An effective recursive partitioning approach for the packing of identical rectangles in a rectangle *

Ernesto G. Birgin † Rafael D. Lobato † Reinaldo Morabito ‡

August 4, 2008.

Abstract

In this work, we deal with the problem of packing (orthogonally and without overlapping) identical rectangles in a rectangle. This problem appears in different logistics settings, such as the loading of boxes on pallets, the arrangements of pallets in trucks and the stowing of cargo in ships. We present a recursive partitioning approach combining improved versions of a recursive five-block heuristic and an L-approach for packing rectangles into larger rectangles and L-shaped pieces. The combined approach is able to rapidly find the optimal solutions of all instances of the pallet loading problem sets Cover I and II (more than 50 thousand instances). It is also effective for solving the instances of problem set Cover III (almost 100 thousand instances) and practical examples of a woodpulp stowage problem, if compared to other methods from the literature. Some theoretical results are also discussed and, based on them, efficient computer implementations are introduced. The computer implementation and the data sets are available for benchmarking purposes at *http://www.ime.usp.br/∼egbirgin/packing/*.

Key words: Cutting and packing, manufacturer's pallet loading problem, woodpulp stowage problem, non-guillotine cutting pattern, dynamic programming, raster points.

1 Introduction

This work is concerned with the problem of arranging (orthogonally and without overlapping) the maximum number of identical rectangles of size (l, w) into a large rectangle of size (L, W). This two-dimensional packing problem is also known as the manufacturer's pallet loading problem, since it appears in the loading of identical boxes on pallets, as well as in the packaging design and truck or rail car loading [7, 8, 13, 14, 23, 32]. It is

*This work was partially supported by PRONEX-Optimization (PRONEX - CNPq / FAPERJ E-26 / 171.510/2006 - APQ1), FAPESP (Grants 2006/53768-0, 2006/03496-3 and 2005/57984-6) and CNPq (Grants PROSUL 490333/2004-4 and 522973/1995-4).

†Department of Computer Science, Institute of Mathematics and Statistics, University of São Paulo, Rua do Matão 1010, Cidade Universitária, 05508-090 São Paulo, SP - Brazil ({egbirgin|lobato}@ime.usp.br).

‡Department of Production Engineering, Federal University of São Carlos, Via Washington Luiz km. 235, 13565-905, São Carlos, SP - Brazil (morabito@ufscar.br).

assumed that there are no constraints related to cargo weight, density, fragility, etc. This problem also appears in other logistics settings, for example, in the problem of finding the maximum number of stowed units of woodpulp into holds of maritime ships, referred to as the woodpulp stowage problem [27]. Apparently easy to be optimally solved, the problem is claimed to be NP-hard although it has not been proven [11, 16, 25]. Various authors have proposed approximate methods to deal with it, as discussed in e.g. [1, 2, 4, 15, 16, 18, 19, 26, 29, 30] and the references therein. Upper bounds for the problem were studied in e.g. [5, 10, 16, 19, 24, 25].

In this paper we present an effective recursive partitioning approach that combines refined versions of the recursive five-block heuristic proposed in [21, 22] and the L-approach for packing rectangles into larger rectangles and L-shaped pieces presented in [17] (see also [6]). We prove some theoretical results and develop some strategies related to the recursive five-block heuristic that enable us to reduce the number of subproblems solved by the method. Regarding the L-approach, two new ways of dividing an L-shaped piece into two L-shaped pieces, which were not considered in [17], are described. Moreover, the usage of the so-called *reduced raster points* (simply called *raster points* from now on) [29] is incorporated into the combined recursive partitioning approach. The refinements based on theoretical results and the integration of these methods enable us to develop a recursive partitioning approach that is very effective for solving difficult pallet loading instances.

For instance, the combined approach is capable of rapidly finding all optimal solutions of the well-known problem sets Cover I and II, which contains more than 50,000 instances involving packing patterns of up to 100 boxes in the pallet surface [1, 25, 29]. Moreover, if compared to the method in [1], the combined approach improves the solutions of 116 instances of problem set Cover III, whose 98,016 instances involve packings from 100 up to 150 boxes. Requiring moderate computational resources, the combined approach also improves the solutions of practical examples of the woodpulp stowage problem, when compared to the method presented in [27]. These are large examples involving packings of hundreds of woodpulp units by hold (i.e., boxes by pallet). Since we were unable to find a counterexample for which the present combined approach fails, we conjecture that it always finds optimal packings, as well as the L-approach in [17].

This paper is organized as follows. In Section 2, we define the problem, discuss some of its properties and introduce the combined recursive partitioning approach. In Sections 3 and 4, we describe the refinements of the recursive five-block heuristic and the L-approach, respectively. We also introduce theoretical results and related strategies that improve the performance of these algorithms. Section 5 presents the time complexity of the recursive partitioning approach. Section 6 analyzes numerical experiments. Finally, some concluding remarks and perspectives for future research are described in Section 7.

2 Problem properties and the combined recursive partitioning approach

As mentioned, the problem consists of packing rectangular boxes with length l and width w into a large rectangular pallet with length L and width W. The boxes have a fixed horizontal orientation, they must be placed orthogonally (i.e., with each side orthogonal to one side of the pallet) and only 90-degree rotations are allowed. The objective is to find a two-dimensional (non-guillotine) packing pattern with the maximum number of boxes packed. Without loss of generality, we assume that L, W, l and w are integers and that $L \geq W$ and $l \geq w$. Thus, a problem instance is determined by the quadruple (L, W, l, w). This packing problem can be classified as 2/B/O/C according to Dyckhoff's typology of cutting and packing problems [14], and as "two-dimensional, rectangular Identical Item Packing Problem (IIPP)" based on Wäescher et al.'s typology [32].

Given a pallet (L, W), we assume that the bottom-left corner of the pallet coincides with the origin of $\mathbb{R}^2$. A *packing* of N boxes for problem (L, W, l, w) is represented by a set of N triplets (x_i, y_i, o_i), $i = 1, \ldots, N$, where (x_i, y_i) corresponds to the coordinate of the bottom-left corner of the i-th box, and $o_i =$ HORIZONTAL means that the i-th box is not rotated, while $o_i =$ VERTICAL means that it is 90-degree rotated. Clearly, the boxes cannot overlap and they must be inside the pallet.

Let (x''_i, y''_i, o_i), $i = 1, \ldots, N$, be a packing for problem (L, W, l, w). In [17], it was shown that there is another packing (x'_i, y'_i, o_i), $i = 1, \ldots, N$, such that x'_i and $y'_i, i = 1, \ldots, N$, are integer conic combinations of l and w, that is, $(x'_i, y'_i) \in S_L \times S_W$, $i = 1, \ldots, N$, where

$$\begin{array}{rcl} S_L & = & \{x \in \mathbb{Z}_+ \mid x = r\, l + s\, w, 0 \leq x \leq L, r, s \in \mathbb{Z}_+\}, \\ S_W & = & \{y \in \mathbb{Z}_+ \mid y = t\, l + u\, w, 0 \leq y \leq W, t, u \in \mathbb{Z}_+\}. \end{array}$$

As a consequence of Assertion 1 in [28] (that deals with non-identical rectangles and whose formal proof is not given), it can be shown that there is another packing (x_i, y_i, o_i), $i = 1, \ldots, N$, such that $(x_i, y_i) \in R_L \times R_W, i = 1, \ldots, N$, where

$$\begin{array}{rcl} R_L & = & \{x \in \mathbb{Z}_+ \mid x = \langle L - \hat{x} \rangle_{S_L} \text{ for some } \hat{x} \in S_L\}, \\ R_W & = & \{y \in \mathbb{Z}_+ \mid y = \langle W - \hat{y} \rangle_{S_W} \text{ for some } \hat{y} \in S_W\}, \end{array} \tag{1}$$

and

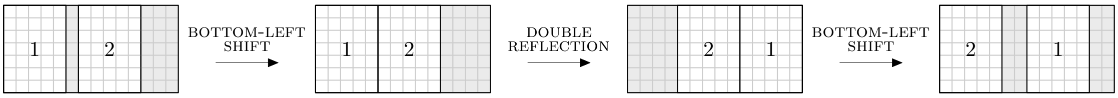
$$\langle \tilde{x} \rangle_{S_L} = \max\{x \in S_L \mid x \leq \tilde{x}\} \text{ and } \langle \tilde{y} \rangle_{S_W} = \max\{y \in S_W \mid y \leq \tilde{y}\}.$$

R_L and R_W are known as the sets of *raster points* [29] for (L, l, w) and (W, l, w), respectively. For completeness, the theorem below formalizes this claim.

Theorem 2.1 *Let (x''_i, y''_i, o_i), $i = 1, \ldots, N$, be a packing for problem (L, W, l, w). There is a packing (x_i, y_i, o_i), $i = 1, \ldots, N$, such that $(x_i, y_i) \in R_L \times R_W$, $i = 1, \ldots, N$.*

Proof: By [17] (p. 779) there is a packing (x'_i, y'_i, o_i), $i = 1, \ldots, N$, such that $(x'_i, y'_i) \in S_L \times S_W$, $i = 1, \ldots, N$. Consider now the packing given by $(L - x'_i - d_1, W - y'_i - d_2, o_i)$, $i = 1, \ldots, N$, where $d_1 = l$ and $d_2 = w$ if $o_i =$ HORIZONTAL and $d_1 = w$ and $d_2 = l$ if $o_i =$ VERTICAL. This packing can be seen as a double reflection of the previous packing. Applying the procedure described in [17] (p. 779), we obtain a packing (x_i, y_i, o_i), $i = 1, \ldots, N$, such that $(x_i, y_i) \in R_L \times R_W$, $i = 1, \ldots, N$. $\square$

Figure 1 illustrates the application of Theorem 2.1 to a very small problem. Consider problem $(14, 7, 7, 5)$. For this problem we have that $S_L = \{0, 5, 7, 10, 12, 14\}$, $S_W =$



(a) (b) (c) (d)

Figure 1: Example of an application of Theorem 2.1. The initial packing in (a) is such that the bottom-left corners of the items do not belong to $R_L \times R_W$ (not even to $S_L \times S_W$); while the packing in (d), obtained by applying the procedure described in the theorem, has the bottom-left corners in $R_L \times R_W$.

$\{0, 5, 7\}$, $R_L = \{0, 7, 14\}$ and $R_W = \{0, 7\}$. Figure 1(a) shows a solution such that the bottom-left corner of item 2 does not belong to $S_L \times S_W$. Figure 1(b) illustrates that, performing a bottom-left shift, it is possible to obtain another packing such that the bottom-left corners of the items belong to $S_L \times S_W$ (note that the bottom-left corner of item 2 does not belong to $R_L \times R_W$). This shift corresponds to the application of the theorem in [17]. Figure 1(c) shows the double reflection mentioned in Theorem 2.1. Finally, Figure 1(d) shows a bottom-left shift applied to the packing in Figure 1(c). The packing of Figure 1(d) is such that the bottom-left corners of the items belong to $R_L \times R_W$.

A corollary of Theorem 2.1 is that it can be assumed, without loss of generality, that $L \in S_L$ and $W \in S_W$. Therefore, problem (L, W, l, w) is equivalent to problem $(\langle L \rangle_{S_L}, \langle W \rangle_{S_W}, l, w)$. If $\langle L \rangle_{S_L} < \langle W \rangle_{S_W}$ then, by convention, we consider the also equivalent problem, $(\langle W \rangle_{S_W}, \langle L \rangle_{S_L}, l, w)$. The process of converting a problem (or subproblem) into an equivalent problem such that the dimensions of the pallet are integer conic combinations and the first dimension is greater than or equal to the second dimension is called *normalization*. The normalization process is useful to detect equivalent problems and solve only one of them. An analogous process also applies to the packing of rectangles within L-shaped pieces; see [17] for details.

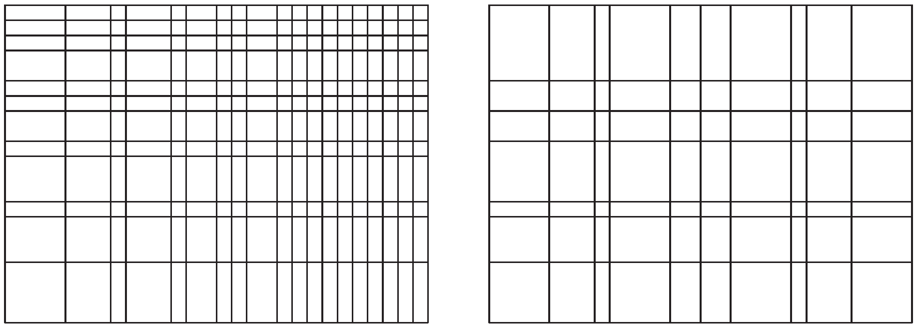
Another corollary of Theorem 2.1 is that a method needs to look for an optimal packing only trying to place the boxes with their bottom-left corners within the set $R_L \times R_W$. For instance, consider the problem $(L, W, l, w) = (28, 21, 7, 4)$. The sets of integer conic combinations S_L and S_W are given by

$$\begin{aligned} S_L &= \{0, 4, 7, 8, 11, 12, 14, 15, 16, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28\}, \\ S_W &= \{0, 4, 7, 8, 11, 12, 14, 15, 16, 18, 19, 20, 21\}, \end{aligned}$$

and the sets of raster points are given by

$$\begin{aligned} R_L &= \{0, 4, 7, 8, 12, 14, 16, 20, 21, 24, 28\}, \\ R_W &= \{0, 4, 7, 8, 12, 14, 16, 21\}. \end{aligned}$$

Figure 2(a) shows all possible positions for the bottom-left corners that might be tried by a method that uses the integer conic combinations (like the one introduced in [17]), while Figure 2(b) shows the smaller set of possible positions that might be considered by a method that uses the raster points.



(a) $S_L \times S_W$ (b) $R_L \times R_W$

Figure 2: Intersections in (a) and (b) represent sets $S_L \times S_W$ and $R_L \times R_W$, respectively, for problem $(L, W, l, w) = (28, 21, 7, 4)$. By definition, $R_L \times R_W \subseteq S_L \times S_W$. Moreover, in this particular example, the number of possibilities that might be tried by a method based on raster points is much smaller than the number of possibilities that might be tried by a method based on integer conic combinations.

The recursive partitioning approach introduced in the present paper consists of the combination of efficient implementations of the five-block recursive heuristic [21, 22] and the L-approach [17, 6]. The combined approach has two phases. First, the five-block recursive algorithm is executed (phase 1) and, if a certificate of optimality is not provided by the method, that is, if the computed upper bound is not equal to the solution value found, the L-approach algorithm is executed (phase 2). Moreover, additional information obtained in phase 1 by the first algorithm is used in phase 2 by the second algorithm in at least two ways.

On one hand, subproblems generated by the first algorithm may also be generated by the second. If this situation occurs, the lower and upper bounds obtained by the first algorithm are used by the second. Therefore, if an optimal solution was already found for the subproblem in phase 1, it is not solved again in phase 2, improving the performance of phase 2. On the other hand, the second algorithm computes lower bounds for L-shaped subproblems by partitioning them into two rectangles (in the two straightforward different ways) and summing up the lower bounds of the rectangles. Having the information saved by the first algorithm at hand, we have often better lower bounds for the rectangles than the ones provided by the homogeneous packings computed by the second algorithm, therefore improving the performance of phase 2.

3 Refinements of the recursive five-block heuristic

The recursive five-block heuristic [21, 22] in phase 1 is, basically, a recursive application of the method presented in [7]. The algorithm divides a rectangle into five (or less) smaller rectangles in a way that is called *first-order non-guillotine cut*; see Figure 3. As shown in this figure, a first-order non-guillotine cut is represented by a quadruple (x_1, x_2, y_1, y_2).

Each small rectangle is recursively cut unless an optimal solution is found, or a depth limit in the recursion (input parameter of the method) is attained. An optimal solution can be detected by closing the gap between known lower and upper bounds on the number of boxes that can be packed. An efficient implementation of the method is to reduce to its minimum, without loss of generality, the number of quadruples (x_1, x_2, y_1, y_2) needed to generate all the possible first-order non-guillotine cuts of a given rectangle. It also depends on developing an efficient strategy to avoid solving the same (or an equivalent) subproblem more than once. In the rest of this section, we describe the strategies developed to improve the algorithm introduced in [21].

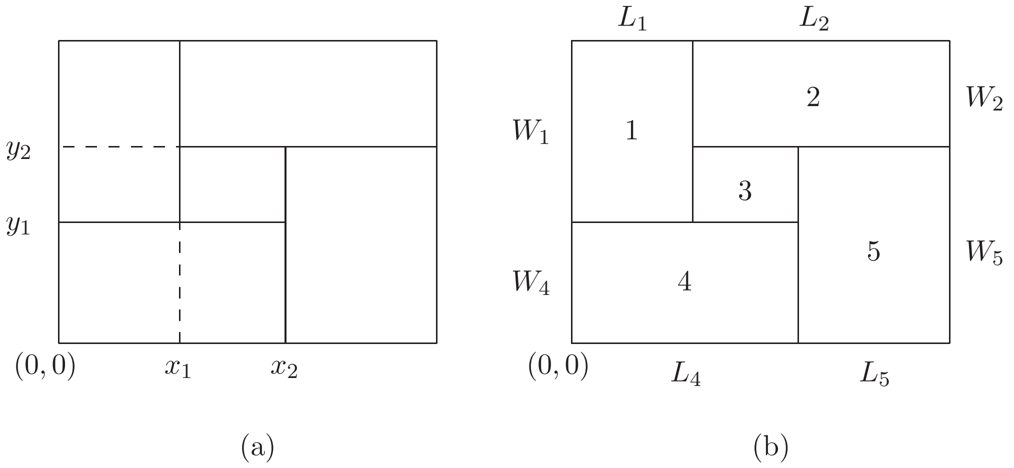


Figure 3: First-order non-guillotine cut. (a) A first-order non-guillotine cut can be defined by a quadruple (x_1, x_2, y_1, y_2) such that $0 \le x_1 \le x_2 \le L$ and $0 \le y_1 \le y_2 \le W$. (b) It determines five subrectangles $(L_1, W_1), \ldots, (L_5, W_5)$ such that $L_1 = x_1$, $W_1 = W - y_1$, $L_2 = L - x_1$, $W_2 = W - y_2$, $L_3 = x_2 - x_1$, $W_3 = y_2 - y_1$, $L_4 = x_2$, $W_4 = y_1$, $L_5 = L - x_2$ and $W_5 = y_2$.

The first two improvements consist of the usage of raster points and the upper bound introduced by Barnes [5]. The raster points are incorporated in connection with the data structures described in [6]. This combination will enable us to arrive to an efficient usage of the raster points (as pointed out in [21, 22]). The two theorems below show that there is no loss of generality by only considering cuts generated by raster points.

Definition 3.1 *Given two rectangular pieces R and R', we say that $R' \ge R$ if there is a transformation (combination of rotation, reflection and/or translation) such that every point $p \in R$ also belongs to R'.*

Definition 3.2 *Let C and C' be two cuts and let $P = \{P_1, P_2, \ldots, P_K\}$ and $P' = \{P'_1, P'_2, \ldots, P'_K\}$ be the sets of pieces defined by each cut, respectively. We say that C' covers C if there exists a bijective function $g : \{1, 2, \ldots, K\} \to \{1, 2, \ldots, K\}$ such that $P'_{g(i)} \ge P_i$ for all $i = 1, \ldots, K$.*

Definition 3.3 *Let $x \in S_L$ and $y \in S_W$, we define*

$$\lceil x \rceil_{R_L} = \min\{r \in R_L \mid x \leq r\} \text{ and } \lceil y \rceil_{R_W} = \min\{r \in R_W \mid y \leq r\}.$$

Theorem 3.1 *Let $x \in S_L$ ($y \in S_W$). Then, $x' = \lceil x \rceil_{R_L}$ ($y' = \lceil y \rceil_{R_W}$) defines a vertical (horizontal) guillotine cut that covers the vertical (horizontal) guillotine cut defined by x (y).*

Proof: The subrectangles generated by the vertical guillotine cut defined by x are $R_1 = (L_1, W_1) = (x, W)$ and $R_2 = (L_2, W_2) = (\langle L - x \rangle_{S_L}, W)$. The subrectangles generated by the vertical guillotine cut defined by x' are $R'_1 = (L'_1, W'_1) = (x', W)$ and $R'_2 = (L'_2, W'_2) = (\langle L - x' \rangle_{S_L}, W)$. It is easy to see that $R'_1 \geq R_1$ and that, by Lemma A.3, $R'_2 = R_2$. The proof for the horizontal cut is analogous. $\square$

Theorem 3.2 *Let $(x_1, x_2, y_1, y_2) \in S_L^2 \times S_W^2$ be a first-order non-guillotine cut. Then, $(x'_1, x'_2, y'_1, y'_2) = (\lceil x_1 \rceil_{R_L}, \lceil x_2 \rceil_{R_L}, \lceil y_1 \rceil_{R_W}, \lceil y_2 \rceil_{R_W}) \in R_L^2 \times R_W^2$ defines a first-order non-guillotine cut that covers the one defined by (x_1, x_2, y_1, y_2).*

Proof: Let R_1, R_2, R_3, R_4 and R_5 the five subrectangles generated by (x_1, x_2, y_1, y_2) (see Figure 3), and R'_1, R'_2, R'_3, R'_4 and R'_5 the five subrectangles generated by (x'_1, x'_2, y'_1, y'_2). It is easy to see that $R'_4 \geq R_4$. By Lemma A.3, $R'_1 \geq R_1$, $R'_2 \geq R_2$ and $R'_5 \geq R_5$; and, by Lemma A.4, $R'_3 \geq R_3$. $\square$

Regarding the use of lower bounds, other than the trivial one obtained by homogeneous packings, preliminary experiments suggested that the effort of computing them does not compensate the poor fathoming of nodes of the search tree provided by them. In [10] it is mentioned that the upper bound introduced by Barnes [5] is better than the trivial bound given by the areas ratio in around 4.2% of the cases. Our experiments confirm this claim. However, computing the Barnes upper bound only once per subproblem and saving them to be used later makes its use profitable. Other upper bounds studied in e.g. [16, 19, 24, 25] could also be considered.

A control of recursion depth is incorporated to avoid multiple resolutions of the same subproblem. The test made in [21] consists of, given a subproblem, solving it again if the current depth is smaller than the depth related to its stored solution, since a deeper recursion could potentially find a better solution. In the present implementation, we also save the information whether the process of computing the solution of the subproblem was influenced by the depth limit. If it was not, the subproblem is never solved again (see Figure 4).

In the following we analyze symmetries. Degenerated first-order non-guillotine cuts in which a rectangle is cut in exactly three or four subrectangles can be eliminated. This elimination is based on the fact that these cuts can be generated by two or three consecutive guillotine cuts. Therefore, without loss of generality, it is possible to consider only guillotine cuts and the first-order non-guillotine cuts that generate exactly five subrectangles.

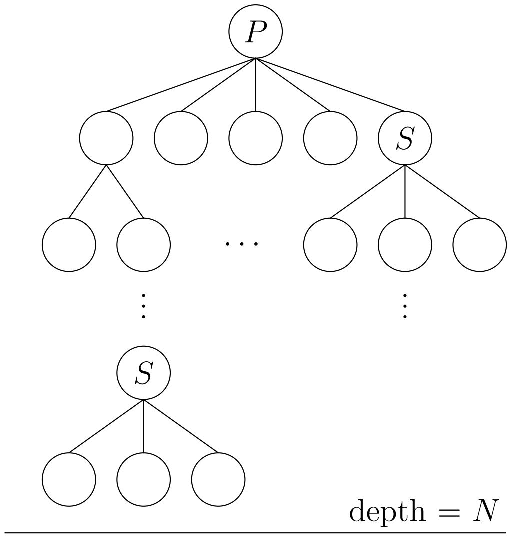


Figure 4: A tree search to represent the algorithm. The root node represents the main problem and each node represents a subproblem. Subproblem S appears twice. If the first occurrence of subproblem S corresponds to the node closer to the root, the stored solution will be used when subproblem S appears for the second time. If, on the other hand, the first occurrence corresponds to the deeper node, there are two possibilities when the subproblem appears for the second time. If the depth limit N was used to stop the recursion when solving the subproblem then it will be solved again. On the other hand, if the depth limit did not interfere in the subproblem resolution, the subproblem will not be solved again, as a better solution cannot be found.

We are interested in symmetries for the guillotine cuts and the non-degenerate first-order non-guillotine cuts. We say that two cuts are *symmetric* or *equivalent* when they both generate the same set of normalized subproblems.

Symmetries for vertical guillotine cuts

Let $(x_1, x_2, y_1, y_2) \in R_L^2 \times R_W^2$ be (a quadruple that represents) a vertical guillotine cut. Without loss of generality, we assume that $y_1 = y_2 = 0$ and $x_1 = x_2 = x$. We claim that it is enough to generate vertical guillotine cuts with $x \in (0, \lfloor L/2 \rfloor]$.

Theorem 3.3 *Every vertical guillotine cut $(x_1, x_2, y_1, y_2) \in R_L^2 \times R_W^2$ such that $y_1 = y_2 = 0$, $x_1 = x_2 = x$, $x > L/2$ is equivalent to a vertical guillotine cut $(x'_1, x'_2, y'_1, y'_2) \in R_L^2 \times R_W^2$ such that $y'_1 = y'_2 = 0$, $x'_1 = x'_2 = x'$, $x' \leq \lfloor L/2 \rfloor$.*

Proof: Let R_1 and R_2 be the two subrectangles generated by (x_1, x_2, y_1, y_2) with $y_1 = y_2 = 0$ and $x_1 = x_2 = x$ (see Figure 5). We show that there is a one-to-one equivalence relation between $\{R_1, R_2\}$ and the set $\{R'_1, R'_2\}$ of subrectangles generated by (x'_1, x'_2, y'_1, y'_2) with

$y'_1 = y'_2 = 0$, $x'_1 = x'_2 = x'$ and

$$x' = \langle L - x \rangle_{S_L} \leq \lfloor L/2 \rfloor. \tag{2}$$

Since

$$\begin{aligned} R_1 &= (L_1, W_1) = (x, W), & R'_1 &= (L'_1, W'_1) = (x', W), \\ R_2 &= (L_2, W_2) = (\langle L - x \rangle_{S_L}, W), & R'_2 &= (L'_2, W'_2) = (\langle L - x' \rangle_{S_L}, W), \end{aligned}$$

by (2) and Lemma A.2 follows that $R_1 = R'_2$ and $R_2 = R'_1$. □

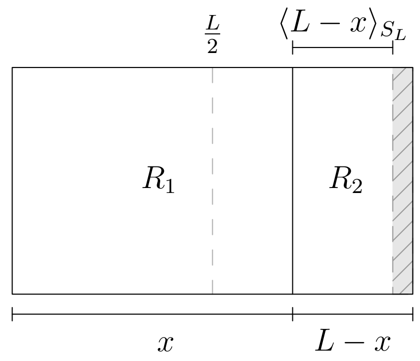


Figure 5: Vertical guillotine cut (x_1, x_2, y_1, y_2) with $x_1 = x_2 = x$ and $y_1 = y_2 = 0$. R_1 and R_2 are the normalized subrectangles generated by the cut. $R_1 = (L_1, W_1) = (x, W)$ and $R_2 = (L_2, W_2) = (\langle L - x \rangle_{S_L}, W)$.

The case of horizontal guillotine cuts is analogous. A corollary of Theorem 3.3 is that a method needs to consider only guillotine cuts $(x_1, x_2, y_1, y_2) \in R_L^2 \times R_W^2$ such that:

1. $y_1 = y_2 = 0$ and $0 < x_1 = x_2 = x \leq \lfloor L/2 \rfloor$; and
2. $x_1 = x_2 = 0$ and $0 < y_1 = y_2 = y \leq \lfloor W/2 \rfloor$.

Symmetries for non-degenerate first-order non-guillotine cuts

We analyze now symmetries for the non-degenerate first-order non-guillotine cuts. In order to do that, we divide the pallet into four regions called A, B, C and D (see Figure 6). From now on, until the end of this section, we will use the term “cut” to refer to a “non-degenerate first-order non-guillotine cut”.

We call p the center of “Subrectangle 3”, the central subrectangle of a cut (refer to Figure 3(b)). The symmetries are analyzed considering the position of p. We claim that a cut with $p \in \operatorname{int} D$ (the interior of D) is equivalent to a cut with $p \in \operatorname{int} A$; see Figure 7. (Analogously, a cut with $p \in \operatorname{int} C$ is equivalent to a cut with $p \in \operatorname{int} B$.) The same claim is also made for a cut with $p \in \{(x, y) \mid x = L/2 \text{ and } y \in [W/2, W]\}$, that is equivalent to a cut with $p \in \{(x, y) \mid x = L/2 \text{ and } y \in [0, W/2]\}$.

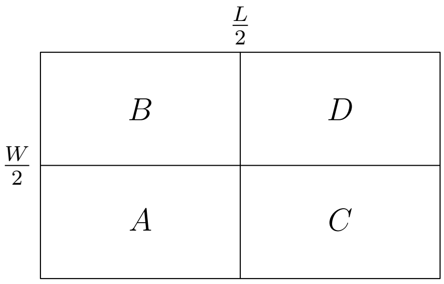


Figure 6: The four regions used to analyze symmetries of non-degenerate first-order non-guillotine cuts.

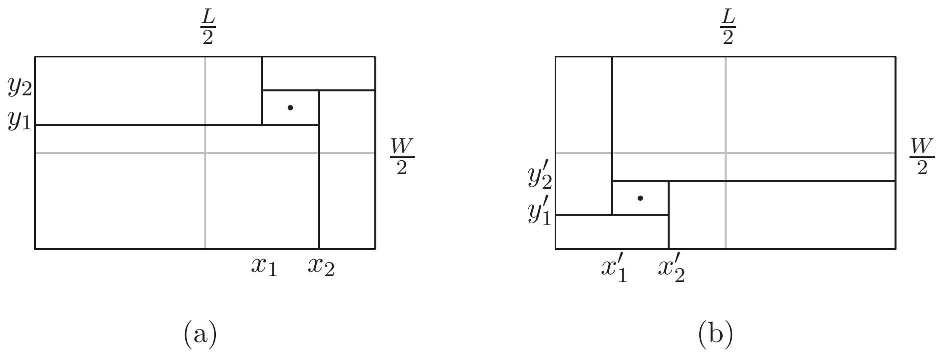


Figure 7: (a) A non-degenerate first-order non-guillotine cut generated by $(x_1, x_2, y_1, y_2) \in R_L^2 \times \in R_W^2$ with the center of Subrectangle 3 in region D. (b) A non-degenerate first-order non-guillotine cut generated by $x'_1 = \langle L - x_2 \rangle_{S_L}, x'_2 = \langle L - x_1 \rangle_{S_L}, y'_1 = \langle W - y_2 \rangle_{S_W}$ and $y'_2 = \langle W - y_1 \rangle_{S_W}$. The latter cut has the center of Subrectangle 3 in region A and is equivalent to the first cut.

Theorem 3.4 *Every non-degenerate first-order non-guillotine cut with the center of Subrectangle 3 in region $F = C \cup D \cup \{(x, y) \mid x = L/2$ and $y \in [W/2, W]\}$ is equivalent to a non-degenerate first-order non-guillotine cut with the center of Subrectangle 3 in region $E = A \cup B \cup \{(x, y) \mid x = L/2$ and $y \in [0, W/2]\}$. Note that the original cut as well as its equivalent cut are determined by quadruples of raster points.*

Proof: Consider problem (L, W, l, w). Let R_L and R_W be the set of raster points defined in (1). Let $(x_1, x_2, y_1, y_2) \in R_L^2 \times R_W^2$ be a non-degenerate first-order non-guillotine cut. Let $(x, y) = ((x_1 + x_2)/2, (y_1 + y_2)/2)$ be the center of Subrectangle 3. We show that if $(x, y) \in \operatorname{int} D$ then there is an equivalent cut $(x'_1, x'_2, y'_1, y'_2) \in R_L^2 \times R_W^2$ such that $(x', y') = ((x'_1 + x'_2)/2, (y'_1 + y'_2)/2) \in \operatorname{int} A$. Moreover,

$$(x'_1, x'_2, y'_1, y'_2) = (\langle L - x_2 \rangle_{S_L}, \langle L - x_1 \rangle_{S_L}, \langle W - y_2 \rangle_{S_W}, \langle W - y_1 \rangle_{S_W}). \tag{3}$$

The five subrectangles generated by (x_1, x_2, y_1, y_2) and (x_1', x_2', y_1', y_2') are:

$$\begin{array}{ll}
R_1 = (L_1, W_1) = (x_1, \langle W - y_1 \rangle_{S_W}), & R_1' = (L_1', W_1') = (x_1', \langle W - y_1' \rangle_{S_W}), \\
R_2 = (L_2, W_2) = (\langle L - x_1 \rangle_{S_L}, \langle W - y_2 \rangle_{S_W}), & R_2' = (L_2', W_2') = (\langle L - x_1' \rangle_{S_L}, \langle W - y_2' \rangle_{S_W}), \\
R_3 = (L_3, W_3) = (\langle x_2 - x_1 \rangle_{S_L}, \langle y_2 - y_1 \rangle_{S_W}), & R_3' = (L_3', W_3') = (\langle x_2' - x_1' \rangle_{S_L}, \langle y_2' - y_1' \rangle_{S_W}), \\
R_4 = (L_4, W_4) = (x_2, y_1), & R_4' = (L_4', W_4') = (x_2', y_1'), \\
R_5 = (L_5, W_5) = (\langle L - x_2 \rangle_{S_L}, y_2), & R_5' = (L_5', W_5') = (\langle L - x_2' \rangle_{S_L}, y_2'),
\end{array}$$

respectively. By (3) and using Lemma A.2, it is easy to verify that $R_1' = R_5$, $R_2' = R_4$, $R_3' = R_3$, $R_4' = R_2$ and $R_5' = R_1$. Proofs for $(x, y) \in \operatorname{int} C$ and for (x, y) such that $x = L/2$ and $y > W/2$ are analogous. $\square$

To end this section, Algorithms 1 and 2 present a detailed description of the original five-block recursive method [21, 22] and its improved version, respectively.

4 Refinements of the L-approach

The L-approach in phase 2 of the recursive partitioning approach is based on the computation of a recursive formula of dynamic programming that deals with a huge number of subproblems [17, 6]. As in the recursive five-block approach in phase 1, we combine the L-approach with the raster points. Its usage is straightforward and the implementation also uses the data structures described in [6].

We present two new ways of dividing an L-shaped piece into two L-shaped pieces that were not considered in [17]. Following the notation in [17], the normalized L-shaped piece represented by the quadruple (X, Y, x, y), with $X \geq x$ and $Y \geq y$, is denoted by $L(X, Y, x, y)$ and defined as the topological closure of the rectangle whose diagonal goes from $(0, 0)$ to (X, Y) minus the rectangle whose diagonal goes from (x, y) to (X, Y). Moreover, the division of an L-shaped piece into two L-shaped pieces can be determined by a pair (x', y'). The two new subdivisions, called B_8 and B_9, are given by:

$$\begin{array}{lllll}
B_8: & x' \in [0, x], & y' \in [y, Y], & L(x, Y, x', Y - y'), & L(X - x', y', x - x', y), \\
B_9: & x' \in [x, X], & y' \in [0, y], & L(x', Y - y', x, y - y'), & L(X, y, X - x', y').
\end{array}$$

Figure 8 shows the nine ways of dividing a rectangle or an L-shaped piece into two L-shaped pieces. The new ones are the last two. Their usage does not show any clear advantage to the method, other than completeness, since we did not find a counter-example for which the absence of subdivisions B_8 and B_9 prevents the approach of obtaining an optimal solution.

It remains as an open problem to prove a statement similar to Theorems 3.1 and 3.2 regarding L-shaped cuts $B_1, \ldots, B_9$, i.e., to prove that there is no loss of generality in considering cuts given by raster points instead of cuts given by integer conic combinations, in the L-approach.

One of the key subjects of the methods analyzed in the present work is related to the storage of information of the subproblems that are previously considered by the algorithm during its execution. In particular, we save a lower and an upper bound on the optimal

Algorithm 1: Pseudo-code of the original version of the recursive five-block heuristic as introduced [22].

Input: $L, W, l, w, n \in \mathbb{Z}$.
Output: Number of (l, w)-boxes packed within the (L, W)-pallet.

```
Recursive-BD(L, W, l, w, n)
begin
    z_lb ← lowerBound[I_L, J_W]
    z_ub ← upperBound[I_L, J_W]
    if z_lb = z_ub or depth[I_L, J_W] ≤ n then
        depth[I_L, J_W] ← min{depth[I_L, J_W], n}
        return z_lb
    Build sets S_L and S_W for (L, W, l, w)
    foreach x_1 ∈ S_L such that x_1 ≤ L − w do
        foreach x_2 ∈ S_L such that x_1 ≤ x_2 ≤ L − w do
            foreach y_1 ∈ S_W such that y_1 ≤ W − w do
                foreach y_2 ∈ S_W such that y_1 ≤ y_2 ≤ W − w do
                    if this pattern is not symmetrical to any other then
                        for i ← 1 to 5 do
                            Compute (L_i, W_i)
                            z^i_lb ← lowerBound[I_{L_i}, J_{W_i}]
                            z^i_ub ← upperBound[I_{L_i}, J_{W_i}]
                        S_lb ← Σ_{i=1}^{5} z^i_lb
                        S_ub ← Σ_{i=1}^{5} z^i_ub
                        if n < N then
                            for i ← 1 to 5 do
                                z_i ← Recursive-BD(L_i, W_i, l, w, n + 1)
                                S_lb ← S_lb + z_i − z^i_lb
                                S_ub ← S_ub + z_i − z^i_ub
                                if z_lb ≥ S_ub then
                                    break
                                if S_lb > z_lb then
                                    z_lb ← S_lb
                                    if z_lb = z_ub then
                                        depth[I_L, J_W] ← n
                                        return z_lb
                        if S_lb > z_lb then
                            z_lb ← S_lb
                            if z_lb = z_ub then
                                depth[I_L, J_W] ← n
                                return z_lb
    depth[I_L, J_W] ← n
    return z_lb
end
```

Algorithm 2: Improved version of the recursive five-block heuristic. This pseudo-code, together with the pseudo-code of routine Solve below, include all the improvements described in the present work.

Input: $L, W, l, w, n \in \mathbb{Z}$.

Output: Number of (l, w)-boxes packed within the (L, W)-pallet.

```
Recursive-BD2(L, W, l, w, n)
begin
    if W > L then
        Swap(L, W)
    z_lb ← lowerBound[I_L, J_W]
    z_ub ← upperBound[I_L, J_W]
    reachedLimit[I_L, J_W] ← false
    if z_lb = z_ub then
        return z_lb
    Build sets R_L and R_W for (L, W, l, w)
    foreach x_1 ∈ R_L such that x_1 ≤ ⌊L/2⌋ do
        foreach x_2 ∈ R_L such that x_1 < x_2 and x_1 + x_2 ≤ L do
            foreach y_1 ∈ R_W such that y_1 < W do
                foreach y_2 ∈ R_W such that y_1 < y_2 and y_1 + y_2 ≤ W do
                    if ¬(x1 + x2 = L and y1 + y2 > W) then
                        Compute (L_i, W_i) for i = 1, ..., 5
                        P ← {(L_1, W_1), ..., (L_5, W_5)}
                        z_lb ← max{z_lb, Solve(L, W, n, z_lb, P)}
                        if z_lb = z_ub then
                            return z_lb
    foreach x_1 ∈ R_L such that x_1 ≤ ⌊L/2⌋ do
        x_2 ← x_1    y_1 ← 0    y_2 ← 0
        Compute (L_1, W_1) and (L_2, W_2)
        P ← {(L_1, W_1), (L_2, W_2)}
        z_lb ← max{z_lb, Solve(L, W, n, z_lb, P)}
        if z_lb = z_ub then
            return z_lb
    foreach y_1 ∈ R_W such that y_1 ≤ ⌊W/2⌋ do
        y_2 ← y_1    x_1 ← 0    x_2 ← 0
        Compute (L_2, W_2) and (L_5, W_5)
        P ← {(L_2, W_2), (L_5, W_5)}
        z_lb ← max{z_lb, Solve(L, W, n, z_lb, P)}
        if z_lb = z_ub then
            return z_lb
    return z_lb
end
```

Algorithm 3: Subroutine Solve used by the improved recursive five-block heuristic.

```
Solve(L, W, n, z_lb, P)
begin
    z_ub ← upperBound[I_L, J_W]
    for i ← 1 to |P| do
        z^i_lb ← lowerBound[I_{L_i}, J_{W_i}]
        z^i_ub ← upperBound[I_{L_i}, J_{W_i}]
    S_lb ← \sum_{i=1}^{|P|} z^i_lb
    S_ub ← \sum_{i=1}^{|P|} z^i_ub
    if n < N then
        if z_lb < S_ub then
            for i ← 1 to |P| do
                if depth[I_{L_i}, J_{W_i}] > n and reachedLimit[I_{L_i}, J_{W_i}] then
                    z_i ← Recursive-BD2(L_i, W_i, l, w, n + 1)
                    lowerBound[I_{L_i}, J_{W_i}] ← z_i
                    depth[I_{L_i}, J_{W_i}] ← n
                    if ¬reachedLimit[I_{L_i}, J_{W_i}] then
                        upperBound[I_{L_i}, J_{W_i}] ← z_i
                else
                    z_i ← lowerBound[I_{L_i}, J_{W_i}]
                if reachedLimit[I_{L_i}, J_{W_i}] then
                    reachedLimit[I_L, J_W] ← true
                S_lb ← S_lb + z_i − z^i_lb
                S_ub ← S_ub + z_i − z^i_ub
                if z_lb ≥ S_ub then
                    return z_lb
                if S_lb > z_lb then
                    z_lb ← S_lb
                    if z_lb = z_ub then
                        reachedLimit[I_L, J_W] ← false
                        return z_lb
    else
        reachedLimit[I_L, J_W] ← true
        if S_lb > z_lb then
            z_lb ← S_lb
            if z_lb = z_ub then
                reachedLimit[I_L, J_W] ← false
                return z_lb
    return z_lb
end
```

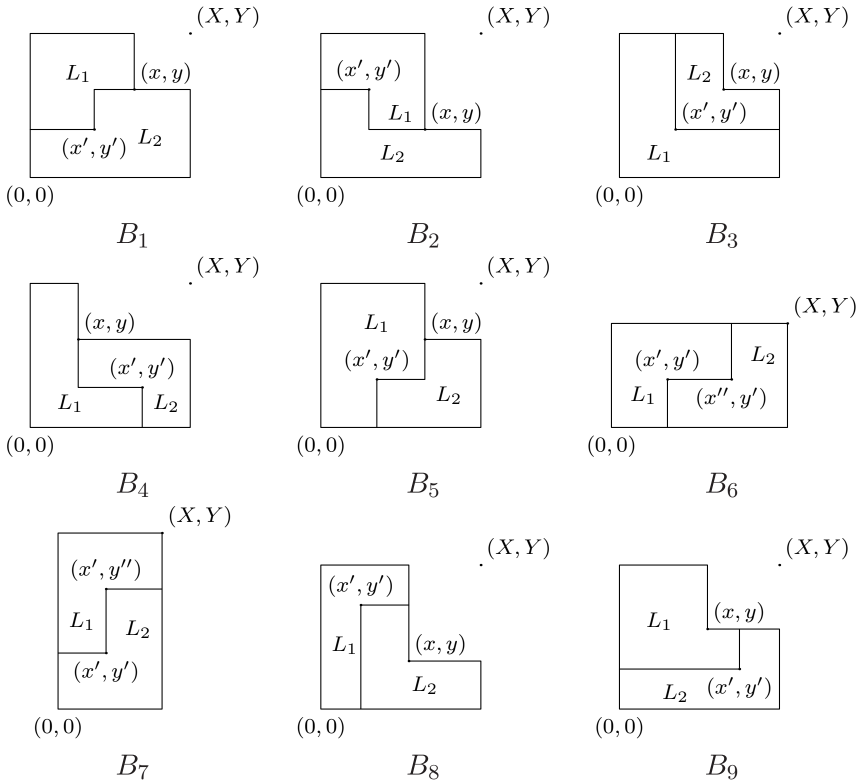


Figure 8: Subdivisions of an L-shaped piece into two L-shaped pieces. The last two are the new ones presented in this work.

value (if they are equal, the solution is optimal). If the lower bound corresponds to a homogeneous packing, we save whether the homogeneous packing is vertical or horizontal. Otherwise, we save the parameters of the cut that determines the lower bound, in order to be able to report the solution found by the method. The relevant amount is the number of all *possible* subproblems, and not the real number of generated subproblems, which is much smaller than the first one. In [6], it was empirically shown that less than 2% of all the possible subproblems is in fact generated by the L-approach. Note that the fact of saving information of the already solved subproblems (of the problem currently being solved) is related to the idea of creating a pool of solutions described by [8].

An important result related to this subject is that "subproblems of subproblems are subproblems of the original problem". This is a valid claim for subproblems generated in the recursive five-block heuristic in phase 1, as well as in the L-approach in phase 2. In other words, all subrectangles $(\hat{L}, \hat{W})$ and L-shaped pieces $(\hat{X}, \hat{Y}, \hat{x}, \hat{y})$ generated through the methods are such that $\hat{L}, \hat{X}, \hat{x} \in R_L$, the set of raster points associated to (L, l, w) (where L is the dimension of the original problem), and $\hat{W}, \hat{Y}, \hat{y} \in R_W$, the set of raster points associated to (W, l, w) (where W is the dimension of the original problem). As a consequence, the number of all possible subproblems is $O(|R_L|\ |R_W|)$ in the recursive five-block heuristic and $O(|R_L|^2 \ |R_W|^2)$ in the L-approach.

Consider a set of raster points R_S (where S is either L or W) as an ordered set (with its elements in increasing order). Consider now an array u of dimension S. If s is the i-th element of R_S, then $u_s = i$. Positions of u that do not correspond to elements in R_S are undefined. Using this kind of indexing arrays it is possible to map, in constant time, a pair of raster points $(\hat{L}, \hat{W})$ or a quadruple of raster points $(\hat{X}, \hat{Y}, \hat{x}, \hat{y})$ into indices $(I_{\hat{L}}, I_{\hat{W}})$ and $(I_{\hat{X}}, J_{\hat{Y}}, i_{\hat{x}}, j_{\hat{y}})$, respectively. In other words, we have a trivial way of associating every possible subproblem with a pair or quadruple of indices [6].

In the case of the recursive five-block heuristic, provided the quantity $|R_L||R_W|$ is not too large, we can simply use a two-dimensional array of dimension $|R_L|$ by $|R_W|$ to save the information related to each subproblem $(\hat{L}, \hat{W})$ into position $(I_{\hat{L}}, I_{\hat{W}})$. In the case of the L-approach, if the quantity $|R_L|^2 \ |R_W|^2$ is affordable (i.e., if the computer has enough memory), a four-dimensional array can be used. Otherwise, we proceed as follows. If $|R_L|^2 \ |R_W|$ is affordable, a three-dimensional array is used whose (I, J, i) element is a balanced binary search tree with key j. If $|R_L|^2 \ |R_W|$ is not affordable but $|R_L| \ |R_W|$ is, we consider a two-dimensional array whose element (I, J) is a balanced binary search tree with key (i, j). Figure 9 illustrates the case of a three-dimensional array with a balanced binary search tree in each position. This strategy to save the information of the subproblems enables us to apply the method to potentially large problems.

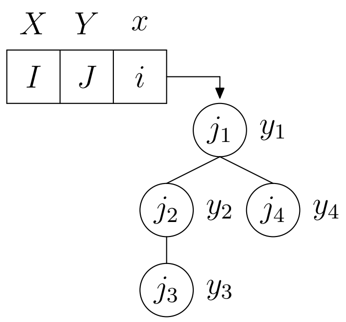


Figure 9: Data structure to store information related to the subproblems. The figure represents a data structure with a three-dimensional array and a binary search tree. The information of subproblems (X, Y, x, y_1), (X, Y, x, y_2), (X, Y, x, y_3) and (X, Y, x, y_4) is being saved in a binary search tree with keys $j_1, \ldots, j_4$ located at position (I, J, i) of a three-dimensional array.

To prove that "subproblems of subproblems are subproblems of the original problem", we need to prove that all subrectangles $(\hat{L}, \hat{W})$ and L-shaped pieces $(\hat{X}, \hat{Y}, \hat{x}, \hat{y})$ generated using the five-block heuristic and the L-approach are such that $\hat{L}, \hat{X}, \hat{x} \in R_L$ and $\hat{W}, \hat{Y}, \hat{y} \in R_W$. By the definition of the methods, $\hat{L}$, $\hat{X}$, $\hat{x}$, $\hat{W}$, $\hat{Y}$ and $\hat{y}$ are:

- raster points by themselves,
- raster points by construction and by the definition of raster points, or

- the raster point of the difference of two raster points.

The first two cases are trivial. Lemma A.1 proves that the normalization of the difference of two raster points is a raster point. In particular, the result is a little more general, as the second element can be an integer conic combination, instead of a raster point.

5 Complexity of the recursive partitioning approach

Our implementation of the recursive partitioning approach is a memoized dynamic programming algorithm (see, for example, [9] pp. 347–349). Each subproblem of the five-block heuristic is given by a rectangle $(X, Y) \in R_L \times R_W$ and each subproblem of the L-approach is given by an L-shaped piece $(X, Y, x, y) \in R_L \times R_W \times R_L \times R_W$. Thus, to save the solution of every possible subproblem, the recursive partitioning approach uses a table of size $|R_L||R_W|$ for phase 1 and a table of size $|R_L|^2|R_W|^2$ for phase 2. In both phases the method also uses a few integer arrays with no more than L elements. Therefore, the memory complexity of phases 1 and 2 is $O(|R_L||R_W|+L)$ and $O(|R_L|^2|R_W|^2+L)$, respectively; the memory complexity of the recursive partitioning approach being $O(|R_L|^2|R_W|^2 + L)$.

To simplify the time complexity analysis below, we ignore from now on the time savings related to symmetries and normalization of subproblems. We assume that no limit to the recursion depth of the five-block heuristic is imposed and, as a consequence, each subproblem is solved at most once. We also assume that there is enough space for the L-approach to save subproblems solutions in a four-dimensional array of size $|R_L|^2|R_W|^2$.

As the worst-case time complexity of a memoized dynamic programming method is equivalent to the complexity of its iterative dynamic programming counterpart, we analyze only the latter one. In terms of the recursive partitioning approach it means that in phase 1, as well as in phase 2, smaller subproblems are solved first. It also means that in the five-block heuristic the recursive call in subroutine Solve (Algorithm 3, line 12) is replaced by the $O(1)$ task of obtaining the solution of the subproblem from a table. Thus, the worst-case time complexity of subroutine Solve (Algorithm 3) is $O(1)$.

Consider a subproblem given by a rectangle (X, Y). In Algorithm 2, the worst-case time complexity for lines 1 to 8 is $O(1)$; the worst-case time complexity of line 9 (generation of raster points sets) is $O(|S_X| + |S_Y|)$; and the time complexities of the loops are $O(|R_X|^2|R_Y|^2)$, $O(|R_X|)$ and $O(|R_Y|)$ (lines 10 to 19, 20 to 26 and 27 to 33, respectively). Therefore, the worst-case time complexity for solving all possible subproblems is given by

$$\sum_{(X,Y)\in R_L\times R_W} O(|S_X| + |S_Y| + |R_X|^2|R_Y|^2 + |R_X| + |R_Y|).$$

The cost of initializing the table and the arrays is $O(|R_L||R_W|+L)$. Therefore, the worst-case time complexity of the five-block heuristic is

$$\begin{aligned} O(|R_L||R_W| + L) + \textstyle\sum_{(X,Y)\in R_L\times R_W} O(|S_X| + |S_Y| + |R_X|^2|R_Y|^2 + |R_X| + |R_Y|) \\ = O(|R_L||R_W|(|R_L|^2|R_W|^2 + |S_L| + |S_W|) + L). \end{aligned} \tag{4}$$

Let us now analyze the worst-case time complexity of the L-approach. We analyze in separate subproblems given by a non-degenerated L-shaped piece and by a rectangular piece. There are seven different ways of partitioning a non-degenerated L-shaped piece $(X, Y, x, y) \in R_L \times R_W \times R_L \times R_W$ into two L-shaped pieces (see B_1, B_2, B_3, B_4, B_5, B_8 and B_9 in Figure 8). Each of them is determined by a pair $(x', y') \in R_X \times R_Y$. Thus, the worst-case complexity for solving a subproblem given by a non-degenerated L-shaped piece (X, Y, x, y), including the complexity $O(|S_X| + |S_Y|)$ for generating its raster points sets, is

$$O(|R_X||R_Y| + |S_X| + |S_Y|). \tag{5}$$

Analogously, the worst-case complexity for solving a subproblem given by a rectangular piece (X, Y) is

$$O(|R_X|^2|R_Y| + |R_X||R_Y|^2 + |S_X| + |S_Y|). \tag{6}$$

The cost of initializing the table and the arrays is $O(|R_L|^2|R_W|^2 + L)$ and summing (5) and (6) over all possible subproblems, we have that the worst-case complexity of the L-approach is

$$\begin{aligned} O(|R_L|^2|R_W|^2 + L) + \textstyle\sum_{(X,Y,x,y)\in R_L\times R_W\times R_L\times R_W} O(|R_X||R_Y| + |S_X| + |S_Y|)+ \\ \textstyle\sum_{(X,Y)\in R_L\times R_W} O(|R_X|^2|R_Y| + |R_X||R_Y|^2 + |S_X| + |S_Y|) \\ = O(|R_L|^2|R_W|^2(|R_L||R_W| + |S_L| + |S_W|) + L). \end{aligned} \tag{7}$$

Finally, if the worst-case complexity of the five-block heuristic is given by (4) and the worst-case complexity of the L-approach is given by (7), we have that the worst-case complexity of the recursive partitioning approach is given by (7).

6 Numerical experiments

In this section, we present the computational results obtained with the refined version of the recursive five-block heuristic in Section 3, the refined version of the L-approach in Section 4 and the combined recursive partitioning approach, here simply referred to as Five-block Algorithm, L-Algorithm and Recursive Partitioning Algorithm, respectively. These algorithms were coded in C/C++ language. All the experiments were run on a 2.4GHz Intel Core2 Quad Q6600 with 4.0GB of RAM memory and Linux Operating System. Compiler option -O3 was adopted.

6.1 Pallet loading instances of Cover I, II and III

In order to evaluate the performances of the algorithms, initially we took five well-known data sets of the manufacturer's pallet loading literature:

Cover IA: 8,274 instances satisfying $1 \le \frac{L}{W} \le 3$, $1 \le \frac{l}{w} \le 4$ and $1 \le \frac{LW}{lw} < 51$;

Cover IIA: 41,831 instances satisfying $1 \le \frac{L}{W} \le 2.2$, $1 \le \frac{l}{w} \le 4$ and $51 \le \frac{LW}{lw} < 101$;

Cover IB: 7,827 instances satisfying $1 \le \frac{L}{W} \le 2$, $1 \le \frac{l}{w} \le 4$ and $1 \le \frac{LW}{lw} < 51$;

Cover IIB: 40,609 instances satisfying $1 \le \frac{L}{W} \le 2$, $1 \le \frac{l}{w} \le 4$ and $51 \le \frac{LW}{lw} < 101$;

Cover IIIB: 98,016 instances satisfying $1 \le \frac{L}{W} \le 2$, $1 \le \frac{l}{w} \le 4$ and $101 \le \frac{LW}{lw} < 151$.

The generation of the cover set problems was introduced in [12]. Each instance in the cover sets is a representative of an equivalence class of problems containing infinite elements. Cover IA and IIA were extensively used in the pallet loading problem literature; see e.g. [16, 25, 29]. Cover IB, IIB and IIIB were recently generated and presented in [1]. Instances in Cover IB, IIB and IIIB are the minimum size instances (see, for example, [20]) of their respective equivalence classes, while instances in Cover IA and IIA are not. See [1] for details.

We start showing empirically computed average sizes of the sets of integer conic combinations and raster points for problems in the cover sets. Consider a problem (L, W, l, w) in Cover IA, IIA, IB, IIB or IIIB. We are interested in computing, for example, $\gamma \in (0, 1]$ such that $|R_L| = \gamma(L + 1)$, i.e., the proportion between the number of raster points and the number of integer numbers between 0 and L. Considering problems from Covers type A, we have that the set of integer conic combinations is, on average, 71% smaller than $L + 1$, while the set of raster points is, on average, 31% smaller than the set of integer conic combinations. For Covers type B these figures are 40% and 33%, respectively. The difference in the average values of γ for Covers type A and B comes from the facts that: (i) Covers type B contain only minimum size instances of each class of problems; and (ii) the sizes of the raster points and integer conic combinations sets are invariant with respect to the units of measurement of the problem.

The aim of the next experiment is to determine the influence of the depth limit N in the recursion of the Five-block Algorithm. The same solutions for all problems of Cover IA, IIA, IB, IIB and IIIB were found using $N \ge 4$, while lower quality solutions were found using $N = 1, 2, 3$. Contrary to the observation in [21], the runtime decreases for increasing values of N, the best results being obtained for $N = \infty$, i.e., no limit in the recursion depth. This is a consequence of the control of recursion depth incorporated in the present work (to avoid multiple resolutions of the same subproblem). The tendency is that for large values of N the depth limit has no influence in the resolution of the subproblems. Saving this information avoids multiple resolutions of the same subproblem, even when it appears in different recursion levels.

Table 1 compares the computer runtimes (in seconds) between the original Five-block Algorithm in [21, 22] and its refined version described in Section 3. To evaluate the influence of each improvement in the overall behaviour of the refined method, the table shows from the third to the sixth column the performance of the refined method without the Barnes's upper bound, without the generation of only non-symmetric cuts, without the proper indexation for the usage of raster points and without the raster points, respectively. For example, note in the second row that the original version of the Five-block Algorithm uses 114.64 seconds (column 2) to solve all the problems in Cover IB, while the refined version presented here uses 4.51 seconds (column 7) to solve the same problems. Note also that the deterioration of the algorithm performance when none of the improvements is used (column 2) is sensitive.

Data set	Runtimes (in seconds)					
	Original version [21, 22]	Without Barnes's bound	Without generating only non-symmetric cuts	Without proper indexation for raster points	Without raster points	Full improved version
Cover IA	32.61	2.74	2.90	8.34	7.23	2.54
Cover IB	114.64	4.84	6.46	15.44	20.19	4.51
Cover IIA	9176.43	324.71	251.91	1049.54	867.22	200.48
Cover IIB	30233.11	775.51	793.97	787.29	3771.49	598.50
Cover IIIB	976871.02	18790.04	16897.56	76215.44	98467.22	13780.30

Table 1: Performance of the Five-block Algorithm. The table shows a comparison between the original version presented in [21, 22] (column 2) and the improved version presented in the present work (column 7). Moreover, it also shows the influence of each new feature in the full improved version (columns 3 to 6).

Table 2 compares the runtimes between the original L-Algorithm in [17] with the integer conic combinations (i.e., without raster points) and its refined version with the raster points as described in Section 4. Note that the reduction of the runtimes is substantial. (The original L-Algorithm was not used to solve problems in Cover IIIB as it would take a few months of computer runtime.)

L-Algorithm **with** raster points					
Data set	Runtimes (in seconds)				
	Total	Average	Standard deviation	Min	Max
Cover IA	4003.09	0.48	0.27	0.00	1.43
Cover IB	3754.65	0.47	0.28	0.00	1.64
Cover IIA	74292.08	1.77	2.22	0.00	21.34
Cover IIB	98414.81	2.42	2.72	0.00	24.26
Cover IIIB	2096308.86	21.38	22.75	0.00	167.44
L-Algorithm **without** raster points					
Data set	Runtimes (in seconds)				
	Total	Average	Standard deviation	Min	Max
Cover IA	6561.17	0.79	1.10	0.00	16.82
Cover IB	8844.76	1.13	1.74	0.00	23.81
Cover IIA	1452125.78	34.71	63.38	0.00	761.73
Cover IIB	2057429.27	50.66	81.19	0.00	836.60
Cover IIIB	–	–	–	–	–

Table 2: Comparison of the performances of the L-Algorithm with and without raster points. Using raster points, the method is around 20 times faster (Cover IIA and IIB). Note that the usage of the raster points in an efficient way was possible due to using the simple-but-effective indexation suggested in [6].

Table 3 presents the performance of the Recursive Partitioning Algorithm. The reductions of the runtimes, when compared with only applying the L-Algorithm, are very large. For example, note that the average runtime to solve each instance of data set Cover IIB reduces from 2.42 seconds (Table 2) to 0.71 seconds (Table 3). The table also shows that the Recursive Partitioning Algorithm consumes, on average, less than one second for solving an instance of the data sets Cover IA, IB, IIA and IIB; and it consumes, on average, less than eight seconds to solve an instance of the data set Cover IIIB. On the other hand, the Recursive Partitioning Algorithm uses a little less than three minutes to solve the most time-consuming problem of all the data sets. These times are very reasonable for a method conjectured (although not proven) to be optimal. Therefore, it is fair to say that, while the Tabu Search approach introduced in [1] runs faster than the Recursive Partitioning Algorithm, the latter finds better quality solutions in some hard problems of the data set Cover IIIB. The stopping criterion of the Tabu Search method is based on the number of iterations and the authors in [1] show some empirical relationships between the number of iterations of the method and the quality of the obtained solution. Thus, it is possible that using more iterations, the Tabu Search method would find better quality solutions.

Recursive Partitioning Algorithm					
Data set	Runtimes (in seconds)				
	Total	Average	Standard deviation	Min	Max
Cover IA	28.94	0.00	0.02	0.00	0.65
Cover IB	150.54	0.01	0.06	0.00	0.87
Cover IIA	6871.80	0.16	0.92	0.00	19.39
Cover IIB	29200.13	0.71	1.95	0.00	22.98
Cover IIIB	745602.23	7.60	17.43	0.00	164.72

Table 3: Performance of the Recursive Partitioning Algorithm. The method is around 3 times faster than the L-Algorithm with raster points and around 60 times faster than the L-Algorithm without raster points, as introduced in [17].

The Recursive Partitioning Algorithm optimally solved all the problems in Cover IA, IIA, IB and IIB, as well as 97,046 (over a total of 98,016) of the instances in Cover IIIB - the solutions of the remaining 970 examples are not proven optimal[1]. It is worth mentioning that for 116 instances out of 98,016, the Recursive Partitioning Algorithm improved the best solution found by the tabu search algorithm in [1]. For the remaining 97,900 instances, the solutions of both algorithms were the same.

6.2 Real cases of a woodpulp stowage problem

In this section we analyze the results obtained by applying the Recursive Partitioning Algorithm to solve practical examples of the woodpulp stowage problem in Brazilian ports. These examples are detailed in [27, 31]. As mentioned, the problem consists of determining the maximum number of stowed units of woodpulp into holds of dedicated maritime ships. Basically, this problem (essentially three-dimensional) can be reduced to

[1]The optimality certificate of the 97,046 problems of Cover IIIB comes from the fact of having reached the upper bounds presented in Cover IIIB. On the other hand, an optimality certificate for other 245 problems of Cover IIIB was obtained by solving the integer programming formulation [21, 17] using CPLEX 7.0. Then, up to the present moment (August 1, 2008), optimality certificates for the solutions found by the Recursive Partitioning Algorithm was obtained for 97,291 out of the 98,016 instances in Cover IIIB.

the two-dimensional case due to constraints provided by transport, and becomes similar to the manufacturer's pallet loading problem.

Table 4 presents, for each instance data (L, W, l, w), the solutions obtained by the Recursive Partitioning Algorithm. The first half of the table shows the solutions obtained in phase 1. When the Five-block Algorithm did not give a certificate of optimality, the second half of the table shows the solution obtained in phase 2 by the L-Algorithm. Note that this data set involves packings of up to 341 units. For all examples, the Recursive Partitioning Algorithm was able to find solutions at least as good as the solutions obtained by the method of Lagrangean relaxation with clusters in [27, 31]. In the examples marked in the table with an asterisk, the Recursive Partitioning Algorithm produced better solutions than the ones in [27, 31], which in turn are better than the stowage plans used in practice. Figure 10 depicts the obtained improved stowage plans.

In seven problems (namely, problems 1, 5, 6, 7, 11, 13 and 14) out of the fifteen woodpulp stowage problems, an optimality certificate was given by phase 1 of the Recursive Partitioning Algorithm. In another five problems (namely, problems 2, 3, 4, 8 and 9), an optimality certificate was obtained comparing the solution given by the Recursive Partitioning Algorithm with the Barnes's upper bound of their minimum size instance[2]. For the remaining problems (namely, problems 10, 12 and 15), an optimality certificate was obtained by solving the relaxation of an integer programming formulation [21, 17] using CPLEX 7.0. Therefore, the solutions obtained for all fifteen problems are proven to be optimal.

It is worth mentioning that the RAM memory of the present computational environment was enough to use four-dimensional arrays to store the information related to the subproblems of all the pallet loading problems in the cover sets. On the other hand, memory was not enough for the large woodpulp stowage problems. Considering the eight problems in which phase 2 of the Recursive Partitioning Algorithm was activated, in five of them a four-dimensional array was used and in the other three (namely, problems 9, 10 and 12) a three-dimensional array was used.

7 Concluding remarks

This study dealt with the problem of packing, orthogonally and without overlapping, identical rectangles in a rectangle. This problem appears in different logistics settings, such as the loading of boxes onto pallets, the arrangements of pallets in trucks and the stowing of cargo in ships. An effective two-phase recursive partitioning approach, combining improved versions of a recursive five-block heuristic and an L-approach for packing rectangles into larger rectangles and L-shaped pieces, was presented. The combined approach was able to rapidly find the optimal solutions of all instances of the well-known manufacturer's pallet loading problem sets Cover I and II. It was also effective for solv-

[2] The execution of phase II of the Recursive Partitioning Algorithm could have been avoided if we had computed this upper bound at the beginning of phase I. With this modification, the CPU time of the Recursive Partitioning Algorithm would have been, for problems 2, 3, 4, 8 and 9, the CPU time of phase I reported in Table 4.

Problem		Recursive Partitioning Algorithm			
ID	(L, W, l, w)	Phase 1 - Five-block Algorithm		Phase 2 - L-Algorithm	
		CPU Time (secs.)	Solution	CPU Time (secs.)	Solution
1	$(2296, 1230, 136, 94)$	0.12	219	–	–
2	$(2536, 1312, 144, 84)$	3.41	273	1036.31	273
3	$(2252, 1470, 144, 84)$	4.43	271	1339.59	271
4	$(1470, 1458, 144, 84)$	0.95	175	144.48	175
5	$(2296, 1230, 135, 92)$	0.00	226	–	–
6	$(1804, 1230, 137, 95)*$	0.33	169	–	–
7	$(2466, 1230, 137, 95)$	0.01	231	–	–
8	$(1804, 1750, 137, 95)*$	14.85	241	894.51	241
9	$(2426, 1230, 137, 95)$	3.33	227	1702.64	227
10	$(2530, 1320, 137, 95)*$	3.40	255	1455.89	255
11	$(2560, 1610, 143, 84)*$	24.42	341	–	–
12	$(2625, 1600, 137, 95)*$	8.41	320	7996.65	320
13	$(1838, 1600, 137, 95)*$	6.70	224	–	–
14	$(2100, 1600, 144, 84)$	0.00	277	–	–
15	$(1600, 1230, 137, 95)$	1.21	147	50.81	147

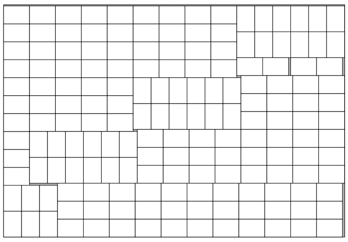
Table 4: Recursive Partitioning Algorithm for solving practical examples of the woodpulp stowage problem in Brazilian ports [27, 31]. In six out of the fifteen cases, better solutions were found. However, all the best solutions were found by the Five-block Algorithm (phase 1) of the Recursive Partitioning Algorithm.

ing the instances of problem set Cover III and practical examples of a woodpulp stowage problem, if compared to other methods from the literature.

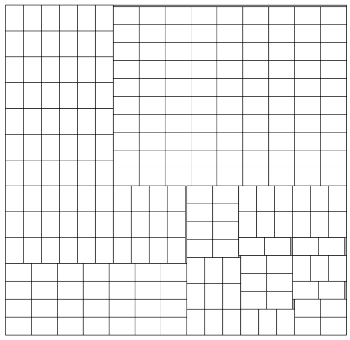
Possible refinements of the approach would be the use of more powerful upper bounds, e.g. based on linear programming or Lagrangean relaxation of a mathematical programming model for the problem. To cope with large scale problems, the recursions of the approach could be modified to consider discarding non-promising paths (subdivisions), based on the lower and upper bound information, as in the and/or-graph approach in [3]. Another interesting perspective for future research is to extend the approach to deal with the packing of different rectangles in a rectangle, i.e. a more general case of non-guillotine packing or cutting.

The current computer implementation of the combined recursive partitioning approach (including the source code in C/C++ language) and the data sets are available for benchmarking purposes at *http://www.ime.usp.br/∼egbirgin/packing/*.

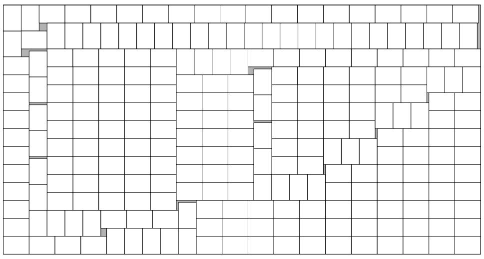
Acknowledgement: The authors would like to thank Dr. W. F. Mascarenhas for his help to prove Lemmas A.1 and A.2 an anonymous referee for his/her useful comments and suggestions.



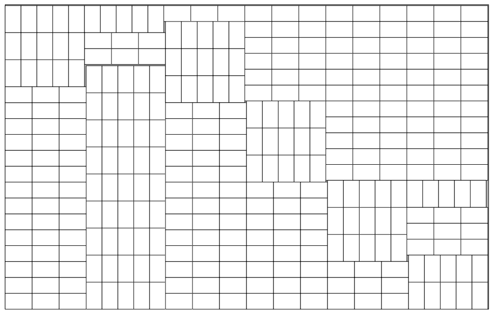
Instance $(1804, 1230, 137, 95)$ with 169 boxes



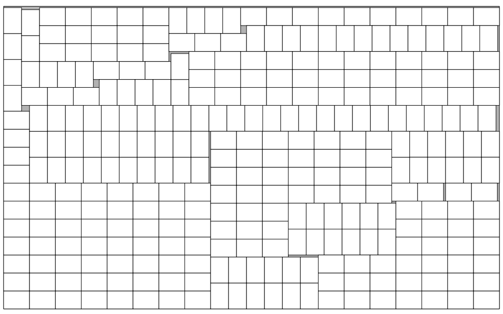
Instance $(1804, 1750, 137, 95)$ with 241 boxes



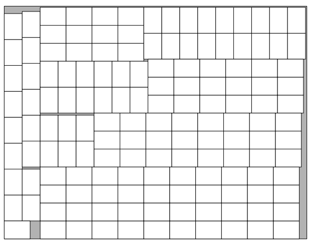
Instance $(2530, 1320, 137, 95)$ with 255 boxes



Instance $(2560, 1610, 143, 84)$ with 341 boxes



Instance $(2625, 1600, 137, 95)$ with 320 boxes



Instance $(1600, 1230, 137, 95)$ with 147 boxes

Figure 10: Six improved solutions found by the Recursive Partitioning Algorithm out of the fifteen practical examples of the woodpulp stowage problem in Brazilian ports presented in [27, 31].

References

[1] R. Alvarez-Valdes, F. Parreno and J. M. Tamarit, A tabu search algorithm for the pallet loading problem, *OR Spectrum* 27, pp. 43–61, 2005.

[2] R. Alvarez-Valdes, F. Parreno and J. M. Tamarit, A Branch-and-Cut Algorithm for the Pallet Loading Problem, *Computers and Operations Research* 32, pp. 3007–3029, 2005.

[3] M. Arenales and R. Morabito, An and/or-graph approach to the solution of two-dimensional non-guillotine cutting problems, *European Journal of Operational Research* 84, pp. 599–617, 1995.

[4] R. Balasubramanian, The pallet loading problem: A survey, *International Journal of Production Economics* 28, pp. 217–225, 1992.

[5] F. W. Barnes, Packing the maximum number of $m \times n$ tiles in a large $p \times q$ rectangle, *Discrete Mathematics* 26, pp. 93–100, 1979.

[6] E. G. Birgin, R. Morabito and F. H. Nishihara, A note on an L-approach for solving the manufacturer's pallet loading problem, *Journal of the Operational Research Society* 56, pp. 1448–1451, 2005.

[7] E. Bischoff and W. B. Dowsland, An application of the micro to product design and distribution, *Journal of the Operational Research Society* 33, pp. 271–280, 1982.

[8] L. Brunetta and P. Gregoire, A general purpose algorithm for three-dimensional packing, *Informs Journal on Computing* 17, pp. 328–338, 2005.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, Second Edition, The MIT Press and McGraw-Hill, Cambridge, 2001.

[10] K. A. Dowsland, Determining an upper bound for a class of rectangular packing problems, *Computers and Operations Research* 12, pp. 201–205, 1985.

[11] K. A. Dowsland, An exact algorithm for the pallet loading problem, *European Journal of Operational Research* 84, pp. 78–84, 1987.

[12] K. A. Dowsland, A combined database and algorithmic approach to the pallet-loading problem, *Journal of the Operational Research Society* 38, pp. 341–345, 1987.

[13] K. A. Dowsland and W. B. Dowsland, Packing problems, *European Journal of Operational Research* 56, pp. 2–14, 1992.

[14] H. Dyckhoff, A Typology of Cutting and Packing Problems, *European Journal of Operational Research* 44, pp. 145–159, 1990.

[15] A. Herbert and K. A. Dowsland, A family of genetic algorithms for the pallet loading problem, *Annals of Operations Research* 63, pp. 415–436, 1996.

[16] A. Letchford and A. Amaral, Analysis of upper bounds for the pallet loading problem, *European Journal of Operational Research* 132, pp. 582–593, 2001.

[17] L. Lins, S. Lins and R. Morabito, An L-approach for packing (l, w)-rectangles into rectangular and L-shaped pieces, *Journal of the Operational Research Society* 54, pp. 777–789, 2003.

[18] K. Maing-Kyu and G. Young-Gun, A fast algorithm for two-dimensional pallet loading problems of large size, *European Journal of Operational Research* 134, pp. 193–200, 2001.

[19] G. H. A. Martins, Packing in two and three dimensions, *Ph.D. Dissertation*, Naval Postgraduate School, CA, 2003.

[20] G. H. A. Martins and R. F. Dell, The minimum size instance of a pallet loading problem equivalence class, *European Journal of Operational Research* 179, pp. 17–26, 2007.

[21] R. Morabito and S. Morales, A simple and effective recursive procedure for the manufacturer's pallet loading problem, *Journal of the Operational Research Society* 49, pp. 819–828, 1998.

[22] R. Morabito and S. Morales, A simple and effective recursive procedure for the manufacturer's pallet loading problem (49, pp. 819–828, 1998), *Journal of the Operational Research Society* 50, pp. 876–876, 1999.

[23] R. Morabito, S. Morales and J. A. Widmer, Loading optimization of palletized products on trucks, *Transportation Research* E36, pp. 285–296, 2000.

[24] R. Morabito and R. Farago, A Tight Lagrangean Relaxation Bound for the Manufacturer's Pallet Loading Problem, *Studia Informatica Universalis* 2, pp. 57–76, 2002.

[25] J. Nelißen, How to use the structural constraints to compute an upper bound for the pallet loading problem, *European Journal of Operational Research* 84, pp. 662–680, 1995.

[26] V. Pureza and R. Morabito, Some Experiments with a Simple Tabu Search Algorithm for the Manufacturer's Pallet Loading Problem, *Computers and Operations Research* 33, pp. 804–819, 2006.

[27] G. M. Ribeiro and L. A. N. Lorena, Optimizing the woodpulp stowage using Lagrangean relaxation with clusters, *Journal of the Operational Research Society* 59, pp. 600-606, 2008.

[28] G. Scheithauer, Equivalence and dominance for problems of optimal packing of rectangles, *Ricerca Operativa* 27, pp. 3–34, 1997.

[29] G. Scheithauer and J. Terno, The G4-Heuristic for the Pallet Loading Problem, *Journal of the Operational Research Society* 47, pp. 511–522, 1996.

[30] G. Scheithauer and G. Sommerweiss, 4-block Heuristic for the Rectangle Packing Problem, *European Journal of Operational Research* 108, pp. 509–526, 1998.

[31] D. R. Sirtoli, G. M. Ribeiro and L. A. N. Lorena, Estivagem de unidades de celulose: uma análise prática com heurísticas de bloco, Annals of the XXVI ENEGEP, Fortaleza, CE, Brazil, 2006.

[32] G. Wäescher, H. Haußner and H. Schumann, An improved typology of cutting and packing problems, *European Journal of Operational Research* 183, pp. 1109–1130, 2007.

A Some raster points properties

Let (L, W, l, w) be a packing problem. Let S_L and S_W be its sets of integer conic combinations and R_L and R_W be its sets of raster points.

Lemma A.1 *Let $x \in R_L$ and $y \in S_L$ such that $x \geq y$. Then, $\langle x-y\rangle_{S_L} = \langle x-y\rangle_{R_L} \in R_L$.*

Proof: Since $x \in R_L$, there is $z \in S_L$ such that $x = \langle L-z\rangle_{S_L}$. We claim that $\langle x-y\rangle_{S_L} = \langle L-z-y\rangle_{S_L}$. Note that this implies the thesis, as $\langle L-(z+y)\rangle_{S_L} \in R_L$ because $z+y \in S_L$ since the sum s of any pair of elements in S_L belongs to S_L provided that $s \leq L$. But $s \leq L$ is implied by $L - s = L - z - y \geq \langle L-z\rangle_{S_L} - y = x - y \geq 0$. To show that $\langle x-y\rangle_{S_L} = \langle L-z-y\rangle_{S_L}$, note that:

(a) $x = \langle L-z\rangle_{S_L} \Longrightarrow x \leq L - z \Longrightarrow x - y \leq L - z - y \Longrightarrow \langle x-y\rangle_{S_L} \leq \langle L-z-y\rangle_{S_L}$.

(b) By definition of $\langle L-z-y\rangle_{S_L}$, there are r_1 and s_1 such that

  (i) $r_1 l + s_1 w = \langle L-z-y\rangle_{S_L}$,

  (ii) $r_1, s_1 \geq 0$, and

  (iii) $r_1 l + s_1 w \leq L - z - y$.

  Since $y \in S_L$, there are $r_y \geq 0$ and $s_y \geq 0$ such that $y = r_y l + s_y w$. Therefore, by (iii), we conclude that $(r_1+r_y)l + (s_1+s_y)w \leq L - z$. Then, since $x = \max\{rl+sw \mid r, s \geq 0,\ rl+sw \leq L-z\}$, we have that $x \geq (r_1+r_y)l + (s_1+s_y)w = r_1 l + s_1 w + y$. Hence, $x - y \geq r_1 l + s_1 w$. Thus, by definition of $\langle x-y\rangle_{S_L}$, we conclude that $\langle x-y\rangle_{S_L} \geq r_1 l + s_1 w$ and (i) shows that $\langle x-y\rangle_{S_L} \geq \langle L-z-y\rangle_{S_L}$.

Items (a) and (b) imply that $\langle x-y\rangle_{S_L} = \langle L-z-y\rangle_{S_L}$ and the proof is complete. □

Lemma A.2 *For all $x \in R_L$, $\langle L - \langle L-x\rangle_{S_L}\rangle_{S_L} = x$.*

Proof: Let $x \in R_L$. We first show that $x \leq \langle L - \langle L-x\rangle_{S_L}\rangle_{S_L}$. First note that $\langle L-x\rangle_{S_L} \leq L - x$ imply that

$$x \leq L - \langle L-x\rangle_{S_L}. \tag{8}$$

Since $\langle L-\langle L-x\rangle_{S_L}\rangle_{S_L} = \max\{u \in S_L \mid u \leq L-\langle L-x\rangle_{S_L}\}$, by $x \in R_L \subseteq S_L$ and (8), we have $x \leq \langle L-\langle L-x\rangle_{S_L}\rangle_{S_L}$. Now, we show that $x \geq \langle L-\langle L-x\rangle_{S_L}\rangle_{S_L}$. By $x \in R_L$, we have $x = \langle L-v\rangle_{S_L}$ for some $v \in S_L$. Since $x = \langle L-v\rangle_{S_L} \leq L-v$, we have $v \leq L-x$. Therefore, as $v \in S_L$, we have

$$v \leq \langle L-x\rangle_{S_L}. \tag{9}$$

Then, by (9), $L-\langle L-x\rangle_{S_L} \leq L-v$ and, in consequence, $\langle L-\langle L-x\rangle_{S_L}\rangle_{S_L} \leq \langle L-v\rangle_{S_L} = x$.
□

Lemma A.3 *Let $x \in S_L$ and $x' = \lceil x \rceil_{R_L}$. Then, $\langle L-x'\rangle_{S_L} = \langle L-x\rangle_{S_L}$.*

Proof: If $x \in R_L$ then $x' = x$ and there is nothing to prove. Suppose that $x \notin R_L$. Since $x < x'$, we have that $L-x' < L-x$ and, thus, $\langle L-x'\rangle_{S_L} \leq \langle L-x\rangle_{S_L}$. Suppose, by contradiction, that $\langle L-x'\rangle_{S_L} < \langle L-x\rangle_{S_L}$. We show that there exists a raster point r, $x < r < x'$, contradicting the definition of x'. Let $t = \langle L-x\rangle_{S_L}$. Then,

$$L-x' < t \leq L-x. \tag{10}$$

By the first inequality in (10), $L-t < x'$ and, thus, $r = \langle L-t\rangle_{S_L} < x'$. By the second inequality in (10), $x \leq L-t$ and, since $x \in S_L$, $x \leq r$. However, $r \in R_L$, by definition of R_L, and $x \notin R_L$. Thus, $x < r$. Therefore, $x < r < x'$. □

Lemma A.4 *Let $x_1, x_2 \in S_L$ such that $x_1 \leq x_2$ and let $x'_1 = \lceil x_1 \rceil_{R_L}$ and $x'_2 = \lceil x_2 \rceil_{R_L}$. Then, $\langle x_2 - x_1\rangle_{S_L} \leq \langle x'_2 - x'_1\rangle_{S_L}$.*

Proof: The proof of this statement is divided in three cases.

Case 1: $x_1 \in R_L$. In this case we have $x'_1 = x_1$. Then $x_2 - x_1 \leq x'_2 - x'_1$ and $\langle x_2 - x_1\rangle_{S_L} \leq \langle x'_2 - x'_1\rangle_{S_L}$ follows trivially.

Case 2: $x_1 \notin R_L$ and $x_2 \in R_L$. Suppose, by contradiction, that $\langle x_2 - x_1\rangle_{S_L} > \langle x'_2 - x'_1\rangle_{S_L}$. So, $t = \langle x_2 - x_1\rangle_{S_L} \in S_L$ is such that $x'_2 - x'_1 = x_2 - x'_1 < t \leq x_2 - x_1$. Subtracting x_2 and multiplying by -1 all terms, we have $x'_1 > x_2 - t \geq x_1$. Since $x_1 \in S_L$, we have $x'_1 > \langle x_2 - t\rangle_{S_L} \geq x_1$. Since $x_2 \in R_L$ and $t \in S_L$, by Lemma A.1, we have that $r = \langle x_2 - t\rangle_{S_L} \in R_L$. However, $x_1 \notin R_L$. Thus, $x'_1 > r > x_1$, which contradicts the definition of x'_1.

Case 3: $x_1, x_2 \notin R_L$. Suppose, by contradiction, that $\langle x_2 - x_1\rangle_{S_L} > \langle x'_2 - x'_1\rangle_{S_L}$. Thus, there exists $t \in S_L$ such that $x'_2 - x'_1 < t \leq x_2 - x_1$. Subtracting x'_2 and multiplying by -1 all terms, we have that $x'_1 > x'_2 - t \geq (x'_2 - x_2) + x_1 > x_1$. Since $x_1 \in S_L$, $x'_1 > \langle x'_2 - t\rangle_{S_L} \geq x_1$. Since $x'_2 \in R_L$ and $t \in S_L$, by Lemma A.1, $\langle x'_2 - t\rangle_{S_L} \in R_L$. By the fact that $x_1 \notin R_L$, we have $x'_1 > \langle x'_2 - t\rangle_{S_L} > x_1$, which contradicts the definition of x'_1.
□