

Algorithms for 3D Guillotine Cutting Problems: Unbounded Knapsack, Cutting Stock and Strip Packing*

Thiago A. de Queiroz, Flávio K. Miyazawa,

Instituto de Computação, IC, UNICAMP,
13084-971, Campinas, SP
e-mail: tqueiroz@ic.unicamp.br, fkm@ic.unicamp.br

Yoshiko Wakabayashi,

Instituto de Matemática e Estatística, IME, USP
05508-090, São Paulo, SP
e-mail: yw@ime.usp.br

Eduardo C. Xavier

Instituto de Computação, IC, UNICAMP,
13084-971, Campinas, SP
e-mail: ecx@ic.unicamp.br

ABSTRACT

We present algorithms for the following three-dimensional (3D) guillotine cutting problems: Unbounded Knapsack, Cutting Stock and Strip Packing. We consider the case where the items have fixed orientation and the case where orthogonal rotations around all axes are allowed. For the Unbounded 3D Knapsack problem, we extend the recurrence formula proposed by Beasley for the Rectangular Knapsack Problem and present a dynamic programming algorithm that uses *reduced raster points*. We also consider a variant of the Unbounded Knapsack problem in which the cuts must be staged. For the 3D Cutting Stock problem and its variants in which the bins have different sizes (and the cuts must be staged), we present column generation based algorithms. Modified versions of the algorithms for the 3D Cutting Stock problems with stages are then used to build algorithms for the 3D Strip Packing problem and its variants. The computational tests performed with the algorithms described in this paper indicate that they are useful to solve instances of moderate size.

KEYWORDS. Guillotine cutting; three-dimensional cutting stock; unbounded knapsack; strip packing; column generation.

*This research was partially supported by CAPES, CNPq and FAPESP.

1 Introduction

The problem of cutting large objects to produce smaller objects has been largely investigated, specially when the objects are one- or two-dimensional. We focus here on the three-dimensional case, restricted to guillotine cuts. In this context, the large objects to be cut are called *bins*, and the small objects (to be produced) are called *boxes* or *items*.

A *guillotine cut* is a cut that is parallel to one of the sides of the bin and goes from one side to the opposite one. For the problems considered here, not only the first cut, but all the subsequent cuts on the smaller parts must be of guillotine type.

A *k-staged cutting* is a sequence of at most k stages of cuts, each stage of which is a set of parallel guillotine cuts performed on the objects obtained in the previous stage. Moreover, the cuts in each stage must be orthogonal to the cuts performed in the previous stage. We assume, without loss of generality, that the cuts are infinitely thin.

Each possible way of cutting a bin is called a *cutting pattern* (or simply, *pattern*). To represent the patterns (and the cuts to be performed), we consider the Euclidean space \mathbb{R}^3 with the xyz coordinate system, and assume that the length, width and height of an object is represented in the axes x , y and z , respectively. We say that a bin (or box) B has dimension (L, W, H) , and write $B = (L, W, H)$, if it has length L , width W and height H . For such a bin, we assume that the position $(0, 0, 0)$ corresponds to its bottom-left front corner, and position (L, W, H) represents its top-right behind corner. Analogously, the same terminology is used for the boxes.

The problems considered in this paper are the following.

THREE-DIMENSIONAL UNBOUNDED KNAPSACK PROBLEM (3DK): We are given a bin $B = (L, W, H)$ and a list T of n types of boxes, each type i with dimension (l_i, w_i, h_i) and value v_i , $i = 1, \dots, n$. We wish to determine how to cut B to produce boxes of some of the types in T so as to maximize the total value of the boxes that are produced. Here, no bound is imposed on the number of boxes of each type that can be produced (some types may not occur). An instance of this problem is denoted by a tuple (L, W, H, l, w, h, v) , where $l = (l_1, \dots, l_n)$ and w, h and v are lists defined likewise.

In the problem 3DK there is no demand associated with a box. Differently, in the cutting stock and strip packing problems, to be defined next, there is a demand associated with each type of box. In this case, a box of type i with dimension (l_i, w_i, h_i) and demand d_i is denoted by a tuple (l_i, w_i, h_i, d_i) ; and a set of n types of boxes is denoted by (l, w, h, d) , where $l = (l_1, \dots, l_n)$, and w, h and d are lists defined analogously.

THREE-DIMENSIONAL CUTTING STOCK PROBLEM (3CS): Given an unlimited quantity of identical bins $B = (L, W, H)$ and a set of n types of boxes (l, w, h, d) , determine how to cut the smallest possible number of bins B so as to produce d_i units of each box type i , $i = 1, \dots, n$. An instance for this problem is given by a tuple (L, W, H, l, w, h, d) .

THREE-DIMENSIONAL CUTTING STOCK PROBLEM WITH VARIABLE BIN SIZES (3CSV): Given an unlimited quantity of b different types of bins B_1, \dots, B_b , each bin B_j with dimension (L_j, W_j, H_j) and value V_j , and a set of n types of boxes (l, w, h, d) , determine how to cut the given bins to generate d_i units of each box type $i, i = 1, \dots, n$, so that the total value of the bins used is the smallest possible. (Some types of bins may not be used.) An instance of this problem is given by a tuple (L, W, H, V, l, w, h, d) .

THREE-DIMENSIONAL STRIP PACKING PROBLEM (3SP): Given a 3D strip $B = (L, W, \infty)$ (a bin with bottom dimension (L, W) and infinite height) and a set of n types of boxes (l, w, h, d) , determine how to cut the strip B so that d_i units of each box type $i, i = 1, \dots, n$, is produced and the height of the part of the strip that is used is minimized. We require the cuts to be k -staged (and horizontal in the first stage); furthermore, the distance between any two subsequent cuts must be at most A (a common restriction imposed by the cutting machines).

For the problems above mentioned, we also consider variants in which orthogonal rotations of the boxes are allowed. These variants are called $3DK^r$, $3CS^r$, $3CSV^r$ and $3SP^r$, respectively. When we allow a box $b_i = (l_i, w_i, h_i)$ to be rotated, this means that its dimension can be considered as being any of the six permutations of (l_i, w_i, h_i) .

Throughout the paper, the dimensions of the bins and the boxes are assumed to be integer. For the staged variant of the 3CS problem, we assume that the first cutting stage is performed in the horizontal direction, that is, parallel to the xy -plane, denoted as ' H' '; followed by a cut in the lateral vertical direction, that is, parallel to the yz -plane, denoted as ' V' '; and then, a cut in the frontal vertical direction (parallel to the xz -plane), denoted as ' D' ' (a depth cut).

All problems above mentioned are NP-hard. The one- and two-dimensional versions of the unbounded knapsack problem have been studied since the sixties. Herz (1972) presented a recursive algorithm for the two-dimensional version, called 2DK, which obtains canonical patterns making use of *discretization points*. Beasley (1985) proposed a dynamic programming formulation that uses the discretization points to solve the staged and non-staged variants of the 2DK problem. Cintra et al. (2008) presented a dynamic programming approach for the 2DK problem and some of its variants. They were able to solve in a small computational time instances of the OR-Library for which no optimal solution was known. Diedrich et al. (2008) proposed approximation algorithms for the 3DK problem with approximation ratios $(9 + \epsilon)$, $(8 + \epsilon)$ and $(7 + \epsilon)$; and for the $3DK^r$ problem they designed an approximation algorithm with ratio $(5 + \epsilon)$.

The first column generation approaches for the one- and two-dimensional Cutting Stock problem, called 1CS and 2CS, were proposed by Gilmore and Gomory (1961, 1963, 1965). They also considered the variant of 2CS in which the bins have different sizes, called 2CSV, and proposed the k -staged version. Alvarez-Valdes et al. (2002) also investigated the 2CS problem, for which they presented a column generation based algorithm that uses the recurrence formulas described in Beasley (1985). Puchinger and Raidl (2007) presented a branch-and-price algorithm for the 3-staged case of 2CS.

For the 3CS problem with unit demand, Csirik and van Vliet (1993) presented an algorithm with asymptotic performance ratio of at most 4.84. Miyazawa and Wakabayashi (2009) showed that the version with orthogonal rotation is as difficult to approximate as the oriented version, and they also presented a 4.89-approximation algorithm for this case. Cintra et al. (2007) showed that these approximation ratios are also preserved in the case of arbitrary demands.

Some approximation algorithms have been proposed for the two-dimensional Strip Packing (2SP) problem. Kenyon and Rémila (2000) presented an AFPTAS for the oriented case and Jansen and van Stee (2005) proposed a PTAS for the case in which rotations are allowed. Other approaches like branch-and-bound and integer linear programming models have also been proposed by Hifi (1998), Lodi et al. (2004) and Martello et al. (2003). Cintra et al. (2008) presented a column generation based algorithm for the staged 2SP problem with and without rotations. For the three-dimensional case (3SP), Jansen and Solis-Oba (2006) proposed an algorithm with asymptotic ratio of $2 + \epsilon$. This ratio was improved to 1.691 by Bansal et al. (2007).

The results we present in this paper are basically extensions of the approaches obtained by Cintra et al. (2008), combined with the use of *reduced raster points* (an idea introduced by Scheithauer). Section 2 focus on the Unbounded Knapsack problems 3DK, 3DK^r and its variants in which the cuts must be k -staged. For all these problems we present exact dynamic programming algorithms.

For the Cutting Stock problems 3CS, 3CS^r, 3CSV and 3CSV^r, we present in Sections 3 and 4 column generation based algorithms that use as a routine the algorithm proposed for the Unbounded Knapsack problem. In Section 5 we focus on the 3SP problem and its variants (with rotations and/or k -staged cuts). The algorithms for all these problems use a column generation technique. The computational experiments with the algorithms described here are reported in Section 6.

2 The 3D Unbounded Knapsack Problem

The algorithms we describe in this section are based on the use of the so-called *raster points*. These are a special sub-set of the *discretization points* (positions where guillotine cutting can be performed) and were first presented by Scheithauer (1997).

Discretization points were used (for the two-dimensional case) by Herz (1972) and also by Beasley (1985) in a dynamic programming algorithm. More recently, Birgin et al. (2008) used raster points to deal with the packing of identical rectangles in another rectangle, obtaining very good results.

Let (L, W, H, l, w, h, v) be an instance of the 3DK problem. A *discretization point of length* (respectively, *of the width* and *of the height*) is a value $i \leq L$ (respectively, $j \leq W$ and $k \leq H$) obtained by an integer conic combination of $l = (l_1, \dots, l_n)$ (respectively, $w = (w_1, \dots, w_n)$ and $h = (h_1, \dots, h_n)$). We denote by P , Q and R the set of all discretization points of length, width and height, respectively.

The set of *reduced raster points* \tilde{P} (relative to P) is defined as $\tilde{P} = \{(L - r) : r \in P\}$, where

$\langle s \rangle = \max\{t \in P : t \leq s\}$. In the same way we define the sets \tilde{Q} (relative to Q) and \tilde{R} (relative to R). To simplify notation, we refer to these points as r -points. An important feature of the r -points is the fact that they are sufficient to generate all possible cutting patterns (that is, for every pattern there is an equivalent one in which the cuts are performed only on r -points). As the set of r -points is a subset of the discretization points, this may reduce the time for the search of an optimum pattern. To refer to these points we define, for any rational number $x_r \leq L$, $y_r \leq W$ and $z_r \leq H$, the following functions:

$$\begin{aligned} p(x_r) &= \max\{i \mid i \in \tilde{P}, i \leq x_r\}; \\ q(y_r) &= \max\{j \mid j \in \tilde{Q}, j \leq y_r\}; \\ r(z_r) &= \max\{k \mid k \in \tilde{R}, k \leq z_r\}. \end{aligned} \tag{1}$$

The algorithm to compute the r -points of a given instance is denoted by RRP. First, it generates the discretization points using the algorithm DDP (*Discretization using Dynamic Programming*) presented by Cintra et al. (2008), and then, it selects those that are r -points, following the above definition.

The time complexity of the algorithm RRP is the same of the algorithm DDP, that is, $O(nD)$ where $D := \max\{L, W, H\}$. This algorithm is pseudo-polynomial; so when D is small, or the dimensions of the boxes are not so small compared to the dimension of the bin, then the algorithm has a good performance, as shown by the computational tests, presented in Section 6.

2.1 Algorithm for the 3DK problem

Let $I = (L, W, H, l, w, h, v)$ be an instance of the 3DK problem, and let \tilde{P} , \tilde{Q} and \tilde{R} be the set of r -points, as defined previously. Let $G(L, W, H)$ be the value of an optimum guillotine pattern for the instance I . The function G can be calculated by the recurrence formula (2). In this formula, $g(l^*, w^*, h^*)$ denotes the maximum value of a box that can be cut in a bin of dimension (l^*, w^*, h^*) . This value is 0 if no box can be cut in such a bin.

$$G(l^*, w^*, h^*) = \max \left\{ \begin{array}{l} g(l^*, w^*, h^*); \\ \max\{G(l', w^*, h^*) + G(p(l^* - l'), w^*, h^*) \mid l' \in \tilde{P}, l' \leq l^*/2\}; \\ \max\{G(l^*, w', h^*) + G(l^*, q(w^* - w'), h^*) \mid w' \in \tilde{Q}, w' \leq w^*/2\}; \\ \max\{G(l^*, w^*, h') + G(l^*, w^*, r(h^* - h')) \mid h' \in \tilde{R}, h' \leq h^*/2\}. \end{array} \right\} \tag{2}$$

We note that the recurrence above is an extension of the recurrence formula of Beasley (1985). It can be solved by the algorithm DP3DK (Dynamic Programming for the Three-dimensional Unbounded Knapsack), which we describe next.

First, the algorithm DP3DK calls the algorithm RRP to compute the sets \tilde{P} , \tilde{Q} and \tilde{R} (lines 1.1 – 1.2). Then (in the lines 1.3 – 1.8), the algorithm stores in $G[i, j, k]$ for each bin of dimension (p_i, q_j, r_k) , with $p_i \in \tilde{P}$, $q_j \in \tilde{Q}$ and $r_k \in \tilde{R}$, the maximum value of a box that can be cut in such a bin. The variable $item[i, j, k]$ indicates the corresponding box type, and the variable $guil[i, j, k]$ indicates the direction of the guillotine cut if its value is not *nil*. The value *nil* indicates that no cut has to be performed, and $pos[i, j, k]$ contains the position (point) at x , y or z -axis where the cut has

Algorithm 1: DP3DK

Input : An instance $I = (L, W, H, l, w, h, v)$ of the 3DK problem.

Output: An optimum solution for I .

```
1.1  $\tilde{P} \leftarrow \text{RRP}(L, l)$ ,  $\tilde{Q} \leftarrow \text{RRP}(W, w)$ ,  $\tilde{R} \leftarrow \text{RRP}(H, h)$ 
1.2 Let  $\tilde{P} = (p_1 < p_2 < \dots < p_m)$ ,  $\tilde{Q} = (q_1 < q_2 < \dots < q_s)$ ,  $\tilde{R} = (r_1 < r_2 < \dots < r_u)$ 
1.3 for  $i \leftarrow 1$  to  $m$  do
1.4   for  $j \leftarrow 1$  to  $s$  do
1.5     for  $k \leftarrow 1$  to  $u$  do
1.6        $G[i, j, k] \leftarrow \max(\{v_d \mid 1 \leq d \leq n; l_d \leq p_i, w_d \leq q_j \text{ and } h_d \leq r_k\} \cup \{0\})$ 
1.7        $item[i, j, k] \leftarrow \max(\{d \mid 1 \leq d \leq n; l_d \leq p_i, w_d \leq q_j, h_d \leq r_k \text{ and } v_d = G[i, j, k]\} \cup \{0\})$ 
1.8        $guil[i, j, k] \leftarrow nil$ 
1.9 for  $i \leftarrow 1$  to  $m$  do
1.10   for  $j \leftarrow 1$  to  $s$  do
1.11     for  $k \leftarrow 1$  to  $u$  do
1.12        $nn \leftarrow \max(d \mid 1 \leq d \leq i \text{ and } p_d \leq \lfloor p_i/2 \rfloor)$ 
1.13       for  $x \leftarrow 1$  to  $nn$  do
1.14          $t \leftarrow \max(d \mid 1 \leq d \leq m \text{ and } p_d \leq p_i - p_x)$ 
1.15         if  $G[i, j, k] < G[x, j, k] + G[t, j, k]$  then
1.16            $G[i, j, k] \leftarrow G[x, j, k] + G[t, j, k]$ 
1.17            $pos[i, j, k] \leftarrow p_x$ 
1.18            $guil[i, j, k] \leftarrow 'V'$  // Vertical cut, parallel to  $yz$ -plane
1.19        $nn \leftarrow \max(d \mid 1 \leq d \leq j \text{ and } q_d \leq \lfloor q_j/2 \rfloor)$ 
1.20       for  $y \leftarrow 1$  to  $nn$  do
1.21          $t \leftarrow \max(d \mid 1 \leq d \leq s \text{ and } q_d \leq q_j - q_y)$ 
1.22         if  $G[i, j, k] < G[i, y, k] + G[i, t, k]$  then
1.23            $G[i, j, k] \leftarrow G[i, y, k] + G[i, t, k]$ 
1.24            $pos[i, j, k] \leftarrow q_y$ 
1.25            $guil[i, j, k] \leftarrow 'D'$  // Depth cut (vertical, parallel to  $xy$ -plane)
1.26        $nn \leftarrow \max(d \mid 1 \leq d \leq k \text{ and } r_d \leq \lfloor r_k/2 \rfloor)$ 
1.27       for  $z \leftarrow 1$  to  $nn$  do
1.28          $t \leftarrow \max(d \mid 1 \leq d \leq u \text{ and } r_d \leq r_k - r_z)$ 
1.29         if  $G[i, j, k] < G[i, j, z] + G[i, j, t]$  then
1.30            $G[i, j, k] \leftarrow G[i, j, z] + G[i, j, t]$ 
1.31            $pos[i, j, k] \leftarrow r_z$ 
1.32            $guil[i, j, k] \leftarrow 'H'$  // Horizontal cut, parallel to  $xy$ -plane
1.33 return  $G(m, s, u)$ .
```

to be made.

Next, (in the lines 1.9 – 1.32) the algorithm iteratively finds the optimum solution for a bin of the current iteration by the best combination of solutions already known for smaller bins. In other words, for a bin of dimension (p_i, q_j, r_k) , the optimum solution is obtained in the following way: for each possible r -point p_x where a vertical cut $'V'$ can be performed, the algorithm determines the best solution by comparing the best solution so far with one that can be obtained with a vertical

cut ' V' ' (lines 1.12 – 1.18); repeat the same process for a depth cut ' D' ' (lines 1.19 – 1.25), and for a horizontal cut ' H' ' (lines 1.26 – 1.32). Finally, (at line 1.33) the algorithm returns the value of an optimum solution.

The algorithm avoids generating symmetric patterns by considering, in each direction, r -points up to half of the size of the respective bin (see lines 1.12, 1.19 and 1.26). In fact, consider a bin of width ℓ and an orthogonal guillotine cut in the x -axis at position $t \in \tilde{P}$, for $t > \frac{\ell}{2}$. This cut divides the current bin into two smaller bins: one with length t and the other with length $\ell - t$. The patterns that can be obtained with these two smaller bins can also be obtained using a guillotine cut at position $t' = \ell - t$ on the original bin. If $t' \in \tilde{P}$, then such a cut is considered as $t' \leq \frac{\ell}{2}$; if $t' \notin \tilde{P}$ then the cut at position $\langle t' \rangle$ generates two bins in which we can obtain the same patterns considered for the cut made on t' .

The time complexity of the algorithm DP3DK is directly affected by the time complexity of the algorithm RRP (line 1.1). Therefore, the time complexity of the algorithm DP3DK is $O(nL + nW + nH + m^2su + ms^2u + msu^2)$ where m , s and u are the total number of r -points of \tilde{P} , \tilde{Q} and \tilde{R} , respectively. On the other hand, the space complexity of the DP3DK is $O(L + W + H + msu)$.

2.2 Algorithm for the k -staged 3DK problem

We present now a dynamic programming algorithm to solve the k -staged 3DK and 3DK^r problems. We consider that in each stage a different cut direction is considered, following the cyclic order: $H - V - D - H - \dots$. A cutting stage may possibly be empty (when no cut has to be performed), and in this case, after it, the next cutting stage is considered.

In the next recurrence formulas, $G(l^*, w^*, h^*, k, V)$, $G(l^*, w^*, h^*, k, H)$ and $G(l^*, w^*, h^*, k, D)$ denote the value of an optimum guillotine k -staged solution for a bin of dimension (l^*, w^*, h^*) . The parameters V , H and D indicate the direction of the first cutting stage.

$$G(l^*, w^*, h^*, 0, V \text{ or } H \text{ or } D) := g(l^*, w^*, h^*);$$

$$G(l^*, w^*, h^*, k, V) := \max \left\{ \begin{array}{l} G(l^*, w^*, h^*, k-1, D); \\ \max \{ G(l', w^*, h^*, k-1, D) + G(p(l^* - l'), w^*, h^*, k, V) \mid l' \in \tilde{P}, l' \leq l^*/2 \} \end{array} \right\},$$

$$G(l^*, w^*, h^*, k, H) := \max \left\{ \begin{array}{l} G(l^*, w^*, h^*, k-1, V); \\ \max \{ G(l^*, w', h^*, k-1, V) + G(l^*, q(w^* - w'), h^*, k, H) \mid w' \in \tilde{Q}, w' \leq w^*/2 \} \end{array} \right\}, \quad (3)$$

$$G(l^*, w^*, h^*, k, D) := \max \left\{ \begin{array}{l} G(l^*, w^*, h^*, k-1, H); \\ \max \{ G(l^*, w^*, h', k-1, H) + G(l^*, w^*, r(h^* - h'), k, D) \mid h' \in \tilde{R}, h' \leq h^*/2 \} \end{array} \right\}.$$

The algorithm DPS3DK (Dynamic Programming for the k -staged 3DK) described in what follows solves the recurrence formulas above. It is very similar to the former algorithm (for the non-staged case). It computes first the sets \tilde{P} , \tilde{Q} and \tilde{R} and stores in $G[0, i, j, l]$ the maximum value of a box that can be cut on a bin of dimension (p_i, q_j, r_l) (lines 2.1–2.8). Then, the algorithm computes, for each stage b , the best solution for cuts done only in one direction, and it uses this information to compute the best solution for the next stage, and so on (guaranteeing that two subsequent stages have cuts in different directions). This is the basic difference between the algorithm DP3DK and DPS3DK. In some cases, the best solution for the stage $b - 1$ is also the solution for the stage b , and no cut is needed in this case. In this case, the value *nil* is stored in the variable *guil* (line 2.15).

The algorithm DPS3DK stores in $G[k, i, j, l]$ the optimum k -staged solution for a bin with dimension (p_i, q_j, r_l) . The variables $guil[k, i, j, l]$, $pos[k, i, j, l]$ and $item[k, i, j, l]$ indicate, respectively, the direction of the first guillotine cut, the position of this cut at x , y or z -axis, and the corresponding item if no cut has to be made in the bin.

The time complexity of the algorithm DPS3DK is the same of the algorithm DP3DK multiplied by the number of cutting stages k . This is also true for the space complexity. On the other hand, if k is limited by some constant, then DPS3DK have the same complexity of the algorithm DP3DK.

2.3 The 3DK^r problem and its variant with k stages

The problem 3DK^r is a variant of 3DK that allows orthogonal rotations of the boxes (to be cut) around any of the axes. This means that each box of type i can be considered as having one of the six dimensions obtained by the permutations of l_i, w_i, h_i (as long as they are feasible). We refer to these feasible dimensions as $PERM(l_i, w_i, h_i)$.

The problem 3DK^r can be solved with the algorithms for the problem 3DK. For that, we only need a preprocessing phase to change the instance. Given an instance I for the 3DK^r, we construct another instance I' by adding to I , for each box i in I of dimension (l_i, w_i, h_i) , the set of new types of boxes $PERM(l_i, w_i, h_i)$, all with the same value v_i . Then, we solve the new instance I' with the algorithm 3DK.

For the k -staged 3DK^r problem, we proceed analogously. We denote the corresponding algorithms for these problems by DP3DK^r and DPS3DK^r.

3 The Three-dimensional Cutting Stock Problem

We first present some heuristics which will be used as subroutines in the column generation approach described in this section for the 3CS problem. We also compare the sole performance of these heuristics with the performance of the column generation approach.

Algorithm 2: DPS3DK

Input : An instance $I = (L, W, H, l, w, h, v, k)$ of the k -staged 3DK problem.

Output: An optimum k -staged solution for I .

```
2.1  $\tilde{P} \leftarrow \text{RRP}(L, l_1, \dots, n)$ ,  $\tilde{Q} \leftarrow \text{RRP}(W, w_1, \dots, n)$ ,  $\tilde{R} \leftarrow \text{RRP}(H, h_1, \dots, n)$ 
2.2 Let  $\tilde{P} = (p_1 < p_2 < \dots < p_m)$ ,  $\tilde{Q} = (q_1 < q_2 < \dots < q_s)$ ,  $\tilde{R} = (r_1 < r_2 < \dots < r_u)$ 
2.3 for  $i \leftarrow 1$  to  $m$  do
2.4   for  $j \leftarrow 1$  to  $s$  do
2.5     for  $l \leftarrow 1$  to  $u$  do
2.6        $G[0, i, j, l] \leftarrow \max(\{v_d \mid 1 \leq d \leq n; l_d \leq p_i, w_d \leq q_j \text{ and } h_d \leq r_l\} \cup \{0\})$ 
2.7        $item[0, i, j, l] \leftarrow \max(\{d \mid 1 \leq d \leq n; l_d \leq p_i, w_d \leq q_j, h_d \leq r_l \text{ and } v_d = G[0, i, j, l]\} \cup \{0\})$ 
2.8        $guil[0, i, j, l] \leftarrow nil$ 
2.9 if  $(k \bmod 3) = 1$  then  $previous \leftarrow 'V'$  else if  $(k \bmod 3) = 2$  then  $previous \leftarrow 'D'$  else
    $previous \leftarrow 'H'$ 
2.10 for  $b \leftarrow 1$  to  $k$  do
2.11   for  $i \leftarrow 1$  to  $m$  do
2.12     for  $j \leftarrow 1$  to  $s$  do
2.13       for  $l \leftarrow 1$  to  $u$  do
2.14          $G[b, i, j, l] \leftarrow G[b-1, i, j, l]$ 
2.15          $guil[b, i, j, l] \leftarrow nil$ 
2.16         if  $previous = 'D'$  then
2.17            $nn \leftarrow \max(d \mid 1 \leq d \leq m \text{ and } p_d \leq \lfloor p_i/2 \rfloor)$ 
2.18           for  $x \leftarrow 1$  to  $nn$  do
2.19              $t \leftarrow \max(d \mid 1 \leq d \leq m \text{ and } p_d \leq p_i - p_x)$ 
2.20             if  $G[b, i, j, l] < G[b-1, x, j, l] + G[b, t, j, l]$  then
2.21                $G[b, i, j, l] \leftarrow G[b-1, x, j, l] + G[b, t, j, l]$ 
2.22                $pos[b, i, j, l] \leftarrow p_x$ 
2.23                $guil[b, i, j, l] \leftarrow 'V'$ 
2.24            $previous \leftarrow 'V'$ 
2.25         else if  $previous = 'V'$  then
2.26            $nn \leftarrow \max(d \mid 1 \leq d \leq u \text{ and } r_d \leq \lfloor r_l/2 \rfloor)$ 
2.27           for  $z \leftarrow 1$  to  $nn$  do
2.28              $t \leftarrow \max(d \mid 1 \leq d \leq u \text{ and } r_d \leq r_l - r_z)$ 
2.29             if  $G[b, i, j, l] < G[b-1, i, j, z] + G[b, i, j, t]$  then
2.30                $G[b, i, j, l] \leftarrow G[b-1, i, j, z] + G[b, i, j, t]$ 
2.31                $pos[b, i, j, l] \leftarrow r_z$ 
2.32                $guil[b, i, j, l] \leftarrow 'H'$ 
2.33            $previous \leftarrow 'H'$ 
2.34         else
2.35            $nn \leftarrow \max(d \mid 1 \leq d \leq s \text{ and } q_d \leq \lfloor q_j/2 \rfloor)$ 
2.36           for  $y \leftarrow 1$  to  $nn$  do
2.37              $t \leftarrow \max(d \mid 1 \leq d \leq s \text{ and } q_d \leq q_j - q_y)$ 
2.38             if  $G[b, i, j, l] < G[b-1, i, y, l] + G[b, i, t, l]$  then
2.39                $G[b, i, j, l] \leftarrow G[b-1, i, y, l] + G[b, i, t, l]$ 
2.40                $pos[b, i, j, l] \leftarrow q_y$ 
2.41                $guil[b, i, j, l] \leftarrow 'D'$ 
2.42            $previous \leftarrow 'D'$ 
2.43 return  $G(k, m, s, u)$ .
```

3.1 Primal heuristics for the three-dimensional cutting stock problem

The primal heuristic we present here – HFF3 – is a hybrid heuristic that generates patterns divided into levels using an algorithm for the 2CS problem combined with an algorithm for the 1CS problem. The first algorithm generates the levels and the second algorithm pack these levels into bins.

The algorithms for the 1CS problem that we use here are the well-known *First Fit* (FF), and *First Fit Decreasing* (FFD) algorithms. We describe here only the algorithm we use for the 2CS problem. It is called HFF2 (*Hybrid First Fit 2*), as it is based on the *Hybrid First Fit* algorithm, designed by Chung et al. (1982). (For convenience, we describe it as ‘packing’ algorithm.)

The algorithm HFF2 includes two variants: HFF^l and HFF^w . Without loss of generality, we suppose that each box has unit demand. Thus, for an instance (L, W, l, w) of the 2CS problem, the algorithm HFF^l considers the items sorted decreasingly by length ($l_1 \geq l_2 \geq \dots \geq l_n$). Then, it considers each item i as a one-dimensional item of size w_i , and applies the algorithm *First Fit*, $\text{FF}(W, w)$, to obtain a packing of those items into recipients S_1, \dots, S_m , which we call *strips*. Finally, each strip S_i is considered as a one-dimensional item of size $s_i = \max\{l_j : j \in S_i\}$ and the algorithm $\text{FFD}(L, s)$ is applied to pack these strips into rectangular (2D) bins. The strips of the algorithm HFF^l are generated in the length direction, whereas the HFF^w generates the strips in the width direction. The algorithm HFF2 executes both variants and returns a solution with the best value.

To deal with the 3CS^r problem (the variant of 3CS in which orthogonal rotations are allowed), we denote by HFF^x (respectively, HFF^y) the variant of the algorithm HFF2 that rotates the rectangles i to obtain $w_i \geq l_i$ (respectively, $l_i \geq w_i$) before applying the algorithms HFF^l and HFF^w . The algorithm HFF2^r executes these algorithms and returns the best solution found.

We present now the algorithm H3CS, which is the core of the algorithm HFF3 (for the 3CS problem). It first sorts the items decreasingly by height. Then, it iteratively generates a new level using an algorithm for the 2CS problem, privileging the packing of the higher items into each level. For each item i , the largest possible number of them is packed without violating its demand and keeping the packing in one level (see line 3.7). When all levels are generated, they are packed into bins by an algorithm for the 1CS problem (see line 3.10).

We denote by HFF3_h (respectively, HFF3_h^r) the algorithm H3CS that uses the algorithms FFD and HFF2 (respectively, HFF2^r) as subroutines. Observe that the algorithms HFF3_h and HFF3_h^r generate and pack the levels in the height direction. We denote by HFF3_w and HFF3_w^r (respectively, HFF3_l and HFF3_l^r) the variants where levels are generated and packed in the width (respectively, length) direction. Finally, the algorithm HFF3 (respectively, HFF3^r) executes the algorithms HFF3_h , HFF3_w and HFF3_l (respectively, HFF3_h^r , HFF3_w^r and HFF3_l^r) and returns the best packing obtained.

Algorithm 4: Simplex_{CS}

Input : An instance $I = (L, W, H, l, w, h, d)$ of the 3CS problem.

Output: An optimum solution for the linear program (4)

Subroutine: An algorithm \mathcal{A} for the 3DK or 3DK^r problem.

- 4.1 Let $x \leftarrow d$ and $B \leftarrow I_{n \times n}$
 - 4.2 Solve $y^T B = [1, 1, \dots, 1]_n^T$
 - 4.3 $z \leftarrow \mathcal{A}(L, W, H, l, w, h, y)$
 - 4.4 **if** $y^T z \leq 1$ **then return** (B, x) **else** solve $Bw = z$
 - 4.5 Let $t \leftarrow \min(\frac{x_j}{w_j} \mid 1 \leq j \leq n, w_j > 0)$ and $s \leftarrow \min(j \mid 1 \leq j \leq n, \frac{x_j}{w_j} = t)$
 - 4.6 **for** $i \leftarrow 1$ **to** n **do**
 - 4.7 $B_{i,s} \leftarrow z_i$
 - 4.8 **if** $i = s$ **then** $x_i \leftarrow t$ **else** $x_i \leftarrow x_i - w_i t$
 - 4.9 Go to line 4.2
-

We present below the algorithm CG3CS that solves the 3CS problem. It receives the solution (possibly fractional) found by the algorithm Simplex_{CS} and returns an integer solution for the 3CS problem. If needed, this algorithm uses a primal heuristic to obtain a cutting pattern that causes a perturbation of some residual instance (see line 14 in 5).

Algorithm 5: CG3CS

Input : An instance $I = (L, W, H, l, w, h, d)$ of the 3CS problem.

Output: A solution for I .

Subroutine: An algorithm \mathcal{A} for the 3CS problem or for the 3CS^r problem.

- 5.1 $(B, x) \leftarrow \text{Simplex}_{CS}(L, W, H, l, w, h, d)$
 - 5.2 **for** $i \leftarrow 1$ **to** n **do** $x_i^* \leftarrow \lfloor x_i \rfloor$
 - 5.3 **if** there is i such that $x_i^* > 0$ for some $1 \leq i \leq n$ **then**
 - 5.4 **return** $(B, x_{1,\dots,n}^*)$ (but do not halt)
 - 5.5 **for** $i \leftarrow 1$ **to** n **do**
 - 5.6 **for** $j \leftarrow 1$ **to** n **do** $d_i \leftarrow d_i - B_{i,j} x_j^*$
 - 5.7 $n' \leftarrow 0, l' \leftarrow (), w' \leftarrow (), h' \leftarrow (), d' \leftarrow ()$
 - 5.8 **for** $i \leftarrow 1$ **to** n **do**
 - 5.9 **if** $d_i > 0$ **then**
 - 5.10 $n' \leftarrow n' + 1, l' \leftarrow l' \parallel (l_i), w' \leftarrow w' \parallel (w_i), h' \leftarrow h' \parallel (h_i), d' \leftarrow d' \parallel (d_i)$
 - 5.11 **if** $n' = 0$ **then HALT**
 - 5.12 $n \leftarrow n', l \leftarrow l', w \leftarrow w', h \leftarrow h', d \leftarrow d'$
 - 5.13 Go to line 5.1
 - 5.14 **return** a pattern of $\mathcal{A}(L, W, H, l, w, h, d)$ that has the largest volume, and update the demands (but do not halt).
 - 5.15 **if** there exists i ($1 \leq i \leq n$) such that $d_i > 0$ **then** go to line 5.1
-

The algorithm CG3CS solves (in each iteration) a linear system for an instance I and obtain B and x (line 5.1). Then, it obtains an integer vector x^* , just by rounding down the vector x (see line 5.2). The vector x^* is a 'partial' solution that possibly fulfills only part of the demands. Thus, if there is a box i with part of its demand fulfilled by x^* , the algorithm returns (B, x^*) ,

and the patterns corresponding to B . After this, the algorithm defines a new *residual instance* $I' = (L, W, H, l, w, h, d')$, where the vector $d' = (d'_1, \dots, d'_n)$ contains the *residual demand* of each item i (see lines 5.7 – 5.12). If d' is a null vector, then the algorithm halts (see line 5.11), as this means that each item i has its demand fulfilled; otherwise, the execution proceeds to solve the new updated instance I .

The vector x returned by the algorithm Simplex_{CS} might have all components smaller than 1. In this case, x^* is a null vector and the subroutine \mathcal{A} is used to obtain a good cutting pattern (line 5.14). Therefore, the demands are updated and if there is some residual demand (lines 5.14 – 5.15) the execution is restarted for the new residual instance (see line 5.1). Note that the number of residual instances solved by the algorithm CG3CS can be exponential in n . But, clearly, the algorithm halts because in each iteration the demands decrease. The algorithm \mathcal{A} used as subroutine by the algorithm CG3CS is the hybrid algorithm HFF3 described in Section 3.1.

An algorithm for the k -staged version of 3CS can be obtained analogously, just by changing the subroutine by the corresponding k -staged versions.

3.3 The 3CS^r problem

We can solve the 3CS^r problem also using the algorithms Simplex_{CS} and CG3CS, each one with the appropriate subroutines. Namely, in the Simplex_{CS} we use the algorithm HFF3^r, and in the algorithm CG3CS, we use the algorithm DP3DK^r. We denote this version by CG3CS^r. The same idea applies to the k -staged 3CS^r problem in which we use the algorithm DPS3DK as subroutine for the algorithm Simplex_{CS} .

4 The 3CSV Problem

We can solve the 3CSV problem using a column generation approach similar to the one described for the 3CS problem. For that, basically we have to adapt the algorithm Simplex_{CS} .

In this problem we are given a list of different bins B_1, \dots, B_b , each bin B_i with dimension (L_i, W_i, H_i) and value V_i , and we want to minimize the total value of the bins used to fulfill the demands. Using an analogous notation as before, the following is a relaxation of the integer linear program for the 3CSV problem:

$$\begin{aligned} & \min \sum_{j \in \mathcal{P}} C_j x_j \\ & \text{subject to } \begin{cases} Px \geq d \\ x_j \geq 0 \quad \text{for all } j \in \mathcal{P}. \end{cases} \end{aligned} \quad (5)$$

The coefficient C_j in the above formulation indicates the value of the bin type used in pattern j . So, each C_j corresponds to some V_i .

Similarly to the 3CS problem, if each box i has value y_i and occurs z_i times in a pattern j , we take a new column with $\sum_{i=1}^n y_i z_i > C_j$. Here, we can also use the algorithms we proposed for the

Three-dimensional Unbounded Knapsack problem to generate the (new) columns. The algorithm to solve (5) is called Simplex_{CSV} . The basic difference between the algorithms Simplex_{CS} and Simplex_{CSV} is that the latter has a vector f that associates one bin with each column of the matrix B . This vector and the variables B , $guil$ and pos are used to reconstruct the solution found.

Algorithm 6: Simplex_{CSV}

Input : An instance $I = (L, W, H, V, l, w, h, d)$ of the 3CSV problem.

Output: An optimum solution for (5), where the columns of P are cutting patterns.

Subroutine: An algorithm \mathcal{A} for the 3DK or $3DK^r$ problem.

- 6.1 Let f be a vector, where f_i is the smallest index j such that $l_i \leq L_j$, $w_i \leq W_j$ and $h_i \leq H_j$
 - 6.2 Let $x \leftarrow d$ and $B \leftarrow I_{n \times n}$
 - 6.3 Solve $y^T B = C_B^T$ // C_B is the vector $C = (C_1, \dots, C_n)$ restricted to the columns of B
 - 6.4 **for** $i \leftarrow 1$ **to** b **do**
 - 6.5 $z \leftarrow \mathcal{A}(L_i, W_i, H_i, l, w, h, y)$
 - 6.6 **if** $y^T z > V_i$ **then** go to line 6.8
 - 6.7 **return** $(B, f, x_{1, \dots, n}^*)$
 - 6.8 Solve $Bw = z$
 - 6.9 Let $t \leftarrow \min(\frac{x_j}{w_j} \mid 1 \leq j \leq n, w_j > 0)$ and $s \leftarrow \min(j \mid 1 \leq j \leq n, \frac{x_j}{w_j} = t)$
 - 6.10 Let $f_j = i$
 - 6.11 **for** $i \leftarrow 1$ **to** n **do**
 - 6.12 $B_{i,s} \leftarrow z_i$
 - 6.13 **if** $i = s$ **then** $x_i \leftarrow t$ **else** $x_i \leftarrow x_i - w_i t$
 - 6.14 Go to line 6.3
-

We describe now the algorithm CG3CSV that solves the 3CSV problem. It uses the algorithm Simplex_{CSV} and is very similar to algorithm CG3CS described for the 3CS problem (we omit the details). The algorithm \mathcal{A} used as subroutine by CG3CSV is the hybrid algorithm HFF3.

The algorithm for the k -staged 3CSV problem also uses the algorithm Simplex_{CSV} , but in this case with the subroutine for the k -staged 3DK problem.

4.1 The $3CSV^r$ problem

For this problem, we use the algorithm CG3CSV with the subroutine HFF3^r ; and the algorithm Simplex_{CSV} with the subroutine DP3DK^r . This version of the algorithm is called CG3CSV^r . For the k -staged $3CSV^r$ problem we use the algorithm Simplex_{CSV} with the subroutine DPS3DK .

5 The Three-dimensional Strip Packing Problem

The 3D strip packing problem (3SP) has been less tackled with the column generation approach. One advantage of this approach is that it is less sensitive to large values of demands. In the 3SP problem the cuts must be k -staged, the first cutting stage has to be horizontal (that is, orthogonal to the height), and the distance between two subsequent cuts must be at most some given value A . We call A -pattern a guillotine cutting pattern between two subsequent horizontal cuts.

Algorithm 7: CG3CSV

Input : An instance $I = (L, W, H, V, l, w, h, d)$ of the 3CSV problem.

Output: A solution for I .

Subroutine: An algorithm \mathcal{A} for the 3CSV problem or for the 3CSV^r problem.

```
7.1  $(B, f, x) \leftarrow \text{Simplex}_{CSV}(L, W, H, V, l, w, h, d)$ 
7.2 for  $i \leftarrow 1$  to  $n$  do  $x_i^* \leftarrow \lfloor x_i \rfloor$ 
7.3 if there is  $i$  such that  $x_i^* > 0$  for some  $1 \leq i \leq n$  then
7.4   return  $(B, f, x_{1^*, \dots, n}^*)$  (but do not halt)
7.5   for  $i \leftarrow 1$  to  $n$  do
7.6     for  $j \leftarrow 1$  to  $n$  do  $d_i \leftarrow d_i - B_{i,j}x_j^*$ 
7.7    $n' \leftarrow 0, l' \leftarrow (), w' \leftarrow (), h' \leftarrow (), d' \leftarrow ()$ 
7.8   for  $i \leftarrow 1$  to  $n$  do
7.9     if  $d_i > 0$  then
7.10     $n' \leftarrow n' + 1, l' \leftarrow l' \parallel (l_i), w' \leftarrow w' \parallel (w_i), h' \leftarrow h' \parallel (h_i), d' \leftarrow d' \parallel (d_i)$ 
7.11   if  $n' = 0$  then HALT
7.12    $n \leftarrow n', l \leftarrow l', w \leftarrow w', h \leftarrow h', d \leftarrow d'$ 
7.13   Go to line 7.1
7.14 Let  $V^* \leftarrow \min(\frac{V_i}{L_i W_i H_i} \mid i = 1, \dots, f)$  and  $j \leftarrow \min(i \mid \frac{V_i}{L_i W_i H_i} = V^*)$ 
7.15 return a pattern of  $\mathcal{A}(L_j, W_j, H_j, l, w, h, d)$  that has the largest volume, and update the demands.
7.16 if there exists  $i$  ( $1 \leq i \leq n$ ) such that  $d_i > 0$  then go to line 7.1
```

Let \mathcal{P} be the set of all A -patterns, $|\mathcal{P}| = m$, and let A_j be the height of an A -pattern $j \in \mathcal{P}$.

The following is a relaxation of the integer linear program for the 3SP problem:

$$\begin{aligned} & \min \sum_{j \in \mathcal{P}} A_j x_j \\ & \text{subject to } \begin{cases} Px \geq d \\ x_j \geq 0 \text{ for all } j \in \mathcal{P}. \end{cases} \end{aligned} \quad (6)$$

We can use the same approach presented for the 3CSV problem to solve the 3SP problem. For that, note that, each A -pattern of height A_j corresponds to a bin with dimension (L, W, A_j) and value precisely A_j in the 3CSV problem. Thus, if $R = \{a_1, \dots, a_b\}$ is the set of discretization points of height at most A , we can assume that $A = \max(a_1, \dots, a_b)$, and we can consider that we are given b different types of bins (A -patterns), each one with dimension (L, W, a_j) .

The algorithm to solve the k -staged 3SP problem, called CG3SP, is basically the algorithm presented for the k -staged 3CSV problem with two modifications. First, to perturb the residual instance we generate a level with maximal volume (considering the height of such level). To do this, we use the algorithm HFF2 (for the 2CS problem). Second, every call to the algorithm Simplex_{CSV} only solves one instance of the k -staged 3DK problem, the one with dimensions (L, W, a_b) . Observe that the variables G , $guil$ and pos computed by the algorithm DPS3DK have the solutions for each height $a_i \in R$. This is an important modification because $|R|$ can be very large, and solving instances for each $a_i \in R$ considering a different bin would consume a lot of time.

For the k -staged 3SP^r problem, we consider the algorithm HFF2^r to generate a perturbed instance. We also consider a modification in the algorithm HFF3 when we compare its solutions with the solution computed by the column generation algorithm. This modification basically consists in packing the levels generated by the algorithm HFF2 (or HFF2^r) one on top of the other in the direction z . We call M-HFF3 this modified algorithm. Finally, the maximum distance between two subsequent cuts is considered as the width of the bin.

6 Computational Tests

The tests were performed on several instances adapted from the literature. We present computational results for the set of instances adapted from Cintra et al. (2008). These instances were obtained in the following way: we considered the instances for the two-dimensional version of the problem, then we added the third dimension for each box (bin) by randomly choosing it from the dimensions already used for the other boxes (bins). These instances are available at the following url: <http://www.loco.ic.unicamp.br/binpack3d/>. We only considered the first 12 instances, which we called $gcut1d, \dots, gcut12d$.

We also considered the set of 700 instances from Bischoff and Ratcliff (1995). In the work of Bischoff and Ratcliff (1995) these instances were used in the Container Loading problem with the objective of maximizing the occupied volume of the container (we ignored the restriction that there were a limited number of copies of each item). We used these instances only for the Knapsack problem, as we would not be able to show the results for each of the problems considered here. These instances were organized in groups of 100 instances. In each group, the dimensions of the container and the number of items are the same: only the dimensions of the items are different. For example, in the first group named $thpack1$, each instance consists of exactly 3 boxes, and the subsequent groups, $thpack2, \dots, thpack7$, have 5, \dots , 20 boxes, respectively.

The algorithms presented in this paper were implemented in C language, and the tests were run on a computer with processor Intel[®] Core[™] 2 Quad 2.4 GHz, 4 GB of memory and operating system *Linux*. The linear systems in the column generation algorithms were solved by the Coin-OR CLP solver.

6.1 Comparing the use of raster points and discretization points

In this section we show, for some of the instances considered, the number of raster points, the number of discretization points, and the corresponding number of subproblems obtained. These numbers are shown in Table 1. We recall that m , s and u denotes the total number of r -points (or discretization points) of length, width and height, respectively. The product msu gives the number of subproblems. In many cases it is very impressive the reduction on the number of subproblems that occurs with the use of the r -points. This has a great impact in the dynamic programming approach.

For the instances gcut1–gcut12, the number of subproblems using r -points were, on average, 0.77% of the number of subproblems using discretization points. For instances thpack1–thpack7, the columns with the numbers of r -points and discretization points indicate the average number (truncated) in each group. For the thpack instances, the number of subproblems using r -points corresponds, on average, to 17.69% of the number of subproblems using discretization points.

Instance	Number of items	Bin Dimensions	Raster Points				Discretization Points				#Subprob(%) Rast./Discr.
			m	s	u	#Subprob	m	s	u	#Subprob	
gcut1_3d	10	(250, 250, 250)	13	5	5	325	68	20	20	27200	1,19
gcut2_3d	20	(250, 250, 250)	17	24	13	5304	95	112	69	734160	0,72
gcut3_3d	30	(250, 250, 250)	44	26	22	25168	143	107	122	1866722	1,35
gcut4_3d	50	(250, 250, 250)	45	50	29	65250	146	146	133	2835028	2,3
gcut5_3d	10	(500, 500, 500)	10	13	8	1040	40	76	26	79040	1,32
gcut6_3d	20	(500, 500, 500)	12	18	8	1728	96	120	41	472320	0,37
gcut7_3d	30	(500, 500, 500)	23	19	17	7429	179	126	140	3157560	0,24
gcut8_3d	50	(500, 500, 500)	44	59	27	70092	225	262	164	9667800	0,73
gcut9_3d	10	(1000, 1000, 1000)	15	7	7	735	92	42	32	123648	0,59
gcut10_3d	20	(1000, 1000, 1000)	14	20	5	1400	89	155	37	510415	0,27
gcut11_3d	30	(1000, 1000, 1000)	20	38	14	10640	238	326	127	9853676	0,11
gcut12_3d	50	(1000, 1000, 1000)	49	42	27	55566	398	363	291	42041934	0,13
thpack1	3	(587, 233, 220)	36	10	22	7920	100	27	53	143100	5,53
thpack2	5	(587, 233, 220)	88	65	48	274560	267	65	113	1961115	14
thpack3	8	(587, 233, 220)	206	37	93	708846	390	114	155	6891300	10,29
thpack4	10	(587, 233, 220)	263	52	110	1504360	425	134	165	9396750	16,01
thpack5	12	(587, 233, 220)	302	65	123	2414490	445	146	172	11174840	21,61
thpack6	15	(587, 233, 220)	339	81	134	3679506	463	157	177	12866307	28,6
thpack7	20	(587, 233, 220)	375	101	147	5567625	481	167	184	14780168	37,67

Table 1: Comparison between the number of subproblems using Raster Points and using Discretization Points for the instances adapted from Cintra et al. (2008) and Bischoff and Ratcliff (1995).

When we consider orthogonal rotations, the use of raster points also leads to a good reduction on the number of subproblems, as we can see in Table 2. In the average, the number of subproblems reduced to 2.13% for gcut1–gcut12 instances and to 45.44% for thpack1–thpack7 instances.

6.2 Computational results for the Three-dimensional Unbounded Knapsack problem

In this section, we present the computational results for the 3D Unbounded Knapsack problem. For this section, we consider the value v_i of each box i equal to its volume. Note that for the thpack instances the values in each group correspond to the average volume for that group. We first observe that for all instances, the computational time required to solve each instance was less than less than 0.001 second.

The columns of the Table 3 have the following information: Instance name, Volume for the case without rotations, Volume for the case with rotations, percentage of volume increased when considering rotations, Volume for the 4-staged case without rotations, Volume for the 4-staged case with rotations, percentage of volume increased when considering rotations in 4-staged patterns.

Instance	Number of items	Bin Dimensions	Raster Points				Discretization Points				# Subprob.(%) Rast./Discr.
			<i>m</i>	<i>s</i>	<i>u</i>	# Subprob.	<i>m</i>	<i>s</i>	<i>u</i>	# Subprob.	
gcut1_3dr	10	(250, 250, 250)	15	15	15	3375	92	92	92	778688	0,43
gcut2_3dr	20	(250, 250, 250)	41	41	41	68921	142	142	142	2863288	2,41
gcut3_3dr	30	(250, 250, 250)	58	58	58	195112	152	152	152	3511808	5,56
gcut4_3dr	50	(250, 250, 250)	81	81	81	531441	166	166	166	4574296	11,62
gcut5_3dr	10	(500, 500, 500)	23	23	23	12167	154	154	154	3652264	0,33
gcut6_3dr	20	(500, 500, 500)	28	28	28	21952	201	201	201	8120601	0,27
gcut7_3dr	30	(500, 500, 500)	43	43	43	79507	232	232	232	12487168	0,64
gcut8_3dr	50	(500, 500, 500)	95	95	95	857375	292	292	292	24897088	3,44
gcut9_3dr	10	(1000, 1000, 1000)	17	17	17	4913	174	174	174	5268024	0,09
gcut10_3dr	20	(1000, 1000, 1000)	32	32	32	32768	294	294	294	25412184	0,13
gcut11_3dr	30	(1000, 1000, 1000)	60	60	60	216000	461	461	461	97972181	0,22
gcut12_3dr	50	(1000, 1000, 1000)	85	85	85	614125	511	511	511	133432831	0,46
thpack1	3	(587, 233, 220)	393	58	50	1139700	490	137	124	8324120	13,69
thpack2	5	(587, 233, 220)	451	99	86	3839814	519	165	152	13016520	29,5
thpack3	8	(587, 233, 220)	486	132	119	7634088	537	183	170	16706070	45,7
thpack4	10	(587, 233, 220)	496	142	129	9085728	542	188	175	17831800	50,95
thpack5	12	(587, 233, 220)	504	150	137	10357200	546	192	179	18764928	55,19
thpack6	15	(587, 233, 220)	511	157	144	11552688	549	195	182	19484010	59,29
thpack7	20	(587, 233, 220)	520	166	153	13206960	554	200	187	20719600	63,74

Table 2: Comparison between the number of subproblems using Raster Points and using Discretization Points for the instances adapted from Cintra et al. (2008) and Bischoff and Ratcliff (1995), considering rotations.

As one would expect, we have a better use of the bin when orthogonal rotations are allowed. Indeed, when we compare the occupied volume of the bin in Table 3, the use of rotations leads to an improvement of 5.63% on gcut instances and of 3.19% on thpack instances, on average. When considering 4-staged patterns, the use of rotations leads to an improvement of 6.65% on gcut instances and of 3.89% on thpack instances, on average.

6.3 Computational results for the Three-dimensional Cutting Stock problem

The results for the 3CS problem and its variants are shown in Tables 4, . . . , 7. For each of them, we indicate the instance name; a lower bound (LB) for the value of an optimum integer solution (obtained by solving the linear relaxation (4) by the algorithm Simplex_{CS}); the difference (in percentage) between the solutions obtained by the algorithm CG3CS and the lower bound (LB); the CPU time in seconds; the total number of columns generated; the solution obtained only by the algorithm HFF3 (or HFF3^r); and the difference between (improvement over) HFF3 (respectively, HFF3^r) and algorithm CG3CS (respectively, CG3CS^r).

We exhibit in Table 4 and Table 5 the results for the non-staged cutting stock problem. In these tables we can see that the difference between the solutions of the algorithm CG3CS (and CG3CS^r) and the lower bound (LB) is 0.407% (and 2.320%), on average. When we compare the performance of the column generation algorithm with the algorithm HFF3 (respectively, HFF3^r) the improvement on the value of the solution is of 20.932% (respectively, 29.819%), on average.

Instance	Unbounded 3DK			4-Staged Unbounded 3DK		
	Without Rot.	With Rot.	Increase (%)	Without Rot.	With Rot.	Increase (%)
gcut1_3d	80,6	86,7	7,58	80,6	85,7	6,4
gcut2_3d	84,9	94,9	11,82	84,4	93,1	10,36
gcut3_3d	92,5	95,3	3	88,1	94,4	7,19
gcut4_3d	95,4	97	1,63	91,6	96,7	5,62
gcut5_3d	84,3	94,1	11,52	83,8	92,5	10,34
gcut6_3d	84,8	90	6,04	81,8	88	7,54
gcut7_3d	88,1	93,3	5,95	87,6	93,1	6,34
gcut8_3d	93,2	96,6	3,67	92,6	96,6	4,4
gcut9_3d	93,2	96,5	3,59	93,2	96,5	3,59
gcut10_3d	85,2	89	4,51	85,2	89	4,51
gcut11_3d	91,4	95	3,88	89,2	95	6,49
gcut12_3d	92,7	96,7	4,39	89,7	96	7,04
thpack1	90,9	98,1	7,94	89,5	97	8,35
thpack2	94,4	98,9	4,73	93	98,1	5,45
thpack3	96,7	99,3	2,74	95,4	98,7	3,5
thpack4	97,3	99,5	2,22	96	99	3,06
thpack5	97,7	99,6	1,88	96,5	99,1	2,67
thpack6	98,2	99,7	1,55	97,1	99,3	2,28
thpack7	98,6	99,8	1,25	97,6	99,5	1,93

Table 3: Results for the 3DK problem on instances adapted from Cintra et al. (2008) and Bischoff and Ratcliff (1995).

Instance	Solution of CG3CS	LB	Difference from LB (%)	Time (s)	Columns generated	HFF3	Improvement over HFF3 (%)
gcut1_3d	177	177	0.000%	0.03	72	181	2.21%
gcut2_3d	220	220	0.000%	0.56	652	245	10.20%
gcut3_3d	142	140	1.429%	6.93	2272	194	26.80%
gcut4_3d	520	517	0.580%	41.41	4230	747	30.39%
gcut5_3d	122	122	0.000%	0.03	48	160	23.75%
gcut6_3d	305	304	0.329%	0.20	338	364	16.21%
gcut7_3d	395	394	0.254%	0.66	607	467	15.42%
gcut8_3d	371	369	0.542%	26.87	3610	558	33.51%
gcut9_3d	60	60	0.000%	0.08	156	70	14.29%
gcut10_3d	217	216	0.463%	0.08	150	276	21.38%
gcut11_3d	191	189	1.058%	1.33	958	281	32.03%
gcut12_3d	429	428	0.234%	8.63	1473	572	25.00%

Table 4: Results for the 3CS problem on instances adapted from Cintra et al. (2008).

The time spent to solve these instances was at most 42 seconds for the 3CS problem and at most 2600 seconds for the 3CS^r problem. For the k -staged version, $k = 4$, we show in Table 6 and Table 7 the results obtained. We omitted the results for $k = 3$, since they are very similar to those for $k = 4$.

Observing Table 6 and Table 7, we have a difference of 0.381% (and 2.402%), on average, between the values of the solutions found by the algorithm CG3CS (and CG3CS^r) and the lower bound (LB). Moreover, comparing them with the HFF3 (respectively, HFF3^r) the gain in the value of the solution was 19.088% (respectively, 29.694%), on average.

Instance	Solution of CG3CS ^r	LB	Difference from LB (%)	Time (s)	Columns generated	HFF3	Improvement over HFF3 (%)
gcut1_3dr	163	161	1.242%	0.15	150	181	9.94%
gcut2_3dr	157	153	2.614%	4.63	466	255	38.43%
gcut3_3dr	135	129	4.651%	154.09	2977	199	32.16%
gcut4_3dr	460	453	1.545%	518.62	3669	666	30.93%
gcut5_3dr	100	98	2.041%	0.31	119	140	28.57%
gcut6_3dr	226	225	0.444%	2.07	438	330	31.52%
gcut7_3dr	372	369	0.813%	17.52	1032	467	20.34%
gcut8_3dr	327	318	2.830%	2554.40	8258	529	38.19%
gcut9_3dr	57	54	5.556%	0.25	187	81	29.63%
gcut10_3dr	198	196	1.020%	1.66	226	269	26.39%
gcut11_3dr	167	161	3.727%	136.24	2432	282	40.78%
gcut12_3dr	375	370	1.351%	525.99	3580	543	30.94%

Table 5: Results for the 3CS^r problem on instances adapted from Cintra et al. (2008).

Instance	Solution of CG3CS	LB	Difference from LB (%)	Time (s)	Columns generated	HFF3	Improvement over HFF3 (%)
gcut1_3d	177	177	0.000%	0.05	106	181	2.21%
gcut2_3d	220	220	0.000%	0.40	428	245	10.20%
gcut3_3d	146	144	1.389%	8.15	2103	194	24.74%
gcut4_3d	519	517	0.387%	39.04	3906	747	30.52%
gcut5_3d	132	132	0.000%	0.03	62	160	17.50%
gcut6_3d	305	304	0.329%	0.06	120	364	16.21%
gcut7_3d	396	394	0.508%	0.56	511	467	15.20%
gcut8_3d	399	397	0.504%	28.84	3243	558	28.49%
gcut9_3d	62	62	0.000%	0.05	91	70	11.43%
gcut10_3d	218	217	0.461%	0.10	157	276	21.01%
gcut11_3d	204	202	0.990%	2.31	1198	281	27.40%
gcut12_3d	434	434	0.000%	14.52	1806	572	24.13%

Table 6: Results for the 4-staged 3CS problem on instances adapted from Cintra et al. (2008).

Instance	Solution of CG3CS ^r	LB	Difference from LB (%)	Time (s)	Columns generated	HFF3	Improvement over HFF3 (%)
gcut1_3dr	163	161	1.242%	0.13	118	181	9.94%
gcut2_3dr	157	153	2.614%	5.34	459	255	38.43%
gcut3_3dr	136	130	4.615%	121.33	2697	199	31.66%
gcut4_3dr	460	453	1.545%	650.82	3883	666	30.93%
gcut5_3dr	100	98	2.041%	0.35	133	140	28.57%
gcut6_3dr	228	225	1.333%	3.51	601	330	30.91%
gcut7_3dr	373	369	1.084%	17.34	908	467	20.13%
gcut8_3dr	325	319	1.881%	2155.94	7789	529	38.56%
gcut9_3dr	57	54	5.556%	0.28	194	81	29.63%
gcut10_3dr	198	196	1.020%	1.24	177	269	26.39%
gcut11_3dr	167	161	3.727%	158.81	2657	282	40.78%
gcut12_3dr	378	370	2.162%	920.53	4376	543	30.39%

Table 7: Results for the 4-staged 3CS^r problem on instances adapted from Cintra et al. (2008).

The algorithm CG3CS found optimum solution for the instances gcut1_3d, gcut2_3d, gcut5_3d, gcut9_3d as shown in Table 4 and Table 6.

6.4 Computational results for the 3CSV problem

We tested the algorithm CG3CSV (and CG3CSV^r) with the instances above mentioned, with three different bins. In these instances, the value of each bin corresponds to its volume. The results are shown in Table 8 and Table 9.

Instance	Solution of CG3CSV	LB	Difference from LB (%)	Time (s)	Columns generated
gcut1_3d	2,431,875,000	2,415,000,000.0	0.699%	0.60	1821
gcut2_3d	2,386,093,750	2,338,125,000.0	2.052%	9.64	10699
gcut3_3d	2,179,687,500	2,137,243,406.8	1.986%	114.06	33936
gcut4_3d	6,894,218,750	6,845,773,809.5	0.708%	1572.03	163072
gcut5_3d	13,342,500,000	13,190,833,333.3	1.150%	0.35	542
gcut6_3d	29,420,000,000	29,130,171,875.0	0.995%	5.73	7796
gcut7_3d	36,553,750,000	36,153,136,160.7	1.108%	44.05	26332
gcut8_3d	41,788,750,000	41,280,158,270.4	1.232%	1622.03	197427
gcut9_3d	59,860,000,000	58,847,226,277.4	1.721%	0.41	646
gcut10_3d	197,420,000,000	196,062,395,833.3	0.692%	0.33	550
gcut11_3d	174,270,000,000	171,061,388,146.2	1.876%	74.32	38689
gcut12_3d	370,100,000,000	366,802,923,728.8	0.899%	143.49	18204

Table 8: Results for the 3CSV problem on instances adapted from Cintra et al. (2008).

Instance	Solution of CG3CSV ^r	LB	Difference from LB (%)	Time (s)	Columns generated
gcut1_3dr	1,582,187,500	1,521,787,500.0	3.969%	5.56	3946
gcut2_3dr	1,917,812,500	1,823,075,945.0	5.197%	169.04	12384
gcut3_3dr	2,011,718,750	1,908,532,902.9	5.407%	4049.10	68502
gcut4_3dr	5,819,531,250	5,652,226,962.2	2.960%	113341.67	373377
gcut5_3dr	10,518,750,000	9,932,319,046.0	5.904%	14.35	4639
gcut6_3dr	22,545,000,000	21,728,068,481.4	3.760%	178.02	20807
gcut7_3dr	31,166,250,000	30,536,088,859.9	2.064%	2464.12	77720
gcut8_3dr	38,084,200,000	37,119,105,733.3	2.534%	335101.12	354237
gcut9_3dr	54,280,000,000	50,302,713,615.5	7.907%	9.61	6112
gcut10_3dr	157,500,000,000	154,562,209,821.4	1.901%	64.77	6583
gcut11_3dr	152,710,000,000	143,306,761,029.1	6.562%	7636.69	102188
gcut12_3dr	300,410,000,000	293,985,781,261.7	2.185%	51168.44	211262

Table 9: Results for the 3CSV^r problem on instances adapted from Cintra et al. (2008).

We can note that the problem with different bins size is harder to solve, demanding more computational time than the 3CS problem. But the results were also very good, where the largest difference from the lower bound for the 3CSV (3CSV^r) problem was 2.052% (7.907%).

Table 10 and Table 11 show the results for the staged version of the problem. We note that some instances like gcut4_3dr, gcut8_3dr and gcut12_3dr require tens of thousand of seconds to be solved. On the other hand, when we compare the solutions found by the algorithm CG3CSV (and CG3CSV^r) and the lower bound, the difference is 0.970% (and 3.920%), on average.

Instance	Solution of CG3CSV	LB	Difference from LB (%)	Time (s)	Columns generated
gcut1_3d	2,432,500,000	2,415,000,000.0	0.725%	0.43	1193
gcut2_3d	2,443,125,000	2,417,773,437.5	1.049%	8.85	8907
gcut3_3d	2,238,750,000	2,205,086,568.8	1.527%	149.56	39142
gcut4_3d	6,963,906,250	6,900,824,728.3	0.914%	1678.22	134327
gcut5_3d	14,187,500,000	14,122,500,000.0	0.460%	0.25	356
gcut6_3d	29,960,000,000	29,722,351,562.5	0.800%	4.88	5936
gcut7_3d	37,412,500,000	37,028,616,071.4	1.037%	33.46	18482
gcut8_3d	43,150,000,000	42,814,590,460.7	0.783%	958.29	90063
gcut9_3d	61,620,000,000	61,051,648,936.2	0.931%	0.17	271
gcut10_3d	198,200,000,000	196,451,666,666.7	0.890%	1.17	1532
gcut11_3d	181,930,000,000	178,705,312,500.0	1.804%	24.25	12067
gcut12_3d	374,660,000,000	371,975,610,351.6	0.722%	258.72	20801

Table 10: Results for the *4-staged* 3CSV problem on instances adapted from Cintra et al. (2008).

Instance	Solution of CG3CSV ^r	LB	Difference from LB (%)	Time (s)	Columns generated
gcut1_3dr	1,597,500,000	1,551,045,372.6	2.995%	3.18	3059
gcut2_3dr	1,969,062,500	1,871,147,927.3	5.233%	253.32	14080
gcut3_3dr	2,051,406,250	1,948,098,696.7	5.303%	4481.70	79799
gcut4_3dr	5,924,218,750	5,756,335,128.3	2.917%	67161.89	276688
gcut5_3dr	10,541,250,000	10,157,437,500.0	3.779%	9.88	2506
gcut6_3dr	23,266,250,000	22,752,722,529.8	2.257%	115.47	14935
gcut7_3dr	31,935,000,000	31,032,417,461.1	2.909%	2782.32	90438
gcut8_3dr	38,182,500,000	37,219,195,377.3	2.588%	116401.04	260380
gcut9_3dr	55,420,000,000	51,183,053,219.9	8.278%	6.19	3217
gcut10_3dr	160,710,000,000	156,510,662,983.4	2.683%	59.95	5476
gcut11_3dr	157,240,000,000	149,207,472,717.2	5.383%	5687.61	79269
gcut12_3dr	305,530,000,000	297,446,392,419.0	2.718%	38753.94	151803

Table 11: Results for the *4-staged* 3CSV^r problem on instances adapted from Cintra et al. (2008).

6.5 Computational results for the Strip Packing problem

The results obtained for the k -staged 3SP and 3SP^r problems with $k = 4$ are shown in Table 12 and Table 13. We omit the results for $k = 3$ because they were very similar to the case $k = 4$. As expected, the computational time required to solve these problems is considerably larger than the time required to solve the respective cutting stock problems. The instance gcut12 for example (and its version with rotation) required 461 seconds (28057 seconds) to be solved for the strip packing problem, and demanded 14 seconds (920 seconds) for the 4-staged version of the cutting stock problem. But the algorithm CG3SP (and CG3SP^r) obtained very good results, computing solutions differ from the lower bound at most 0.835% (and 1.534%). Moreover the improvement over M-HFF3 was 7.779% (and 22.325%), on average.

Instance	Solution of CG3SP	LB	Difference from LB (%)	Time (s)	Columns generated	M-HFF3	Improvement over M-HFF3 (%)
gcut1_3d	35,510	35,458.2	0.146%	0.02	41	35,648	0.39%
gcut2_3d	45,400	45,372.2	0.061%	0.72	138	48,151	5.71%
gcut3_3d	37,632	37,565.5	0.177%	26.60	1201	43,537	13.56%
gcut4_3d	112,507	112,334.5	0.154%	303.90	5077	134,169	16.15%
gcut5_3d	54,311	54,208.8	0.188%	0.02	39	55,413	1.99%
gcut6_3d	114,387	114,114.7	0.239%	0.37	408	127,178	10.06%
gcut7_3d	162,829	162,551.2	0.171%	2.88	370	182,543	10.80%
gcut8_3d	185,854	185,425.5	0.231%	440.59	5993	208,859	11.01%
gcut9_3d	58,804	58,317.3	0.835%	0.04	79	61,002	3.60%
gcut10_3d	191,638	190,937.9	0.367%	0.34	238	205,111	6.57%
gcut11_3d	192,456	191,962.8	0.257%	19.61	1915	209,980	8.35%
gcut12_3d	399,664	398,647.1	0.255%	461.47	3930	421,417	5.16%

Table 12: Results for the *4-staged* 3SP problem on instances adapted from Cintra et al. (2008).

Instance	Solution of CG3SP ^r	LB	Difference from LB (%)	Time (s)	Columns generated	M-HFF3	Improvement over M-HFF3 (%)
gcut1_3dr	24,863	24,757.1	0.428%	0.73	246	32,808	24.22%
gcut2_3dr	30,824	30,440.4	1.260%	105.28	2276	43,364	28.92%
gcut3_3dr	32,246	31,922.2	1.014%	978.46	7776	41,750	22.76%
gcut4_3dr	90,838	90,175.2	0.735%	14590.06	31584	117,003	22.36%
gcut5_3dr	40,931	40,263.0	1.659%	1.82	147	52,695	22.32%
gcut6_3dr	87,297	86,758.8	0.620%	39.90	1660	113,529	23.11%
gcut7_3dr	121,259	120,707.1	0.457%	441.89	5446	159,555	24.00%
gcut8_3dr	149,917	148,765.7	0.774%	23620.73	25946	190,709	21.39%
gcut9_3dr	52,314	51,523.6	1.534%	1.22	219	60,608	13.68%
gcut10_3dr	151,532	150,583.5	0.630%	22.70	303	193,338	21.62%
gcut11_3dr	150,444	149,160.8	0.860%	2005.83	4856	193,689	22.33%
gcut12_3dr	296,361	294,843.5	0.515%	28057.17	24402	376,038	21.19%

Table 13: Results for the *4-staged* 3SP^r problem on instances adapted from Cintra et al. (2008).

7 Concluding Remarks

We presented algorithms and computational tests for the problems 3DK, 3CS, 3CSV and 3SP and its variants with k stages and orthogonal rotations.

For the Three-dimensional Unbounded Knapsack and its variants, the results obtained showed that the use of raster points in the dynamic programming approach was very successful, with considerable reduction on the number of subproblems. On the oriented 3DK problem with gcut instances, for example, the number of subproblems reduced to less than 0.77% of the number of subproblems using discretization points.

When orthogonal rotations are allowed, the occupied volume of the bin increases significantly (on average, this improvement was 5.63% on gcut instances). This is natural, since the domain of the feasible solutions increases too. The highlight is for the computational time, since all instances were solved (to optimality) in at most 0.01 seconds.

For the Three-dimensional Cutting Stock Problem and its variants, the column generation algo-

rithm found solutions, on average, within 1.8% of the lower bound. And, when we compare with the primal heuristic we have high improvements. The computational time was high for the case when orthogonal rotations are allowed. We had instances solved in about 2600 seconds.

For the Three-dimensional Cutting Stock Problem with Variable Bin Size (and its variants) the column generation algorithm found solutions differing 2.6%, on the average, from the lower bound. On the other hand, a lot of computational time (more than 100 thousand seconds), was required to solve some instances, mainly for the case in which orthogonal rotations are allowed. So this problem showed to be harder to solve than the 3CS problem.

The column generation algorithms for the Strip Packing problem and its variants also obtained solutions very close to the lower bound: the difference was at most 1.6%. As in the case of the 3CS and 3CSV problems, the improvement over the solutions returned by the primal heuristics was larger than 22.5%, on average. It is important to note that the solutions for the k -staged version of the 3CS, 3CSV and 3SP problems for $k = 3$ were very similar to those for $k = 4$. The main difference was in the little increase of computational time when $k = 4$.

The computational results indicate that the algorithms proposed in this paper may be useful to solve real-world instances of moderate size. For the instances considered here, the algorithms found optimum or quasi-optimum solutions in a satisfactory amount of computational time.

Acknowledgement. We would like to thank CAPES, CNPq and FAPESP for supporting this research.

References

- R. Alvarez-Valdes, A. Parajon, and J. M. Tamarit. A computational study of lp-based heuristic algorithms for two-dimensional guillotine cutting stock problems. *OR Spectrum*, 24(2):179–192, 2002.
- N. Bansal, X. Han, K. Iwama, M. Sviridenko, and G. Zhang. Harmonic algorithm for 3-dimensional strip packing problem. In *Proc. of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1197–1206, 2007.
- J. E. Beasley. Algorithms for unconstrained two-dimensional guillotine cutting. *Journal of the Operational Research Society*, 36(4):297–306, 1985.
- E. G. Birgin, R. D. Lobato, and R. Morabito. An effective recursive partitioning approach for the packing of identical rectangles in a rectangle. *Journal of the Operational Research Society*, 2008. Aceito; DOI: 10.1057/jors.2008.141.
- E.E. Bischoff and M.S.W. Ratcliff. Issues in the development of approaches to container loading. *OMEGA*, 23(4):377–390, 1995.

- F. R. K. Chung, M. R. Garey, and D. S. Johnson. On packing two-dimensional bins. *SIAM Journal on Algebraic and Discrete Methods*, 3:66–76, 1982.
- V. Chvátal. *Linear Programming*. W. H. Freeman and Company, New York, 1980.
- G. Cintra, F. K. Miyazawa, Y. Wakabayashi, and E. C. Xavier. A note on the approximability of cutting stock problems. *European Journal on Operations Research (Elsevier Science)*, 34: 2589–2603, 2007.
- G. F. Cintra, F. K. Miyazawa, Y. Wakabayashi, and E. C. Xavier. Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation. *European Journal of Operational Research*, 191:59–83, 2008.
- Coin-OR CLP. Linear programming solver: An open source code for solving linear programming problems. <http://www.coin-or.org/Clp/index.html>.
- J. Csirik and A. van Vliet. An on-line algorithm for multidimensional bin packing. *Operations Research Letters*, 13:149–158, 1993.
- F. Diedrich, R. Thöle R. Harren, K. Jansen, and H. Thomas. Approximation algorithms for 3D orthogonal knapsack. *Journal of Computer Science and Technology*, 23(5):749–762, 2008.
- P. Gilmore and R. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9:849–859, 1961.
- P. Gilmore and R. Gomory. A linear programming approach to the cutting stock problem - part II. *Operations Research*, 11:863–888, 1963.
- P. Gilmore and R. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13:94–120, 1965.
- J. C. Herz. A recursive computational procedure for two-dimensional stock-cutting. *IBM Journal of Research Development*, pages 462–469, 1972.
- Mhand Hifi. Exact algorithms for the guillotine strip cutting/packing problem. *Computers & Operations Research*, 25:925–940, 1998.
- K. Jansen and R. Solis-Oba. An asymptotic approximation algorithm for 3D-strip packing. In *Proc. of the 17th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 143–152, 2006.
- Klaus Jansen and Rob van Stee. On strip packing with rotations. In *Proc. of the 37th ACM Symposium on Theory of Computing*, 2005.
- C. Kenyon and E. Rémila. A near-optimal solution to a two-dimensional cutting stock problem. *Mathematics of Operations Research*, 25:645–656, 2000.

- A. Lodi, S. Martello, and D. Vigo. Models and bounds for two-dimensional level packing problems. *Journal of Combinatorial Optimization*, 8:363–379, 2004.
- Silvano Martello, Michele Monaci, and Daniele Vigo. An exact approach to the strip-packing problem. *INFORMS Journal on Computing*, 15(3):310–319, 2003.
- F. K. Miyazawa and Y. Wakabayashi. Three-dimensional packings with rotations. *Computers and Operations Research*, 36:2801–2815, 2009.
- J. Puchinger and G. R. Raidl. Models and algorithms for three-stage two-dimensional bin packing. *European J. Operational Research*, 183(3):1304–1327, 2007.
- G. Scheithauer. Equivalence and dominance for problems of optimal packing of rectangles. *Ricerca Operativa*, 27:3–34, 1997.