INSTITUT FÜR INFORMATIK

An $OPT + 1$ algorithm for the cutting stock problem with a constant number of object lengths

Klaus Jansen
Roberto Solis-Oba

Bericht Nr. 1007
July 2010

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

An $OPT + 1$ algorithm for the cutting stock problem with a constant number of object lengths

Klaus Jansen*
Institut für Informatik
Universität zu Kiel, Kiel, Germany
kj@informatik.uni-kiel.de

Roberto Solis-Oba†
Department of Computer Science
University of Western Ontario, London, Canada
solis@csd.uwo.ca

Abstract

In the *cutting stock problem* we are given a set $T = \{T_1, T_2, \ldots, T_d\}$ of object types, where objects of type T_i have integer length $p_i > 0$. Given a set $\mathcal{O}$ of n objects containing n_i objects of type T_i, for each $i = 1, \ldots, d$, the problem is to pack $\mathcal{O}$ into the minimum number of bins of capacity β. In this paper we consider the version of the problem in which the number d of different object types is constant and we present an algorithm that computes a solution using at most $OPT + 1$ bins, where OPT is the value of an optimum solution.

1 Introduction

In the *cutting stock problem* we are given a set $T = \{T_1, T_2, \ldots, T_d\}$ of object types, where objects of type T_i have positive integer length p_i. Given an infinite set of bins, each of integer capacity β, the problem is to pack a set $\mathcal{O}$ of n objects into the minimum possible number of bins in such a way that the capacity of the bins is not exceeded; in set $\mathcal{O}$ there are n_i objects of type T_i, for all $i = 1, \ldots, d$. In this paper we consider the version of the problem in which the number d of different object types is constant.

In the related bin packing problem, given an infinite set of identical bins the goal is to pack a set of n objects with positive integer lengths into the minimum possible number of bins. The cutting stock problem can be considered as the *high multiplicity* version of bin packing, as defined by Hochbaum and Shamir [10]. In a high multiplicity problem, the input objects are partitioned into types and all objects of the same type are identical. The number of objects of a given type is called the type's multiplicity. Note that a high multiplicity problem allows a compact representation of the input, as the attributes of each type need to be listed only once along with the multiplicity of the type. Hence, any instance of the cutting stock problem with a constant number of object types can be represented with a number of bits that is logarithmic in the number of objects.

*Research partially supported by the European Project AEOLUS, contract 015964

†Research partially supported by the Natural Sciences and Engineering Research Council of Canada, grant 227829-2009.

There is extensive research literature on the bin packing and cutting stock problems, attesting to their importance, both, from the theoretical and practical points of view (see e.g. the survey by Coffman et al. [1]). The cutting stock problem was introduced by Eisemann [3] in 1957 under the name of the "Trim problem". The cutting stock and bin packing problems are known to be strongly NP-hard and no approximation algorithm for them can have approximation ratio smaller than 3/2 unless P = NP. In 1985 Marcotte [19] showed that the cutting stock problem with two different object types has the so called integer round-up property and so the algorithm by Orlin in [21] can solve this particular version of the problem in polynomial time. Later, McCormick et al. [20] presented a more efficient $O(\log^2 \beta \log n)$ time algorithm for this same version of the problem.

Filippi and Agnetis [5] proposed an algorithm for the cutting stock problem that uses at most $OPT + d - 2$ bins, where OPT is the value of an optimum solution; hence, this algorithm also finds an optimum solution for the case of $d = 2$. Recently, Filippi [6] improved on the above algorithm for the case when $d \geq 4$ by providing an algorithm that uses at most $OPT + 1$ bins for $2 < d \leq 6$ and at most $OPT + 1 + \lfloor (d-1)/3 \rfloor$ bins for $d > 6$. ¿From the asymptotic point of view, the best known algorithm for the problem is by Karmarkar and Karp [14] and it produces solutions of value at most $OPT + O(\log^2 d)$. This algorithm can be implemented to run in $d^{O(d)} \times O(\log^7 n)$ time if the algorithm of Kannan [13] is used to solve certain instances of the knapsack problem that arise during the execution of the algorithm of Karmakar and Karp.

It is not known whether the cutting stock problem can be solved in polynomial time for every fixed value d. Similarly, it is not known whether there is any polynomial time algorithm for bin packing that produces a solution of value at most $OPT + k$ for some constant value k. In this paper we make further progress towards answering these questions by providing an algorithm for the cutting stock problem that uses at most $OPT + 1$ bins, for any fixed value d.

Theorem 1.1 *There is a* $d^{O(d2^d)} \times 2^{O(8^d)} \times O((\log^2 n + \log \beta)^3 \log^5 n)$ *time algorithm for the cutting stock problem with a constant number* d *of different object types, that solves the problem using at most* $OPT + 1$ *bins, where* OPT *is the value of an optimum solution.*

When computing time complexities we use the *log-cost* RAM model, where each arithmetic operation requires time proportional to the logarithm of the size of its operands. Our algorithm uses a variation of the integer programming formulation (IP) for the cutting stock problem of Gilmore and Gomory [7]; furthermore, we take advantage of a result by Eisenbrand and Shmonin [4] stating that IP has an optimum solution with only a constant number of positive variables. By partitioning the set of objects into two groups of small and big objects, we can re-write IP so that only a constant number of constraints is needed to restrict the placement of big objects in the bins. Then, by relaxing the integrality constraints on the variables controlling the packing for the small objects, we obtain a mixed integer program with a constant number of integer variables. We show that this mixed integer program can be solved in polynomial time using Lenstra's algorithm [17], and a simple rounding procedure can be then used to transform this solution into a feasible solution for the cutting stock problem that uses at most $OPT + 1$ bins.

2 Mixed Integer Program Formulation

Let $\varepsilon = \frac{1}{2d-1}$. We partition the set O of objects in two sets: The *big objects*, with lengths at least $\varepsilon\beta$, and the *small objects*, with lengths smaller than $\varepsilon\beta$. Without loss of generality, let $p_1, \ldots, p_\alpha$ be the different lengths of the big objects and let $p_{\alpha+1}, \ldots, p_d$ be the lengths of the small objects. Note that a bin can store at most $1/\varepsilon$ big objects in it.

A *configuration* C_i is a set of objects of total length at most β, so all objects in a configuration can be packed in a bin. Given a configuration C_i, the subset C_i^B of big objects in C_i is called a *big configuration* and the subset C_i^S of small objects in C_i is called a *small configuration*. Observe that C_i^B could be empty. A configuration C_i can be specified with a d-dimensional vector $C_i = \langle a(C_i, 1), a(C_i, 2) \ldots, a(C_i, d)\rangle$ in which the j-th entry, $a(C_i, j)$, specifies the number of objects of length p_j in C_i. As the number of different object lengths is d, the number of different configurations is at most n^d; similarly, the number of different big configurations is at most $1/\varepsilon^d$.

Let $\mathcal{C}$ be the set of all configurations. The cutting stock problem can be formulated as the following integer program, first proposed by Gilmore and Gomory [7].

$$\begin{aligned} \text{IP}: \quad \min \quad & \sum_{C_i \in \mathcal{C}} x_{C_i} \\ \text{s.}t. \quad & \sum_{C_i \in \mathcal{C}} a(C_i, j) x_{C_i} \geq n_j, \quad \text{for } j = 1, \ldots, d \qquad (1) \\ & x_{C_i} \in \mathbb{Z}_{\geq 0}, \quad \text{for all } C_i \in \mathcal{C} \end{aligned}$$

In this integer program, n_j is the total number of objects of length p_j, and for each configuration C_i, variable x_{C_i} indicates the number of bins storing objects according to C_i. Constraint (1) ensures that all objects are placed in the bins.

Eisenbrand and Shmonin [4] show that this integer program has an optimum solution x^* in which at most 2^d of the variables $x^*_{C_i}$ are non-zero. We will use this result to re-write IP so that the number of big configurations used is at most 2^d. To do this, let us first split each configuration $C_i \in \mathcal{C}$ into a big, C_i^B, and a small, C_i^S, configuration. Let $\mathcal{C}^B$ be the set of all big configurations. Note that $\mathcal{C}^B$ includes the empty configuration.

IP, then, can be re-written as the following feasibility problem.

$$\begin{aligned} \text{IP1}: \quad & \sum_{\substack{C_i \in \mathcal{C} \\ C_i^B = B_\ell}} x_{C_i} \leq y_\ell, \quad \text{for all } B_\ell \in \mathcal{C}^B \qquad (2) \\ & \sum_{B_\ell \in \mathcal{C}^B} y_\ell = m^* \qquad (3) \\ & \sum_{B_\ell \in \mathcal{C}^B} a(B_\ell, j) y_\ell \geq n_j, \quad \text{for all } j = 1, \ldots, \alpha \qquad (4) \\ & \sum_{C_i \in \mathcal{C}} a(C_i, j) x_{C_i} \geq n_j, \quad \text{for all } j = \alpha + 1, \ldots, d \qquad (5) \\ & x_{C_i} \in \mathbb{Z}_{\geq 0}, \quad \text{for all } C_i \in \mathcal{C} \\ & y_\ell \in \mathbb{Z}_{\geq 0}, \quad \text{for all } B_\ell \in \mathcal{C}^B \end{aligned}$$

In this integer program $a(B_\ell, j)$ is the number of objects of length p_j in configuration B_ℓ, y_ℓ is a variable indicating the number of bins in which the big objects are packed according to big configuration B_ℓ, and m^* is the minimum number of bins needed to pack all the objects. Constraint (3) ensures that the number of bins used is the optimum one, while constraints (4) and (5) guarantee that all big and small objects are packed in the bins.

Lemma 2.1 *$x = \langle x_{C_i}, x_{C_2}, \ldots, x_{C_{|\mathcal{C}|}} \rangle$ is an optimum solution for IP if and only if (x, y) is a feasible solution for IP1, where $y = \langle y_1, y_2, \ldots, y_{|\mathcal{C}^B|} \rangle$, and $y_\ell = \sum_{\substack{C_i \in \mathcal{C} \\ C_i^B = B_\ell}} x_{C_i}$ for every index $\ell = 1, 2, \ldots, |\mathcal{C}^B|$.*

Proof: If x is an optimum solution of IP then $\sum_{C_i \in \mathcal{C}} x_{C_i} = m^*$ and thus (x, y) satisfies constraints (2) and (3). Constraint (5) follows from (1), while constraint (4) is satisfied because for all $j = 1, \ldots, \alpha$,

$$\sum_{B_\ell \in \mathcal{C}^B} a(B_\ell, j) y_\ell = \sum_{B_\ell \in \mathcal{C}^B} \left(a(B_\ell, j) \sum_{\substack{C_i \in \mathcal{C} \\ C_i^B = B_\ell}} x_{C_i} \right) = \sum_{C_i \in \mathcal{C}} a(C_i, j) x_{C_i} \geq n_j, \quad \text{from (1).}$$

Similarly, if (x, y) is a feasible solution for IP1, then x is a feasible solution for IP since, as shown above, as x satisfies constraints (4) and (5), then it also satisfies constraint (1). Furthermore,

$$\sum_{C_i \in \mathcal{C}} x_{C_i} = \sum_{B_\ell \in \mathcal{C}^B} y_\ell = m^*,$$

so x is an optimum solution for IP. □

¿From Corollary 6 in [4] and Lemma 2.1, IP1 has an optimum solution (x^*, y^*) with at most 2^d non-zero variables y^*. Let S^{B^*} be the set of at most 2^d big configurations corresponding to the non-zero variables in y^*. We can reduce the number of constraints of type (2) by not considering all big configurations $\mathcal{C}^B$, but only those in S^{B^*}. Since we do not know the optimum solution (x^*, y^*) we do not know either which big configurations to select. However, since the number of big configurations is $1/\varepsilon^d$, there is only a constant number $\binom{\varepsilon^{-d}}{2^d}$ of subsets S^B of 2^d big configurations, so we can try them all knowing that one of them will lead to an optimum solution.

Note that we do not know the value of m^* either, but since $m^* \leq n$, we can use binary search to find in $O(\log n)$ iterations the smallest value for m^* for which IP1 has a feasible solution. Finally, we relax the integrality constraints on the variables x_{C_i} of IP1 to get the following mixed integer linear program, where m is the number of bins and S^B is a subset of big configurations:

$$\text{MILP}(m, S^B): \quad \sum_{\substack{C_i \in \mathcal{C} \\ C_i^B = B_\ell}} x_{C_i} \leq y_\ell, \quad \text{for all } B_\ell \in S^B \tag{6}$$

$$\sum_{B_\ell \in S^B} y_\ell = m \tag{7}$$

$$\sum_{B_\ell \in S^B} a(B_\ell, j) y_\ell \geq n_j, \quad \text{for all } j = 1, \ldots, \alpha \tag{8}$$

$$\sum_{\substack{C_i \in \mathcal{C} \\ C_i^B \in S^B}} a(C_i, j) x_{C_i} \geq n_j, \quad \text{for all } j = \alpha + 1, \ldots, d \tag{9}$$

$$x_{C_i} \geq 0, \quad \text{for all } C_i \in \mathcal{C} \text{ such that } C_i^B \in S^B$$

$$y_\ell \in \mathbb{Z}_{\geq 0}, \quad \text{for all } B_\ell \in S^B$$

$$x_{C_i} = 0, \quad \text{for all } C_i \in \mathcal{C} \text{ such that } C_i^B \notin S^B$$

$$y_\ell = 0, \quad \text{for all } B_\ell \notin S^B.$$

3 Rounding

In Section 4 we show how to solve MILP(m, S^B) using Lenstra's algorithm [17]. Let (x^+, y^+) be the solution produced by Lenstra's algorithm for MILP(m^*, S^{B^*}). We show now how to obtain an integer solution from (x^+, y^+) that uses at most $m^* + 1$ bins.

Consider the following linear program, that we will use to determine how to pack the small objects in the bins:

$$\text{LP1}: \quad \min \sum_{C_i \in \mathcal{C}} x_{C_i}$$

$$\sum_{\substack{C_i \in \mathcal{C} \\ C_i^B = B_\ell}} x_{C_i} = y_\ell^+, \quad \text{for all } B_\ell \in S^{B^*} \tag{10}$$

$$\sum_{\substack{C_i \in \mathcal{C} \\ C_i^B \in S^{B^*}}} a(C_i, j) x_{C_i} = n_j, \quad \text{for all } j = \alpha + 1, \ldots, d \tag{11}$$

$$x_{C_i} \geq 0, \quad \text{for all } C_i \in \mathcal{C} \text{ such that } C_i^B \in S^{B^*} \tag{12}$$

$$x_{C_i} = 0, \quad \text{for all } C_i \in \mathcal{C} \text{ such that } C_i^B \notin S^{B^*}$$

Let x' be a basic feasible solution for LP1. Note that any constraints of type (10) where $y_\ell^+ = 0$ can be discarded since then all variables x_{C_i} for which $C_i^B = B_\ell$ must have value 0 by constraints (12). Let Y be the number of variables y_ℓ^+ with positive value. Since the number of constraints (11) is no more than d, then, at most $Y + d$ variables x'_{C_i} from basic feasible solution x' can have positive value. For each variable y_ℓ^+, let Y_ℓ^+ be the set of positive variables x'_{C_i} with $C_i^B = B_\ell$. Note that $|Y_\ell^+| \geq 1$ for each positive variable y_ℓ^+.

Since at most $Y + d$ variables x'_{C_i} have positive value, then at most d positive variables y^+_ℓ can have associated sets Y^+_ℓ of size larger than 1; this implies that at most $2d$ variables x'_{C_i} have non-integer values, since all variables y^+_ℓ are integer.

Let $\bar{x}_{C_i} = \lfloor x'_{C_i} \rfloor$ and $\tilde{x}_{C_i} = x'_{C_i} - \bar{x}_{C_i}$, for each configuration $C_i \in \mathcal{C}$. We now have an integral solution $(\bar{x}, y^+)$ that packs all the big objects and some of the small objects into m^* bins. For each configuration $C_i \in \mathcal{C}$ this solution has $\bar{x}_{C_i}$ bins in which objects are packed according to configuration C_i.

The remaining small objects (which are fractionally assigned to configurations in $\tilde{x}$) are greedily packed in the m^* bins using the next fit algorithm, taking the small objects in non-decreasing order of length. Since there are at most $2d$ positive variables in $\tilde{x}$, then at most $2d - 1$ small objects are left unpacked by the next fit algorithm. Each one of these objects has length smaller than $\varepsilon\beta$ and, since $\varepsilon = 1/(2d - 1)$, all of them can be packed in a single bin.

Lemma 3.1 *The above rounding procedure packs all the objects in at most $m^* + 1$ bins.*

4 Solving the Mixed Integer Linear Program

Lenstra's algorithm [17] can be used to solve mixed integer linear programs in which the number of integer variables is constant. The time complexity of the algorithm is $O(P(N'))$, where P is a polynomial and $N' = O(M\vartheta \log k)$ is the maximum number of bits needed to specify the input; M is the number of constraints in the mixed integer linear program, ϑ is the number of variables, and k is the maximum of the absolute values of the coefficients of the constraints. Since MILP(m, S^B) has $O(n^d)$ variables, it might seem that the time complexity of Lenstra's algorithm is too high for our purposes as an instance of the cutting stock problem is specified with only $N = \sum_{i=1}^{d}(\log p_i + \log n_i) + \log \beta = O(\log \beta + \log n)$ bits, and so $P(N')$ is not a polynomial function of N. In this section we show that Lenstra's algorithm can, in fact, be implemented to run in time polynomial in N.

First, let us write the set of constraints of MILP(m, S^B) in the form $A(y, x) \leq b$. Let K' denote the closed convex set

$$K' = \{(y, x) \in \mathbb{R}^{2^d + n^d} \mid A(y, x) \leq b\}$$

and let

$$K = \{y \in \mathbb{R}^{2^d} \mid \text{there exists } x \in \mathbb{R}^{n^d} \text{ such that } (y, x) \in K'\};$$

then, deciding whether MILP(m, S^B) has a feasible solution is equivalent to deciding whether $K \cap \mathbb{Z}^{2^d} \neq \emptyset$. For completeness we give below a description of Lenstra's algorithm.

Algorithm LENSTRA(K)
Input: Compact convex set K of dimension D.
Output: A point in $K \cap \mathbb{Z}^D$, if one exists, or null if $K \cap \mathbb{Z}^D = \emptyset$.

1. Reduce the dimension of K until we ensure that K has positive volume.
2. Compute a linear transformation τ that maps K into a ball-like set τK such that there is a point σ and radii r, R with $\frac{R}{r} = 2D^{3/2}$ for which $\mathcal{B}(\sigma, r) \subset \tau K \subset \mathcal{B}(\sigma, R)$, where $\mathcal{B}(\sigma, z) \subset \mathbb{R}^D$ is the closed ball with center σ and radius z.
3. Compute a *reduced basis* $b_1, b_2, \ldots, b_D$ for the lattice $\tau\mathbb{Z}^D$: a basis such that $\prod_{i=1}^{D} \|b_i\| \leq 2^{D(D-1)/4} \times |\text{determinant}(b_1, b_2, \ldots, b_D)|$, where $\| \ \|$ denotes the Euclidean norm.
4. Find a point $v \in \tau\mathbb{Z}^D$ such that $\|v - \sigma\| \leq \frac{1}{2}\sqrt{D}\ \max\{\|b_i\| \mid i = 1, \ldots, D\}$.
5. If $v \in \tau K$ then output $\tau^{-1}v$.
6. If $v \notin \tau K$ let $H = \sum_{i=1}^{D-1}(\mathbb{R}b_i)$ be the $(D-1)$-hyperplane spanned by $b_1, \ldots, b_{D-1}$.

   For each integer i such that $H + ib_D$ intersects $\mathcal{B}(\sigma, R)$ **do** :

   Let $\vec{K}$ be the intersection of K with $H + ib_D$.

   If $v =$LENSTRA($\vec{K}$) is not null, then output $\tau^{-1}(v, ib_D)$ and terminate.

   Output null.

In the following sections we compute the time complexity of every step of Lenstra's algorithm when used to solve MILP(m, S^B).

4.1 Step 1: Ensuring that K has positive volume

Step 1 of Lenstra's algorithm requires finding 2^d vertices of K whose convex hull is a 2^d-simplex $\mathcal{S}$ of positive volume (for details see [17]). Finding each one of these vertices requires maximizing up to 2^d linear functions on K, thus in total $O(2^{2d})$ linear functions need to be maximized on K. Note that maximizing a linear function on K is equivalent to maximizing on K' a linear function $f(y) = f_1y_1 + f_2y_2 + \cdots + f_{2^d}y_{2^d}$ that depends only on the 2^d variables y. These maximization problems can be solved using the ellipsoid algorithm [15, 8] as described below. Each linear function $f(y)$ can be computed by using Gaussian elimination over an $O(2^d) \times O(2^d)$ matrix whose null space spans a set of $O(2^d)$ vertices of K. Notice that by constraints (7) and (8) of MILP(m, S^B) each vertex of K is a 2^d-vector whose components are values of at most $\log n$ bits. Hence, each linear function $f(y)$ can be computed in $O(2^{7d} \log^2 n)$ time and each coefficient f_i of $f(y)$ can be encoded by $O(2^{2d} \log n)$ bits (see [23] Theorem 3.3). Then, the total time needed to compute all $O(2^{2d})$ linear functions is $O(2^{9d} \log^2 n)$.

If 2^d vertices cannot be found as above, then the Hermite normal form algorithm of Kannan and Bachem [12] is used to reduce the dimension of K and to find a simplex of positive volume of the reduced dimension. The algorithm in [12] requires the computation of $O(2^{2d})$ determinants of matrices of size $O(2^d) \times O(2^d)$ each of whose entries is encoded with $O(\log n)$ bits, and so it runs in $O(2^{4.7d} \log n)$ time by using the algorithm of [11] for computing the determinants.

The problem of maximizing a linear function $f(y) = f_1y_1 + f_2y_2 + \cdots + f_{2^d}y_{2^d}$ over K' can be written as follows.

$$
\begin{aligned}
\text{LP}: \quad \max \quad & \sum_{B_\ell \in S^B} f_\ell y_\ell \\
s.t. \quad & y_\ell - \sum_{\substack{C_i \in \mathcal{C} \\ C_i^B = B_\ell}} x_{C_i} \geq 0, \quad \text{for all } B_\ell \in S^B \\
& - \sum_{B_\ell \in S^B} y_\ell \geq -m \\
& \sum_{B_\ell \in S^B} a(B_\ell, j) y_\ell \geq n_j, \quad \text{for all } j = 1, \ldots, \alpha \\
& \sum_{\substack{C_i \in \mathcal{C} \\ C_i^B \in S^B}} a(C_i, j) x_{C_i} \geq n_j, \quad \text{for all } j = \alpha + 1, \ldots, d \\
& y_\ell \geq 0 \quad \text{for all } B_\ell \in S^B \\
& x_{C_i} \geq 0, \quad \text{for all } C_i \in \mathcal{C} \text{ such that } C_i^B \in S^B
\end{aligned}
$$

LP has $2^d + d + 1$ constraints, but it might have a very large number, $O(n^d)$, of variables so we deal with its dual instead:

$$
\begin{aligned}
\text{DLP}: \quad \min \quad & \delta_0 m - \sum_{i=1}^{d} (\lambda_i n_i) \\
s.t. \quad & \delta_\ell - \delta_0 + \sum_{i=1}^{\alpha} (a(B_\ell, i)\lambda_i) \leq -f_\ell, \text{ for all } B_\ell \in S^B \qquad (13) \\
& -\delta_\ell + \sum_{i=\alpha+1}^{d} (a(C_i, j)\lambda_i) \leq 0, \text{ for all } C_i \in \mathcal{C},\ B_\ell \in S^B \text{ s.t. } C_i^B = B_\ell \qquad (14) \\
& \delta_0 \geq 0, \delta_\ell \geq 0 \quad \text{for all } B_\ell \in S^B \\
& \lambda_i \geq 0, \quad i = 1, \ldots, d
\end{aligned}
$$

By Lemmas 2.1 and 8.5 of [22] the basic feasible solutions of DLP are 2^d-vectors of rational numbers whose numerators and denominators have absolute values at most $L = (2^d)!n^{d2^d}$. Therefore, we can use the version of the ellipsoid algorithm described in [9] with precision L^{-1} to solve DLP in time polynomial in $\log n$ and $\log \beta$. For convenience, we present the ellipsoid algorithm for solving DLP below.

Algorithm ELLIPSOID(A, b, c)
Input: Constraint matrix A, right hand side vector b, and cost vector c for DLP.
Output: An optimum solution for DLP, if one exists.

0. Let η be the number of columns of A.

1. Set $(\delta,\lambda)_0 = 0$ and $A_0 = \eta^2 L^2 I$, where I is the identity matrix of size $\eta \times \eta$ and $(\delta,\lambda)_0$ is an η-vector.

2. For all values $k = 0, 1, \ldots, 4 \times \eta^2 \lceil \log(2L^3 n^{d+1}) \rceil$ do: {

   2.1 If $(\delta,\lambda)_k$ is a feasible solution for DLP, label k as a *feasible index* and set $v = c$. Else let $(\delta,\lambda)_k$ violate the i-th constraint of DLP; set v to be the i-th row of A and label k as an *infeasible index*.

   2.2 Compute

$$\begin{aligned} q_k &= \frac{A_k v}{\sqrt{v^T A_k v}} \\ (\delta,\lambda)_{k+1} &= (\delta,\lambda)_k + \frac{q_k}{2^d+1} \\ A_{k+1} &= \frac{2^{2d+1}+3}{2^{2d+1}} \left(A_k - \frac{2}{2^d+1} q_k q_k^T \right) \end{aligned}$$

   All these values are rounded to $\log L$ bits.

   }

3. Output the solution $(\delta,\lambda)_k$ for which k is a feasible index and $c^T(\delta,\lambda)_k = \max\{c^T(\delta,\lambda)_i \mid i \text{ is a feasible index}\}$.

DLP has only $2^d + d + 1$ variables, but it has a large number $O(n^d)$ of constraints; hence, for the ellipsoid algorithm to solve DLP in time polynomial in N, the size of the input for an instance of the cutting stock problem, Step 2.1 of Ellipsoid requires the use of an efficient separation oracle that given a vector $(\delta,\lambda) = \langle \delta_0, \ldots, \delta_{2^d}, \lambda_1, \ldots, \lambda_d \rangle$ it either determines that (δ,λ) is a feasible solution for DLP or it finds a constraint of DLP that is violated by (δ,λ).

4.1.1 Separation Oracle for DLP

To design the separation oracle, we can think that each object $o_i \in \mathcal{O}$ has weight p_i and value λ_i. Each constraint (13) can be tested in $O(d^3 2^{7d} \log^2 n)$ time as it just requires us to verify whether for a big configuration $B_\ell \in S^B$, the total value, $\sum_{i=1}^{\alpha} (a(B_\ell, i)\lambda_i)$, of the big objects in B_ℓ is at most $\delta_0 - \delta_\ell - f_\ell$; as mentioned at the beginning of Section 4.1, each coefficient f_ℓ can be encoded with $O(2^{2d} \log n)$ bits and by Lemma 4.2 below, each λ_i and δ_j has $O(d^2 2^{7d} \log^2 n)$ bits. Therefore, all constraints (13) can be tested in $O(d^3 2^{8d} \log^2 n)$ time.

Constraints (14) are a bit harder to test. Since a configuration C_j for which $C_j^B = B_\ell \in S^B$ includes small objects of total weight at most $\beta - \beta_\ell$, where β_ℓ is the total weight of the big objects in B_ℓ, then constraints (14) check that for each $C_j \in \mathcal{C}$ and $B_\ell \in S^B$ such that $C_j^B = B_\ell$, the set of small objects in C_j has total value at most δ_ℓ; in other words, it checks whether every set of small objects of total weight at most $\beta - \beta_\ell$ has value at most δ_ℓ.

Hence, (as it was also observed by Karmakar and Karp [14]) to determine whether the constraints (14) are satisfied we need to solve an instance of the *knapsack problem* where

the input is the set of small objects and the knapsack has capacity $\beta - \beta_\ell$; in the knapsack problem the goal is to find a maximum value set of objects of total weight at most the capacity of the knapsack. If the maximum value of any subset of small items of weight at most $\beta - \beta_\ell$ is larger than δ_ℓ then we know that a constraint of type (14) is violated; furthermore, the solution of the knapsack problem indicates exactly which constraint is not satisfied by δ. Similarly, if the maximum value of all subsets of small objects of total weight at most $\beta - \beta_\ell$ is at most δ_ℓ, then all constraints (14) are satisfied.

Therefore, a separation oracle for DLP needs to be able to efficiently solve any instance of the knapsack problem formed by a set of objects of $d' \leq d$ different types, where objects of type i have weight p_i and value λ_i. This knapsack problem can be formulated as the following integer program.

$$\begin{aligned} \text{KN}: \quad \max \quad & \sum_{i=1}^{d'} \lambda_i z_i \\ s.t. \quad & \sum_{i=1}^{d'} p_i z_i \leq \beta' \\ & z_i \leq n_i, \quad \text{for all } i = 1, \ldots, d' \\ & z_i \in \mathbb{Z}_{\geq 0}, \quad \text{for all } i = 1, \ldots, d' \end{aligned}$$

where β' is the capacity of the knapsack, n_i is the number of objects of type i, and z_i is the number of objects of type i selected by the solution. Integer program KN has a constant number $d' \leq d$ of variables and a constant number of constraints, so it can be solved, for example, by using Kannan's algorithm [13].

Lemma 4.1 *Integer program KN can be solved in $O(d^{9+17d/2}2^{21d}(\log^2 n + \log \beta)^3)$ time.*

Proof: To compute the amount of time required by Kannan's algorithm to solve KN, we first need to determine how big the coefficients λ_i and β' are. Note that $\beta' \leq \beta$, so β' can be encoded with $\log \beta$ bits. As for the coefficients λ_i, they are computed by algorithm Ellipsoid as it solves DLP. By below Lemma 4.2, each λ_i is encoded with $O(d^2 2^{7d} \log^2 n)$ bits.

Furthermore, each coefficient p_i has $O(\log \beta)$ bits and each n_i has $O(\log n)$ bits. By Theorem 5.4 in [13], Kannan's algorithm can solve KN in $O(d^{9d/2}s)$ arithmetic operations, where s is the length of the input. The integers produced by Kannan's algorithm have $O(d^{2d}s)$ bits, so the time complexity of the algorithm is $O(d^{9d/2}s(d^{2d}s)^2)$ since a multiplication of 2 values of k bits each can be performed in $O(k^2)$ time. The number of bits needed to encode all the coefficients of KN is $O(d^3 2^{7d} \log^2 n + d \log \beta)$, so Kannan's algorithm runs in $O(d^{9+17d/2}2^{21d}(\log^2 n + \log \beta)^3)$ time. □

Lemma 4.2 *The maximum number of bits needed to encode each value in the solution (δ, λ) computed by the ellipsoid algorithm for DLP, is $O(d^2 2^{7d} \log^2 n)$.*

Proof: Each iteration of the loop of step 2 of Ellipsoid, excluding step 2.1, performs $O(2^{3d})$ arithmetic operations and the loop is repeated $O(2^{2d} \log(L^3 n^{d+1}))$ times, so the total

number of arithmetic operations performed in this step of the algorithm is $O(d2^{6d}\log n)$ as $L = (2^d)!n^{d2^d}$. Each entry in A_0 has $O(d2^d \log n)$ bits since $\eta = 2^d + d + 1$. Furthermore, every component of the cost vector c for DLP has $O(\log n)$ bits. Therefore, throughout all the iterations of the loop, the maximum number of bits needed to encode any of the values in solution $(\delta, \lambda)_k$ is at most $O((d2^{6d}\log n) \times (d2^d \log n)) = O(d^2 2^{7d}\log^2 n)$.

Note that step 2.1 of ELLIPSOID was excluded from the above calculations as the repeated execution of this step does not contribute to increase the size of the components of the vectors $(\delta, \lambda)_k$. □

4.1.2 Time Complexity of Step 1 of Lenstra's Algorithm

We first compute the time complexity of algorithm ELLIPSOID.

Lemma 4.3 *The running time of* ELLIPSOID *is* $O(\eta^2 d^{10+17d/2} 2^{22d}(\log^2 n + \log\beta)^3 \log n)$, *where* η *is the number of columns of the constraint matrix* A.

Proof: Each execution of step 2.2 of ELLIPSOID performs $O(2^{3d})$ arithmetic operations on values encoded with $O(d^2 2^{7d}\log^2 n)$ bits each (Lemma 4.2). Since a multiplication of 2 values of k bits each can be performed in $O(k^2)$ time, then step 2.2 can be performed in $O(2^{3d} \times (d^2 2^{7d}\log^2 n)^2) = O(d^4 2^{17d}\log^4 n)$ time. Step 2.1 requires the use of the separation oracle of Section 4.1.1, hence by Lemma 4.1 each iteration of the loop of step 2 needs $O(d^4 2^{17d}\log^4 n + d^{9+17d/2}2^{21d}(\log^2 n + \log\beta)^3) = O(d^{9+17d/2}2^{21d}(\log^2 n + \log\beta)^3)$ time.

Since the loop is repeated $4 \times \eta^2 \lceil \log(2L^3 n^{d+1}) \rceil$ times and this loop dominates the execution time of the algorithm, the time complexity of ELLIPSOID is

$$O(\eta^2(\log L + d\log n)(d^{9+17d/2}2^{21d}(\log^2 n + \log\beta)^3)) = O(\eta^2 d^{10+17d/2}2^{22d}(\log^2 n + \log\beta)^3 \log n).$$

□

Lemma 4.4 *Step 1 of Lenstra's algorithm can be performed in time*

$$O(d^{12+17d/2}2^{30d}(\log^2 n + \log\beta)^3 \log^3 n).$$

Proof: As mentioned above, step 1 of algorithm LENSTRA needs the maximization of $O(2^{2d})$ linear functions. Using the ellipsoid algorithm as described above, each maximization can be performed in $O(d^{10+17d/2}2^{24d}(\log^2 n + \log\beta)^3 \log n)$ time, as $\eta = 2^d + d + 1$. Note, however that algorithm ELLIPSOID only solves DLP, while step 1 of algorithm LENSTRA requires the actual solution of LP. A solution for LP can be computed as follows. Algorithm ELLIPSOID makes $\rho = O(2^{2d}\log(L^3 n^{d+1})) = O(d2^{3d}\log n)$ invocations to the separation oracle. Each call to the separation oracle discovers at most one constraint that is violated by the current solution. Let $\mathcal{V}$ be the set of all these violated constraints. Note that $|\mathcal{V}| \leq \rho$.

We now construct a new dual linear program DLP$'$ containing only those constraints in $\mathcal{V}$. Note that the ellipsoid algorithm will produce the same solution for DLP$'$ and for DLP, as only the violated constraints determine the way in which the algorithm will search for a solution. The interesting fact about DLP$'$ is that it contains $2^d + d + 1$ variables and at most ρ constraints. Therefore, the dual LP$'$ of DLP$'$ has $2^d + d + 1$ constraints and

at most ρ variables. We can, then, use the ellipsoid algorithm again to solve LP$'$ in time $O(\rho^2 d^{10+17d/2} 2^{22d}(\log^2 n + \log\beta)^3 \log n) = O(d^{12+17d/2} 2^{28d}(\log^2 n + \log\beta)^3 \log^3 n)$.

The total time needed to maximize all $O(2^{2d})$ linear functions is $O(d^{12+17d/2} 2^{30d}(\log^2 n + \log\beta)^3 \log^3 n)$. This time dominates the time needed to compute the linear functions $f(y)$ and the running time of the algorithm of Kannan and Bachem, which needs to be run at most once. □

4.2 Steps 2-6

Step 2 of algorithm Lenstra requires finding a linear transformation τ that maps K into a set τK that is nearly "spherical". To do this the simplex $\mathcal{S}$ found in step 1 is transformed into a polyhedron of "large" volume through an iterative procedure that at each step modifies $\mathcal{S}$ by replacing one of its vertices with some vertex v of K such that the volume of the polyhedron increases by at least a factor of 3/2. Each vertex v can be found by maximizing $O(2^{2d})$ linear functions $g(y)$ on K. To compute each linear function we need to solve a system of linear equations with $O(2^d)$ variables each, where the coefficients are taken from the vertices of the simplex. If we use Gaussian elimination to solve a system of linear equations, then, by Theorem 3.3 in [23], each linear function $g(y)$ can be computed in $O(2^{7d}\log^2 n)$ time. Furthermore, each coefficient of a linear function $g(y)$ is encoded with $O(2^{2d}\log n)$ bits. The total time needed to compute the $O(2^{2d})$ linear functions is, then, $O(2^{7d}\log^2 n)$.

Lemma 4.5 *Step 2 of Lenstra's algorithm can be performed in time*

$$O(d^{12+17d/2} 2^{31d}(log^2 n + \log\beta)^3 \log^4 n).$$

Proof: Since the volume of the simplex $\mathcal{S}$ found in step 1 is at least $1/(2^d)!$ (see [17]) and by constraint (7) of MILP(m, S^B) the volume of K is at most $m^{2^d} \leq n^{2^d}$, then the number of iterations in the above procedure is at most $\log_{3/2}(n^{2^d}(2^d)!) = O(2^d \log n)$. Therefore, for step 2 of algorithm Lenstra in total we need to maximize $O(2^{3d}\log n)$ linear functions $g(y)$ over K. By using algorithm Ellipsoid to maximize each linear function, as explained in the proof of Lemma 4.4, the total time needed to perform step 2 of Lenstra's algorithm is $O(d^{12+17d/2} 2^{31d}(\log^2 n + \log\beta)^3 \log^4 n)$. □

For Step 3 of algorithm Lenstra we can use the algorithm of Lenstra, Lenstra, and Lovász [16] to find a reduced basis. By Proposition 1.26 in [16] this step can be performed in $O(2^{6d})$ time. Step 4 requires projecting σ over each one of the vectors b_i from the reduced basis and then rounding each component of the projection down to the nearest integer. This step requires $O(2^d)$ multiplications on numbers encoded with $O(2^{6d})$ bits, so this step can be performed in $O(2^{12d})$ time.

Finally, in Step 5, to decide whether $y \in \tau K$, we need to determine whether $y' = \tau^{-1} y \in K$. This requires us to solve MILP when the values of the variables y_ℓ are known: this is just linear program LP when the objective function f is constant and the values for the variables y_ℓ are given. The dual of this linear program is DLP without constraints (13), so we can solve it using the ellipsoid algorithm as described above in $O(d^{10+17d/2} 2^{24d}(\log^2 n + \log\beta)^3 \log n)$ time, by Lemma 4.3.

Lemma 4.6 *The running time of Lenstra's algorithm is*

$$O(d^{12+17d/2}2^{31d+8^d}(\log^2 n + \log\beta)^3 \log^4 n).$$

Proof: By Lemmas 4.4, 4.5 and the above discussion, steps 1-5 of the algorithm can be performed in time $T = O(d^{12+17d/2}2^{31d}(\log^2 n + \log\beta)^3 \log^4 n)$. As shown in [17] in the for loop of step 6 we need to consider at most $2^{1+2d+2^d(2^d-1)/4}$ different values for i. In each iteration of the for loop we perform a recursive call to the algorithm, and the recursive call dominates the running time of every iteration of the loop.

Let $f(D)$ be the time complexity of the algorithm when the input convex set has dimension D. Then, $f(D)$ satisfies the following recurrence:

$$f(D) = T + 2^{1+d+2^d(2^d-1)/4} f(D-1).$$

Therefore, $f(D) = O\left(T(2^{1+d+2^d(2^d-1)/4})^D\right)$. Since $D = O(2^d)$, the complexity of the algorithm is $O(d^{12+17d/2}2^{31d+8^d}(\log^2 n + \log\beta)^3 \log^4 n)$. □

Theorem 4.1 *We can compute a solution for MILP(m^*, S^{B^*}) in time*

$$O(d^{12+17d/2}2^{31d+8^d}(\log^2 n + \log\beta)^3 \log^4 n).$$

Proof: Let $y^+ \in \mathbb{Z}^{2^d}$ be the solution produced by algorithm LENSTRA. Note that this solution does not include values for the variables x_{C_i}. These values can be computed by solving a modified version of LP in which we give values to the variables y_ℓ as indicated by y^+ and we use any linear cost function $f(y)$. By Lemma 4.6 and 4.3 a solution for MILP(m^*, S^{B^*}) can be computed in $O(d^{12+17d/2}2^{31d+8^d}(\log^2 n + \log\beta)^3 \log^4 n)$ time. □

5 The Algorithm for the Cutting Stock Problem

A description of our algorithm for the cutting stock problem is given below.

Algorithm BINPACKING(P, U, β)
Input: Sets $P = \{p_1, \ldots, p_d\}$, $U = \{n_1, \ldots, n_d\}$ of object lengths and their multiplicities; capacity β of each bin.
Output: A packing for the objects into at most $OPT + 1$ bins.

1. Set $\varepsilon = \frac{1}{2d-1}$ and then partition the set of objects into big (of length at least $\varepsilon\beta$) and small (of length smaller than $\varepsilon\beta$). Set $m^* = n$.
2. **For** each set S^B of 2^d big objects **do** :

   Use Lenstra's algorithm and binary search over the set $V = \{1, 2, \ldots, m^*\}$ to find the smallest value $j \in V$, if any, for which MILP(j, S^B) has a solution.

   If a value $j < m^*$ was found for which MILP(j, S^B) has a solution, then set $m^* = j$ and let (x^+, y^+) be the solution computed by Lenstra's algorithm for MILP(j, S^B).
3. Round (x^+, y^+) as described in Section 3 and output the corresponding packing of the objects into $m^* + 1$ bins.

We are now ready to prove our main theorem.

Theorem 1.1 *There is a* $d^{O(d2^d)} \times 2^{O(8^d)} \times O((\log^2 n + \log \beta)^3 \log^5 n)$ *time algorithm for the cutting stock problem with a constant number* d *of different object types, that solves the problem using at most* $OPT + 1$ *bins, where* OPT *is the value of an optimum solution.*

Proof: By Lemmas 2.1 and 3.1, algorithm BinPacking produces a solution for the cutting stock problem that uses at most $OPT+1$ bins. By Theorem 4.1 the time complexity of step 2 of BinPacking is $d^{O(d2^d)} \times 2^{O(8^d)} \times O((\log^2 n + \log \beta)^3 \log^5 n)$ as the algorithm needs to solve $O\left(\binom{\varepsilon^{-d}}{2^d} \log n\right)$ mixed integer linear programs.

Step 3 of the algorithm needs to find a basic feasible solution for LP1. A solution for LP1 can be computed using algorithm Ellipsoid in $O(d^{10+17d/2} 2^{24d} (\log^2 n + \log \beta)^3 \log n)$ time. This solution can be transformed into a basic feasible solution by using the algorithm of Dantzig and Thapa (Algorithm 4.2 in [2]) in $O(d^4 2^{7d} \log^2 n)$ time. So the time complexity of algorithm BinPacking is $d^{O(d2^d)} \times 2^{O(8^d)} \times O((\log^2 n + \log \beta)^3 \log^5 n)$. □

6 Conclusion

We have presented an approximation algorithm for the cutting stock problem that packs a given set of objects into at most $OPT + 1$ bins, where OPT is the value of an optimum solution. The time complexity of the algorithm is polynomial in $\log n$ and, interestingly, the exponent of $\log n$ in the running time is independent of d. However, the dependency of the running time on d is doubly exponential. We note that the time complexity of the algorithm could be made simply exponential in d if we could strengthen the result of Eisenbrand and Shmonin in [4] by showing that IP has an optimum solution with at most $p(d)$ positive variables, for some polynomial function p.

References

[1] E.G. Coffman, Jr., M.R. Garey, and D.S. Johnson, Approximation algorithms for bin packing: a survey, in *Approximation algorithms for NP-hard problems*, edited by Dorit S. Hochbaum, PWS Publishing Company, 1997, 46–86.

[2] G.B. Dantzig and M.N. Thapa, *Linear Programming: Theory and extensions*, Springer, 2003.

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

[4] F. Eisenbrand and G. Shmonin, Carathéorody bounds for integer cones. *Operations Research Letters*, 34, 2006, 564–568.

[5] C. Filippi and A. Agnetis, An asymptotically exact algorithm for the high-multiplicity bin packing problem, *Mathematical Programming*, 104 (1), 2005, 21–57.

[6] C. Filippi, On the bin packing problem with a fixed number of object weights, *European Journal of Operational Research*, 181, 2007, 117–126.

[7] P.C. Gilmore and R.E. Gomory, A linear programming approach to the cutting stock problem, *Operations Research*, 9, 1961, 849–859.

[8] M. Grötschel, L. Lovász, A. Schrijver, The ellipsoid method and its consequences in Combinatorial Optimization, *Combinatorica*, 1 (2), 1981, 169–197.

[9] M. Grötschel, L. Lovász, A. Schrijver, *Geometric algorithms and combinatorial optimization*, Springer-Verlag, Berlin, 1988.

[10] D.S. Hochbaum and R. Shamir, Strongly polynomial algorithms for the high multiplicity scheduling problem, *Operations Research*, 39, 1991, 648–653.

[11] E. Kaltofen and G. Villard, On the complexity of computing determinants, *Computational Complexity*, 13 (3-4), 2004, 91–130.

[12] R. Kannan and A. Bachem, Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix, *SIAM Journal on Computing*, 8, 1979, 499-507.

[13] R. Kannan, Minkowski's convex body theorem and integer programming, *Mathematics of Operations Research*, 12 (3), 1987, 415–440.

[14] N. Karmakar and R.M. Karp, An efficient approximation scheme for the one-dimensional bin packing problem, Proceedings FOCS, 1982, 312-520.

[15] L.G. Khachiyan, A polynomial algorithm in linear programming, *Dokl. Akad Nauk SSSR*, 244, 1979, 1093-1096 (English translation: *Soviet Math Dokl.* 20, 1979, 191-194).

[16] Lenstra A.K., Lenstra H.W., Jr. and Lovász, L., Factoring Polynomials with rational coefficients, *Math. Ann.*, 261, 1982, 515–534.

[17] H.W. Lenstra, Jr., Integer programming with a fixed number of variables, *Mathematics of Operations Research*, 8 (4), 1983, 538–548.

[18] L. Lovász, *Complexity of algorithms*, 1998.

[19] O. Marcotte, The cutting stock problem and integer rounding, *Mathematical Programming*, 33, 1985, 82–92.

[20] S.T. McCormick, S.R. Smallwood, F.C.R. Spieksma, A polynomial algorithm for multiprocessor scheduling with two job lengths, *Math. Op. Res.*,26, 2001, 31–49.

[21] J.B. Orlin, A polynomial algorithm for integer programming covering problems satisfying the integer round-up property, *Mathematical Programming*, 22, 1982, 231–235.

[22] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: Algorithms and complexity*, Prentice-Hall, Inc., 1982.

[23] A. Schrijver, *Theory of Linear and Integer Programming*, John Wiley, 1986.