x} \in P$ that attains the maximum $C_{\mathcal{C}}(y)$, and we modify x by moving a small amount towards $\tilde{x}$. This will decrease the potential function $\Phi = y^t b$, and gradually increase λ.

The following lemma is similar to Lemma 2.2.

Lemma 3.2 If $\alpha \geq 2\lambda^{-1}\epsilon^{-1}\ln(4m\epsilon^{-1})$ and $0 < \epsilon < 1$, then any feasible solution (x, λ) and its corresponding dual solution y satisfy $\mathcal{C}1$.

Proof: For this proof, it is useful to introduce a localized version of $\mathcal{C}1$.

($\hat{\mathcal{C}}1$) For each $i = 1, \ldots, m$, $(1 + \epsilon/2)\lambda b_i \geq a_i x$ or $a_i x y_i \leq \frac{\epsilon}{2m}\lambda y^t b$.

Note that $\hat{\mathcal{C}}1$ implies that

$$a_i x y_i \leq (1+\epsilon/2)\lambda y_i b_i + \frac{\epsilon}{2m}\lambda y^t b.$$

Summing up over all i, we see that $\hat{C}1$ implies $C1$.

Next we show that the hypothesis of the lemma implies that $\hat{C}1$ is satisfied. Notice that $y^t b = \sum_i e^{-\alpha a_i x/b_i}$. By the maximality property of λ, we have that $y^t b$ is at least $e^{-\alpha\lambda}$. Consider any row i for which $(1+\epsilon/2)\lambda b_i < a_i x$ and let $\lambda_i = a_i x/b_i$. This implies that $\lambda_i > (1+\epsilon/2)\lambda$. If $\alpha z \geq 1$, then $ze^{-\alpha z}$ is a monotonically nonincreasing function of z. Since, by definition of α, we have $\lambda_i \alpha > \lambda\alpha \geq 1$, we get that

$$\frac{a_i x y_i}{y^t b} = \frac{(a_i x/b_i)e^{-\alpha a_i x/b_i}}{y^t b} = \frac{\lambda_i e^{-\alpha\lambda_i}}{y^t b} < \frac{(1+\epsilon/2)\lambda e^{-\alpha\lambda(1+\epsilon/2)}}{e^{-\alpha\lambda}} = (1+\epsilon/2)\lambda e^{-\alpha\epsilon\lambda/2} \leq \frac{\epsilon\lambda}{2m}. \quad \blacksquare$$

Next we prove that for an appropriately chosen σ, the new solution significantly reduces the potential function $\Phi = y^t b = \sum_i e^{-\alpha a_i x/b_i}$.

Lemma 3.3 Consider a point $x \in P$ and an error parameter ϵ, $0 < \epsilon \leq 1$, such that x and its corresponding dual solution y have potential function value Φ and do not satisfy $C2$. Let $\tilde{x} \in P$ attain the maximum $C_C(y)$. Assume that $\sigma \leq \frac{\epsilon}{4\alpha\rho}$. Define a new solution by $\hat{x} = (1-\sigma)x + \sigma\tilde{x}$, and let $\hat{\Phi}$ denote the potential function value for $\hat{x}$ and its corresponding dual solution $\hat{y}$. Then $\Phi - \hat{\Phi} \geq \epsilon\alpha\sigma\Phi$.

Proof: By the definition of ρ, $Ax \leq \rho b$ and $A\tilde{x} \leq \rho b$. This implies that $\alpha\sigma|a_i x - a_i\tilde{x}|/b_i \leq \epsilon/4 \leq 1/4$. Using the second-order Taylor theorem we see that if $|\delta| \leq \epsilon/4 \leq 1/4$, then, for all x, $e^{x+\delta} \leq e^x + \delta e^x + \frac{\epsilon}{2}|\delta|e^x$. Setting $\delta = \alpha\sigma(a_i x - a_i\tilde{x})/b_i$, we see that

$$\begin{aligned}
\hat{y}_i &\leq y_i + \frac{1}{b_i}\frac{\alpha\sigma(a_i x - a_i\tilde{x})}{b_i}e^{-\alpha a_i x/b_i} + \frac{1}{b_i}\frac{\epsilon\alpha\sigma|a_i x - a_i\tilde{x}|}{2b_i}e^{-\alpha a_i x/b_i} \\
&\leq y_i + \alpha\sigma\frac{1}{b_i}(a_i x - a_i\tilde{x})y_i + \epsilon\alpha\sigma\frac{1}{2b_i}(a_i x + a_i\tilde{x})y_i.
\end{aligned}$$

Using this inequality to bound the change in the potential function, we get

$$\begin{aligned}
\Phi - \hat{\Phi} &= \sum_i (y_i - \hat{y}_i)b_i \geq \alpha\sigma\sum_i (a_i\tilde{x} - a_i x)y_i - \alpha\sigma\frac{\epsilon}{2}\sum_i (a_i\tilde{x} + a_i x)y_i \\
&= \alpha\sigma(y^t A\tilde{x} - y^t Ax) - \alpha\sigma\frac{\epsilon}{2}(y^t A\tilde{x} + y^t Ax) \\
&= \alpha\sigma(C_C(y) - y^t Ax) - \alpha\sigma\frac{\epsilon}{2}(C_C(y) + y^t Ax) \geq \alpha\sigma(C_C(y) - y^t Ax) - \alpha\sigma\epsilon C_C(y).
\end{aligned}$$

Since $C2$ is not satisfied, the decrease in Φ is at least $\alpha\sigma\epsilon\Phi$. $\blacksquare$

Next we show that the chosen value of α is large enough to guarantee that condition $C1$ is always satisfied during the execution of IMPROVE-COVER.

Lemma 3.4 If $0 < \epsilon < 1$, then throughout the execution of IMPROVE-COVER, the current solution (x, λ) has $\lambda \geq 3\lambda_0/4$.

Proof: The value of the potential function Φ does not increase during the execution of IMPROVE-COVER. Initially, $\Phi \leq me^{-\alpha\lambda_0}$, and for any current solution (x, λ), $\Phi \geq e^{-\alpha\lambda}$. Therefore

$$\lambda_0 - \lambda \leq \frac{1}{\alpha} \ln m \leq \epsilon\lambda_0/4 \leq \lambda_0/4.$$

This implies that $\lambda \geq 3\lambda_0/4$. ∎

Since we set $\alpha = 4\epsilon^{-1}\lambda_0^{-1}\log(m\epsilon^{-1})$, Lemma 3.2 implies that $\mathcal{C}1$ is satisfied throughout the execution of IMPROVE-COVER. Since σ is set equal to $\frac{\epsilon}{4\alpha\rho}$, the decrease in the potential function due to a single iteration is $\Omega(\frac{\epsilon^2}{\rho}\Phi)$. Observe that during a single call to IMPROVE-COVER we have $e^{-2\alpha\lambda_0} \leq \Phi \leq me^{-\alpha\lambda_0}$. If the initial solution is 6ϵ-optimal for $\epsilon \leq 1/12$, then we have the tighter bound, $e^{-\alpha(1+12\epsilon)\lambda_0} \leq e^{-\alpha(1-6\epsilon)^{-1}\lambda_0} \leq \Phi \leq me^{-\alpha\lambda_0}$. This, together with Lemma 3.3, can be used to bound the number of iterations in a single call to IMPROVE-COVER.

Theorem 3.5 The procedure IMPROVE-COVER terminates in $O(\epsilon^{-3}\rho\log(m\epsilon^{-1}))$ iterations. If the initial solution is 6ϵ-optimal for $\epsilon \leq 1/12$, then IMPROVE-PACKING terminates in $O(\epsilon^{-2}\rho\log(m\epsilon^{-1}))$ iterations.

We use the procedure IMPROVE-COVER repeatedly to find an ϵ_0-approximate solution. Before this, we must specify how to obtain an initial solution of sufficient quality, which is somewhat more involved than for the packing analog (where any initial solution in P suffices). For each $i = 1, \ldots, m$, we find $x_i \in P$ that maximizes $a_i x$; this takes m calls to the subroutine. If there exists an i such that $max(a_i x : x \in P) < b_i$, then we can conclude that no exact solution for the COVERING problem exists. Otherwise, we take $(1/m)\sum_i x_i$ as the initial solution, for which the corresponding value of λ is at least $1/m$.

Lemma 3.6 With m calls to the subroutine (7) for P and A, we can either find a solution $x \in P$ satisfying $Ax \geq (1/m)b$, or conclude that there does not exist an exact solution to the COVERING problem.

The basic approach to using IMPROVE-COVER to obtain an ϵ_0-approximate solution closely parallels the packing algorithm. First set $\epsilon = 1/6$ and start with the point $x \in P$ given by the previous lemma; this is a solution (x, λ) with $\lambda \geq 1/m$. Between consecutive calls to IMPROVE-COVER, we increase λ_0 by at least a factor of 2, and so within $\log m$ iterations, IMPROVE-COVER must output an exact solution or else find feasible primal and dual solutions (x, λ) and y that satisfy $\mathcal{C}1$ and $\mathcal{C}2$. Suppose the latter occurs. If $\lambda \leq 1 - 3\epsilon = 1/2$, then the algorithm concludes that no exact solution exists (by Lemma 3.1); otherwise, $\lambda > 1/2$ and x is 1/2-optimal. If $\epsilon_0 \geq 1/2$ then the algorithm outputs x and stops; otherwise, we proceed to ϵ-scaling. Each scaling phase decreases ϵ by a factor of 2, and then makes a single call to IMPROVE-COVER. The algorithm checks if the resulting solution (x, λ) is an ϵ_0-approximate solution, and if so, outputs x and stops. Otherwise, it checks if $\lambda \leq 1 - 3\epsilon$, and if so, claims that no exact solution exists and stops. If neither termination criterion is satisfied, the algorithm proceeds to the next phase. The input to each scaling phase has $\lambda \geq 1 - 3\epsilon \geq 1/2$, and so the output is either an

exact solution, or else satisfies the relaxed optimality conditions. Note that the output must be 3ϵ-optimal in the only case when further phases are needed. Hence, each call to IMPROVE-COVER has an input that is a 6ϵ-optimal solution with the new $\epsilon \leq 1/12$. Furthermore, the algorithm is guaranteed to stop by the point when $\epsilon \leq \epsilon_0/3$. IMPROVE-COVER uses $O(\rho \log m)$ iterations for each of the $O(\log m)$ iterations with $\epsilon = 1/6$, and the number of iterations during ϵ-scaling is dominated by the the number of iterations during the final call to IMPROVE-COVER, which implies the following theorem.

Theorem 3.7 For $0 < \epsilon < 1$, repeated calls to IMPROVE-COVER can be used so that the algorithm either finds an ϵ-approximate solution for the fractional covering problem, or proves that no exact solution exists; the algorithm uses $O(m+\rho \log^2 m+\epsilon^{-2}\rho \log(m\epsilon^{-1}))$ calls to the subroutine (7) for P and A, plus the time to compute Ax for the current iterate x between consecutive calls.

Randomized Version. As was true for the fraction packing problem, we can use randomization to speed up the fractional covering algorithm if the polytope is in product form. Suppose that $P = P^1 \times \cdots \times P^k$ and the inequalities, when written to reflect this structure, are $\sum A^\ell x^\ell \geq b$. In this case, a subroutine (7) to compute $C_{\mathcal{C}}(y)$ for P and A consists of a call to each of k subroutines: subroutine (7) for P^ℓ and A^ℓ, $\ell = 1, \ldots, k$. Instead of calling all k subroutines each iteration, the randomized algorithm will make a call to the subroutine (7) for P^s and A^s for a single value $s \in \{1, \ldots, k\}$. The choice of s is made at random, according to a probability distribution that we will describe below, independently of choices made in other iterations. A tentative modification of the current iterate $(x^1, \ldots, x^k)$ is made, where only the coordinate x^s is updated. If this change causes the potential function to decrease, then it is really made; otherwise, the current iterate is unchanged by this iteration. Since $C_{\mathcal{C}}(y)$ is not computed each iteration, we cannot check if $\mathcal{C}2$ is satisfied; instead, in each iteration, with probability $1/k$, the algorithm does the additional work needed to check this condition.

Let ρ^ℓ denote the width of P^ℓ subject to $A^\ell x^\ell \geq b$, $\ell = 1, \ldots, k$. As in the case of the PACKING problem, randomization speeds up the algorithm by roughly a factor of k if $\rho^1 = \rho^2 = \cdots = \rho^k$, or the k subroutines have the same time bound. The key to the randomized version of our algorithm is the following lemma, which is analogous to Lemma 2.6.

Lemma 3.8 Consider a point $(x^1, \ldots, x^k) \in P^1 \times \ldots \times P^k$ with corresponding dual solution y and potential function value Φ, and an error parameter ϵ, $0 < \epsilon < 1$. Let $\tilde{x}^s$ be a point in P^s that maximizes the cost $c^s x^s$, where $c^s = y^t A^s$, $s = 1, \ldots, k$, and assume that $\sigma^s \leq \min\{\epsilon/(4\rho^s\alpha), 1\}$. Define a new point by changing only x^s, where $x^s \leftarrow (1-\sigma^s)x^s + \sigma^s\tilde{x}^s$. If $\hat{\Phi}$ denotes the potential function value of the new solution, then $\Phi - \hat{\Phi} \geq \alpha\sigma^s((1-\epsilon)y^t A^s\tilde{x}^s - y^t A^s x^s)$.

Since the algorithm updates x only when Φ decreases, the change in Φ associated with updating x_s is $\alpha\sigma^s\Delta_s$, where $\Delta_s = \max\{((1-\epsilon)y^t A^s\tilde{x}^s - y^t A^s x^s), 0\}$. We have restricted $\sigma^s \leq 1$ to ensure that the new point is in P; to get the maximum improvement, the algorithm sets $\sigma^\ell = \min\{1, \epsilon/(4\alpha\rho^\ell)\}$. Let $S = \{s : 4\alpha\rho^s \leq \epsilon\}$ and define $\rho' = \sum_{s \notin S} \rho^s$. The probability $\beta(s)$ with which we pick an index s is defined as follows:

$$\beta(s) = \begin{cases} \frac{\rho^s}{2\rho'} & \text{for } s \notin S \\ \frac{1}{2|S|} & \text{for } s \in S \end{cases}$$

Using Lemma 3.8 instead of Lemma 3.3, we get the following theorem:

Theorem 3.9 For $0 < \epsilon < 1$, repeated calls the randomized version of IMPROVE-COVER can be used so that the algorithm either finds an ϵ-optimal solution for the fractional covering problem defined by the polytope $P = P^1 \times \cdots \times P^k$ and inequalities $\sum_\ell A^\ell x^\ell \geq b$, or else proves that no exact solution exists; it is expected to use a total of $O(mk + \rho \log^2 m + k \log \epsilon^{-1} + \epsilon^{-2}\rho \log(m\epsilon^{-1}))$ calls to any of the subroutines (7) for P^ℓ and A^ℓ, $\ell = 1, \ldots, k$, plus the time to compute $\sum_\ell A^\ell x^\ell$ between consecutive calls.

Proof: We find an initial solution using Lemma 3.6. This requires m calls to the subroutine (7) for each P^ℓ and A^ℓ, $\ell = 1, \ldots, k$. Hence, we can assume that our initial solution (x, λ) has $\lambda \geq 1/m$.

Next we analyze a single call to the randomized variant of IMPROVE-COVER, in a way completely analogous to the analysis of the randomized version of IMPROVE-PACKING. There are two types of iterations: those where $C2$ is satisfied, and those where it isn't. We separately bound the expected number of each of these. For the former, since $C2$ is checked with probability $1/k$ in each iteration, there are $O(k)$ of these iterations expected before the algorithm detects that $C2$ is satisfied, and terminates. In the latter case, we shall show that the expected decrease of Φ during each iteration is $\Omega\{\min\{\epsilon^2/\rho, \lambda^{-1}\log(m\epsilon^{-1})/k\}\Phi$. Since $C2$ is not satisfied, $\sum_s \Delta_s \geq \epsilon\Phi$, where $\alpha\sigma^s\Delta_s$ is the decrease in Φ associated with updating x_s. Using this fact and applying Lemma 3.8, we see that the expected decrease in Φ is

$$\begin{aligned} \sum_s \alpha\sigma^s\Delta_s\beta(s) &= \sum_{s\notin S} \alpha\frac{\epsilon}{4\rho^s\alpha}\cdot\frac{\rho^s}{2\rho'}\Delta_s + \sum_{s\in S}\frac{\alpha}{2|S|}\Delta_s \\ &\geq \min\left\{\frac{\epsilon}{8\rho}, \frac{\alpha}{2k}\right\}\sum_{s=1}^{k}\Delta_s \geq \min\left\{\frac{\epsilon}{8\rho}, \frac{\alpha}{2k}\right\}\epsilon\Phi. \end{aligned}$$

Since $\alpha = \Omega(\epsilon^{-1}\lambda^{-1}\log(m\epsilon^{-1}))$, we get the claimed decrease in Φ.

To analyze the number of iterations, we once again apply the result of Karp [14]. This implies that the randomized version of IMPROVE-COVER is expected to terminate in $O(\rho\epsilon^{-3}\log(m\epsilon^{-1}) + k\epsilon^{-1}\lambda_0)$ iterations, and is a factor of ϵ^{-1} faster if the initial solution is 6ϵ-optimal for $\epsilon \leq 1/12$.

We use this randomized version of IMPROVE-COVER repeatedly to find an ϵ_0-approximate solution in exactly the same way as in the deterministic case. First we set $\epsilon = 1/6$, and then use ϵ-scaling. The contribution of terms that depend on λ_0 or ϵ^{-1} can be bounded by their value for calls in which these parameters are largest. For the remaining terms, we need only observe that there are $O(\log m)$ calls to IMPROVE-COVER with $\epsilon = 1/6$, and $O(\log \epsilon_0^{-1})$ calls during ϵ-scaling. To complete the proof, we note that condition $C2$ is expected to be checked in an $O(1/k)$ fraction of the iterations. ∎

Relaxed Version. It is not hard to see that a subroutine that finds a point in P of cost not much less than the maximum can be used in the algorithm, and gives a bound on the number of iterations of the same order of magnitude.

Theorem 3.10 **If the optimization subroutine (7) used in each iteration of IMPROVE-COVER is replaced by an algorithm that finds a point $\tilde{x} \in P$ such that $y^t A\tilde{x} \geq (1-\epsilon/2)C_C(y) - (\epsilon/2)\lambda y^t b$ for any given $y \geq 0$, then the resulting procedure yields a relaxed decision procedure for the fractional covering problem; furthermore, in either the deterministic or the randomized implementations, the number of iterations can be bounded exactly as before, in Theorems 3.7 and 3.9, respectively.**

Proof: It is easy to prove that the analogs of Lemmas 3.3 and 3.8 remain valid. The rest of the proof follows from these lemmas. ∎

In some of our applications, the optimization over P required by the original version of the algorithm is difficult or even NP-hard. However, if there is a fast fully polynomial approximation scheme for the problem, then Theorem 3.10 allows us to use the approximation algorithm instead.

4 The General Problem

Consider the class of problems in the following form:

GENERAL: $\exists ? x \in P$ such that $Ax \leq b$, where A is an arbitrary $m \times n$ matrix, b is an arbitrary vector, and P is a convex set in $\mathbf{R}^n$.

We shall assume that we are given a fast subroutine to solve the following optimization problem for P and A:

Given an m-dimensional vector $y \geq 0$, find a point $\tilde{x} \in P$ such that:

$$c\tilde{x} = \min(cx : x \in P) \text{ where } c = y^t A. \tag{10}$$

Given an error parameter $\epsilon > 0$ and a positive vector d, we shall say that a point $x \in P$ is an *ϵ-approximate solution* if $Ax \leq b + \epsilon d$; an *exact solution* is a point $x \in P$ such that $Ax \leq b$. The running time of the relaxed decision procedure for this problem depends on the *width* ρ of P relative to $Ax \leq b$ and d, which is defined in this case by

$$\rho = \max_i \max_{x \in P} \frac{|a_i x - b_i|}{d_i}.$$

This formulation of the problem is quite general. We shall also be interested in a special case of this problem, where the constraints can be viewed as simultaneous packing and covering constraints.

SIMULTANEOUS PACKING AND COVERING: $\exists ? x \in P$ such that $Ax \leq b$, and $\hat{A}x \geq \hat{b}$ where $\hat{A}$ and A are $\hat{m} \times n$ and $(m - \hat{m}) \times n$ nonnegative matrices, $b > 0$ and $\hat{b} > 0$, and P is a convex set in the positive orthant of $\mathbf{R}^n$.

In other words, this is the special case where the coefficients of each row of $Ax \leq b$ are either all positive (a packing constraint) or all negative (a covering constraint). Furthermore, given this interpretation it is natural to define d as $d_i = |b_i|$, for $i = 1, \ldots, m$.

To simplify notation for this special case, we will let $a_i x \leq b_i$ denote the ith row of $Ax \leq b$ and let $\hat{a}_i x \geq \hat{b}_i$ denote the ith row of $\hat{A}x \geq \hat{b}$. Then, for a given error parameter ϵ, $0 < \epsilon < 1$, an *ϵ-approximate solution* is a vector $x \in P$ such that $Ax \leq (1 + \epsilon)b$ and $\hat{A}x \geq (1 - \epsilon)\hat{b}$. In the next section we will give techniques to reduce the width ρ, which for this problem is $\max_{x \in P} \max\{\max_i a_i x / b_i, \max_i \hat{a}_i x / \hat{b}_i\}$.

Relaxed Optimality. Consider the following optimization version of the GENERAL problem:

$$\min(\lambda : Ax \leq b + \lambda d \text{ and } x \in P), \tag{11}$$

and let λ^* denote its optimal value. There exists an exact solution to the GENERAL problem if $\lambda^* \leq 0$. For each $x \in P$, there is a corresponding minimum value λ such that $Ax \leq b + \lambda d$. We shall use the notation (x, λ) to denote that λ is the minimum value corresponding to x. A solution (x, λ) with $x \in P$ is an ϵ-approximate solution if $\lambda \leq \epsilon$.

The algorithm for this problem is similar to the packing and covering algorithms discussed in the previous two sections. Let $y \geq 0$, $y \in \mathbf{R}^m$ denote a *dual solution*, and let $C_{\mathcal{G}}(y)$ denote the minimum $C_{\mathcal{G}}(y) = \min(y^t Ax - y^t b : x \in P)$. Let (x, λ) denote a feasible solution, let y denote a dual solution and consider the following chain of inequalities:

$$\lambda y^t d \geq y^t Ax - y^t b \geq C_{\mathcal{G}}(y). \tag{12}$$

It follows that for any dual solution y, $\lambda^* \geq C_{\mathcal{G}}(y)/y^t d$. Notice that if there exists an exact solution to the GENERAL problem, then $C_{\mathcal{G}}(y) \leq 0$ for each dual solution y. The relaxed optimality conditions for this problem are defined as follows:

($\mathcal{G}1$) $\lambda y^t d \leq 4y^t(Ax - b)$

($\mathcal{G}2$) $y^t(Ax - b) - C_{\mathcal{G}}(y) \leq (\lambda/5)y^t d$.

IMPROVE-GENERAL(x, ϵ)

$\lambda_0 \leftarrow \max_i(a_i x - b_i)/d_i$; $\alpha \leftarrow 4\lambda_0^{-1} \ln(6m\rho\lambda_0^{-1})$; $\sigma \leftarrow \frac{\lambda_0}{24\alpha\rho^2}$.

While $\max_i(a_i x - b_i)/d_i \geq \lambda_0/2$ **and** x and y do not satisfy $\mathcal{G}2$

  For each $i = 1, \ldots, m$: set $y_i \leftarrow \frac{1}{d_i} e^{\alpha(a_i x - b_i)/d_i}$.

  Find a minimum-cost point $\tilde{x} \in P$ for costs $c = y^t A$.

  Update $x \leftarrow (1 - \sigma)x + \sigma\tilde{x}$.

Return x.

Figure 4: Procedure IMPROVE-GENERAL.

The relaxation in $\mathcal{G}2$ seems weaker than in the case of the packing or covering problems, since neither $C_{\mathcal{G}}(y)$ nor $y^t Ax$ is included on the right-hand side. However, $C_{\mathcal{G}}(y) \leq 0$ whenever there exists an exact solution, and $y^t Ax$ is not known to be positive. The following lemma is the analog to Lemma 3.1.

Lemma 4.1 Suppose that (x, λ) and $y \neq 0$ are feasible primal and dual solutions that satisfy the relaxed optimality conditions $\mathcal{G}1$ and $\mathcal{G}2$, and $\lambda > 0$. Then there does not exist an exact solution to the GENERAL problem.

Proof: Conditions $\mathcal{G}1$ and $\mathcal{G}2$ imply that

$$\lambda y^t d \geq 5(y^t(Ax - b) - C_{\mathcal{G}}(y)) \geq \frac{5}{4}\lambda y^t d - 5C_{\mathcal{G}}(y),$$

which implies that

$$C_{\mathcal{G}}(y) \geq \frac{1}{20}\lambda y^t d.$$

The assumptions that $d > 0$ and $y \neq 0$ imply that $y^t d > 0$. Since $\lambda > 0$,

$$\lambda^* \geq \frac{C_{\mathcal{G}}(y)}{y^t d} > \frac{1}{20}\lambda > 0,$$

and hence there does not exist an exact solution. ∎

The Algorithm. The heart of the algorithm is procedure IMPROVE-GENERAL (see Figure 4), which is the analog of procedures IMPROVE-PACKING and IMPROVE-COVER. For each x, let y be the *dual solution corresponding to* x, where $y_i = \frac{1}{d_i} e^{\alpha(a_i x - b_i)/d_i}$ for some choice of the parameter α. Condition $\mathcal{G}1$ will be satisfied throughout the procedure. If the current solution (x, λ) has $\lambda \leq \epsilon$, then x is an ϵ-approximate solution. If $\lambda > \epsilon$, and $\mathcal{G}2$ is satisfied as well, then, by Lemma 4.1, we can conclude that there does not exist an exact solution. Otherwise, we use the point $\tilde{x} \in P$ that attains the minimum $C_{\mathcal{G}}(y)$ to modify x by moving a small amount towards $\tilde{x}$. This decreases the potential function $\Phi = y^t d$, and gradually decreases λ.

We first prove that if α is sufficiently large, then $\mathcal{G}1$ is satisfied.

Lemma 4.2 If $\alpha \geq 2\lambda^{-1}\ln(6m\rho\lambda^{-1})$, then any feasible solution (x,λ) and its corresponding dual solution y satisfy $\mathcal{G}1$.

Proof: Consider the following localized version of $\mathcal{G}1$:

($\hat{\mathcal{G}}1$) for each $i = 1,\ldots,m$, $2(a_ix - b_i) \geq \lambda d_i$ or $y_id_i \leq \lambda y^td/(6m\rho)$.

We first show that $\hat{\mathcal{G}}1$ implies $\mathcal{G}1$. Let I denote the set of indices such that $2(a_ix - b_i) \geq \lambda d_i$. Then, using $\lambda \leq \rho$, we see that

$$\begin{aligned}
\lambda y^td &= \lambda\sum_{i\in I} y_id_i + \lambda\sum_{i\notin I} y_id_i \leq 2\sum_{i\in I}(a_ix - b_i)y_i + \sum_{i\notin I}\lambda^2 y^td/(6m\rho) \\
&\leq 2y^t(Ax - b) + 2\sum_{i\notin I} y_i|a_ix - b_i| + \lambda y^td/6 = 2y^t(Ax - b) + 2\sum_{i\notin I} y_id_i\frac{|a_ix - b_i|}{d_i} + \lambda y^td/6 \\
&\leq 2y^t(Ax - b) + 2\rho\sum_{i\notin I} y_id_i + \lambda y^td/6 \leq 2y^t(Ax - b) + (1/2)\lambda y^td.
\end{aligned}$$

Hence, condition $\hat{\mathcal{G}}1$ implies that $\lambda y^td \leq 4y^t(Ax - b)$.

Next we show that our choice of α implies that $\hat{\mathcal{G}}1$ is satisfied. Consider any row i such that $\lambda d_i > 2(a_ix - b_i)$. By the definition of y_i, we have that $y_id_i < e^{\alpha\lambda/2}$. Also, by the minimality property of λ, $y^td \geq e^{\alpha\lambda}$. Therefore,

$$\frac{y_id_i}{y^td} < e^{-\alpha\lambda/2} \leq \frac{\lambda}{6m\rho}. \quad \blacksquare$$

We prove next that for an appropriate choice of σ, updating the solution significantly reduces the potential function $\Phi = y^td = \sum_i e^{\alpha(a_ix-b_i)/d_i}$.

Lemma 4.3 Consider a point $x \in P$ and an error parameter ϵ, $0 < \epsilon < 1$, such that (x,λ) and its corresponding dual solution y have potential function value Φ and do not satisfy $\mathcal{G}2$. Let $\tilde{x} \in P$ attain the minimum $C_{\mathcal{G}}(y)$. Assume that $\sigma \leq \frac{\lambda}{24\alpha\rho^2}$. Define a new solution $\hat{x} = (1-\sigma)x + \sigma\tilde{x}$, and let $\hat{\Phi}$ denote the potential function value for $\hat{x}$ and its corresponding dual solution $\hat{y}$. Then $\Phi - \hat{\Phi} = \Omega(\alpha\sigma\lambda\Phi)$.

Proof: By the definition of ρ, we have that $\lambda \leq \rho$, as well as $|a_ix - b_i|/d_i \leq \rho$ and $|a_i\tilde{x} - b_i|/d_i \leq \rho$, $i = 1,\ldots,m$. Hence, $\sigma \leq 1/(24\alpha\rho)$, and this implies that $\alpha\sigma|a_ix - a_i\tilde{x}|/d_i = \alpha\sigma|(a_ix - b_i) - (a_i\tilde{x} - b_i)|/d_i \leq 1/12$. Using the second-order Taylor theorem, we see that if $|\delta| \leq 1/2$, then, for all x, $e^{x+\delta} \leq e^x + \delta e^x + \delta^2e^x$. Setting $\delta = \alpha\sigma(a_i\tilde{x} - a_ix)/d_i$, we see that

$$\begin{aligned}
\hat{y}_i &\leq y_i + \frac{1}{d_i}\frac{\alpha\sigma(a_i\tilde{x} - a_ix)}{d_i}e^{\alpha(a_ix-b_i)/d_i} + \frac{1}{d_i}\frac{\alpha^2\sigma^2|a_i\tilde{x} - a_ix|^2}{d_i^2}e^{\alpha(a_ix-b_i)/d_i} \\
&\leq y_i + \alpha\sigma\frac{1}{d_i}((a_i\tilde{x} - b_i) - (a_ix - b_i))y_i + \frac{\alpha^2\sigma^2}{d_i^2}|(a_i\tilde{x} - b_i) - (a_ix - b_i)|^2y_i.
\end{aligned}$$

We use this inequality to bound the change in the potential function:

$$\begin{aligned}\Phi - \hat{\Phi} &= \sum_i (y_i - \hat{y}_i) d_i \\ &\geq \alpha\sigma \sum_i ((a_i x - b_i) - (a_i \tilde{x} - b_i)) y_i - \alpha^2\sigma^2 \sum_i \frac{|(a_i x - b_i) - (a_i \tilde{x} - b_i)|^2}{d_i^2} y_i d_i \\ &= \alpha\sigma(y^t(Ax - b) - C_{\mathcal{G}}(y)) - 4\alpha^2\sigma^2\rho^2 y^t d > \alpha\sigma(\lambda/5 - 4\alpha\sigma\rho^2)\Phi.\end{aligned}$$

Since $\sigma \leq \lambda/(24\alpha\rho^2)$, we see that the decrease in Φ is $\Omega(\lambda\alpha\sigma\Phi)$. ∎

During IMPROVE-GENERAL, σ is set equal to $\frac{\lambda_0}{24\alpha\rho^2}$, and so the decrease in the potential function due to a single iteration is $\Omega(\frac{\lambda_0^2}{\rho^2}\Phi)$. Observe that during a single call to IMPROVE-GENERAL we have $e^{\alpha\lambda_0/2} \leq \Phi \leq me^{\alpha\lambda_0}$. This, together with the previous lemma, can be used to bound the number of iterations in a single call to IMPROVE-GENERAL.

Theorem 4.4 The procedure IMPROVE-GENERAL terminates in $O(\rho^2\lambda_0^{-2}\log(m\rho\lambda_0^{-1}))$ iterations.

We use the procedure IMPROVE-GENERAL repeatedly to find an ϵ-approximate solution. We start by calling IMPROVE-GENERAL with any point in P and let (x, λ) denote its output. We check if $\lambda \leq \epsilon$, and if so output the ϵ-approximate solution x, and stop. If $\lambda > \epsilon$ and condition $\mathcal{G}2$ is satisfied, then we conclude that there does not exist a solution and stop. If neither termination condition is satisfied, then this process is repeated, where the new input to IMPROVE-GENERAL is its previous output. Note that if the output (x, λ) of any call to IMPROVE-GENERAL does not satisfy $\mathcal{G}2$, then the value λ must have decreased by at least a factor of 2. By observing that the number of iterations during the last call to IMPROVE-GENERAL dominates the total number of iterations, we obtain the following theorem.

Theorem 4.5 For any ϵ, $0 < \epsilon < 1$, repeated calls to IMPROVE-GENERAL yields an algorithm that finds an ϵ-approximate solution to the GENERAL problem or proves that no exact solution exists; the algorithm uses $O(\rho^2\epsilon^{-2}\log(m\rho\epsilon^{-1}))$ calls to the subroutine (10) for P and A, plus the time to compute Ax for the current iterate x between consecutive calls.

Randomized Version. As was true for the fraction packing and covering problems, we can use randomization to speed up the algorithm for the general problem if the polytope is in product form. Suppose that $P = P^1 \times \cdots \times P^k$ and the inequalities, when written to reflect this structure, are $\sum A^\ell x^\ell \leq b$. Let ρ^ℓ be the width of P^ℓ relative to $A^\ell x^\ell \leq b$ and d, $\ell = 1, \ldots, k$. If $\rho^1 = \rho^2 = \cdots = \rho^\ell = \hat{\rho}$, then we can speed up the algorithm by roughly a factor of k. The idea of the improved version is analogous to the randomized versions of the packing and covering algorithms.

A subroutine to compute $C_{\mathcal{G}}(y)$ for P consists of calls to k subroutines: subroutine (10) for P^ℓ and A^ℓ, $\ell = 1, \ldots, k$. In each iteration, the modified IMPROVE-GENERAL randomly picks

an index $s \in \{1, \ldots, k\}$, calls subroutine (10) for P^s and A^s, and uses the solution computed by this call to compute a tentative update for x^s, whereas all other components of the current iterate x are unchanged. We update x^s only if this results in a decrease in Φ. Since $\mathcal{G}2$ is not computed by this subroutine call, in each iteration, with probability $1/k$, the algorithm does the additional work needed to check if $\mathcal{G}2$ is satisfied.

The key to the improved version of our algorithm is the following lemma, which is analogous to Lemmas 2.6 and 3.8.

Lemma 4.6 Consider a point $(x^1, \ldots, x^k) \in P^1 \times \ldots \times P^k$ with corresponding dual solution y and potential function value Φ, and an error parameter ϵ, $0 < \epsilon < 1$. Let $\tilde{x}^s$ be a point in P^s that minimizes the cost $c^s x^s$, where $c^s = y^t A^s$, $s = 1, \ldots, k$, and assume that $\sigma^s \leq \min\{\lambda/(24\alpha\rho^s\rho), 1\}$. Let the new solution be defined by changing only x^s where $x^s \leftarrow (1 - \sigma^s)x^s + \sigma^s\tilde{x}^s$. If $\hat{\Phi}$ denotes the potential function value of the new solution, then $\Phi - \hat{\Phi} = \Omega(\alpha\sigma^s(y^t A^s x^s - y^t A^s \tilde{x}^s) - 4\alpha^2(\sigma^s)^2(\rho^s)^2 y^t d)$.

Since the algorithm updates x only when Φ decreases, the change in Φ associated with updating x_s is $\alpha\sigma^s\Delta_s$, where $\Delta_s = \max\{(y^t A^s x^s - y^t A^s \tilde{x}^s) - 4\alpha\sigma^s(\rho^s)^2 y^t d, 0\}$. We have restricted $\sigma^s \leq 1$ to ensure that the new point is in P; to get the maximum improvement, the algorithm sets $\sigma^s = \min\{\lambda/(24\alpha\rho^s\rho), 1\}$. Let $S = \{s : 24\alpha\rho^s\rho \leq \lambda\}$ and $\rho' = \sum_{s \notin S} \rho^s$. The probability $\beta(s)$ with which we pick an index s is defined as follows:

$$\beta(s) = \begin{cases} \frac{\rho^s}{2\rho'} & \text{for } s \notin S \\ \frac{1}{2|S|} & \text{for } s \in S \end{cases}$$

The following theorem is analogous to Theorems 2.7 and 3.9.

Theorem 4.7 For $0 < \epsilon < 1$, repeated calls the randomized version of IMPROVE-GENERAL can be used so that the algorithm either finds an ϵ-optimal solution for the general problem defined by the polytope $P = P^1 \times \cdots \times P^k$ and inequalities $\sum_\ell A^\ell x^\ell \leq b$ and tolerance vector d, or else proves that no exact solution exists; it is expected to use a total of $O(\rho^2\epsilon^{-2}\log(\rho m\epsilon^{-1}) + k\log(\rho\epsilon^{-1}))$ calls to any of the subroutines (10) for P^ℓ and A^ℓ, $\ell = 1, \ldots, k$, plus the time to compute $\sum_\ell A^\ell x^\ell$ between consecutive calls.

Proof: The proof of this theorem is analogous to the proofs of the randomized algorithm analyzed above. We first consider one call to IMPROVE-GENERAL. Once again, there are two types of iterations: those where $\mathcal{G}2$ is satisfied, and those where it isn't. To bound the expected number of the first type, we note that we terminate after each such iteration with probability $1/k$, and hence we expect that $O(k)$ suffice. For the second type, we first give a lower bound on the expected decrease of the potential function Φ. Recall that $\alpha\sigma^s\Delta_s$ is the decrease in Φ associated with updating x_s. Furthermore, since $\sigma^s \leq \lambda/(24\alpha\rho^s\rho)$, $\sum \alpha\sigma^s(\rho^s)^2 \leq \lambda/24$. Applying this along with the fact that $\mathcal{G}2$ is not satisfied, we have that $\sum_s \Delta_s \geq \lambda\Phi/5 -$

$4\lambda\Phi/24 = \lambda\Phi/30$. Hence, we get that the expected decrease in Φ is

$$\begin{aligned}\sum_s \alpha\sigma^s\Delta_s\beta(s) &= \sum_{s\notin S}\alpha\frac{\lambda}{24\alpha\rho^s\rho}\cdot\frac{\rho^s}{2\rho'}\Delta_s + \sum_{s\in S}\frac{\alpha}{2|S|}\Delta_s \\ &\geq \min\left\{\frac{\lambda}{48\rho^2},\frac{\alpha}{2k}\right\}\sum_{s=1}^{k}\Delta_s \geq \min\left\{\frac{\lambda}{48\rho^2},\frac{\alpha}{2k}\right\}\frac{\lambda}{30}\Phi.\end{aligned}$$

Once again, we use the result of Karp [14] to analyze the running time. The above bound on the expected decrease in Φ implies that the randomized version of IMPROVE-GENERAL is expected to terminate after $O(\rho^2\lambda_0^{-2}\log(\rho m\lambda_0^{-1}) + k)$ iterations. The number of times we call the modified IMPROVE-GENERAL is bounded by $O(\log(\rho\epsilon^{-1}))$. The overall time bound follows from the fact that λ_0 decreases by a factor of 2 each time we invoke the procedure, as well as the fact that the routine to check $\mathcal{G}2$ is expected to be called in a $O(1/k)$ fraction of the iterations. ∎

5 Decreasing the width ρ

The running times of our algorithms are proportional to the width ρ. In this section we present techniques that transform the original problem into an equivalent one, while reducing the width. Each of the techniques assumes the existence of a particular fast subroutine related to optimization over P; different subroutines might be available in different applications.

Relaxation of Integer Programming. In some cases, the primary interest in solving a particular fractional packing problem is to obtain a lower bound on an integer program of the form: minimize cx subject to $A'x \leq b'$ and $x \in P$, where the constraints constitute an integer packing problem and $c \geq 0$. As a result, we wish to decide if there exists a fractional solution $x \in P$ that satisfies packing constraints $Ax \leq b$, that consist of $A'x \leq b'$ and $cx \leq b_0$, for some value b_0. We shall give a technique to reduce the width of fractional packing problems that arise in this way.

The assumptions that $A \geq 0$ and P is in the nonnegative orthant imply that any integer solution must satisfy $x_j = 0$ whenever there exists an index i such that $a_{ij} > b_i$. Hence, instead of using P, we can tighten the fractional relaxation, and use $\bar{P} = \{x : x \in P,\ x_j = 0 \text{ if } j \in J\}$, where $J = \{j : \exists i \text{ such that } a_{ij} > b_i\}$. The width $\bar{\rho}$ of $\bar{P}$ relative to $Ax \leq b$ is bounded by $\zeta = \max_{x\in P}\sum_j x_j$. For example, if the variables of the integer program are restricted to be 0 or 1, we get $\zeta \leq n$.

Theorem 5.1 For any ϵ, $0 < \epsilon \leq 1$, there is an ϵ-relaxed decision procedure to solve a fractional packing problem that is derived from an integer packing problem in $O(\epsilon^{-2}\zeta\log(m\epsilon^{-1}))$ calls to the subroutine that finds a minimum-cost point in the restricted polytope $\bar{P}$, plus the time to compute the value of the left-hand side of the packing constraints for the current iterate after each update. If $x \leq 1$ for all $x \in P$, then $\zeta \leq n$.

If P is a direct product of convex sets, then so is $\bar{P}$, and hence the same technique can be applied for speeding up the randomized version of the packing algorithm as well.

Restricting P^ℓ. The next technique can be applied for some packing problems where P is a product of convex sets in smaller dimension. The idea is to define the same packing problem using a different polytope that has a smaller width. This technique can be applied for multicommodity flow problems (to obtain the formulation used in [18]) and for the preemptive machine scheduling problem, which will be discussed in the next section.

Consider a packing problem defined by the convex set $P = P^1 \times \cdots \times P^k$, and the inequalities $\sum_\ell A^\ell x^\ell \leq b$. It is easy to see that the convex set $\hat{P} = \hat{P}^1 \times \cdots \times \hat{P}^k$, where $\hat{P}^\ell = \{x^\ell \in P^\ell : A^\ell x^\ell \leq b\}$, $\ell = 1, \ldots, k$, and the same inequalities define the same fractional packing problem, and has $\rho \leq k$. It is possible that one of the polytopes, $\hat{P}^\ell$, $\ell = 1, \ldots, k$, is empty, and if so, the optimization routine for $\hat{P}^\ell$ will detect this, and thereby prove that there does not exist an exact solution to the original problem.

Theorem 5.2 For any ϵ, $0 < \epsilon \leq 1$, there is an ϵ-relaxed decision procedure to solve a fractional packing problem defined by $P = P^1 \times \cdots \times P^k$ and $\sum_\ell A^\ell x^\ell \leq b$ that is expected to use a total of $O(\epsilon^{-2} k \log(m\epsilon^{-1}) + k \log k)$ calls to a subroutine that finds a minimum-cost point in $\hat{P}^\ell$, $\ell = 1, \ldots, k$, and a deterministic version that uses $O(\epsilon^{-2} k^2 \log(m\epsilon^{-1}))$ such calls, plus the time to compute the value of the left-hand side of the packing constraints for the current iterate after each update.

Recall that the multicommodity flow problem can be defined with P^ℓ being the dominant of the convex combinations of all paths from the source of commodity ℓ to its sink. In this case, optimization over P^ℓ is a shortest path computation, but the parameter ρ defined by the problem can be arbitrarily high. The above technique imposes capacity constraints on the flows of individual commodities (since a flow is a convex combinations of paths). The resulting equivalent formulation has $\hat{\rho} \leq k$, but the required subroutine is the more time consuming minimum-cost flow computation.

Decomposition for Packing Problems. Consider the packing problem defined by a polytope P and inequalities $Ax \leq b$. We introduce a decomposition technique that defines a related problem with decreased width by replacing P and A by an equivalent problem in the product form. This decomposition can be used in cases where P is a polytope and we are given a subroutine that is more sophisticated than an optimization routine for P; the details of this routine will be given below. This technique will be used to solve the minimum-cost multicommodity flow problem in the next section.

For simplicity of presentation, we shall initially work with fractional packing problems where the polytope is a simplex. This is, in fact, without loss of generality, since each packing problem is equivalent to a problem in this form. To see this, let $v_1, \ldots, v_s$ denote a list of the vertices of P. Each point $x \in P$ can be written as a convex combination of the vertices of P: $x = \sum_j \xi_j v_j$, where $\sum_j \xi_j = 1$ and $\xi_j \geq 0$, $j = 1, \ldots, s$. If we let ξ_j, $j = 1, \ldots, s$, be the variables of the transformed problem, then this yields a problem in which the polytope is a simplex, possibly

with exponentially many variables; the packing constraints are now represented as $H\xi \leq b$, where $H = (h_{ij})$ and $h_{ij} = a_i v_j$. Observe that this change of coordinates does not change the width. In order to apply the packing algorithm to the transformed problem, we need a subroutine that finds j such that the jth coordinate of the vector $y^t H$ is minimum. Substituting the definition of H, this means that we need a subroutine that finds a vertex v_j of P that has minimum cost cv_j where $c = y^t A$.

Now we show how to obtain an equivalent problem for which the width is roughly half of its original value ρ. In order to facilitate recursion, we will assume that the simplex is defined by $S = \{(\xi_1, \xi_2, \ldots, \xi_s) : \sum \xi_j = d, \xi_j \geq 0, j = 1, \ldots, s\}$ for some d. We introduce two copies of each variable ξ_j: ξ_j^1 and ξ_j'. If we let $J^1 = \{j : \exists i \text{ such that } h_{ij} d > 2mb_i\}$, then the new polytope is $S^1 \times S'$, where $S' = (1/2)S = \{\xi : \sum_j \xi_j = d/2, \xi \geq 0\}$ and $S^1 = \{\xi^1 : \xi^1 \in S', \xi_j^1 = 0 \text{ if } j \in J^1\}$. The new system of packing inequalities is $H\xi^1 + H\xi' \leq b$. Note that the width of S' relative to $H\xi' \leq b$ is $\rho/2$; for any $\xi^1 \in S^1$,

$$\sum_j h_{ij}\xi_j^1 \leq \sum_{j \notin J^1} (2mb_i/d)\xi_j^1 \leq (2mb_i/d) \sum_j \xi_j^1 \leq mb_i, \quad i = 1, \ldots, m,$$

and hence the width of S^1 relative to $H\xi^1 \leq b$ is at most m.

If we apply the same transformation to S', after $k = \lceil \log \rho \rceil$ applications we obtain a fractional packing problem where the polytope is a product of $k+1$ polytopes, $S^1 \times \cdots \times S^k \times S'$, and a set of inequalities of the form $\sum_\ell H\xi^\ell + H\xi' \leq b$, where $S' = 2^{-k}S$, and $S^\ell = \{\xi^\ell : \xi^\ell \in 2^{-\ell}S, \xi_j^\ell = 0 \text{ if } j \in J^\ell\}$ where $J^\ell = \{j : \exists i \text{ such that } h_{ij} d > 2^\ell mb_i\}$. Since, for any $\xi^\ell \in S^\ell$,

$$\sum_j h_{ij}\xi_j^\ell \leq \sum_{j \notin J^\ell} (2^\ell mb_i/d)\xi_j^\ell \leq (2^\ell mb_i/d) \sum_j \xi_j^\ell \leq mb_i, \quad i = 1, \ldots, m,$$

the width of S^ℓ relative to $H\xi^\ell \leq b$ is at most m, $\ell = 1, \ldots, k$. Furthermore, the following lemma implies that the new problem is equivalent to the original one.

Lemma 5.3 If the fractional packing problem defined by S and $H\xi \leq b$ has an exact solution, then so does the problem defined by $S^1 \times \cdots \times S^k \times S'$ and $\sum_\ell H\xi^\ell + H\xi' \leq b$. For any $\epsilon > 0$, any ϵ-approximate solution to the latter problem can be used to find an ϵ-approximate solution to the former.

Proof: We first note that if $(\xi^1, \ldots, \xi^k, \xi')$ is an ϵ-approximate solution to the transformed problem, then $\xi = \sum_\ell \xi^\ell + \xi'$ is an ϵ-approximate solution to the original problem.

Now assume that we have an exact solution ξ to the original problem. We will show that this implies the existence of an exact solution of the transformed problem, and in fact, give an algorithm to construct such a solution, given the solution for the original problem. Initially, set $\xi' = \xi$ and $\xi^\ell = 0$, $\ell = 1, \ldots, k$. The algorithm consists of k phases. In phase ℓ, $\ell = 1, \ldots, k$, $\sum_j \xi_j'$ decreases by $d/2^\ell$, and $\sum_j \xi_j^\ell$ increases by the same amount. All other variables are unchanged. Hence, the resulting solution $(\xi^1, \xi^2, \ldots, \xi^k, \xi')$ is such that $\xi^\ell \in 2^{-\ell}S$, $\ell = 1, \ldots, k$, and $\xi' \in 2^{-k}S = S'$. We will perform each phase to ensure that, in fact, each $\xi^\ell \in S^\ell$, $\ell = 1, \ldots, k$.

In phase ℓ, $\ell = 1, \ldots, k$, while $\sum_j \xi'_j > d/2^\ell$, find $j \notin J^\ell$ with $\xi'_j > 0$, and simultaneously increase ξ^ℓ_j and decrease ξ'_ℓ by the same amount. This maintains that $\sum_\ell H\xi^\ell + H\xi' \leq b$. If we continue until $\sum_j \xi'_j = d/2^\ell$, this ensures that the resulting solution $\xi^\ell \in S^\ell$. We claim that we can always continue while $\sum_j \xi'_j > d/2^\ell$. Assume, for a contradiction, that for the current solution $(\xi^1, \ldots, \xi^k, \xi')$, we have $\sum_j \xi'_j > d/2^\ell$, and $j \in J^\ell$ whenever $\xi'_j > 0$. Hence, for each j such that $\xi'_j > 0$, we have a row $i(j)$ that contains a large coefficient in the jth column: $h_{i(j)j} > 2^\ell m b_{i(j)}/d$. Furthermore, $d/2^\ell < \sum_j \xi'_j = \sum_i \sum_{j:i(j)=i} \xi'_j$. Choose i such that $\sum_{j:i(j)=i} \xi'_j > d/(2^\ell m)$, and consider row i of $\sum_s H\xi^s + H\xi' \leq b$. We get that

$$\sum_s \sum_j h_{ij}\xi^s_j + \sum_j h_{ij}\xi'_j \geq \sum_{j:i(j)=i} h_{ij}\xi'_j > \frac{2^\ell m b_i}{d} \sum_{j:i(j)=i} \xi'_j > b_i,$$

which contradicts $\sum_s H\xi^s + H\xi' \leq b$. ∎

Having obtained this decomposition, we would like to express these polytopes and these constraints in terms of the original coordinates; we also would like to express the optimization subroutine (1) for S^ℓ and H in terms of a different, more intricate, subroutine for P and A. For the former, we can restate the decomposed problem as $P^1 \times \cdots \times P^k \times P'$ subject to the system of inequalities $\sum_\ell Ax^\ell + Ax' \leq b$, where $P' = 2^{-k}P$ and $P^\ell = \{x^\ell : x^\ell = \sum_j \xi^\ell_j v_j, \xi^\ell \in S^\ell\}$, $\ell = 1, \ldots, k$. In other words, a point in P^ℓ is $2^{-\ell}$ times a convex combination of vertices v_j of P, each of which satisfies $Av_j \leq 2^\ell mb$. Thus, in these terms, the basic decomposition lemma can be restated in the following way.

Lemma 5.4 **If the fractional packing problem defined by P and $Ax \leq b$ has an exact solution, then so does the transformed problem defined by $P^1 \times \ldots \times P^k \times P'$ and $\sum_\ell Ax^\ell + Ax' \leq b$, where $k \leq \lceil \log \rho \rceil$. For any $\epsilon > 0$, any ϵ-approximate solution to the latter problem can be used to find an ϵ-approximate solution to the former.**

In order to apply the packing algorithms in Theorem 2.5 and Theorem 2.7 to the transformed problem, we need subroutines that, given a dual solution y, find a point $x^\ell \in P^\ell$ that minimizes the cost $c^\ell x^\ell$, where $c^\ell = y^t A$, $\ell = 1, \ldots, k$. There is a vertex of P^ℓ that attains this minimum; the vertices of P^ℓ are those vertices v_j of P that satisfy $Av_j \leq 2^\ell mb$. Hence, it is sufficient to have the following subroutine with $\nu = 2^\ell m$:

Given a constant ν and a dual solution y, find a vertex $\tilde{x} \in P$ such that:

$$\begin{aligned} A\tilde{x} &\leq \nu b, \text{ and} \\ y^t A\tilde{x} &= \min(y^t Ax : x \text{ a vertex of } P \text{ s.t. } Ax \leq \nu b). \end{aligned} \tag{13}$$

Observe that there need not be a feasible solution to this further constrained optimization problem; if this is detected, however, then Lemma 5.4 implies that there does not exist an exact solution to the original packing problem. Since each polytope P^ℓ has width at most m with respect to $Ax \leq b$, we have the following theorem:

Theorem 5.5 For any ϵ, $0 < \epsilon \leq 1$, there is a randomized ϵ-relaxed decision procedure for the fractional packing problem that is expected to use $O(\epsilon^{-2} m \log \rho \log(m\epsilon^{-1}) + \log \rho \log\log \rho)$ calls to the subroutine (13), and a deterministic version that uses $O(\epsilon^{-2} m \log^2 \rho \log(m\epsilon^{-1}))$ calls, plus the time to compute the value of the left-hand side of the packing constraints for the current iterate after each update.

Observe that in order to obtain a decomposition where each subproblem has width at most m, it would suffice to take the above decomposition with $k = \log(\rho/m)$. This implies an improved version of the theorem with $\log \rho$ replaced by $\log(\rho/m)$. However, in our applications of this theorem, ρ will be large relative to m, and so that this improvement will not be relevant.

Subroutine (13) is, in some sense, similar to optimization over $\hat{P}$, which is required by Theorem 5.2, and was discussed in the previous subsection. However, if the packing problem is not in the product form, then optimization over $\hat{P}$ solves the original problem, whereas (13) does not. Instead of finding a minimum-cost point in P that satisfies $Ax \leq \nu b$, subroutine (13) finds a minimum-cost *vertex* of P that satisfies the same condition. Even if an instance is feasible, all vertices of P might violate $Ax \leq b$, and hence we cannot directly use subroutine (13) with $\nu = 1$ to solve the packing problem.

In the case of the minimum-cost multicommodity flow problem, the vertices of the polytopes defining the individual commodities are paths, and subroutine (13) for a commodity finds a shortest path in an appropriate subgraph of the original graph (induced by edges with relatively large capacity and relatively small cost). On the other hand, optimization over the polytope $\hat{P}^\ell$ used in Theorem 5.2 is a minimum-cost flow computation.

Taking advantage of the fact that we are only interested in approximate solutions, we can improve the previous theorem by replacing $\log \rho$ by $\log \epsilon^{-1}$. Consider the packing problem defined by the inequalities $\sum_\ell Ax^\ell \leq b$ and the convex set $P^1 \times \cdots \times P^k$ with $k = \lceil \log(3\epsilon^{-1}) \rceil$ and P^ℓ, $\ell = 1, \ldots, k$, as defined above.

Lemma 5.6 Let $0 < \epsilon \leq 1$. If the fractional packing problem defined by P and $Ax \leq b$ has an exact solution, then so does the transformed problem defined by $P^1 \times \cdots \times P^k$ and $\sum_\ell Ax^\ell \leq b$, where $k = \lceil \log 3\epsilon^{-1} \rceil$. An $\epsilon/3$-approximate solution to the transformed problem can be used to find an ϵ-approximate solution to the original problem.

Proof: By Lemma 5.4, if there is an exact solution to the original problem then there is an exact solution to $P^1 \times \cdots \times P^k \times P'$. By ignoring the component of this solution for P', we obtain an exact solution for the transformed problem of this lemma.

Now assume that we have an $\epsilon/3$-approximate solution $(x^1, \ldots, x^k)$ to the transformed problem. We claim that $x = \frac{1}{1-2^{-k}} \sum_\ell x^\ell$ is an ϵ-approximate solution for the transformed problem. Observe that $x \in P$ and

$$Ax = \frac{1}{1-2^{-k}} \sum_\ell Ax^\ell \leq \frac{1}{1-2^{-k}}(1+\frac{\epsilon}{3})b \leq (\frac{1}{1-\epsilon/3})(1+\frac{\epsilon}{3})b \leq (1+\epsilon)b. \quad \blacksquare$$

Theorem 5.7 For any ϵ, $0 < \epsilon \leq 1$, there exists a randomized ϵ-relaxed decision procedure for the fractional packing problem that is expected to use $O(\epsilon^{-2} m \log \epsilon^{-1} \log(m\epsilon^{-1}))$ calls to the subroutine (13), and by a deterministic version that uses $O(\epsilon^{-2} m \log^2 \epsilon^{-1} \log(m\epsilon^{-1}))$ calls, plus the time to compute the value of the left-hand side of the packing constraints for the current iterate after each update.

Subroutine (13) will not be available for the minimum-cost multicommodity flow problem. Instead, we will have the following relaxed subroutine, for some parameters $\gamma_i \geq 1$, $i = 1, \ldots, m$.

Given a constant ν and a dual solution y, find a vertex $\tilde{x} \in P$ such that:

$$\begin{aligned} A\tilde{x} &\leq \nu b, \text{ and} \\ y^t A\tilde{x} &\leq \min(y^t Ax : x \text{ a vertex of } P \text{ with } a_i x \leq \frac{\nu}{\gamma_i} b_i \quad \forall i). \end{aligned} \tag{14}$$

Subroutine (13) is the special case of (14) when $\gamma_i = 1$, $i = 1, \ldots, m$.

In order to use subroutine (14) in place of (13) and still obtain a similar time bound, we need a generalized decomposition instead of the one in Lemmas 5.4 or 5.7. For simplicity of presentation we shall again focus initially on a fractional packing problem given by a simplex $S = \{\xi : \sum_j \xi_j = d, \xi \geq 0\}$, and packing constraints $H\xi \leq b$.

Let $\Gamma = \sum_i \gamma_i$, $K^\ell = \{j : \exists i \text{ such that } h_{ij} d > 2^\ell (\Gamma/\gamma_i) b_i\}$, and $S^\ell = \{\xi : \xi \in 2^{-\ell} S,\ \xi_j^\ell = 0 \text{ if } j \in K^\ell\}$, $\ell = 1, \ldots, k$, and $S' = 2^{-k} S$. It is not hard to show that Lemma 5.3 still holds under these more general definitions. The proof of this lemma is trivially adapted: we merely replace J^ℓ by K^ℓ, and substitute Γ/γ_i for m; in particular, the choice of row i in the proof by contradiction is made so that it satisfies $\sum_{j:i(j)=i} \xi'_j > (d/2^\ell)(\gamma_i/\Gamma)$. The width of S^ℓ relative to $H\xi^\ell \leq b$ can now be bounded by Γ.

In terms of the original coordinates, this decomposition yields a polytope $Q = Q^1 \times \cdots \times Q^k \times Q'$, and a system of inequalities $\sum_\ell Ax^\ell + Ax' \leq b$, where $Q' = 2^{-k} P$, and $Q^\ell = \{x^\ell : x^\ell = \sum_j \xi_j^\ell v_j, \xi^\ell \in S^\ell\}$, $\ell = 1, \ldots, k$. In other words, a point in Q^ℓ is $2^{-\ell}$ times a convex combination of vertices v_j of P, each of which satisfies $a_i v_j \leq 2^\ell \Gamma b_i / \gamma_i$, $i = 1, \ldots, m$.

Lemma 5.8 If the fraction packing problem defined by P and $Ax \leq b$ has a solution, then so does the transformed problem defined by $Q^1 \times \cdots \times Q^k \times Q'$ and $\sum_\ell Ax^\ell + Ax' \leq b$. For any $\epsilon > 0$, an ϵ-approximate solution to the latter problem can be used to find an ϵ-approximate solution to the former.

If we were to apply Theorem 2.5 to solve the transformed problem, we need subroutines that find minimum-cost points in the polytopes Q^ℓ, $\ell = 1, \ldots, k$, where the cost vector $c = y^t A$. Instead, we will formulate a relaxed packing problem, and apply Theorem 2.9; to do this, we must first define the polytopes $\hat{Q}^\ell \supseteq Q^\ell$, $\ell = 1, \ldots, k$, on which this relaxation is based. Define $\hat{Q}^\ell$, $\ell = 1, \ldots, k$, to be the polytope formed by taking the convex hull of all vertices v_j of P that

satisfy $Av_j \leq 2^\ell \Gamma b$, and rescaling by $2^{-\ell}$; $\hat{Q} = \hat{Q}^1 \times \cdots \times \hat{Q}^k \times Q'$. Since $\gamma_i \geq 1$, $i = 1, \ldots, m$, it is clear that $Q^\ell \subseteq \hat{Q}^\ell$, $\ell = 1, \ldots, k$. Finally, note that subroutine (14) serves the role required by subroutine (6) for Theorem 2.9: it produces a vertex of $\hat{Q}^\ell$ with cost no more than the cost of a minimum-cost vertex in Q^ℓ. Hence, we can use Theorem 2.9 to produce a point $\hat{x} = (\hat{x}^1, \ldots, \hat{x}^k, x') \in \hat{Q}$ such that $\sum_\ell A^\ell \hat{x}^\ell + Ax' \leq (1+\epsilon)b$, or else to determine that there is no solution $x \in Q$ such that $\sum_\ell Ax^\ell + Ax' \leq b$. In the latter case, we can conclude that the original problem does not have an exact solution. Otherwise, since $\hat{Q}^\ell \subseteq 2^{-\ell} P$, $\ell = 1, \ldots, k$, and $Q' = 2^{-k} P$, if the algorithm returns a solution $\hat{x} \in \hat{Q}$, then $x = \sum_\ell \hat{x}^\ell + x'$ is in P, and is an ϵ-approximate solution to the original problem. Since the width of each $\hat{Q}^\ell$, $\ell = 1, \ldots, k$, is at most Γ, we obtain the following theorem.

Theorem 5.9 For any ϵ, $0 < \epsilon \leq 1$, there is a randomized ϵ-relaxed decision procedure for the fractional packing problem that is expected to use $O(\epsilon^{-2} \Gamma \log \rho \log(m\epsilon^{-1}) + \log \rho \log(\Gamma \log \rho))$ calls to the subroutine (14), and a deterministic version that uses $O(\epsilon^{-2} \Gamma \log^2 \rho \log(m\epsilon^{-1}))$ calls, plus the time to compute the value of the left-hand side of the packing constraints for the current iterate after each update.

An analogous result can be proved where the $\log \rho$ terms in this theorem are replaced by $\log \epsilon^{-1}$, by using the convex sets $Q^1 \times \cdots \times Q^k$ and $\hat{Q}^1 \times \cdots \times \hat{Q}^k$ with $k = \lceil \log(3\epsilon^{-1}) \rceil$ as suggested by Lemma 5.6.

Theorem 5.10 For any ϵ, $0 < \epsilon \leq 1$, there is a randomized ϵ-relaxed decision procedure for the fractional packing problem that is expected to use $O(\epsilon^{-2} \Gamma \log \epsilon^{-1} \log(m\epsilon^{-1}))$ calls to the subroutine (14), and a deterministic version that uses a factor of $\log \epsilon^{-1}$ more calls, plus the time to compute the value of the left-hand side of the packing constraints for the current iterate after each update.

Analogous results can be proved if the convex set P is already in product form, $P = P^1 \times \cdots \times P^k$. We use the decomposition technique to replace each set P^ℓ, $\ell = 1, \ldots, k$, by a product of $O(\log \epsilon^{-1})$ sets. Consequently, this approach assumes that the subroutine (14) is available for each P^ℓ and $A^\ell x^\ell \leq b$, $\ell = 1, \ldots, k$. We get the following theorem.

Theorem 5.11 For any ϵ, $0 < \epsilon \leq 1$, there is a randomized ϵ-relaxed decision procedure for the fractional packing problem defined by $P = P^1 \times \cdots \times P^k$ and $\sum_\ell A^\ell x^\ell \leq b$, that is expected to use $O(\epsilon^{-2} k \Gamma \log \epsilon^{-1} \log(m\epsilon^{-1}) + k \log k \log \epsilon^{-1})$ calls to the subroutine (14) for any of P^ℓ and $A^\ell x^\ell \leq b$, $\ell = 1, \ldots, k$, and a deterministic version that uses a total of $O(\epsilon^{-2} k^2 \Gamma \log^2 \epsilon^{-1} \log(m\epsilon^{-1}))$ such calls, plus the time to compute the value of the left-hand side of the packing constraints for the current iterate after each update.

Decomposition for Covering Problems. We present a decomposition technique for the covering problem, which is analogous to the technique used for the packing problem. The subroutine required for this approach is given by (15). This technique will be used by our algorithm for the cutting-stock problem, as described in Section 6.

Consider the covering problem defined by the polytope P and the inequalities $Ax \geq b$. For simplicity of presentation, we shall again assume that the problem was converted into the form $H\xi \geq b$, $\xi \in S$, where S is the simplex: $S = \{\xi : \sum_{j=1}^{s} \xi_j = d, \xi \geq 0\}$ for some d. As before, this reformulation does not change the width.

We first show how to obtain an equivalent problem for which the width is roughly half of its original value ρ. We introduce two copies of each ξ_j: ξ_j^1 and ξ_j'. The new polytope is $S^1 \times S'$, where $S' = S^1 = (1/2)S$. The new set of inequalities is $H^1\xi^1 + H\xi' \geq b$ where $H^1 = (h_{ij}^1)$ and h_{ij}^1 is h_{ij} if $h_{ij}d \leq 2mb_i$, and 0 otherwise. Note that the width of S' relative to $H\xi \geq b$ is $\rho' = \rho/2$, and, since $\sum_j h_{ij}^1 \xi_j^1 \leq \sum_j (2mb_i/d)\xi_j^1 \leq mb_i$, $i = 1, \ldots, m$, the width of S^1 relative to $H^1\xi \geq b$ is $\rho^1 \leq m$.

If we apply the same transformation to S', after $k = \lceil \log \rho \rceil$ applications we obtain an equivalent covering problem with a polytope which is a product of $k+1$ polytopes, $S^1 \times \cdots \times S^k \times S'$, and a system of inequalities of the form $\sum_\ell H^\ell \xi^\ell + H\xi' \geq b$,where $S' = 2^{-k}S$, $S^\ell = 2^{-\ell}S$, $H^\ell = (h_{ij}^\ell)$, and h_{ij}^ℓ is h_{ij} if $h_{ij}d \leq 2^\ell mb_i$, and 0 otherwise. For each of these subproblems, the width is at most m.

We shall give a slightly generalized version of the covering analog of Lemma 5.3. In this section, we shall use this lemma with $\lambda = 1$. The more general version with $\lambda < 1$ will be used in the cutting-stock application.

Lemma 5.12 Let $\lambda \leq 1$. If there exists $\xi \in S$ such that $H\xi \geq \lambda b$, then there exists $(\xi^1, \xi^2, \ldots, \xi^k, \xi') \in S^1 \times S^2 \times \cdots \times S'$ such that $\sum_\ell H^\ell \xi^\ell + H\xi' \geq \lambda b$. For any $\epsilon > 0$, an ϵ-approximate solution to the latter problem can be used to find an ϵ-approximate solution to the former.

Proof: We first note that if $(\xi^1, \xi^2, \ldots, \xi')$ is an ϵ-approximate solution to the transformed problem, then $\xi = \sum_\ell \xi^\ell + \xi'$ is an ϵ-approximate solution to the original problem.

Now assume that we have a solution $\xi \in S$ such that $H\xi \geq \lambda b$. We claim that any such ξ corresponds to an exact solution of the transformed problem, and we will give an algorithm that does this conversion. Initially, set $\xi' = \xi$ and $\xi^\ell = 0$, $\ell = 1, \ldots, k$. The algorithm consists of k phases. In phase ℓ, $\ell = 1, \ldots, k$, $\sum_j \xi_j'$ decreases by $d/2^\ell$, and $\sum_j \xi_j^\ell$ increases by the same amount. All other variables are unchanged. Hence, the resulting solution $(\xi^1, \xi^2, \ldots, \xi^k, \xi')$ is such that $\xi^\ell \in S^\ell$, $\ell = 1, \ldots, k$, and $\xi' \in 2^{-k}S = S'$.

We will perform each phase so that the covering constraints remain satisfied. In phase ℓ, $\ell = 1, \ldots, k$, while $\sum_j \xi_j' > d/2^\ell$, find j with $\xi_j' > 0$ such that for each i with $\sum_s h_i^s \xi^s + h_i \xi' = \lambda b_i$, we have that $h_{ij}^\ell = h_{ij}$. Simultaneously increase ξ_j^ℓ and decrease ξ_j' by the same amount, so that $\sum_s H^s \xi^s + H\xi' \geq \lambda b$ is maintained. We claim that we can always continue while $\sum_j \xi_j' > d/2^\ell$. Let $(\xi^1, \ldots, \xi^k, \xi')$ be the current solution, and assume, for a contradiction, that for each $\xi_j' > 0$, we can select a row $i(j)$ such that $h_{i(j)j}^\ell \neq h_{i(j)j}$ and $\sum_s h_{i(j)}^s \xi^s + h_{i(j)}\xi' = \lambda b_{i(j)}$. By the definition of the matrix H^ℓ, we have that $h_{i(j)j} > 2^\ell mb_{i(j)}/d$. Furthermore, since $d/2^\ell < \sum_j \xi_j' = \sum_i \sum_{j:i(j)=i} \xi_j'$, there exists an index i such that $\sum_{j:i(j)=i} \xi_j' > d/(2^\ell m)$. Since

$i = i(j)$ for some j,

$$\lambda b_i = \sum_s \sum_j h_{ij}\xi_j^s + \sum_j h_{ij}\xi_j' \geq \sum_{j:i(j)=i} h_{ij}\xi_j' > \frac{2^\ell m b_i}{d} \sum_{j:i(j)=i} \xi_j' > b_i.$$

This implies that $\lambda > 1$, which is a contradiction. ∎

In order to apply the covering algorithm in Theorem 3.7 or Theorem 3.9 to solve the transformed problem, we need a subroutine that finds a vertex v_j of $(2^{-\ell})P$ such that $\sum_i y_i h_{ij}^\ell$ is maximum. By the definition of H, $h_{ij}^\ell = a_i v_j$ if $a_i v_j \leq 2^\ell m b_i$, and 0 otherwise, and hence the required subroutine is as follows:

Given a constant ν and a dual solution y, find a vertex $\tilde{x} \in P$ such that:

$$\sum_{i \in I(\nu,\tilde{x})} y_i a_i \tilde{x} = \max\Big(\sum_{i \in I(\nu,x)} y_i a_i x : x \text{ a vertex of } P\Big), \tag{15}$$

where $I(\nu, x) = \{i : a_i x \leq \nu b_i\}$.

Theorem 5.13 For any ϵ, $0 < \epsilon \leq 1$, there is a randomized ϵ-relaxed decision procedure for the fractional covering problem that is expected to use $O(m \log \rho\ (\log^2 m + \epsilon^{-2} \log(m\epsilon^{-1})))$ calls to the subroutine (15), and a deterministic version that uses a factor of $\log \rho$ more calls, plus the time to compute the value of the left-hand side of the covering constraints for the current iterate after each update.

Analogous results can be proved if the convex set P is in product form, $P = P^1 \times \cdots \times P^k$. Assuming that the subroutine (15) is available for each P^ℓ and $A^\ell x \geq b$, $\ell = 1, \ldots, k$, we use the same technique to replace the set P^ℓ by a product of $1 + \lceil \log \rho \rceil$ sets. We get the following theorem.

Theorem 5.14 For any ϵ, $0 < \epsilon \leq 1$, there is a randomized ϵ-relaxed decision procedure for the fractional covering problem defined by $P = P^1 \times \cdots \times P^k$ and $\sum_\ell A^\ell x^\ell \geq b$ that is expected to use a total of $O(mk \log \rho\ (\log^2 m + \epsilon^{-2} \log(m\epsilon^{-1})))$ calls to any of the subroutines (15) for P^ℓ and $A^\ell \geq b$, $\ell = 1, \ldots, k$ and a deterministic version that uses a factor of $k \log \rho$ more calls, plus the time to compute the value of the left-hand side of the covering constraints for the current iterate after each update.

Decomposition for Simultaneous Packing and Covering. A combination of the techniques used to derive Theorems 5.5 and 5.13 can be used to obtain an analogous version for problems with simultaneous packing and covering constraints. The subroutine that we will use is given by (16).

For simplicity of presentation, we shall again reformulate the problem so that the polytope is the simplex: $S = \{\xi : \sum_j \xi_j = d, \xi \geq 0\}$ for some d. Let $H\xi \leq b$ and $\hat{H}\xi \geq \hat{b}$ denote the packing and covering constraints in the converted form.

The polytope for the equivalent problem constructed by the decomposition technique is a product of $k+1$ simplices $S^1 \times \cdots \times S^k \times S'$ where $k = \lceil \log \rho \rceil$. If we let $J^\ell = \{j : \exists i \text{ s.t. } h_{ij}d > 2^\ell m b_i\}$, then the simplices are $S' = 2^{-k}S$ and $S^\ell = \{\xi^\ell : \xi^\ell \in 2^{-\ell}S,\ \xi^\ell_j = 0 \text{ if } j \in J^\ell\}$, $\ell = 1, \ldots, s$. The packing and covering constraints are $\sum_\ell H\xi^\ell + H\xi' \leq b$ and $\sum_\ell \hat{H}^\ell\xi^\ell + \hat{H}\xi' \geq \hat{b}$, where $\hat{H}^\ell = (\hat{h}^\ell_{ij})$ and $\hat{h}^\ell_{ij}$ is $\hat{h}_{ij}$ if $\hat{h}_{ij}d \leq 2^\ell m\hat{b}_i$, and 0 otherwise. The proof that this is an equivalent formulation is nearly identical to the separate proofs of the decomposition for packing and for covering. The only difference is that in the proof by contradiction, for each $\xi'_j > 0$, we identify either a packing inequality i such that $h_{ij}d > 2^\ell m b_i$, or else we identify a tight covering inequality i such that $\hat{h}_{ij}d > 2^\ell m\hat{b}_i$; by averaging over *all* inequalities, we identify one that provides the contradiction. The width of each subproblem resulting from the decomposition is at most m. The optimization subroutine over S^ℓ, required for our algorithm, is as follows: among thoses indices j such that $h_{ij}d \leq 2^\ell m b_i$, $i = 1, \ldots, m - \hat{m}$, find one that minimizes $\sum_i y_i(h_{ij} - \hat{h}^\ell_{ij})$. Converting back to the original coordinates x, the required subroutine is the following, with $\nu = 2^\ell m$:

Given a constant ν and a dual solution $(y, \hat{y})$, find a vertex $\tilde{x} \in P$ such that:

$$\begin{aligned} A\tilde{x} &\leq \nu b, \text{ and} \\ y^t A\tilde{x} - \sum_{i \in I(\nu,\tilde{x})} \hat{y}_i \hat{a}_i \tilde{x} &= \min(y^t Ax - \sum_{i \in I(\nu,x)} \hat{y}_i \hat{a}_i x : x \text{ a vertex of } P \text{ such that } Ax \leq \nu b), \end{aligned} \tag{16}$$

where $I(\nu, x) = \{i : \hat{a}_i x \leq \nu b_i\}$.

Theorem 5.15 For any ϵ, $0 < \epsilon \leq 1$, there is a randomized ϵ-relaxed decision procedure for the general problem that is expected to use $O(m^2(\log^2 \rho)\epsilon^{-2} \log(\epsilon^{-1} m \log \rho))$ calls to the subroutine (16), and a deterministic version that uses a factor of $\log \rho$ more calls, plus the time to compute the value of the left-hand side of the covering constraints for the current iterate after each update.

An analogous results can be proved if the convex set P is in product form, $P = P^1 \times \cdots \times P^k$, assuming that the subroutine (16) is available for P^ℓ and the corresponding inequalities, for each $\ell = 1, \ldots, k$.

6 Applications

In this section, we will show how to apply the techniques presented in the previous four sections to a variety of linear programs related to packing and covering problems. For an optimization problem, an ϵ-approximation algorithm delivers a solution of value within a factor of $(1 + \epsilon)$ of optimal in polynomial time. Although we will focus on ϵ-approximation algorithms for fixed

ϵ, this is only to simplify the discussion of running times. In each of the applications except for the Held-Karp bound and the bin-packing problem, we obtain a significant speedup over previously known algorithms. When we cite bounds based on Vaidya's algorithm [25] for the dual problem, then this algorithm is used in conjunction with the techniques of Karmarkar & Karp [13] to obtain a primal solution.

Scheduling unrelated parallel machines: with and without preemption. Suppose that there are N jobs and M machines, and each job must be scheduled on exactly one of the machines. For simplicity of notation, assume that $N \geq M$. Job j takes p_{ij} time units when processed by machine i. The *length* of a schedule is the maximum total processing time assigned to run on one machine; the objective is to minimize the schedule length. This problem, often denoted $R||C_{\max}$, is NP-complete, and in fact, Lenstra, Shmoys, & Tardos showed that there does not exist an ϵ-approximation algorithm with $\epsilon < 1/2$ unless $P = NP$. Lenstra, Shmoys, & Tardos [20] also gave a 1-approximation algorithm for it, based on a 1-relaxed decision procedure. If there exists a schedule of length T, then the following linear program has a feasible solution:

$$\sum_{j=1}^{N} p_{ij}x_{ij} \leq T, \quad i = 1, \ldots, M; \tag{17}$$

$$\sum_{i=1}^{M} x_{ij} = 1, \; j = 1, \ldots, N; \tag{18}$$

$$x_{ij} = 0 \text{ if } p_{ij} > T, \; i = 1, \ldots, M, \; j = 1, \ldots, N, \tag{19}$$

$$x_{ij} \geq 0 \text{ if } p_{ij} \leq T, \; i = 1, \ldots, M, \; j = 1, \ldots, N. \tag{20}$$

Lenstra, Shmoys, & Tardos showed that any vertex of this polytope can be rounded to a schedule of length $2T$. We shall call $x \geq 0$ an *assignment* if it satisfies (18). Let the *length* of an assignment x be the minimum value T such that it is a feasible solution to this linear program.

To apply Theorem 2.5, we let P be defined by the constraints (18–20). It is easy to see that $\rho \leq N$: for any $x \in P$, $x_{ij} > 0$ implies that $p_{ij} \leq T$, and so $\sum_{j=1}^{N} p_{ij}x_{ij} \leq NT$ for each machine $i = 1, \ldots, M$. Each dual variable y_i corresponds to one of the machine load constraints (17), and the coefficient of x_{ij} in the aggregated objective function y^tAx is y_ip_{ij}. Since $P = P^1 \times \cdots \times P^N$, where each P^j is a simplex, we can minimize this objective function by separately optimizing over each P^j. For a given P^j, this is done by computing the minimum modified processing time y_ip_{ij}, where the minimization is restricted to those machines for which $p_{ij} \leq T$. This approach is quite similar to the ascent method that Van de Velde [27] used to solve this linear program; he also used a Lagrangean method that, in each iteration, constructs a schedule by assigning each job to its fastest machine with respect to the modified processing times, but uses a much simpler rule to update the dual variables y.

Each iteration takes $O(MN)$ time and $\rho \leq N$. Hence, for any fixed $\epsilon > 0$ we can find an assignment $\bar{x}$ of length at most $(1 + \epsilon)T$ in $O(MN^2 \log M)$ time, if one of length T exists.

However, in order to apply the rounding procedure to produce a schedule, $\bar{x}$ must first be converted to a vertex of the polytope.

We can represent any assignment x as a weighted bipartite graph $G = (V_1, V_2, E)$, where V_1 and V_2 correspond to machines and jobs, respectively, and $(i,j) \in E$ if and only if $x_{ij} > 0$. If x is a vertex, then each connected component of the corresponding graph is either a tree or a tree plus one additional edge; we call such a graph a *pseudoforest.* The rounding procedure of Lenstra, Shmoys, & Tardos can be applied to any assignment represented by a pseudoforest, and takes $O(M+N)$ time. We will give a procedure which, given any assignment of length T represented by $G = (V_1, V_2, E)$, converts it in $O(|E|M)$ time into another assignment of length at most T that is represented by a forest. Since $|E| \leq MN$, the time to preprocess $\bar{x}$ for rounding is dominated by the time taken to find $\bar{x}$.

Lemma 6.1 Let $\bar{x}$ be an assignment represented by the graph $G = (V_1, V_2, E)$. Then $\bar{x}$ can be converted in $O(|E|M)$ time into another assignment $\hat{x}$ of no greater length, where $\hat{x}$ is represented by a forest.

Proof: To show that the assignment $\bar{x}$ can be easily converted to one represented by a forest without increasing its length, first consider the case when a connected component of G is a cycle. Let $e_1, \ldots, e_{2r}$ denote the edges of the cycle. It is always possible to obtain another assignment of the same length, either by increasing the coordinate of $\bar{x}$ for each edge e_{2i}, $i = 1, \ldots, r$ and decreasing those for e_{2i-1}, $i = 1, \ldots, r$, or vice versa. If e_i and e_{i+1} meet at a node in V_2 (a job node), then the perturbations have the same magnitude; if they meet at a node in V_1, the perturbations are linearly related based on machine load constraint for that machine node. By choosing the largest such perturbation for which the non-negativity constraints are satisfied, at least one of these coordinates of $\bar{x}$ is forced to 0, and so this connected component has been transformed into a forest.

To generalize this to a procedure that converts an arbitrary assignment into one represented by a forest, we perform a modified depth-first search of G: when a cycle is found by detecting a back edge, this perturbation is computed for that cycle, and the search is restarted; when the search detects that an edge does not belong to any cycle in G (because the search from one of its endpoints has been exhausted and is retracing its path towards the root) the edge is deleted to avoid repeatedly searching that part of the graph, and the coordinate of $\bar{x}$ for this edge is fixed to its current value.

Consider the time that it takes to find the next cycle, and divide it into two parts: time spent searching edges that are deleted due to the fact that they are not contained in any cycle, and time spent searching the cycle as well as the path from the root to the cycle in the depth-first search tree. Since the depth of tree is at most $2M$, the time spent for the latter in one search is $O(M)$; since at least one edge is deleted in each search (by the perturbation), the total time for this is $O(|E|M)$. On the other hand, the time spent searching edges that are deleted in this phase of the search is $O(1)$ per edge, and hence is clearly $O(|E|)$, in total for the algorithm. Finally, the time spent computing the correct perturbation is $O(M)$ per perturbation, and hence $O(|E|M)$ in total. ∎

By applying Theorem 2.5, we obtain a deterministic $(2+\epsilon)$-relaxed decision procedure for $R||C_{\max}$ that runs in $O(MN^2 \log M)$ time. Recall that $P = P^1 \times \cdots \times P^N$, and we can also take advantage of this structure using randomization. Observe that $\rho^j \leq 1$, $j = 1, \ldots, N$, and we can optimize over P^j in $O(M)$ time; we can also compute the updated values Ax in $O(M)$ time. Applying Theorem 2.7, we get a randomized algorithm that takes $O(N \log N)$ iterations, each of which takes $O(M)$ time. Furthermore, the solution $\bar{x}$ is expected to have $O(N \log N)$ positive components, since at most one is added at each iteration, and so it can be preprocessed for rounding in $O(MN \log N)$ time. This gives a randomized analogue that runs in $O(MN \log N)$ expected time. To convert either relaxed decision procedure into an approximation algorithm, we use bisection search to find the best length T. Since the schedule in which each job is assigned to the machine on which it runs fastest is within a factor of M of the optimum, $O(\log M)$ iterations of this search suffice.

Although it is most natural to formulate the linear program for $R||C_{\max}$ as a packing problem, a faster deterministic algorithm can be obtained by using a covering formulation. Let $\sum_i x_{ij} \geq 1$, $j = 1, \ldots, N$ be the covering constraints; let $P = P^1 \times P^2 \times \cdots \times P^M$, where

$$P^i = \{(x_{i1}, \ldots, x_{iN}) : \sum_j p_{ij} x_{ij} \leq T;\ x_{ij} = 0, \text{ if } p_{ij} > T, \text{ and } 0 \leq x_{ij} \leq 1, \text{ otherwise.}\}$$

In this case, optimizing over P^i is merely solving a fractional knapsack problem with N pieces, which can be solved in $O(N)$ time using a linear-time median finding algorithm. As a consequence, each iteration again takes $O(MN)$ time, but for this formulation, $\rho = M$. We will not apply Theorem 3.7 directly, but instead give a simple way to compute an initial solution with $\lambda = 1/M$. If $\hat{x}$ is the 0-1 solution in which each job is assigned to the machine on which it runs fastest, then $\hat{x}/M$ is such a solution: if $\hat{x} \notin P$ then there is no feasible solution, since this implies that the minimum total load of the jobs is greater than the machines' total capacity. As a result, the N calls (one per covering constraint) to the subroutine to optimize over P are not needed to construct an initial solution. Given an ϵ-optimal solution $\bar{x}$, it can be converted to a feasible solution to our original linear program by rescaling the variables for each job so that they sum to exactly 1; as a result, the machine load constraints will be satisfied with right-hand side set to $T/(1-\epsilon)$. By Lemma 6.1, this solution can be converted into one represented by a forest in $O(M^2 N)$ time.

Theorem 6.2 For any fixed $r > 1$, there is a deterministic r-approximation algorithm for $R||C_{\max}$ that runs in $O(M^2 N \log^2 N \log M)$ time, and a randomized analog than runs in $O(MN \log M \log N)$ expected time.

The fastest previously known algorithm for solving this problem is the FAT-PATH generalized flow algorithm of Goldberg, Plotkin, and Tardos [8]. In order to convert the packing problem defined by (17-20) into a generalized flow problem, we construct a bipartite graph with nodes representing jobs and machines and introduce an edge from machine node i to job node j with gain $1/p_{ij}$ if $p_{ij} \leq T$. There is a source which is connected to all the machine nodes with edges of gain $P_{\max} = \max\{p_{ij}\}$ and capacity T, and the job nodes are connected to a sink with edges of unit gain and unit capacity. A generalized flow in this network that results in an excess of

N at the source corresponds to a solution of the packing problem. On the other hand, if the maximum excess that can be generated at the source is below N, the original packing problem is infeasible, *i.e.*, the current value of T is too small.

The running time of the FAT-PATH algorithm given in [8] is $O(m^2n^2 \log n \log^2 B)$, where n, m, and B are the number of nodes, edges, and the maximum integer used to represent gains or capacities, respectively. In our case, we have $O(N)$ nodes, $O(MN)$ edges, and the maximum integer used to represent gains and capacities is bounded by $O(NP_{\max})$, where $P_{\max} = \max\{p_{ij}\}$. It is possible to show that the FAT-PATH algorithm is significantly faster for our specific case as compared to the general case. First, it is sufficient to compute an approximate solution. Also, the maximum length of the cycle in our graph is $O(M)$. Finally, in order to eliminate dependence on $P_{\max}$, we can round p_{ij} and T so that they will be represented by $O(\log N)$-bit integers. The running time of the resulting algorithm is $O((M^2N^2 + M^3N \log^2 N) \log N)$, which is worse than the running times of our deterministic and randomized algorithms by an $\Omega^*(N)$ and $\Omega^*(MN)$ factors, respectively.

In a related model, we consider schedules with preemptions: a job may be started on one machine, interrupted, and then continued later on another. Lawler & Labetoulle [17] showed that an optimal preemptive schedule for this problem, $R|pmtn|C_{\max}$, can be found by minimizing T subject to

$$\sum_{j=1}^{N} p_{ij}x_{ij} \leq T,\ i = 1, \ldots, M, \tag{21}$$

$$\sum_{i=1}^{M} p_{ij}x_{ij} \leq T,\ j = 1, \ldots, N, \tag{22}$$

$$\sum_{i=1}^{M} x_{ij} = 1,\ j = 1, \ldots, N, \tag{23}$$

$$x_{ij} \geq 0,\ i = 1, \ldots, M,\ j = 1, \ldots, N. \tag{24}$$

We can again use a weighted bipartite graph G to represent the *assignments* satisfying (23)-(24); the length of an assignment x is the minimum value T such that x and T satisfy (21)-(24). If the weights are represented as integers over a common denominator, then this can be viewed as compactly represented multigraph, where the numerator of the weight of each edge specifies its multiplicity. An optimal edge coloring of this multigraph for the optimal solution to (21)-(24) gives an optimal schedule, where the matching of jobs and machines given by each color class represents a fragment of the schedule. If we use relatively few distinct matchings, then we introduce few preemptions, and it can be shown that $O(N)$ matchings suffice [17].

In order to apply our relaxed decision procedure, we shall do a bisection search for the minimum value of T for which we find an ϵ-approximate solution. We will have deterministic and randomized variants for performing one iteration, which apply, respectively, Theorems 2.5 and 2.7 to the constraints of the linear program (21)-(24) for a particular target $\bar{T}$. As in the

previous packing formulation, the system $Ax \leq b$ is given by (21), and let

$$P^j = \{x^j : \sum_{i=1}^{M} p_{ij}x_{ij} \leq \bar{T},\ \sum_{i=1}^{M} x_{ij} = 1,\ x_{ij} \geq 0,\ i = 1, \ldots, M\}.$$

It is easy to see that $\rho^j \leq 1$. To optimize over P^j, note that this is the dual of a 2-variable linear program with M constraints, and, in fact, it is a fractional multiple-choice knapsack problem with M variables. Dyer [1] has shown that this problem can be solved in $O(M)$ time. For the deterministic version, when $P = P^1 \times \cdots \times P^N$, we have $\rho \leq N$. To optimize over P, we solve N disjoint multiple-choice knapsack problems, each with M variables, in $O(MN)$ time. Similarly, each iteration of the randomized variant can be implemented in $O(M)$ time.

Given an assignment $\bar{x}$ of length T represented by a weighted graph $G = (V_1, V_2, E)$, we must still compute a schedule. If we are interested in computing a schedule that completes in exactly time T, then it takes $O(|E|(|E| + M))$ time to compute such a schedule [17]. However, since $\bar{x}$ is itself only approximately optimal, there is little point to computing the best schedule corresponding to $\bar{x}$: we can more efficiently compute a somewhat longer schedule.

Given G, we compute rescaled values $\bar{p}_{ij} = (p_{ij}\bar{x}_{ij}) \cdot (Q/T)$, where the value of Q will be specified below. As a result, the rescaled total load on each machine and total processing time of each job is at most Q. Round these rescaled times by forming the multigraph $\bar{G}$, where each $ij \in E$ occurs with multiplicity $\lceil \bar{p}_{ij} \rceil$. Thus, the maximum degree Δ of this graph is at most $Q + N$. Using an algorithm of Gabow [4], we can color this graph in $O(M\Delta \log \Delta)$ time with $2^{\lceil \log_2 \Delta \rceil}$ colors. By choosing $Q = 2^l - N$, where $l = \lceil \log_2(N + N\epsilon^{-1}) \rceil$, it follows that $\Delta \leq 2^l = O(N/\epsilon)$. Each matching given by a color class corresponds to a fragment of the schedule of length T/Q. The total length of this schedule is at most

$$2^{\lceil \log_2 \Delta \rceil} T/Q \leq \frac{2^l T}{2^l - N} = T + \frac{NT}{2^l - N} \leq T + \frac{N}{N + N\epsilon^{-1} - N} T = (1 + \epsilon)T.$$

Theorem 6.3 For any constant $\epsilon > 0$, there are deterministic and randomized ϵ-approximation algorithms for $R|pmtn|C_{\max}$ that run in $O(MN^2 \log^2 M)$ time and $O(MN \log M \log N)$ expected time, respectively.

The previous best algorithm is obtained by using the linear programming algorithm of Vaidya [26]. Our running time marks an $\Omega^*(M^{2.5}N^{1.5})$ improvement over this algorithm for the deterministic algorithm, and an $\Omega^*((MN)^{2.5})$ improvement for the randomized algorithm.

Job shop scheduling. In the job shop scheduling problem, there are N jobs to be scheduled on a collection of M machines; each job j consists of a specified sequence of operations, $O_{1j}, O_{2j}, \ldots, O_{\mu j}$, where O_{ij} must be processed on a particular machine m_{ij} for p_{ij} time units without interruption; the operations of each job must be processed in the given order, and each machine can process at most one operation at a time; the aim is to schedule the jobs so as to minimize the time by which all jobs are completed. Let $p_{\max}$ denote $\max_{i,j} p_{ij}$, the maximum processing time of any operation, let $P_{\max}$ denote $\max_j \sum_i p_{ij}$, the maximum total processing

time of any job, and finally, let $\Pi_{\max}$ denote $\max_i \sum_{j,k:m_{kj}=i} p_{kj}$, the maximum total processing time assigned to a machine.

Shmoys, Stein, & Wein [24] give a randomized $O(\log^2(M+\mu))$-approximation algorithm for this problem and a deterministic variant that uses the randomized rounding technique of Raghavan & Thompson [22] and its deterministic analogue due to Raghavan [21]. The overwhelming computational bottleneck of the deterministic algorithm is the solution of a certain fractional packing problem.

The algorithms work by first performing a preprocessing phase that reduces the problem to the following special case in $O(M^2\mu^2N^2)$ time: $N = O(M^2\mu^3)$, $p_{\max} = O(N\mu)$, $\Pi_{\max} = O((N\mu)^2)$, and $P_{\max} = O(N\mu^2)$. We shall use $\tilde{N}$ to denote $\min\{N, M^2\mu^3\}$. For each job, the randomized algorithm selects, uniformly and independently, an initial delay in the range $\{1, 2, \ldots, T\}$, where $T = \Pi_{max}$. A straightforward counting argument proves that if each job is scheduled to be processed continuously with its first operation starting at the chosen delay, then, with high probability, this assignment has placed $O(\log(M+\mu))$ jobs on any machine, at any time. The remainder of the algorithm carefully slows down this attempted schedule by an $O(\log(M+\mu)\log p_{\max})$ factor in order to produce a schedule in which each machine is assigned to process at most one job at a time.

This algorithm can be made deterministic by formulating the problem of choosing initial delays so that each machine is always assigned $O(\log(M+\mu))$ jobs as an integer packing problem, and then applying the techniques of Raghavan & Thompson [22] and Raghavan [21] to approximately solve this packing problem. The computational bottleneck of this procedure is solving the fractional relaxation of the integer packing problem. The variables for the fractional packing problem are x_{jd}, for each job $j = 1, \ldots, N$ and each possible delay $d = 1, \ldots, T$; the polytope is $P = P^1 \times \cdots \times P^N$, where P^j is a T-dimensional unit simplex, where each vertex corresponds to a particular delay selected for job $j, = 1, \ldots, N$. There are $M(P_{\max} + T) = M(P_{\max} + \Pi_{\max}) = O(M\tilde{N}^2\mu^2)$ packing constraints: for each machine and each time unit, we wish to ensure that the particular selection of initial delays results in $O(\log(M+\mu))$ jobs on that machine at that time.

One way in which our results can be applied to this problem is to use our algorithm to solve the fractional packing problem, and then apply the algorithm of Raghavan to round this fractional solution to an integer one. However, we can obtain a simpler and more efficient solution by applying the integer packing algorithm of Theorem 2.11, which directly produces an integer solution of sufficient quality.

In the worst case, all N jobs can be assigned to the same machine at a particular time, and hence the parameters of the packing problem are $\bar{\rho} = O(1/\log(M+\mu))$ (since the right-hand sides are $O(\log(M+\mu))$), $\rho = N\bar{\rho}$, $k = N$, $m = O(M\tilde{N}^2\mu^2)$, and $d = 1$. Since the random selection of delays yields a feasible integer packing with high probability, $\lambda^* = O(1)$. By Theorem 2.11, we can find an integral solution to the above packing problem with $\lambda = O(1)$ in $O(N\log(M+\mu))$ iterations of this algorithm.

It remains to show how to implement a single iteration of the integral packing algorithm. Until the algorithm terminates, each $\sigma^j = 1$, $j = 1, \ldots, N$; any decrease in any σ^j causes the

algorithm to terminate. As a consequence, the algorithm maintains a solution $(x_1, \ldots, x_N)$ such that each x_j is a vertex of P^j, $j = 1, \ldots, N$. In each iteration, only one of these components is changed. This change involves only two variables: for one job j, its assigned delay is changed from one value to another. If we change one variable x_{jd}, then this affects at most $P_{\max}$ dual variables y_{it}, corresponding to the time units $t \in (d, d + P_{\max} - 1)$ since x_{jd} corresponds to processing job j starting at time d. This change in $P_{\max}$ dual variables affects the costs $c_{j'd'}$ for at most $2P_{\max}$ delays, $d' \in (d - P_{\max}, d + P_{\max} - 1)$, for each $j' = 1, \ldots, N$. However, given the updated cost $c_{j'd'}$ for a job j', we can update $c_{j'd'+1}$ in $O(\mu)$ time. Therefore, the time required to update all of the costs after the change in 2 primal variables is $O(\tilde{N}\mu P_{\max}) = O(\tilde{N}^2\mu^3)$. For each job j, $j = 1, \ldots, N$, we must select the delay of minimum cost, and then update the job for which this represents the maximum improvement. To efficiently select the minimum cost variable x_{jd}, for each $j = 1, \ldots, N$, we maintain a heap for each job j, $j = 1, \ldots, N$, which contains the costs c_{jd}, $d = 1, \ldots, T$.

Lemma 6.4 One iteration of the integer version of IMPROVE-PACKING can be implemented in $O(\tilde{N}^2\mu^3 \log(M + \mu))$ time.

Applying Theorem 2.11 and the results of Shmoys, Stein, and Wein, we get the following.

Theorem 6.5 A job shop schedule with maximum completion time that is a factor $O(\log^2(M+\mu))$ more than optimal can be found deterministically in $O(M^2\mu^2N^2 + \tilde{N}^3\mu^3 \log^2(M + \mu))$ time.

The fastest previously known algorithm is obtained by using the linear programming algorithm of Vaidya [26], which solves the fractional packing problem in $O(\tilde{N}^{10.5}\mu^7 \log(M + \mu))$ time. Then one can apply the techniques of Raghavan and Thompson [22] and Raghavan [21] to round to an integer solution. Our algorithm marks a very large improvement over this running time.

Network embeddings. Let $G = (V, E_G)$ and $H = (V, E_H)$ denote two constant-degree graphs on the same set of N nodes. We define the *flux* of G by $\alpha = \min\{\delta(S)/|S| : S \subset V, |S| \leq N/2\}$, where $\delta(S)$ denotes the number of edges in E_G leaving S, i.e., one endpoint is in S and the other is in $V - S$. An *embedding* of H in G is defined by specifying a path in G from i to j for each edge $ij \in E_H$. The *dilation* of the embedding is the maximum number of edges on one of the paths used, and the *congestion* is the maximum number of paths that contain the same edge in G. Leighton and Rao [19] gave an algorithm to embed H in G with dilation and congestion both $O(\frac{\log N}{\alpha})$. If H is an expander, and hence each subset S of at most $N/2$ nodes has $\Omega(|S|)$ edges leaving it in H, then every embedding of H in G must have congestion $\Omega(\frac{1}{\alpha})$.

The computational bottleneck of the Leighton-Rao algorithm is finding an approximately optimal dual solution to a certain fractional packing problem that directly corresponds to the problem of routing the edges of H. To search for an appropriate choice of L, we repeatedly double the candidate value; our algorithm that attempts to embed H into G with congestion and dilation L has running time proportional to L, and hence the time for the final value of L will dominate the total. Leighton and Rao [19] show that when $L = \Theta(\frac{\log N}{\alpha})$, then there exists

an embedding with dilation and congestion at most L. The variables of the packing problem are as follows: for each edge $ij \in E_H$, there is a variable for each path in G from i to j of length at most L. The polytope P is a product of simplices, one simplex for each edge $ij \in E_H$, which ensures that one path connecting i and j is selected. There is a packing constraint for each edge $ij \in E_G$, which ensures that ij is not contained in more than L of the paths selected. An integer solution to this packing problem is an embedding of H in G with congestion and dilation at most L.

We will apply Theorem 2.11 to obtain an approximately optimal integral solution. The width $\bar{\rho} = 1/L$ for this problem, $d = 1$; k is the number of edges in H, which is $O(N)$; and m is the number of edges in G, which is also $O(N)$. Leighton and Rao proved that if an appropriate $L = \Theta(\frac{\log N}{\alpha})$ is used then $\lambda^* = O(1)$. The assumption that G has bounded degree implies that $\alpha = O(1)$, and therefore λ' in the theorem is $O(1)$. Theorem 2.11 implies that an integral solution to the packing problem with $\lambda = O(1)$ can be obtained in $O(N \log N)$ iterations of the packing algorithm.

To implement one iteration, we need the following subroutine. Given nonnegative costs on the edges of G, and an edge $ij \in E_H$, we must select a minimum-cost path from i to j consisting of at most L edges. In the randomized version of IMPROVE-PACKING, we pick an edge of H at random; in the deterministic version, such a path has to be selected for each edge of H. A minimum-cost path from i to j consisting of at most L edges, can be found in $O(NL)$ time using dynamic programming, since G has $O(N)$ edges.

Theorem 6.6 For any two constant-degree graphs, H and G, on the same set of nodes, an embedding of the edges of H into the edges of G with congestion and dilation $O(\frac{\log N}{\alpha})$ can be found by a randomized algorithm in $O(N^2 \frac{1}{\alpha} \log^2 N)$ expected time, or in a factor of N more time deterministically.

This running time marks a major improvement over the best previously known time that is obtained by using Vaidya's algorithm [25]. To obtain the dual solution, Vaidya's algorithm has $O(N^2 \log N)$ iterations, each of which consists of inverting an $O(N)$ by $O(N)$ matrix, plus $O(N^2 L)$ time for the deterministic version of the above subroutine; since $L = O(\frac{\log N}{\alpha})$, the total time is $O(N^2 \log N(N^2 \frac{\log N}{\alpha} + \mathcal{M}(N)))$. When used in conjunction with the techniques of Karmarkar & Karp to produce a primal solution, this appears to yield an algorithm that runs in $O(N^7 \log^3 N)$ time, and a randomized analog that is a factor of N faster.

The Held-Karp bound for the TSP with triangle inequality. One of the most useful ways to obtain a lower bound on the length of the optimum tour for the traveling salesman problem was proposed by Held & Karp [11], and is based on the idea of Lagrangean relaxation. We shall assume that an instance of the *Traveling Salesman Problem* (TSP) is given by a symmetric $N \times N$ cost matrix $C = (c_{ij})$ that satisfies the triangle inequality, *i.e.*, $c_{ij} + c_{jk} \geq c_{ik}, \forall i, j, k$, and has minimum tour length $TSP(C)$. A *1-tree* consists of 2 edges incident to node 1, and a spanning tree on $\{2, \ldots, N\}$. Since every tour is a 1-tree, the cost of the minimum 1-tree is at most $TSP(C)$; furthermore, it can be computed by a minimum spanning tree computation. Each node i is then given a price p_i and reduced costs $\bar{c}_{ij} = c_{ij} + p_i + p_j$ are formed; if γ is

the cost of a minimum 1-tree with respect to the reduced costs, then $\gamma - 2\sum_i p_i \leq TSP(C)$. The Held-Karp bound is attained by choosing the vector p to maximize this lower bound. Held & Karp gave a subgradient optimization method to find such a p by iteratively computing the minimum 1-tree T with respect to the current reduced costs, and then adjusting p by taking a step proportional to $d_i - 2$, where d_i is the degree of node i in T.

It is possible to formulate the Held-Karp bound as a linear program where the variables are the prices (plus one additional variable γ), and there is one constraint for each possible 1-tree: maximize γ subject to the constraint that the reduced cost of each 1-tree is at least γ. We shall instead focus on the dual of this linear program. Let $T_1, \ldots, T_s$ be a complete enumeration of all 1-trees, let d_{ij} denote the degree of node i in T_j, and let c_j denote the total cost of T_j (with respect to the original costs C). If x_j denotes the variable corresponding to T_j, then we can formulate this dual as follows:

minimize $\sum_j c_j x_j$ subject to

$$\sum_{j=1}^{s} d_{ij} x_j \leq 2, \quad i = 1, \ldots, N, \tag{25}$$

$$\sum_{j=1}^{s} x_j = 1, \quad x_j \geq 0, \; j = 1, \ldots, s. \tag{26}$$

We apply Theorem 2.5 by using a bisection search for the minimum feasible cost K, so that there is feasible solution to the fractional packing problem, where P is defined by (26), $Ax \leq b$ is given by (25) and $\sum_{j=1}^{s} c_j x_j \leq K$. The bisection search can be initialized with lower and upper bounds of $c_{\min}$ and $2c_{\min}$, respectively, where $c_{\min} = \min_j c_j$.

Lemma 6.7 The width ρ of the above formulation of the Held-Karp bound is at most N.

Proof: We can assume that $K \geq c_{\min}$. By the triangle inequality, each entry in C is at most the cost of the minimum 1-tree, $c_{\min}$. This implies that $c_j/c_k \leq N$, for each $j, k = 1, \ldots, s$, and so for each $x \in P$, $\sum_j c_j x_j \leq NK$; furthermore, for each 1-tree, the degree of each node is less than N, and hence, $\sum_j d_{ij} x_j < N$. ∎

To minimize a linear objective over P, we choose the 1-tree with minimum objective coefficient. If $z \in \mathbf{R}$ denotes the dual variable for $\sum_j c_j x_j \leq K$ and $y \in \mathbf{R}^N$ denotes the vector of dual variables for (25), then the objective coefficient of x_j, in the corresponding optimization over P, is $c_j z + \sum_{i=1}^{N} d_{ij} y_i$. This implies that the minimum 1-tree found in this iteration is precisely the 1-tree found by minimizing with respect to the reduced costs $\bar{c}$ with $p = y/z$, which was used in each iteration of the Held-Karp subgradient optimization method. Of course, we use a rather different rule to compute the new vector p for the next iteration. Using the minimum spanning tree algorithm of Fredman & Tarjan [3], we get the following result.

Lemma 6.8 The subroutine required by the packing algorithm in Theorem 2.5 for this problem can be implemented in $O(N^2)$ time.

The bisection search produces a value $\tilde{K}$ and a solution $\tilde{x} \in P$ that satisfies $Ax \leq (1+\epsilon)b$ with $K = \tilde{K}$, whereas for $K \leq \tilde{K}/(1+\epsilon)$, there does not exist $x \in P$ of cost K that satisfies (25). However, this does *not* imply that $\tilde{K}$ is within a factor of $(1+\epsilon)$ of the optimum Held-Karp bound; it is possible that any $x \in P$ that satisfies (25) has a much larger cost. However, we have the following lemma.

Lemma 6.9 For any point $\tilde{x} \in P$ of cost at most $\tilde{K}$ such that $\sum_j d_{ij}x_j \leq (1+\epsilon)2$, $i = 1, \ldots, N$, we can find, in $O(N^2)$ time, a point $x \in P$ that satisfies (25) and has cost at most $\tilde{K}(1+2N\epsilon)$.

Proof: We construct x by carefully perturbing $\tilde{x}$. We start by setting $x = \tilde{x}$. The current solution is a convex combination of 1-trees. We will maintain the property that x is a convex combination of *multi-1-trees*, where a multi-1-tree is a spanning tree on nodes $\{2, \ldots, n\}$ and any two edges incident to node 1, which might be two copies of the same edge. Let $d_i(x) = \sum_j d_{ij}x_j$. If x does not satisfy (25), then there must exist a node k with $d_k(x) > 2$. By an averaging argument, we see that in this case there is at least one node j with $d_j(x) < 2$. So we have partitioned the nodes into three sets, $S_<$, $S_=$, and $S_>$, depending on whether a node i has $d_i(x)$ less than, equal to, or greater than 2, respectively. We construct a multi-1-tree T_ℓ as follows: form a spanning tree on $S_<$, add a cycle through node $j \in S_<$ and all nodes in $S_=$, as well as an edge from each node in $S_>$ to node j. Observe that T_ℓ has $d_{i\ell} = 1$ if and only if $d_i(x) > 2$, and $d_{i\ell} = 2$ for each node i with $d_i(x) = 2$. We use T_ℓ to perturb x; let $\bar{x}$ denote the incidence vector of T_ℓ. We set $x = (1-\sigma)x + \sigma\bar{x}$ where σ is chosen to be such that $d_i(x) \leq (1+\epsilon-\sigma/2)2$, for each $i = 1, \ldots, N$, and the number of points with $d_i(x) = 2$ is increased. This can be achieved by setting

$$\sigma = \min_{i:d_i(x)\neq 2} \frac{d_i(x)-2}{d_i(x)-d_{i\ell}}.$$

We repeat this procedure until there are no more points with $d_i(x) > 2$ left, and hence the resulting solution satisfies (25). Since each iteration increases the number of points with $d_i(x) = 2$, the procedure terminates after at most N iterations, and takes $O(N^2)$ time.

Now consider the cost of the resulting solution. If the point $\tilde{x} \in P$ has cost at most $\tilde{K}$, then there must exist a 1-tree of cost at most $\tilde{K}$, and therefore, by the triangle inequality, $c_j \leq N\tilde{K}$, for each $j = 1, \ldots, s$. Each perturbation with corresponding value σ ensures that $\max_i d_i(x)$ decreases by σ. Since a total decrease of at most 2ϵ is needed to ensure that the resulting solution satisfies (25), the values of σ for all of the perturbations sum to at most 2ϵ. Therefore, these perturbations have increased the cost by at most $2\epsilon N\tilde{K}$. Finally, it is not hard to show that there must exist a solution that is a convex combination of 1-trees of no greater cost, by showing that x can be interpreted as a feasible solution to the subtour elimination polytope, which is an alternative formulation of the Held-Karp bound. ∎

If we use $\epsilon = \epsilon_0/(2N)$, we obtain the Held-Karp bound within a factor of $1+\epsilon_0$. Unfortunately, this implies that the algorithm might run for $O(N^3 \log N)$ iterations, where each iteration takes $O(N^2)$ time. In contrast, Vaidya's algorithm [25] takes $O(N^2\mathcal{M}(N)\log N)$ time.

The cutting-stock problem. In the *cutting-stock problem*, we wish to subdivide a minimum number of *raws* of width W, in order to satisfy a demand d_i for *finals* of width w_i, $i = 1, \ldots, M$.

This can be formulated as an integer program with a variable x_j for each feasible pattern for subdividing a single raw; that is, a *pattern* is an vector $a^t \in \mathrm{N}^M$, such that $\sum_i a_i w_i \leq W$, and $a_i \leq d_i$, $i = 1, \ldots, M$. Let $(a_{1j}, \ldots, a_{Mj})^t$, $j = 1, \ldots, N$, be a list of all patterns. Then we wish to minimize $\sum_j x_j$ subject to

$$\sum_{j=1}^{N} a_{ij} x_j \geq d_i, \quad i = 1, \ldots, M, \tag{27}$$

and $x_j \geq 0$, integer, $j = 1, \ldots, N$. Although we want an integer solution, the linear relaxation of this formulation has been extremely useful in practical applications; furthermore, there are applications in which patterns may be used fractionally [2]. Also, finding an approximate solution to this linear relaxation is the key ingredient of Karmarkar & Karp's [13] fully polynomial approximation scheme for the bin-packing problem.

Given a possible number of raws r, we try to find $x \in P = \{x_j : \sum_{j=1}^{N} x_j = r,\ x_j \geq 0,\ j = 1, \ldots, N\}$ that satisfies (27). Since the width ρ can only be bounded by r, we will use the decomposition result, and so we need subroutine (15) for this application. Consider a vertex x of P, where $x_k = r$ and $x_j = 0$, $j \neq k$. The *profit* of this kth pattern is $\sum((y_i r)a_{ik} : i$ such that $a_{ik} \leq \nu d_i/r)$; each final of width w_i that is used in this pattern has a profit of $y_i r$, unless more than $\nu d_i/r$ finals of width w_i are used, in which case none of those finals has any profit. For each pattern a, any vector b such that $b \leq a$ is also a pattern, and so we can find the optimal vertex of P among those patterns j for which $a_{ij} \leq \nu d_i/r$, $i = 1, \ldots, M$. Hence, subroutine (15) is equivalent to solving the following knapsack problem: there are M types of pieces, such that type i has weight w_i and has profit $y_i r$ and the total knapsack has capacity W; at most $\nu d_i/r$ pieces of type i can be used, and we wish to fill our knapsack as profitably as possible. Although this is NP-hard, recall that by Theorem 3.10 an $\epsilon/2$-approximation algorithm would suffice for our purposes. Lawler [16] gave efficient approximation algorithms for the M-piece ordinary knapsack problem that run in $O(M\epsilon^{-2})$ and $O(M \log \epsilon^{-1} + \epsilon^{-4})$ time. Next we adapt both algorithms for the above version of the knapsack problem, where instead of M different pieces, we have M different types of pieces.

Lawler's ϵ-approximation algorithm for the knapsack problem is roughly as follows. First compute the optimum value P_0 of the linear programming relaxation of the knapsack problem. Lawler shows that $P_0 \leq P^* \leq 2P_0$, where P^* denotes the optimum integer knapsack value. Next the algorithm considers *large* pieces with profit at least $T = \epsilon P_0/2$. By rounding the profits and then using dynamic programming, it compiles a list of $O(\epsilon^{-2})$ candidate solutions using just these large pieces. Next it augments each of these candidate solutions with a subset of the remaining small pieces using a greedy algorithm, and selects the best among the resulting solutions.

We shall briefly describe a modification of Lawler's algorithm that can be applied to the version of the knapsack problem with a specified number of copies of each type of piece. We first solve the linear programming relaxation; even with multiple copies, this can be done in $O(M)$ time using median finding within a divide-and-conquer strategy. The only part of the algorithm that is non-trivial to adapt to problems with multiple copies of pieces is the rounding

and dynamic programming. We first round the profits of the large pieces as follows: for an item with profit $p_i \in (2^\ell T, 2^{\ell+1}T]$, we let the rounded profit $q_i = \left\lfloor \frac{p_i}{2^\ell K} \right\rfloor 2^\ell$, where $K = \epsilon T/2 = \epsilon^2 P_0/4$. This rounding guarantees that there are at most $2P_0/K = O(\epsilon^{-2})$ distinct values for the total rounded profit of any knapsack solution.

If (C, P) and (C', P') represent the unused capacity and total profit for two knapsack solutions x and x', respectively, then x dominates x' if $C \geq C'$ and $P \geq P'$. The dynamic programming algorithm finds the set of all undominated solutions. The algorithm works in M stages; after stage j, it has found the set of undominated solutions using pieces of only the first j types of pieces. Stage j can be implemented to run in time proportional to the the multiplicity of piece type j times number of different rounded total profits. Therefore, the time required for the dynamic programming can be estimated as the total number of large pieces (counting multiplicities) times the number of possible rounded total profits. Lawler observed that after rounding, one can use median finding to discard all but $O(\epsilon^{-2})$ large pieces in $O(M)$ time. To see this, note that at most $2^{1-\ell}\epsilon^{-1}$ pieces of profit more than $2^\ell T$ can be used in any solution of total profit at most $2P_0$; hence, for each rounded profit value in the interval $[2^\ell T, 2^{\ell+1}T]$, we need keep only the $2^{1-\ell}\epsilon^{-1}$ pieces of least weight. Since there are $O(\epsilon^{-1})$ distinct rounded profits in this interval there are $\sum_l 2^{1-\ell}\epsilon^{-2} = O(\epsilon^{-2})$ large pieces needed in total. As in Lawler's algorithm, these pieces can be selected in $O(M)$ time.

As a consequence, the dynamic programming algorithm produces $O(\epsilon^{-2})$ solutions using just the large pieces, and the algorithm automatically arranges them in increasing order of unused knapsack capacity. Lawler has shown that these solutions can all be augmented to include a greedily selected extension of small pieces in $(M \log \epsilon^{-1})$ time, using median finding (see section 6 in [16]). This algorithm can also be used for problems with multiple items. The resulting implementation of the subroutine runs in $O(M \log \epsilon^{-1} + \epsilon^{-4})$ time.

In order to obtain the $O(M\epsilon^{-2})$ bound, we have to further modify Lawler's algorithm, using another technique that was introduced in [16] in a somewhat different context. We can assume without loss of generality that the multiplicity of each of the large pieces is at most $O(\epsilon^{-1})$. Next we construct an instance of the ordinary knapsack problem that is equivalent to this instance with multiplicities; the new instance has fewer pieces in total, by replacing a piece of weight w_i, profit p_i and multiplicity m_i by $\lceil \log m_i \rceil$ items, with weights and profits $[w_i, p_i], [2w_i, 2p_i], \ldots, [2^k w_i, 2^k p_i]$ and $[(m_i - 2^k)w_i, (m_i - 2^k)p_i]$ where $k = \lfloor \log m_i \rfloor$. Since we can simulate selecting any number of copies of type i that is at most m_i by choosing an appropriate subset of the new pieces, we get the following lemma.

Lemma 6.10 The resulting instance of the ordinary knapsack problem with $O(M \log \epsilon^{-1})$ pieces is equivalent to the instance of the knapsack problem of the large pieces.

Next we round the resulting pieces as was done above. Notice that although there are $O(M \log \epsilon^{-1})$ pieces, no profit interval $(2^\ell T, 2^{\ell+1}T]$ has more than M items. We will run the dynamic programming starting with the items whose profits fall in the top interval. These items were rounded more roughly. The number of different rounded profits possible using items with profits at least $2^\ell T$ is at most $P^*/(2^\ell K)$. Therefore, the dynamic programming for items with

profits in the interval $(2^\ell T, 2^{\ell+1}T]$ can be implemented in $O(MP^*/(2^\ell K))$ time, and the time spent on the last interval, which is $O(MP^*/K) = O(M\epsilon^{-2})$, dominates the total time for the computation.

Theorem 6.11 An ϵ-approximation algorithm that is analogous to the optimization subroutine (15) can be implemented in $O(\min\{M\epsilon^{-2}, M\log\epsilon^{-1} + \epsilon^{-4}\})$ time.

The covering algorithm of Theorem 3.7 starts by finding an initial solution with $\lambda \geq 1/m$. For the cutting-stock problem, it is easy to provide an initial solution with $\lambda \geq 1/2$. For final i of width w_i, consider the pattern $j(i)$ that consists of $\lfloor W/w_i \rfloor$ finals of type i, $i = 1, \ldots, M$. Set $x'_{j(i)} = d_i/\lfloor W/w_i \rfloor$ for each $i = 1, \ldots, M$, and set each other component $x'_j = 0$. Let $r' = \sum_j x'_j$. Since each selected pattern is at least half used for finals, $r^* \geq r'/2$. Hence, we can initialize the bisection search for the minimum number of raws with r' and $r'/2$ as upper and lower bounds, respectively. For any candidate number of raws r, the vector $x_0 = (r/r')x'$ serves as an initial solution with $\lambda \geq 1/2$ to the covering problem formulated for the cutting-stock problem. However, we are using the decomposition technique, so we must provide an initial solution to the transformed version of the covering problem. Since $\lambda = r/r' \leq 1$, the algorithm used to prove Lemma 5.12 finds a solution of identical quality for the transformed problem $P^1 \times \cdots \times P^k \times P'$. The initial solution x' satisfies each covering constraint with equality, and all patterns used consist of a single type of final. Therefore, each phase of this algorithm can be implemented to run in $O(M)$ time. Since $k = \log r$, we obtain a solution for the transformed problem in $O(M \log r)$ time. Since we have an initial solution with $\lambda \geq 1/2$, we no longer need the $\log M$ calls to IMPROVE-COVER with $\epsilon = 1/6$, and can start with ϵ-scaling; this improves the running time in Theorem 5.13 by deleting the $\log^2 m$ term in the parenthesis.

We apply this improved version of Theorem 5.13 and Theorem 6.11 to obtain the following result. For simplicity, we state the resulting bound using the $O(M\epsilon^{-2})$ bound for the knapsack problem with multiple copies.

Theorem 6.12 For any $\epsilon > 0$, there is a randomized ϵ-approximation algorithm for the fractional cutting-stock problem that is expected to run in $O(M^2\epsilon^{-4}\log(\epsilon^{-1}M)\log r^*\log\epsilon^{-1})$ time, and a deterministic analog that takes a factor of $\log r^*$ more time.

It is not too hard to notice that by somewhat modifying the covering algorithm used we can eliminate the need for the bisection search for the required number of raws; this improves the running time by a $\log\epsilon^{-1}$ factor.

The best previously known algorithm is obtained by using Vaidya's algorithm [25] to solve the linear programming dual of the problem, and then use the techniques of Karmarkar & Karp and the algorithm of Vaidya [26] to obtain a primal solution. The resulting deterministic algorithm runs in $O^*(M^4\mathcal{M}(M) + M^3\epsilon^{-2})$ time. A randomized version runs in $O^*(M^3\mathcal{M}(M) + M^3\epsilon^{-2})$ time. As for our algorithm, these bounds use the $O(M\epsilon^{-2})$ bound for the approximation algorithm. For fixed ϵ, our algorithm is a significant improvement over Vaidya's algorithm.

The integer version of the cutting-stock problem is equivalent to the bin-packing problem, which is usually stated in terms of pieces of specified sizes that are to be packed into the minimum number of bins. Karmarkar & Karp [13] gave a fully polynomial approximation scheme for the bin-packing problem which uses an algorithm (based on the ellipsoid method) for the fractional cutting-stock problem. Our algorithm can be used to replace the ellipsoid method in this application to yield the fastest known deterministic algorithm for this problem.

Karmarkar & Karp give a fully polynomial approximation scheme for the bin-packing problem that, for an instance with N pieces and optimum value r^*, delivers a solution that uses $(1+\epsilon)r^*+O(\epsilon^{-2})$ bins. In fact, the additive term in the performance guarantee can be improved to $O(\epsilon^{-1}\log(\epsilon^{-1}))$.

We can assume without loss of generality that the size W of the bins is 1. Given a bin-packing instance I, let $opt(I)$ denote the minimum number of bins required for this instance, and let $size(I)$ denote the sum of the piece sizes. Clearly, $size(I) \leq opt(I)$. The Karmarkar & Karp algorithm first deletes any piece of size at most $\epsilon/2$. Let I' denote the resulting instance. These small pieces can be added back to a packing of the remaining pieces, arbitrarily filling up the bins without effecting the performance guarantee (by Lemma 3 in [13]).

Next the algorithm uses grouping of pieces to have a small number of distinct piece sizes. Karmarkar & Karp use linear grouping for one version of the algorithm, but they use geometric grouping for a more sophisticated version. An improved guarantee (where the additive error term is $O(\epsilon^{-1}\log\epsilon^{-1})$) is obtained by using geometric grouping with parameter $k = size(I)\epsilon/\log(2\epsilon^{-1})$. This grouping yields a rounded instance J which satisfies $opt(J) \leq opt(I) \leq opt(J) + k\log 2\epsilon^{-1}$ (by Lemma 5 in [13]). The Karmarkar & Karp algorithm approximately solves the fractional cutting-stock problem corresponding to instance J to obtain a vertex x, which is converted to the integer solution $\lceil x \rceil$. The number of additional bins introduced by this rounding is at most the number of non-zeros in x; since x is a vertex, this is at most the number of different piece sizes. It is not hard to show that M, the number of different piece sizes in the rounded instance is at most $(2/k)size(I) + \lceil \log 2\epsilon^{-1} \rceil$ (by Lemma 5 in [13]). The choice of k implies that $M = O(\epsilon^{-1}\log\epsilon^{-1})$. Therefore, the total number of bins used is at most $opt(I) + M + k\log 2\epsilon^{-1} = (1+\epsilon)opt(I) + O(\epsilon^{-1}\log\epsilon^{-1})$.

The geometric grouping can be constructed in $O(N\log M)$ time. We use Theorem 6.12 to solve the resulting cutting stock problem. Karmarkar & Karp find a vertex of the covering problem. Instead, we will find a solution consisting of at most M non-zeros. The randomized version of the algorithm increases the number of non-zeros by at most 1 every iteration. Therefore, the final number of zeros is at most $O(M\epsilon^{-2}\log(\epsilon^{-1}M))\log N)$. By implementing the randomized version deterministically (choosing the best commodity every iteration, rather than a random one) we obtain the same bound on the number of non-zeros also for the deterministic version. Given a solution with more than M non-zeros, the number of non-zeros can be decreased by one using matrix inversion without affecting the quality of the solution. Therefore, in $O(M\mathcal{M}(M)\epsilon^{-2}\log(\epsilon^{-1}M))\log N) = O(\epsilon^{-6}\log^4\epsilon^{-1}\log N)$ time we can find a solution with at most M non-zeros.

We combine the bound given above for M, and Theorems 6.12 and 6.11. Observe that, for

any constant c, $O(N \log \epsilon^{-1} + \epsilon^{-6} \log^4 \epsilon^{-1} \log^c N)$ can be bounded by $O(N \log \epsilon^{-1} + \epsilon^{-6} \log^{4+c} \epsilon^{-1})$.

Theorem 6.13 There is a randomized fully polynomial approximation scheme for the bin-packing problem that, for an instance with N pieces and optimum value r^*, delivers a solution that uses $(1+\epsilon)r^* + O(\epsilon^{-1} \log \epsilon^{-1})$ bins in $O(N \log \epsilon^{-1} + \epsilon^{-6} \log^5 \epsilon^{-1})$ time, a deterministic analog runs in $O(N \log \epsilon^{-1} + \epsilon^{-6} \log^6 \epsilon^{-1})$ time.

Our cutting stock algorithm was a significant improvement over the best previously know algorithm when M is large relative to ϵ^{-1}. However, in this application $M = O(\epsilon^{-1} \log \epsilon^{-1})$. Using the algorithms of Vaidya [26, 25] to solve the fractional cutting stock problems, as mentioned after Theorem 6.12, and plugging in $M = O(\epsilon^{-1} \log \epsilon^{-1})$ gives a deterministic algorithm that runs in $O^*(N \log \epsilon^{-1} + \epsilon^{-4} \mathcal{M}(M))$ time, and a randomized version that runs in $O^*(N \log \epsilon^{-1} + \epsilon^{-3} \mathcal{M}(M))$ time. Thus, our deterministic algorithm improves on the deterministic implementation of Vaidya's algorithm.

Minimum-cost multicommodity flow. The input for the minimum-cost multicommodity flow problem consists of an N-node, M-edge directed graph $G = (V, E)$, a non-negative cost $c(e)$ and a non-negative capacity $u(e)$ for each edge $e \in E$, and source-sink pairs $s_j, t_j \in V$ with a non-negative demand d_j, $j = 1, \ldots, K$, that specify the K commodities. For notational convenience we assume that the graph G is connected and has no parallel edges.

For each commodity j, we have a function $f_j(e) \geq 0$, that specifies the flow of that commodity on each edge $e \in E$, $j = 1, \ldots, K$. The total flow function is then $f(e) = \sum_j f_j(e)$, for each $e \in E$. The *conservation constraints* ensure that

$$\sum_{w:wv \in E} f_j(wv) - \sum_{w:vw \in E} f_j(vw) = 0 \text{ for each } v \notin \{s_j, t_j\},\ j = 1, \ldots, K. \tag{28}$$

We require also that

$$\sum_{w:vw \in E} f_j(vw) - \sum_{w:wv \in E} f_j(wv) = d_j \text{ for } v = s_j. \tag{29}$$

We say that a multicommodity flow f in G is *feasible* if $f(e) \leq u(e)$ for each edge $e \in E$. The cost of a flow f is $\sum_{e \in E} c(e) f(e)$ and the objective is to find a feasible flow of minimum cost.

To apply our relaxed decision procedure, we once again use bisection search for the minimum feasible cost B. We define P by (28), (29), and the constraint $f_j(e) \geq 0$, for each commodity $j = 1, \ldots, K$ and each edge $e \in E$; this is in the product form $P^1 \times \cdots \times P^K$, where P^j denotes these constraints on commodity $j = 1, \ldots, K$. Let $Ax \leq b$ be given by the feasibility constraints, as well as the budget constraint $\sum_{e \in E} c(e) f(e) \leq B$.

We shall use the decomposition technique of Theorem 5.11. To do so, we first specify γ_i for each inequality in $Ax \leq b$, and then show how to compute subroutine (14) for each P^j. For each edge $e \in E$, let $\gamma(e) = 1$ (corresponding to the inequality $f(e) \leq u(e)$) and let $\gamma = N$ (corresponding to the budget constraint), so that $\Gamma = M + N = O(M)$.

Lemma 6.14 Subroutine (14) can be implemented in $O(M + N \log N)$ time for each P^j, $j = 1, \ldots, K$.

Proof: Each vertex of P^j corresponds to an (s_j, t_j) path with d_j units of flow of commodity j along it. For each $e \in E$, let $y(e)$ denote the dual variable for its capacity constraint, and let z denote the dual variable for the budget constraint. Given y, z and ν, we must find an (s_j, t_j) path $\tilde{Q}$ such that

$$u(e) \geq d_j/\nu \text{ for each } e \in \tilde{Q}, \text{ and } \sum_{e \in Q} c(e) \leq \nu B/d_j, \tag{30}$$

and for which,

$$\sum_{e \in \tilde{Q}} y(e) + zc(e) \quad \leq \quad \sum_{e \in Q} y(e) + zc(e), \quad \text{for all } Q \in \mathcal{Q},$$

where $\mathcal{Q}$ is the set of (s_j, t_j) paths Q such that $u(e) \geq d_j/\nu$ for each $e \in Q$, and $\sum_{e \in Q} c(e) \leq (\nu/N)B/d_j$. Observe that all paths in $\mathcal{Q}$ are contained in the subgraph of edges e that satisfy $u(e) \geq d_j/\nu$ and $c(e) \leq (\nu/N)B/d_j$; furthermore, each (s_j, t_j) path in this subgraph satisfies (30). Therefore, by computing the shortest (s_j, t_j) path with respect to the modified costs $y(e) + zc(e)$ in this subgraph, we find a suitable path $\tilde{Q}$. This takes $O(M + N \log N)$ time. ∎

We use our relaxed decision procedure within a bisection search for the appropriate choice for the budget B, which can be at most $C = \sum_e c(e)u(e)$. By applying Theorem 5.11, we obtain the following result; note that an ϵ-optimal flow may exceed the optimum cost and the capacity constraints by a $(1 + \epsilon)$ factor.

Theorem 6.15 For any fixed $\epsilon > 0$, there exists a deterministic algorithm for the minimum-cost multicommodity flow problem that finds an ϵ-optimal flow and runs in $O(K^2 M \log N \ (M + N \log N) \log C)$ time, and a randomized analog that runs in $O(KM \log N \ (M + N \log N) \log C)$ time.

The best previously known algorithm is due to Vaidya [26]. For the randomized version, our algorithm is an $\Omega^*(M^{.5} N K^{2.5})$ factor faster than Vaidya's.

Acknowledgments

We are grateful to Andrew Goldberg and Cliff Stein for many helpful discussions. In particular, we would like to thank Cliff for allowing us to include his observation that an integer version of the packing algorithm could be applied to the job-shop scheduling problem.

References

[1] M. E. Dyer. An $O(n)$ algorithm for the multiple-choice knapsack linear program. *Mathematical Programming*, 29:57-63, 1984.

[2] K. Eisemann. The trim problem. *Management Science*, 3:279–284, 1957.

[3] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. Assoc. Comput. Mach.*, 34:596–615, 1987.

[4] H. N. Gabow. Using Euler partitions to edge-color bipartite multi-graphs. *Int. J. Comput. Inform. Sci.*, 5:345–355, 1976.

[5] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9:839–859, 1961.

[6] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem – Part II. *Operations Research*, 11:863–888, 1963.

[7] A. V. Goldberg. A natural randomization strategy for multicommodity flow and related algorithms. Unpublished manuscript, 1991.

[8] A. V. Goldberg, S. A. Plotkin, and É. Tardos. Combinatorial algorithms for the generalized flow problem. *Mathematics of Operations Research*, 16:351–381, 1990.

[9] A. V. Goldberg, S. A. Plotkin, D. B. Shmoys, and É. Tardos. Interior point methods for fast parallel algorithms for bipartite matching and related problems. *SIAM J. on Computing*, to appear.

[10] 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, New Brunswick, NJ, 1991.

[11] M. Held and R. M. Karp. The traveling-salesman problem and minimum cost spanning trees. *Operations Research*, 18:1138–1162, 1970.

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

[13] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 206–213, 1982.

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

[15] P. Klein, S. A. Plotkin, C. Stein, and É. Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. Technical Report 961, School of Operations Research and Industrial Engineering, Cornell

University, 1991. A preliminary version of this paper appeared in *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 310–321, 1990.

[16] E. L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4:339–356, 1979.

[17] E. L. Lawler and J. Labetoulle. On preemptive scheduling on unrelated parallel processors by linear programming. *J. Assoc. Comput. Mach.*, 25:612–619, 1978.

[18] T. Leighton, F. Makedon, S. Plotkin, C. Stein, É. Tardos, and S. Tragoudas. Fast approximation algorithms for multicommodity flow problems. In *Proceedings of the 23rd Annual ACM Symposium on the Theory of Computing*, pages 101–111, 1991.

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

[20] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming, A*, 24:259–272, 1990.

[21] P. Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *J. Comput. System Sciences*, 37:130–143, 1988.

[22] P. Raghavan and C. D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.

[23] F. Shahrokhi and D. W. Matula. The maximum concurrent flow problem. *J. Assoc. Comput. Mach.*, 37:318–334, 1990.

[24] D. B. Shmoys, C. Stein and J. Wein. Improved approximation algorithms for shop scheduling problems. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 148–157, 1991.

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

[26] P. M. Vaidya. Speeding up linear programming using fast matrix multiplication. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 332–337, 1989.

[27] S. L. van de Velde. Machine scheduling and Lagrangian relaxation. Doctoral thesis, Centre for Mathematics and Computer Science, Amsterdam, 1991.