Generating unconstrained two-dimensional non-guillotine cutting patterns by a recursive partitioning algorithm

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

April 3, 2010

Abstract

In this study, a dynamic programming approach to deal with the unconstrained two-dimensional non-guillotine cutting problem is presented. The method extends the recently introduced recursive partitioning approach for the manufacturer's pallet loading problem. The approach involves two phases and uses bounds based on unconstrained two-staged and non-staged guillotine cutting. Since a counterexample for which the approach fails to find known optimal solutions was not found, it is conjectured that it always finds an optimal unconstrained non-guillotine cutting. The method is able to find the optimal cutting pattern of a large number of problem instances of moderate sizes known in the literature. For the instances that the required computer runtimes is excessive, the approach is combined with simple heuristics to reduce its running time. Detailed numerical experiments show the reliability of the method.

Key words: Cutting and packing, two-dimensional non-guillotine cutting pattern, dynamic programming, recursive approach, distributor's pallet loading problem.

1 Introduction

Cutting and packing problems have been widely studied in the literature in the last decades. They appear in different classes and practical applications, as discussed in several surveys [27, 26, 47, 64], annotated bibliographies [62, 28] and special issues [29, 48, 49, 16, 5, 65, 40, 55, 52] (additional references can be found in the ESICUP web site [67]). In the present paper, we study the generation of *two-dimensional non-guillotine cutting (or packing) patterns*, also referred by some authors as two-dimensional knapsack problem or two-dimensional distributor's pallet loading. This problem is classified as 2/B/O/ according to Dyckhoff's typology of cutting and packing problems [27], and as two-dimensional rectangular Single Large Object Packing Problem (SLOPP) based on Waescher et al.'s typology [64]. Besides the inherent complexity of this problem (it is NP-hard [9]), we are also motivated by its practical relevance in different industrial

*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. e-mail: {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. e-mail: morabito@ufscar.br

and logistics settings, such as in the cutting of steel and glass stock plates into required sizes, the cutting of wood sheets and textile materials to make ordered pieces, the loading of different items on the pallet surface or the loading of different pallets on the truck or container floor, the cutting of cardboards into boxes, the placing of advertisements on the pages of newspapers and magazines, the positioning of components on chips when designing integrated circuit, among others.

Given a large rectangle of length L and width W (i.e. a stock plate), and a set of rectangular pieces grouped into m different types of length l_i, width w_i and value v_i, $i = 1, \ldots, m$ (i.e. the ordered items), the problem is to find a cutting (packing) pattern which maximizes the sum of the values of the pieces cut (packed). The cutting pattern is referred as two-dimensional since it involves two relevant dimensions, the lengths and widths of the plate and pieces. A feasible two-dimensional pattern for the problem is one in which the pieces placed into the plate do not overlap each other, they have to be entirely inside the plate, and each piece must have one edge parallel to one edge of the plate (i.e., an orthogonal pattern). In this paper we assume that there are no imposed lower or upper bounds on the number of times that each type of piece can be cut from the plate; therefore, the two-dimensional pattern is called *unconstrained* (i.e., 0 and $\lfloor \frac{L}{l_i} \rfloor \lfloor \frac{W}{w_i} \rfloor$ turn out to be the obvious minimum and maximum number of pieces of type i in the pattern); otherwise, it would be called *constrained* or *doubly-constrained*.

Without loss of generality, we also assume that the cuts are infinitely thin (otherwise we consider that the saw thickness was added to L, W, l_i, w_i), the orientation of the pieces is fixed (i.e., a piece of size (l_i, w_i) is different from a piece of size (w_i, l_i) if $l_i \neq w_i$) and that L, W, l_i, w_i are positive integers. We note that if the 90^o-rotation is allowed for cutting or packing the piece type i of size (l_i, w_i), this situation can be handled by simply considering a fictitious piece type $m + i$ of size (w_i, l_i) in the list of ordered items, since the pattern is unconstrained. Depending on the values v_i, the pattern is called *unweighted*, if $v_i = \gamma l_i w_i$ for $i = 1, \ldots, m$ and $\gamma > 0$ (i.e., proportional to the area of the piece), or *weighted*, otherwise.

Moreover, we assume that the unconstrained two-dimensional cutting pattern is *non-guillotine* as it is not limited by the guillotine type cuts imposed by some cutting machines (an orthogonal *guillotine* cut on a rectangle is a cut from one edge of the rectangle to the opposite edge, parallel to the remaining edge). Some industrial cutting processes also limit the way of producing a guillotine cutting pattern. At a first stage the cuts are performed parallel to one side of the plate and then, at the next stage, orthogonal to the preceding cuts, and so on. If there is an imposed limit on the number of stages, say k, the guillotine pattern is called a *k-staged* pattern; otherwise, it is *non-staged* (note that a non-staged pattern is equivalent to define k large enough).

A large number of studies in the literature have considered staged and non-staged two-dimensional guillotine cutting problems. Much less studies have considered two-dimensional non-guillotine cutting problems (constrained and unconstrained), and only a few of them have proposed exact methods to generate non-guillotine patterns. For example, [8] and [36] presented 0–1 linear programming formulations for the problem using 0–1 decision variables for the positions of the pieces cut from the plate; they developed Lagrangean relaxations and use them as bounds in tree search procedures (subgradient optimization was used to optimize the bounds). In [63] and [20], the problem is formulated as 0–1 linear models using left, right, top and bottom decision variables for the relative positions of each pair of pieces cut from the plate (with multiple choice disjunctive constraints); they suggested solving the models by branch-and-bound algorithms exploring particular structures of these constraints. Other related 0–1 linear formulations appear in [18, 30], and a 0–1 non-linear formulation was presented in [9] (see also [59, 56]). Other exact and branch-and-bound approaches based on the so called two-level approach (the first level selects the set of pieces to be cut without taking into account the layout, the second checks if a feasible cutting layout exists for the pieces selected) are found in [19, 6, 32]. The method presented in [32] is based on a two-level tree search that combines the use of a special data structure for characterizing feasible cuttings with upper bounds.

Different heuristics based on random local search, bottom-left placement, network flow, graph search, etc., and different metaheuristics based on genetic algorithms, tabu search, simulated annealing, GRASP, etc., for generating constrained and unconstrained two-dimensional non-guillotine cuttings are found in the literature. Some examples are in [15, 25, 4, 43, 45, 57, 9, 2, 3, 61, 35, 44, 21, 17, 30, 66]. Nonlinear-programming-based method for packing rectangles within arbitrary convex regions, considering different types of positioning constraints, were presented in [12, 13, 10, 50].

Most of these approaches were developed for the constrained problem, which can be more interesting for certain practical applications with relatively low demands of the ordered items. However, part of these methods may not perform well when solving the unconstrained problem, especially those whose computational performance is highly dependent on the total number of ordered items. On the other hand, the unconstrained problem is particularly interesting for cutting stock applications with large-scale production and weakly heterogeneous items (i.e., relatively few piece types but many copies per type), in which the problem plays the role of a column generation procedure, as pointed out by several authors since the pioneer study in [34].

In the present paper we extend a *Recursive Partitioning Approach* presented in [11] for the manufacturer's pallet loading to deal with the unconstrained two-dimensional orthogonal non-guillotine cutting (unweighted and weighted, without and with piece rotation). This Recursive Partitioning Approach combines refined versions of both the *Recursive Five-block Heuristic* presented in [53, 54] and the *L-approach* for cutting rectangles from larger rectangles and L-shaped pieces presented in [46, 14]). This combined approach also uses bounds based on unconstrained two-staged and non-staged guillotine cutting patterns. Since we were unable to find a counterexample for which the approach fails, we conjecture that it always finds an optimal unconstrained non-guillotine cutting, as well as the L-approach in [46, 14, 11] for the manufacturer's pallet loading. The approach was able to find an optimal solution of a large number of problem instances of moderate sizes known in the literature. For the instances that the required computer runtimes were excessive, we combined the approach with simple heuristics to reduce its running time.

The paper is organized as follows. In Section 2 we briefly review a 0–1 linear programming formulation of the problem and some lower and upper bounds known in the literature. In Section 3 we present the two phases of the Recursive Partitioning Approach and some heuristics that reduce the time and memory requirements of the procedure to deal with large problem instances. Then in Section 4 we analyze the computational performance of the approach, with and without the heuristics, in some numerical experiments. Finally, in Section 5 we present the concluding remarks of this study.

2 Problem modeling and bounds

2.1 Problem modeling

The unconstrained two-dimensional non-guillotine cutting problem can be modeled as a 0–1 linear formulation proposed in [8]. Let (x, y) be the coordinates of the left-lower-corner of a piece placed on the plate of size (L, W) (for simplicity we assume that the left-lower-corner of the plate is $(0, 0)$). Without loss of generality, it can be shown that x and y belong, respectively, to the sets of conic combinations of l_i and w_i, $i = 1, \ldots, m$, for L and W, defined by:

$$\begin{aligned} S_L &= \{x \in \mathbb{Z}_+ \mid x = \sum_{i=1}^{m} r_i l_i,\ 0 \le x \le L,\ r_i \in \mathbb{Z}_+,\ i = 1, \ldots, m\} \text{ and} \\ S_W &= \{y \in \mathbb{Z}_+ \mid y = \sum_{i=1}^{m} s_i w_i,\ 0 \le y \le W,\ s_i \in \mathbb{Z}_+,\ i = 1, \ldots, m\}. \end{aligned}$$

Note that as we decide to place a piece of type i in a position (x, y), we cannot place another piece in a position (x', y') such that $x \le x' \le x + l_i - 1$ and $y \le y' \le y + w_i - 1$, $x, x' \in S_L$, $y, y' \in S_W$. In order to avoid piece overlapping, let $g_{ixyx'y'}$ be the mapping:

$$g_{ixyx'y'} = \begin{cases} 1, & \text{if } x \le x' \le x + l_i - 1 \text{ and } y \le y' \le y + w_i - 1, \\ 0, & \text{otherwise,} \end{cases}$$

which can be computed *a priori* for each i, (x, y) and (x', y'). By defining the decision variables a_{ixy}, $i = 1, \ldots, m$, $x \in S_L$, $y \in S_W$, as:

$$a_{ixy} = \begin{cases} 1, & \text{if a piece of type } i \text{ is placed in a position } (x, y), \\ 0, & \text{otherwise,} \end{cases}$$

the model can be formulated as the following 0–1 integer program:

$$\max \quad \sum_{i=1}^{m} \sum_{x \in S_L} \sum_{y \in S_W} v_i a_{ixy} \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{m} \sum_{x \in S_L} \sum_{y \in S_W} g_{ixyx'y'} a_{ixy} \le 1, \quad x' \in S_L,\ y' \in S_W, \tag{2}$$

$$a_{ixy} \in \{0, 1\},\ i = 1, \ldots, m,\ x \in S_L,\ y \in S_W. \tag{3}$$

Note that model (1–3) has $O(m|S_L||S_W|)$ 0–1 variables and $|S_L||S_W|$ constraints[1]. It can be shown that, without loss of generality, sets S_L and S_W can be replaced by the smaller sets R_L and R_W known as reduced raster points [60] and defined as:

$$\begin{aligned} R_L &= \{x \in \mathbb{Z}_+ \mid x = \langle L - \hat{x} \rangle_{S_L} \text{ for some } \hat{x} \in S_L\}, \\ R_W &= \{y \in \mathbb{Z}_+ \mid y = \langle W - \hat{y} \rangle_{S_W} \text{ for some } \hat{y} \in S_W\}, \end{aligned} \tag{4}$$

[1] By defining $S_L^i = \{x \in S_L | 0 \le x \le L - l_i\}$ and $S_W^i = \{y \in S_W | 0 \le y \le W - w_i\}$, it is possible to reduce the number of binary variables of the model substituting S_L and S_W by S_L^i and S_W^i, respectively, in the summations of (1) and (2) and in (3).

where

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

However, since both S_L, S_W and R_L, R_W can be large in practical cases, it may be hard to solve the model above, as illustrated in Section 4. As mentioned before, other 0–1 linear models for the non-guillotine cutting are found in the literature, but authors have pointed out that their LP-relaxations provide bounds in general far from the optimal solution values [9, 30].

2.2 Lower and upper bounds

A simple lower bound on the value of an optimal cutting can be obtained from the best homogeneous packing considering all types of pieces:

$$\max_{i \in \{1,\ldots,m\}} \left\{ v_i \left\lfloor \frac{L}{l_i} \right\rfloor \left\lfloor \frac{W}{w_i} \right\rfloor \right\}. \tag{5}$$

Similarly, a simple upper bound on the value of an optimal cutting is given by

$$\left\lfloor LW \max_{i \in F(L,W)} \left\{ \frac{v_i}{l_i w_i} \right\} \right\rfloor,$$

where $F(L,W) = \{i \in \{1,\ldots,m\} \mid l_i \leq L \text{ and } w_i \leq W\}$.

Other lower and upper bounds are found in [8, 58, 33, 9, 3, 6]. However, most of these lower and upper bounds were proposed for the constrained non-guillotine cutting, and they are less effective for the unconstrained problem. In our implementation of the algorithm described in Section 3, we have used the lower bound given by the best homogeneous packing (5) and the upper bound introduced in [33]. According to [33], the upper bound for a rectangle with length L and width W considering all types of pieces is given by

$$\min\{u_h(L,W), u_v(L,W)\}, \tag{6}$$

where

$$u_h(L,W) = \left\lfloor W \max_{t \in \mathbb{Z}_+^m} \left\{ \sum_{i \in F(L,W)} \frac{v_i}{w_i} t_i \;\middle|\; \sum_{i \in F(L,W)} l_i t_i \leq L \right\} \right\rfloor$$

and

$$u_v(L,W) = \left\lfloor L \max_{t \in \mathbb{Z}_+^m} \left\{ \sum_{i \in F(L,W)} \frac{v_i}{l_i} t_i \;\middle|\; \sum_{i \in F(L,W)} w_i t_i \leq W \right\} \right\rfloor.$$

Upper bounds for all subproblems can be obtained by solving only two unconstrained knapsack problems at the beginning of the algorithm. A dynamic programming procedure for solving these problems is presented in details in [33].

3 Description of the algorithm

The Recursive Partitioning Algorithm presented here is an extension of the algorithm described in [11] for the manufacturer's pallet loading problem. It has basically two phases: in phase 1 it applies a recursive five-block heuristic based on the procedure presented in [53] and in phase 2 it uses an L-approach based on a dynamic programming recursive formula presented in [46, 14]. Firstly, phase 1 is executed and, if a certificate of optimality is not provided by the Recursive Five-block Heuristic, then phase 2 is executed. Additionally, information obtained in phase 1 is used in phase 2 in at least two ways, according to [11]. If an optimal solution was already found for a subproblem in phase 1, it is not solved again in phase 2, improving the performance of phase 2. Moreover, having the information obtained in phase 1 at hand, phase 2 is often able to obtain better lower bounds for its subproblems than the ones provided by homogeneous cuttings (5), therefore improving the performance of phase 2. These two phases are detailed in the sequel.

3.1 Phase 1

In phase 1, the Recursive Five-block Heuristic divides a rectangle into five (or less) smaller rectangles in a way that is called *first-order non-guillotine cut* [4]. Figure 1 illustrates this kind of cut represented by a quadruple (x_1, x_2, y_1, y_2), such that $0 \leq x_1 \leq x_2 \leq L$ and $0 \leq y_1 \leq y_2 \leq W$. This cut 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 $W_5 = y_2$. Each rectangle is recursively cut unless the (sub)problem related to this rectangle has already been solved.

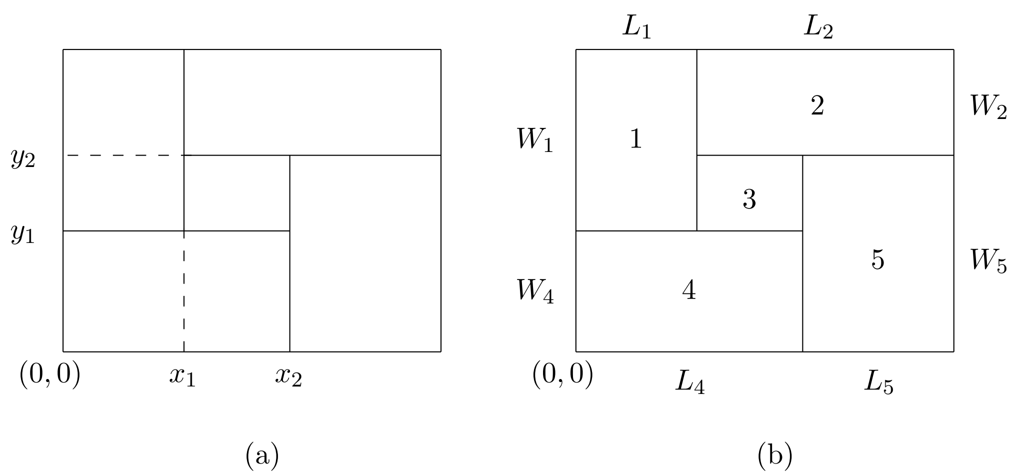

Figure 1: Representation of a first-order non-guillotine cut.

A pseudo-code of the Recursive Five-block Algorithm for the unconstrained non-guillotine cutting problem is presented in Algorithms 1[2] and 2. The method is composed by two proce-

[2] It is worth mentioning that there is a minor typographical error in line 13 of Algorithm 2 in [11] (which

dures: Five-block-Algorithm and Solve. The second one is an auxiliary subroutine used by the first. Input parameters of Five-block-Algorithm include the dimensions of the plate and the pieces, and n and N correspond to the current depth and the maximum depth limit of the search, respectively.

In Five-block-Algorithm, the lower bound starts with the value of the best homogeneous solution (5), and it is updated as better solutions are found by the algorithm. The procedure uses the upper bound (6). Both the initial lower and upper bounds are computed *a priori*. In this way, an optimal solution can be detected in the procedure by closing the gap between these bounds. Note that without loss of generality, the procedure only consider the sets of raster points as defined in Section 2. Moreover, the same symmetry constraints for both guillotine and first-order non-guillotine cuttings considered in [11] are used in the present algorithm to avoid equivalent patterns.

In the Solve procedure, we note that if the depth limit N is sufficiently large, the Five-block Algorithm is able to find the optimal first-order non-guillotine cutting pattern. Otherwise, by choosing small values of N, it is possible to control the tradeoff between the computer runtime and the quality of the solution found.

The same subproblem may appear in different levels of the tree search. In the case N is not sufficiently large, a subproblem may be solved more than once (if its solution is not proven to be optimal) in order to try to improve the previous solution found for it. In this case, if the depth n at which a subproblem is faced is greater than or equal to the depth of its stored solution, the subproblem is not solved again and the stored solution is used. Otherwise, if the current depth is smaller, there are two possibilities. If the depth limit N was used to stop the recursion when obtaining the stored subproblem solution, then it is worth trying to solve it 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.

Using the same reasoning as in [11] for the manufacturer's pallet loading, it can be shown that the worst-case time complexity of the Five-block Heuristic for the unconstrained two-dimensional non-guillotine cutting is $O(|R_L||R_W|(|R_L|^2|R_W|^2 + |S_L| + |S_W|) + L + W)$ and the memory complexity is $O(|R_L||R_W| + L + W)$.

3.2 Phase 2

Phase 2 of the Recursive Partitioning Approach applies the L-approach [46, 14, 11] which is based on the computation of a dynamic programming recursive formula [46]. This procedure divides a rectangle or an L-shaped piece into two L-shaped pieces. An L-shaped piece is determined by a quadruple (X, Y, x, y), with $X \geq x$ and $Y \geq y$, and is 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). Figure 2 depicts the nine possible divisions [11] of a rectangle or an L-shaped piece into two L-shaped pieces.

A pseudo-code of the L-Algorithm for the unconstrained non-guillotine cutting problem is presented in Algorithms 3 and 4. In subroutine Solve-L (Algorithm 4), $\mathcal{L}_i(L, k, i_k)$ and $I_k(L)$, for $i = 1, 2$ and $k = 1, \ldots, 9$, are defined as in [46, 11]. Similarly to Algorithm 1, input parameters

corresponds to Algorithm 1 here), although the computer implementation is correct – the correct constraint should be $y_1 < y_2$ and $y_2 < W$, instead of $y_1 < y_2$ and $y_1 + y_2 \leq W$.

Algorithm 1: Recursive Five-block Algorithm for the unconstrained non-guillotine cutting problem.

Input: $L, W, l_1, w_1, v_1, \ldots, l_m, w_m, v_m, n, N \in \mathbb{Z}$.
Output: Value of the best cutting pattern found.
Five-block-Algorithm$(L, W, l_1, w_1, v_1, \ldots, l_m, w_m, v_m, n, N)$

```
begin
    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 set R_L for L, (l_1, ..., l_m), and set R_W for W, (w_1, ..., w_m)
    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_2 < W do
                    if ¬(x_1 + x_2 = L and y_1 + y_2 > W) then
                        Compute (L_1, W_1), ..., (L_5, W_5)
                        P ← {(L_1, W_1), ..., (L_5, W_5)}
                        z_lb ← max{z_lb, Solve(L, W, l_1, w_1, v_1, ..., l_m, w_m, v_m, n, 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, l_1, w_1, v_1, ..., l_m, w_m, v_m, n, 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, l_1, w_1, v_1, ..., l_m, w_m, v_m, n, N, z_lb, P)}
        if z_lb = z_ub then
            return z_lb
    return z_lb
end
```

Algorithm 2: Subroutine SOLVE used by FIVE-BLOCK-ALGORITHM

SOLVE$(L, W, l_1, w_1, v_1, \dots, l_m, w_m, v_m, n, N, z_{lb}, \mathcal{P})$
begin
$\quad$ $z_{ub} \leftarrow upperBound[I_L, J_W]$
$\quad$ **for** $i \leftarrow 1$ **to** $|\mathcal{P}|$ **do**
$\quad\quad$ $z^i_{lb} \leftarrow lowerBound[I_{L_i}, J_{W_i}]$
$\quad\quad$ $z^i_{ub} \leftarrow upperBound[I_{L_i}, J_{W_i}]$
$\quad$ $S_{lb} \leftarrow \sum_{i=1}^{|\mathcal{P}|} z^i_{lb}$
$\quad$ $S_{ub} \leftarrow \sum_{i=1}^{|\mathcal{P}|} z^i_{ub}$
$\quad$ **if** $n < N$ **then**
$\quad\quad$ **if** $z_{lb} < S_{ub}$ **then**
$\quad\quad\quad$ **for** $i \leftarrow 1$ **to** $|\mathcal{P}|$ **do**
$\quad\quad\quad\quad$ **if** $depth[I_{L_i}, J_{W_i}] > n$ and $reachedLimit[I_{L_i}, J_{W_i}]$ **then**
$\quad\quad\quad\quad\quad$ $z_i \leftarrow$ FIVE-BLOCK-ALGORITHM$(L_i, W_i, l_1, w_1, v_1, \dots, l_m, w_m, v_m, n+1, N)$
$\quad\quad\quad\quad\quad$ $lowerBound[I_{L_i}, J_{W_i}] \leftarrow z_i$
$\quad\quad\quad\quad\quad$ $depth[I_{L_i}, J_{W_i}] \leftarrow n$
$\quad\quad\quad\quad\quad$ **if** $\neg reachedLimit[I_{L_i}, J_{W_i}]$ **then**
$\quad\quad\quad\quad\quad\quad$ $upperBound[I_{L_i}, J_{W_i}] \leftarrow z_i$
$\quad\quad\quad\quad$ **else**
$\quad\quad\quad\quad\quad$ $z_i \leftarrow lowerBound[I_{L_i}, J_{W_i}]$
$\quad\quad\quad\quad$ **if** $reachedLimit[I_{L_i}, J_{W_i}]$ **then**
$\quad\quad\quad\quad\quad$ $reachedLimit[I_L, J_W] \leftarrow$ true
$\quad\quad\quad\quad$ $S_{lb} \leftarrow S_{lb} + z_i - z^i_{lb}$
$\quad\quad\quad\quad$ $S_{ub} \leftarrow S_{ub} + z_i - z^i_{ub}$
$\quad\quad\quad\quad$ **if** $z_{lb} \geq S_{ub}$ **then**
$\quad\quad\quad\quad\quad$ **return** z_{lb}
$\quad\quad\quad\quad$ **if** $S_{lb} > z_{lb}$ **then**
$\quad\quad\quad\quad\quad$ $z_{lb} \leftarrow S_{lb}$
$\quad\quad\quad\quad\quad$ **if** $z_{lb} = z_{ub}$ **then**
$\quad\quad\quad\quad\quad\quad$ $reachedLimit[I_L, J_W] \leftarrow$ false
$\quad\quad\quad\quad\quad\quad$ **return** z_{lb}
$\quad$ **else**
$\quad\quad$ $reachedLimit[I_L, J_W] \leftarrow$ true
$\quad\quad$ **if** $S_{lb} > z_{lb}$ **then**
$\quad\quad\quad$ $z_{lb} \leftarrow S_{lb}$
$\quad\quad\quad$ **if** $z_{lb} = z_{ub}$ **then**
$\quad\quad\quad\quad$ $reachedLimit[I_L, J_W] \leftarrow$ false
$\quad\quad\quad\quad$ **return** z_{lb}
$\quad$ **return** z_{lb}
end

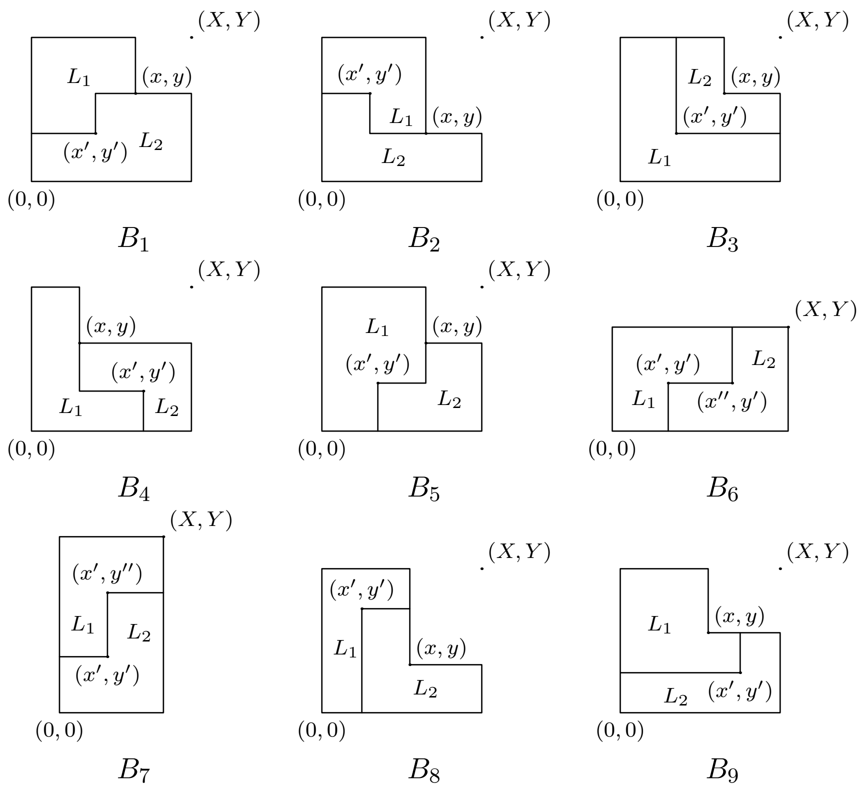

Figure 2: Subdivisions of an L-shaped piece into two L-shaped pieces.

of L-Algorithm include the dimensions of the plate and the pieces, the current depth n and the maximum depth limit N of the search. As Algorithm 1, the L-Algorithm takes into account the storage of information of the subproblems previously solved during its execution. The same data structure used in [11] to store this information is used in the implementation of the present algorithm (a four dimensional array or a combination of an array and a balanced binary search tree). See [11] for details.

Algorithm 3: L-Algorithm for the unconstrained non-guillotine cutting problem.

Input: $L, W, l_1, w_1, v_1, \ldots, l_m, w_m, v_m, n, N \in \mathbb{Z}$.
Output: Value of the best cutting pattern found.
L-Algorithm$(L, W, l_1, w_1, v_1, \ldots, l_m, w_m, v_m, n, N)$
begin
    Build set S_L for L, $(l_1, \ldots, l_m)$, and set S_W for W, $(w_1, \ldots, w_m)$
    return Solve-L$((L, W, L, W), S_L, S_W, 0, N)$
end

Algorithm 4: Subroutine Solve-L used by the L-Algorithm.

Input: An L-shaped piece L (X, Y, x, y), sets S_L and S_W, and a nonnegative integer N.
Output: Value of the best cutting pattern found.
Solve-L(L, S_L, S_W, n, N)
begin
    $I \leftarrow$ Index(X, Y, x, y)
    if $n = N$ *or* Solved(I) **then**
        return $solution[I]$
    $z_{lb} \leftarrow$ L-LowerBound(I)
    $z_{ub} \leftarrow$ L-UpperBound(I)
    Build set R_X for (X, S_L), and set R_Y for (Y, S_W)
    foreach $k \in 1, \ldots, 9$ **do**
        foreach $i_k \in I_k(X, Y, x, y)$ **do**
            $L_1 \leftarrow \mathcal{L}_1(L, k, i_k)$
            $L_2 \leftarrow \mathcal{L}_2(L, k, i_k)$
            if L-UpperBound(L_1) + L-UpperBound$(L_2) > z_{lb}$ **then**
                $z_1 \leftarrow$ Solve-L$(L_1, S_L, S_W, n+1, N)$
                $z_2 \leftarrow$ Solve-L$(L_2, S_L, S_W, n+1, N)$
                if $z_1 + z_2 > z_{lb}$ **then**
                    $z_{lb} \leftarrow z_1 + z_2$
                    if $z_{lb} = z_{ub}$ **then**
                        return z_{lb}
    return z_{lb}
end

In the L-Algorithm, the lower bound for a rectangle (X, Y) is given, initially, by the solution found in phase 1 for the subproblem (X, Y). As the algorithm proceeds, the lower bound for the subproblem (X, Y) may be improved and updated, so that the lower bound for (X, Y) in future requests is no longer the solution found in the phase 1, but the best solution found so far for the subproblem (X, Y). A lower bound for a non-degenerate L-shaped piece can be computed by dividing it into two rectangles and summing up the lower bounds of these two rectangles. The two straightforward different ways of dividing an L-shaped piece into two rectangles are considered and the lower bound for the L-shaped piece is given by the best one out of these two ones. As the algorithm proceeds, the lower bounds for L-shaped pieces are similarly updated as those for rectangles.

The upper bound for a rectangle (X, Y) is the same one used in the previous phase, that is, it is computed by (6). The upper bound for a non-degenerate L-shaped piece is given by the area ratio. Among all pieces that fit into the L-shaped piece, the one with the greatest value by area ratio is considered. The upper bound for the L-shaped piece is then given by its area multiplied by the value per unit area of the most valuable piece, that is,

$$\left\lfloor [XY - (X - x)(Y - y)] \max_{i \in G(X,Y,x,y)} \left\{ \frac{v_i}{l_i w_i} \right\} \right\rfloor,$$

where $G(X, Y, x, y) = \{i | i \in F(X, y) \text{ or } i \in F(x, Y)\}$.

Following the same steps in [11] for the manufacturer's pallet loading, it can be shown that the worst-case time complexity of the L-Algorithm for the unconstrained two-dimensional non-guillotine cutting is

$$O(|R_L|^2 |R_W|^2 (|R_L||R_W| + |S_L| + |S_W|) + L + W) \tag{7}$$

and the memory complexity is

$$O(|R_L|^2 |R_W|^2 + L + W). \tag{8}$$

It should be mentioned that the worst-case time and memory complexities of the Recursive Partitioning Approach are given by (7) and (8), respectively.

3.3 Improving the lower bound and problem preprocessing

In order to improve the computational performance of the Recursive Partitioning Approach, we start computing better lower bounds (feasible solutions) than the ones provided by the homogeneous solutions (5). The idea is to generate cutting patterns simpler than generic unconstrained non-guillotine cutting patterns, however, more elaborated than homogeneous patterns.

In this way, before performing phase 1 of Section 3.1, the approach computes optimal unconstrained two-staged guillotine cutting patterns for the plate (L, W) and for a number of subrectangles of (L, W). This computation uses the exact procedure in [34], based on a solution of $O(m)$ one-dimensional knapsack problems. Then, the approach computes optimal non-staged guillotine cutting patterns for the plate (L, W) and a number of subrectangles of (L, W), using the value of the two-staged solutions as initial lower bounds. This computation employs the Recursive Five-block Algorithm specialized for the guillotine case (i.e., considering only vertical and horizontal guillotine cuts). These optimal guillotine solutions are used to reduce the tree search of the Recursive Five-block Heuristic in phase 1. Finally, the optimal first-order non-guillotine solutions found in phase 1 for the plate (L, W) and a number of subrectangles of

(L, W) are used to reduce the dynamic programming recursions of the L-approach in phase 2 (Section 3.2). We note that each phase of the approach is executed only if the solution given by the previous phase cannot be proven to be optimal. Solutions found for the subproblems in a given phase are used as lower bounds for the same subproblems in the next phase. In this way, solutions found in higher phases are at least as good as the ones found in earlier phases.

Therefore, besides providing as by-products (i) the optimal unconstrained two-staged guillotine pattern, (ii) the optimal unconstrained non-staged guillotine pattern and (iii) the optimal unconstrained first-order non-guillotine pattern, we conjecture that the Recursive Partitioning Algorithm also provides the optimal unconstrained (general) non-guillotine pattern.

A final remark in this section is that as a preprocessing of the input of the Recursive Partitioning Algorithm, our implementation of the algorithm may discard pieces that are not required to be in an optimal solution. Note that any piece type i that satisfies $l_i \geq l_j$, $w_i \geq w_j$, and $v_i \leq v_j$, for some piece type $j \neq i$, does not need to be considered, since j is at least as valuable as i and it is not larger than i. If this is the case, we say that piece type j dominates piece type i. It is worth mentioning that this procedure may be useful for weighted instances only. This process could be generalized to the case where not only one, but a combination of pieces dominates another piece. That is, when there is a subset of pieces P and a piece $i \notin P$, such that we can pack all pieces that belong to P (maybe using more than one of the same type) in rectangle i, and the packing has value greater than or equal to v_i. For unweighted instances, if a piece i is dominated by pieces in set P, then there is a packing of the pieces in P in the rectangle i so that rectangle i is completely covered (i.e, a zero waste packing). Hence, all integer conic combinations produced by piece i are also produced by the ones in the set P and, therefore, the exclusion of piece i does not contribute to a reduction on the number of patterns that the algorithm must inspect. As the generalized process described above is a computational expensive task, in the present work we implemented a preprocessing procedure that considers the elimination of a piece dominated by another piece.

3.4 Heuristics for large problems

The generation of all patterns by the Recursive Partitioning Approach may be prohibitive for large instances. Moreover, the amount of memory required by these algorithms may not be available. For this reason, we propose heuristics that reduce both the time and memory requirements of the algorithms. These procedures, however, may lead to a loss of quality of the solution found. Since the time and memory complexities of generating all possible cuttings highly depends on the sizes of the integer conic combinations and raster points sets (as pointed out in (7) and (8), respectively), we can significantly reduce time and memory requirements in two ways: (i) by limiting the search depth of the recursions (i.e., setting parameter $N \neq \infty$ in Algorithms 1–4); and (ii) by replacing the integer conic combinations and raster points sets by smaller sets.

In order to reduce the sets of conic combinations S_L and S_W, it was suggested in [8] a simple iterative procedure that starts eliminating the piece with the smallest length (width) until the desired sizes of the sets are achieved. In this work we decided by an alternative procedure that replaces the sets S_L and S_W by the shrunk sets $\tilde{S}_L$ and $\tilde{S}_W$, so that the product $|\tilde{S}_L||\tilde{S}_W|$ is limited by a given threshold parameter M. Moreover, to substitute the raster points sets, the shrunk sets $\tilde{R}_L$ and $\tilde{R}_W$ are obtained by using definition (4) with $\tilde{S}_L$ and $\tilde{S}_W$ instead of S_L and S_W.

Algorithms 5–7 describe the implemented reduction strategy. Basically, the strategy consists in selecting equally distributed elements within ordered versions of sets S_L and S_W. Algorithm 7 describes the main routine named ReduceSets that uses Algorithms 5 and 6 as subroutines. Specifically, the heuristic version of the Recursive Partitioning Algorithm makes use of routine ReduceSets in line 7 of Algorithm 1 to compute $\tilde{R}_L$ and $\tilde{R}_W$ based on $\tilde{S}_L$ and $\tilde{S}_W$. Similarly, Algorithm 3 uses routine ReduceSets in line 2 to build sets $\tilde{S}_L$ and $\tilde{S}_W$ and call subroutine Solve-L (Algorithm 4) in line 3 with them, instead of S_L and S_W.

Algorithm 5: Select n numbers between s and e.

Input: The number n of elements to be selected, the first element s to be considered and the last element e to be considered. We suppose that $s \leq e$ and $n \leq e - s + 1$.
Output: A subset of $\{s, \dots, e\}$ with n elements.
Select(n, s, e)
begin
  $I \leftarrow \emptyset$
  if $n > 0$ **then**
    $I \leftarrow \lfloor \frac{s+e}{2} \rfloor$
    if $n > 1$ **then**
      $I \leftarrow I \cup \text{Select}(\lfloor \frac{n-1}{2} \rfloor, s, \lfloor \frac{s+e}{2} \rfloor - 1)$
      $I \leftarrow I \cup \text{Select}(\lceil \frac{n-1}{2} \rceil, \lfloor \frac{s+e}{2} \rfloor + 1, e)$
  return I
end

Algorithm 6: Select n elements of S equally distributed. We suppose that $n \leq |S|$. S_i denotes the i-th smallest element of S.

Input: S and the number n of elements in the reduced set.
Output: A subset of S with size n.
ReduceSet(S, n)
begin
  $\tilde{S} \leftarrow \emptyset$
  $I \leftarrow \text{Select}(n, 1, |S|)$
  foreach $i \in I$ **do**
    $\tilde{S} \leftarrow \tilde{S} \cup \{S_i\}$
  return $\tilde{S}$
end

4 Numerical experiments

We implemented the Recursive Partitioning Approach and its heuristic counterpart for the unconstrained two-dimensional non-guillotine cutting problem as described in Algorithms 1–7. The algorithms were coded in C/C++ language. The experiments were performed in a 2.4GHz Intel Core2 Quad Q6600 with 8.0GB of RAM memory and Linux Operating System. Compiler option -O3 has been adopted. The computer implementation of the algorithms as well as the

Algorithm 7: Given sets S and T, and a positive integer n, it creates sets $\tilde{S} \subseteq S$ and $\tilde{T} \subseteq T$ such that $|\tilde{S}||\tilde{T}| \approx M$ if $M < |S||T|$. Otherwise, it returns S and T.

Input: Sets S and T and an integer M.
Output: Sets $\tilde{S} \subseteq S$ and $\tilde{T} \subseteq T$ such that $|\tilde{S}||\tilde{T}| \approx M$.
ReduceSets(S, T, M)
begin
  $p \leftarrow |S||T|$
  if $p \leq M$ **then**
    return S and T
  $s \leftarrow \left\lceil \sqrt{\frac{M}{p}} |S| \right\rceil$
  $t \leftarrow \left\lceil \sqrt{\frac{M}{p}} |T| \right\rceil$
  $\tilde{S} \leftarrow$ ReduceSet(S, s)
  $\tilde{T} \leftarrow$ ReduceSet(T, t)
  $\tilde{S} \leftarrow \tilde{S} \cup \{S_1, S_{|S|}\}$
  $\tilde{T} \leftarrow \tilde{T} \cup \{T_1, T_{|T|}\}$
  return $\tilde{S}$ and $\tilde{T}$
end

data sets used in our experiments and the solutions found are publicly available for benchmarking purposes at [68].

In the numerical experiments, we considered 95 problem instances found in the literature: H, HZ1 and HZ2 from [37], [41] and [42], respectively; GCUT1–GCUT13 and GCUT14–GCUT17 from [7] and [22], respectively; M1–M5 and MW1–MW4 from [51] and [39], respectively; U1–U3 and W1–W3 from [38]; U4, W4, LU1–LU4, LW1–LW4 from [39]; UU1–UU11 and UW1–UW11 from [31]; APT10–APT29 from [1]; and B1–B7 from [23]. Table 1 presents some characteristics of these instances. In the table, m_2 is the original number of piece types while m_1 is the reduced number of piece types obtained by applying the preprocessing procedure described in Subsection 3.3, L and W are the length and width of the stock plate, and $|S_L|$, $|S_W|$, $|R_L|$ and $|R_W|$ are the number of elements of the sets of conic combinations and raster points, respectively, after applying the preprocessing procedure for the elimination of dominated pieces described in Subsection 3.3. It should be noted in Table 1 that the number of piece types of the instances varies from 5 to 200 and the number of conic combinations (or raster points) from a few tens to thousands. Therefore, the number of variables and constraints of model (1–3) for these instances varies from a few hundreds to billions.

We arbitrarily divided the experiments with these instances into two parts: the first consists of experiments with instances of moderate size, defined as $\max\{|S_L|, |S_W|\} < 1{,}000$, and the second part contains experiments with the remaining large-sized instances.

4.1 Instances of moderate size

In this set of experiments, we imposed no limit to the search depth, i.e., we set $N = \infty$. Table 2 presents the values of the unconstrained two-stage, guillotine, first-order non-guillotine (Five-block Algorithm) and (superior-order) non-guillotine (L-algorithm) cutting patterns obtained by the Recursive Partitioning Approach for the instances of moderate size. The two-

Instance	m		L	W	$\lvert S_L\rvert$	$\lvert S_W\rvert$	$\lvert R_L\rvert$	$\lvert R_W\rvert$	$\lvert S_L\rvert\lvert S_W\rvert$
	m_1	m_2							
H	5	5	127	98	35	61	25	30	2135
HZ1	6	6	78	67	70	44	61	23	3080
HZ2	5	5	99	80	25	56	17	31	1400
GCUT1	10	10	250	250	68	20	13	5	1360
GCUT2	20	20	250	250	95	112	17	24	10640
GCUT3	30	30	250	250	143	107	44	26	15301
GCUT4	50	50	250	250	146	146	45	50	21316
GCUT5	10	10	500	500	40	76	10	13	3040
GCUT6	20	20	500	500	96	120	12	18	11520
GCUT7	30	30	500	500	179	126	23	19	22554
GCUT8	50	50	500	500	225	262	44	59	58950
GCUT9	10	10	1000	1000	92	42	15	7	3864
GCUT10	20	20	1000	1000	89	155	14	20	13795
GCUT11	30	30	1000	1000	238	326	20	38	77588
GCUT12	50	50	1000	1000	398	363	49	42	144474
GCUT13	32	32	3000	3000	1821	2425	647	1849	4415925
GCUT14	42	42	3500	3500	2681	2976	1861	2451	7978656
GCUT15	52	52	3500	3500	2690	3048	1879	2595	8199120
GCUT16	62	62	3500	3500	2824	3058	2147	2615	8635792
GCUT17	82	82	3500	3500	2929	3068	2357	2635	8986172
M1	10	10	100	156	48	74	23	17	3552
M2	10	10	253	294	154	87	63	17	13398
M3	10	10	318	473	72	156	13	32	11232
M4	10	10	501	556	78	116	15	15	9048
M5	10	10	750	806	124	147	23	16	18228
MW1	10	10	100	156	48	74	23	17	3552
MW2	9	10	253	294	145	87	53	17	12615
MW3	8	10	318	473	60	130	13	28	7800
MW4	9	10	501	556	106	112	22	15	11872
MW5	8	10	750	806	99	100	22	12	9900
U1	10	10	4500	5000	2910	1984	1327	253	5773440
U2	10	10	5050	4070	2225	373	351	38	829925
U3	20	20	7350	6579	3948	3711	832	979	14651028
U4	40	40	7350	6579	6151	4898	4951	3216	30127598
W1	6	20	5000	5000	1288	1228	190	179	1581664
W2	9	20	3427	2769	407	382	60	58	155474
W3	7	40	7500	7381	1029	1477	129	180	1519833
W4	12	80	7500	7381	5412	5494	3323	3606	29733528
UU1	25	25	500	500	171	205	29	39	35055
UU2	30	30	750	800	334	322	61	38	107548
UU3	25	25	1100	1000	389	317	44	37	123313
UU4	38	38	1000	1200	444	570	65	83	253080
UU5	50	50	1450	1300	694	643	111	116	446242
UU6	38	38	2050	1457	649	495	59	50	321255
UU7	50	50	1465	2024	743	944	144	139	701392
UU8	55	55	2000	2000	914	830	114	95	758620
UU9	60	60	2500	2460	1047	1044	120	127	1093068
UU10	55	55	3500	3450	1304	1542	110	177	2010768
UU11	25	25	3500	3765	3014	2461	2527	1156	7417454

Table 1: Characteristics of the instances.

Instance	m		L	W	$\|S_L\|$	$\|S_W\|$	$\|R_L\|$	$\|R_W\|$	$\|S_L\|\|S_W\|$
	m_1	m_2							
UW1	7	25	500	500	72	70	10	18	5040
UW2	8	35	560	750	172	80	31	16	13760
UW3	9	35	700	650	96	98	18	21	9408
UW4	10	45	1245	1015	193	251	23	33	48443
UW5	6	35	1100	1450	76	68	19	9	5168
UW6	15	47	1750	1542	310	379	37	38	117490
UW7	11	50	2250	1875	185	354	27	33	65490
UW8	13	55	2645	2763	450	540	49	39	243000
UW9	14	45	3000	3250	357	384	38	39	137088
UW10	17	60	3500	3650	690	797	61	53	549930
UW11	8	25	555	632	259	327	64	104	84693
LU1	97	100	20789	23681	10075	22602	9753	21522	227715150
LU2	98	100	25587	34563	24478	33421	23368	32278	818079238
LU3	149	150	37587	27563	37022	13684	36456	13584	506609048
LU4	200	200	45237	35983	22546	17894	22471	17794	403438124
LW1	56	100	20789	23681	10057	11516	9717	11189	115816412
LW2	48	100	25587	34563	12447	33403	12098	32242	415767141
LW3	83	150	37587	27563	18627	13677	18458	13570	254761479
LW4	112	200	45237	35983	22546	17883	22471	17772	403190118
ATP10	51	51	2097	1713	1760	1478	1422	1242	2601280
ATP11	58	58	2600	1612	2156	1435	1711	1257	3093860
ATP12	35	35	2662	1941	1966	1420	1269	898	2791720
ATP13	54	54	1674	2090	1422	1802	1169	1513	2562444
ATP14	42	42	2090	2138	1739	1681	1387	1223	2923259
ATP15	49	49	2222	2726	1741	2264	1259	1801	3941624
ATP16	53	53	2899	2614	2292	2104	1684	1593	4822368
ATP17	59	59	2313	1962	1980	1643	1646	1323	3253140
ATP18	31	31	2775	2105	2163	1524	1550	942	3296412
ATP19	35	35	2284	2994	1729	2325	1173	1655	4019925
ATP20	34	45	2840	2858	2284	2260	1727	1661	5161840
ATP21	34	48	2866	1784	2066	1417	1265	1049	2927522
ATP22	37	52	2711	2110	2168	1708	1624	1305	3702944
ATP23	35	53	1856	2636	1401	2048	945	1459	2869248
ATP24	33	44	2070	2729	1632	2308	1193	1886	3766656
ATP25	24	36	2885	1715	2246	1307	1606	898	2935522
ATP26	34	48	2359	1656	1842	1340	1324	1023	2468280
ATP27	33	48	1793	1875	1429	1397	1064	918	1996313
ATP28	36	54	2020	2796	1604	2383	1187	1969	3822332
ATP29	41	59	1839	2829	1505	2261	1170	1692	3402805
B1	30	30	4000	2000	3452	1343	2903	685	4636036
B2	30	30	4000	2000	3325	1429	2649	857	4751425
B3	30	30	4000	2000	3370	1411	2739	821	4755070
B4	30	30	4000	2000	3388	1426	2775	851	4831288
B5	30	30	4000	2000	3324	1393	2647	785	4630332
B6	30	30	4000	2000	3299	1441	2597	881	4753859
B7	180	180	4000	2000	3654	1664	3307	1327	6080256

Table 1: Characteristics of the instances (cont.).

Instance	Two-stage cuts		Guillotine cuts		Five-block Algorithm		L-Algorithm		Total
	Solution	Time (sec)	Solution	Time (sec)	Solution	Time (sec)	Solution	Time (sec)	time
H	12132	0.62	**12348**	0.00	–	–	–	–	1.67
HZ1	5226	0.67	–	–	–	–	–	–	1.30
HZ2	8046	0.57	**8226**	0.00	**8443**	0.01	8443	2.18	4.23
GCUT1	56460	0.57	56460	0.00	**58480**	0.00	58480	0.72	2.84
GCUT2	60076	0.69	**60536**	0.00	**61146**	0.00	61146	0.96	3.03
GCUT3	60133	0.58	**61036**	0.00	**61275**	0.02	61275	4.21	6.41
GCUT4	61698	0.70	61698	0.00	**61918**	0.06	61918	20.22	22.35
GCUT5	246000	0.54	246000	0.00	246000	0.00	246000	0.67	2.74
GCUT6	238998	0.71	238998	0.00	**243598**	0.00	243598	0.74	2.82
GCUT7	242567	0.56	242567	0.00	**244306**	0.00	244306	0.92	2.98
GCUT8	245758	0.73	**246633**	0.00	**247815**	0.10	247815	32.57	34.82
GCUT9	971100	0.54	971100	0.00	971100	0.00	971100	0.68	2.79
GCUT10	982025	0.71	982025	0.00	982025	0.00	982025	0.76	2.83
GCUT11	980096	0.56	980096	0.16	980096	0.00	980096	1.92	4.05
GCUT12	978776	0.53	**979986**	0.16	979986	0.06	979986	16.95	19.06
M1	15024	0.55	15024	0.00	**15054**	0.00	**15073**	1.13	3.20
M2	72172	0.66	**73176**	0.00	**73255**	0.04	73255	7.90	10.02
M3	141810	0.57	**142817**	0.00	**147386**	0.00	147386	1.10	3.16
M4	265768	0.47	265768	0.00	**266233**	0.00	266233	0.76	2.80
M5	577882	0.71	577882	0.00	**579883**	0.00	579883	1.08	3.14
MW1	3882	0.50	3882	0.00	3882	0.00	3882	1.22	3.23
MW2	24950	0.67	24950	0.00	24950	0.01	24950	6.17	8.24
MW3	37068	0.49	37068	0.00	37068	0.00	37068	1.01	3.04
MW4	59364	0.50	**59576**	0.00	59576	0.00	59576	1.07	3.05
MW5	189924	0.54	189924	0.00	189924	0.00	189924	0.94	2.98
W2	34520	0.70	**35159**	0.00	**35822**	0.96	35822	229.76	232.81
UU1	240346	0.58	**242919**	0.00	**245205**	0.02	245205	5.48	7.62
UU2	595288	0.70	595288	0.00	595288	0.08	595288	34.24	36.39
UU3	1065051	0.50	**1072764**	0.16	**1088154**	0.04	1088154	16.51	18.61
UU4	1177371	0.52	**1179050**	0.17	**1191071**	0.82	1191071	330.16	333.02
UU5	1868985	0.49	**1868999**	0.18	**1870038**	4.52	1870038	3057.23	3063.82
UU6	2950760	0.52	2950760	0.16	2950760	0.16	2950760	61.50	63.80
UU7	2925362	0.54	**2930654**	0.22	**2943852**	10.76	2943852	7620.64	7633.53
UU8	3959352	0.55	3959352	0.19	**3969784**	3.30	**3970877**	1822.15	1827.64
UW1	6036	0.58	6036	0.00	6036	0.00	6036	0.72	2.83
UW2	8468	0.66	8468	0.00	**8720**	0.00	8720	1.45	3.48
UW3	5888	0.50	**6302**	0.00	**6652**	0.00	6652	1.20	3.18
UW4	7748	0.52	**8326**	0.16	8326	0.01	8326	3.24	5.31
UW5	7780	0.64	7780	0.00	7780	0.00	7780	0.74	2.76
UW6	6548	0.67	**6615**	0.00	**6803**	0.03	6803	14.68	16.77
UW7	10464	0.65	10464	0.00	10464	0.01	10464	4.42	6.47
UW8	7692	0.66	7692	0.00	7692	0.05	7692	26.79	28.92
UW9	7038	0.72	7038	0.00	**7128**	0.02	7128	13.87	16.02
UW10	7461	0.67	**7507**	0.00	7507	0.12	7507	99.79	101.96
UW11	15747	0.70	15747	0.01	**16400**	1.56	16400	1015.00	1018.74

Table 2: Results obtained by the Recursive Partitioning Algorithm for instances of moderate size.

stage and guillotine solutions are computed before phase 1 to obtain non-trivial lower bounds as described in Section 3.3. The Five-block Algorithm solution and the L-Algorithm solution correspond to phases 1 and 2 of the Recursive Partitioning Approach, respectively. The symbol "–" in the table means that an optimality certificate was found by closing the gap between the upper and the lower bound of the problem and, in consequence, the execution of the method was terminated. Columns with times represent the CPU time of each phase of the method. The time in the last column is the sum of the times of each phase plus the time used to read and preprocess the input data, compute lower and upper bounds for each possible subrectangle, draw and save the solution.

In the table, the numbers in bold indicate that the corresponding phase of the method improved the solution given by the previous phase. For example, note that for instances M1 and UU8, the optimal guillotine pattern value coincides with the optimal two-stage pattern value, while the optimal five-block pattern is better and the L-Algorithm pattern is even better. Figures 3 and 4 depict the corresponding optimal cutting patterns for instances M1 and UU8,

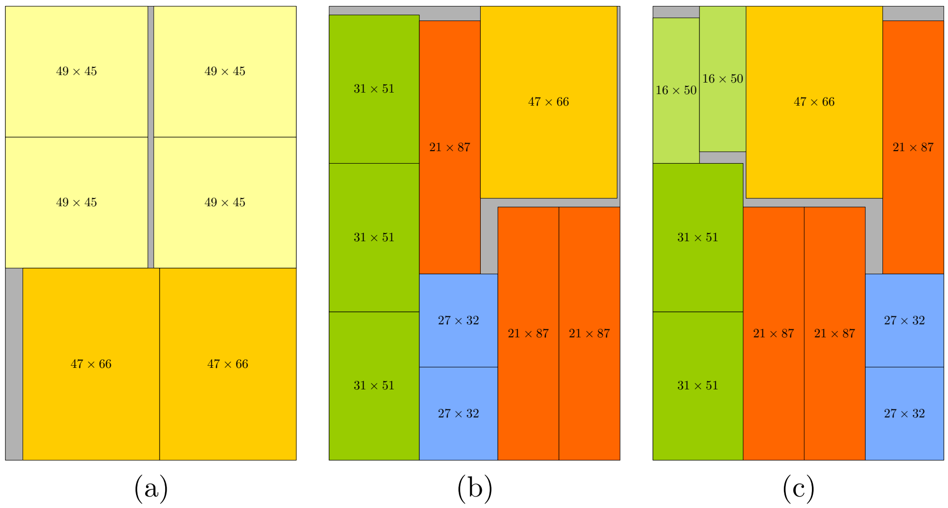

Figure 3: Graphical representations of solutions found by the Recursive Partitioning Algorithm for instance M1. (a) Optimal guillotine pattern with value 15,024. (b) First-order guillotine pattern with value 15,054. (c) L-Algorithm pattern with value 15,073.

respectively.

As expected, the optimal unconstrained two-stage and guillotine pattern values coincide with the ones reported in the literature [37, 7, 51, 41, 42, 38, 31, 39, 1, 24, 22]. Moreover, the unconstrained non-guillotine solutions found by the Five-block Algorithm and the L-Algorithm improve the guillotine ones in many instances. To the best of our knowledge, there are no studies in the literature reporting unconstrained non-guillotine solutions for these instances. An interesting result in Table 2 is that most of the best solutions of the Recursive Partitioning Approach were found by the Five-block Algorithm, requiring affordable runtimes (only a few seconds).

We conjecture that all unconstrained non-guillotine cutting patterns obtained by the Recursive Partitioning Approach for these moderate-sized instances are optimal. For instances H and HZ1, the Recursive Partitioning Approach gives an optimality certificate. For instances HZ2, GCUT1–GCUT3, GCUT5–GCUT7, GCUT9, GCUT10, M1–M5 and MW1–MW5, an optimality certificate was obtained solving the LP-relaxation of model (1–3) using the modelling language GAMS 19.8 with the solver CPLEX 7. For all the remaining moderate-sized instances, we were unable to prove the optimality because the CPLEX execution was aborted with a computer memory overflow error.

To empirically reinforce our conjecture about the exactness of the Recursive Partitioning Approach, we also performed additional experiments with other 22 problem instances of the literature [43, 45], for which the optimal solutions are known by construction. These unweigthed instances were generated in such a way that there exists at least one optimal cutting pattern with zero waste, that is, the value of this cutting pattern coincides with the area of the stock plate. For all instances C1P1–C4P3 in [43] and LCT1–LCT10 in [45], the Recursive Partitioning Approach was able to find an optimal solution. In all cases the optimal pattern is two-stage

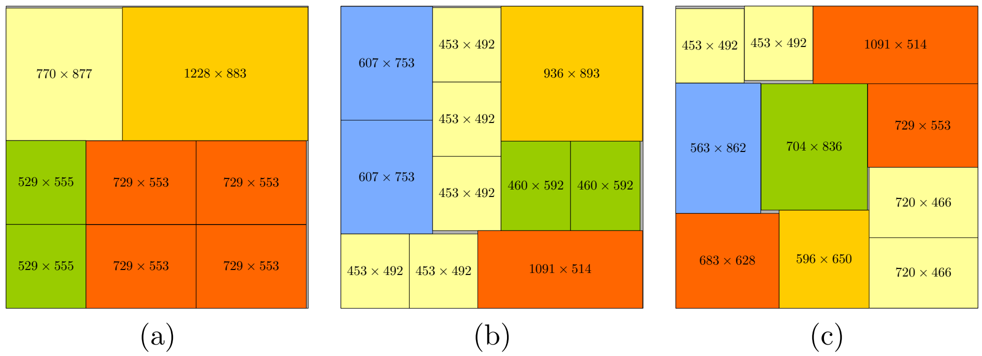

Figure 4: Graphical representations of solutions found by the Recursive Partitioning Algorithm for instance UU8. (a) Optimal guillotine pattern with value 3,959,352. (b) First-order guillotine pattern with value 3,969,784. (c) L-Algorithm pattern with value 3,970,877.

guillotine, with the exception of instance LCT3 for which the optimal pattern was obtained by the Five-block Algorithm.

4.2 Large instances

In a first set of experiments with the large instances, we applied an heuristic version of the Recursive Partitioning Approach, without imposing limits to the search depth (i.e., setting $N = \infty$). Sets S_L and S_W were replaced in the guillotine, the five-block and the L-Algorithm phases by sets $\tilde{S}_L$ and $\tilde{S}_W$ generated by Algorithm 7 with the product threshold parameter $M = 40,000,000$, $M = 150,000$ and $M = 10,000$, respectively. Therefore, the best guillotine, five-block and non-guillotine solutions obtained by these algorithms may not be optimal.

Table 3 presents the values of the unconstrained two-stage, guillotine, first-order non-guillotine (Five-block Algorithm) and superior-order non-guillotine (L-algorithm) cutting patterns obtained by the heuristic version of the Recursive Partitioning Approach for the large-sized instances. We first note that the two-staged phase of this heuristic version is identical to the one of the exact version of the method, i.e., all two-staged solutions are optimal. Moreover, looking at the quantity $|S_L| \times |S_W|$ of each instance on Table 1, and considering that guillotine cuts of instances with $|S_L| \times |S_W| \leq 40,000,000$ are being solved to optimality, we know that, with the exception of instances LU1–LU4 and LW1–LW4, the guillotine cuts of all the large-sized instances are optimal too. Among the optimal guillotine cuts generated by our method, we were able to verify that, for instances GCUT13–GCUT17, U1–U3, W1, W3, UU9–UU11, and APT10–APT29, solutions found by our method are in agreement with previously published solutions [38, 31, 39, 1, 23, 24, 22]. For the remaining instances there are no optimal guillotine solutions reported in the literature. Even without this double checking, guillotine solutions for instances U4, W4 and B1–B7 are optimal, but whether or not guillotine solutions for instances LU1–LU4 and LW1–LW4 are optimal remains to be verified. A remark for instances U2 and U3 is in order. Instances U2 and U3 were introduced in [38]. [24] reports their optimal guillotine pattern values as being 20,232,224 and 48,142,840, respectively, mentioning that these optimal values can be obtained by running the algorithm for staged patterns presented in [23] (setting the maximum allowed number of stages to infinity). These values are slightly larger than the ones reported in Table 3, that should be optimal. We implemented and run the method introduced in [23], and the same values reported in Table 3 were found, which suggests that the optimal values for instances U2 and U3 reported in [24] may be incorrect. To reinforce this idea, we note that, as far as we know, there is no publication introducing a method (exact or heuristic) that finds better quality solutions than the ones being reported here, while some publications report equal or lower quality solutions [38, 1, 24].

Regarding Table 3, it is worth remarking that instances LU1–LU4, LW1–LW4 are huge instances and involve tens of thousands of conic combinations. Note that the guillotine solution found for instance B7 is proven to be optimal. Note also that most of the best solutions were obtained by applying only guillotine cuts, but the demanded runtimes are generally much higher than the ones in the moderate-sized instances of Table 2. Exceptions are instances GCUT13, W3, W4, UU9 and LU4 for which none of the possible guillotine partitions (neither horizontal nor vertical) provides a lower bound greater than the value of the optimal two-stage pattern. Other exceptions are instances APT21 and APT22 for which the Five-block Algorithm was able to improve the lower bound given by the guillotine partitions. On the other hand, note that the optimal two-stage solutions in the table are reasonably good, if compared to the guillotine solutions, and they are obtained in just a couple of seconds.

In a second set of experiments, we applied another heuristic version of the Recursive Partitioning Approach, using the complete conic combinations sets in all the phases of the method, fixing the search depth limit parameter $N = \infty$ in the guillotine phase and $N = 1$ in the five-block phase, and skipping the L-Algorithm phase of the method. Table 4 presents the values of the unconstrained two-stage, guillotine and first-order non-guillotine (Five-block Algorithm) cuts obtained by this heuristic version of the Recursive Partitioning Approach for some of the large-sized instances. The unconstrained two-stage and guillotine pattern values coincide with the ones of Table 3. Note, however, that the Five-block Algorithm was able to find improved non-guillotine solutions for most of these examples (if compared to Table 3), at the expenses of much higher running times. In particular, since the upper bound on the optimal value for instance GCUT14–GCUT17 provided by the algorithm is 12,250,000, we note that the five-block solution found for instance GCUT17 is proven to be optimal. Besides that, note that, for instance APT22, the heuristic version of the Five-block Algorithm with $N = 1$ and using the complete set of conic combinations found a better solution than the previous heuristic version with $N = \infty$ and a shrunk set of conic combinations (see Table 3).

In addition to the experiments described above, we arbitrarily decided to solve the two large instances UU9 and UU10 with the non-heuristic version of the Recursive Partitioning Approach. While for instance UU9 the best solution obtained corresponds to a two-stage pattern with value 6,100,692, for instance UU10, each phase of the method improved the solution found by the previous phase. Figure 5 illustrates the corresponding cutting patterns obtained by each phase for instance UU10. The total runtimes for solving instances UU9 and UU10 were 4,321.59 and 8,908.24 seconds, respectively.

Instance	Two-stage cuts		Guillotine cuts		Five-block Algorithm		L-Algorithm		Total time
	Solution	Time (sec)	Solution	Time (sec)	Solution	Time (sec)	Solution	Time (sec)	
GCUT13	8997780	0.70	8997780	21.24	8997780	10.40	8997780	0.72	36.72
GCUT14	12236280	0.68	**12245410**	206.46	12245410	37.47	12245410	46.84	298.22
GCUT15	12239904	0.58	**12246032**	225.11	12246032	29.00	12246032	876.49	1139.46
GCUT16	12243100	0.57	**12248836**	326.06	12248836	72.24	12248836	1492.92	1901.73
GCUT17	12242998	0.73	**12248892**	430.19	12248892	28.10	12248892	1763.37	2235.80
U1	22167051	0.55	**22370130**	6.27	22370130	3.90	22370130	1.83	15.16
U2	20031845	0.55	**20232223**	0.23	20232223	0.14	20232223	0.91	3.42
U3	47912942	0.84	**48142836**	14.24	48142836	0.89	48142836	1.06	23.75
U4	48227224	0.76	**48304289**	2169.51	48304289	289.91	48304289	435.73	2917.68
W1	161424	0.65	**162867**	0.16	162867	0.24	162867	0.88	3.53
W3	234108	0.70	234108	0.03	234108	0.17	234108	0.82	3.31
W4	377910	0.73	377910	27.14	377910	65.67	377910	158.99	261.15
UU9	6100692	0.62	6100692	0.19	6100692	0.03	6100692	0.88	3.97
UU10	11929561	0.69	**11955852**	0.08	11955852	0.04	11955852	0.70	4.37
UU11	13066737	0.71	**13157811**	217.67	13157811	988.84	13157811	244.20	1455.46
LU1	492259840	1.52	**492278922**	6792.50	492278922	154.47	492278922	1.14	7001.03
LU2	884318400	1.54	**884341464**	9620.75	884341464	88.56	884341464	0.89	9763.91
LU3	1035962256	2.27	**1035971468**	15245.45	1035971468	116.47	1035971468	0.92	15444.27
LU4	1627681752	2.35	1627681752	0.10	1627681752	154.87	1627681752	0.90	261.50
LW1	170590913	0.94	**171245481**	10313.17	171245481	145.54	171245481	15008.65	25499.52
LW2	324615667	0.99	**325011119**	9446.71	325011119	19766.82	325011119	20104.52	49346.43
LW3	430291142	1.15	**430706636**	16192.94	430706636	94.64	430706636	19676.79	36013.23
LW4	565839770	1.49	**566102201**	11459.56	566102201	93.23	566102201	22218.03	33833.78
APT10	3585612	0.56	**3589703**	56.52	3589703	190.66	3589703	1727.17	1978.77
APT11	4175414	0.54	**4188915**	81.59	4188915	50.09	4188915	2748.75	2885.51
APT12	5148302	0.54	**5156065**	21.16	5156065	270.93	5156065	174.00	469.79
APT13	3493072	0.73	**3498302**	56.41	3498302	60.40	3498302	0.87	122.27
APT14	4458052	0.55	**4463550**	51.57	4463550	89.45	4463550	1082.57	1227.69
APT15	6028426	0.72	**6047188**	70.68	6047188	81.15	6047188	669.11	826.36
APT16	7533987	0.70	**7566719**	98.33	7566719	62.66	7566719	893.13	1060.57
APT17	4529371	0.56	**4535302**	88.67	4535302	70.07	4535302	1540.74	1704.73
APT18	5807988	0.66	**5825956**	39.50	5825956	129.76	5825956	215.25	388.37
APT19	6811338	0.51	**6826674**	48.88	6826674	28.93	6826674	337.51	419.70
APT20	5490828	0.54	**5545818**	112.36	5545818	8114.96	5545818	1039.89	9272.29
APT21	3479634	0.52	**3484406**	25.14	**3491545**	247.33	3491545	358.85	635.07
APT22	4111542	0.55	**4145317**	70.56	**4153426**	5139.29	4153426	1224.32	6438.58
APT23	3494778	0.53	**3546535**	25.99	3546535	1067.21	3546535	559.38	1656.34
APT24	3898694	0.69	**3948037**	73.38	3948037	12497.31	3948037	1639.73	14214.76
APT25	3485589	0.67	**3507615**	36.38	3507615	2026.71	3507615	612.73	2679.23
APT26	2639964	0.64	**2683689**	33.17	2683689	2290.99	2683689	985.56	3313.24
APT27	2382270	0.66	**2438174**	19.20	2438174	1283.44	2438174	412.72	1718.58
APT28	3969356	0.52	**4065011**	79.46	4065011	7946.36	4065011	2079.59	10109.82
APT29	3619071	0.50	**3652858**	63.10	3652858	5255.54	3652858	1276.47	6599.48
B1	7990710	0.68	**7993031**	118.17	7993031	25.13	7993031	337.64	485.55
B2	7984124	0.74	**7993849**	108.62	7993849	149.35	7993849	395.13	657.75
B3	7966146	0.67	**7991615**	129.17	7991615	35.04	7991615	406.55	575.43
B4	7979928	0.54	**7989673**	135.13	7989673	1316.49	7989673	492.63	1948.81
B5	7978160	0.55	**7994752**	101.15	7994752	30.28	7994752	319.74	455.67
B6	7989831	0.66	**7992075**	107.35	7992075	458.75	7992075	413.29	983.88
B7	7999488	0.83	**8000000**	47.09	–	–	–	–	66.58

Table 3: Results obtained by a heuristic version of the Recursive Partitioning Algorithm for the large instances. The sets of conic combinations were limited in the guillotine, five-block and L-Algorithm phases of the method.

Instance	Two-stage		Guillotine cuts		Five-block		Total
	Solution	Time (sec)	Solution	Time (sec)	Solution	Time (sec)	time
GCUT14	12236280	0.70	**12245410**	205.18	**12249200**	150212.43	150422.13
GCUT15	12239904	0.60	**12246032**	224.15	**12249200**	172000.08	172229.49
GCUT16	12243100	0.70	**12248836**	312.87	**12249200**	227771.90	228091.34
GCUT17	12242998	0.61	**12248892**	408.31	**12250000**	245000.98	245417.90
APT10	3585612	0.68	**3589703**	54.98	**3589833**	22263.79	22319.45
APT11	4175414	0.56	**4188915**	81.36	**4189860**	33014.50	33099.53
APT12	5148302	0.72	**5156065**	20.22	**5156618**	9256.83	9279.67
APT13	3493072	0.71	**3498302**	55.75	3498302	22523.98	22583.18
APT14	4458052	0.54	**4463550**	50.40	4463550	20573.89	20627.23
APT15	6028426	0.73	**6047188**	70.08	**6052277**	36959.17	37032.87
APT16	7533987	0.52	**7566719**	96.11	**7571590**	51597.46	51697.39
APT17	4529371	0.57	**4535302**	87.79	**4536372**	33850.66	33942.20
APT18	5807988	0.72	**5825956**	36.83	**5829172**	15196.78	15236.43
APT19	6811338	0.56	**6826674**	48.43	6826674	27025.49	27076.86
APT20	5490828	0.70	**5545818**	111.39	**5548505**	59086.20	59201.01
APT21	3479634	0.67	**3484406**	24.58	**3491545**	12537.07	12564.35
APT22	4111542	0.56	**4145317**	69.35	**4161384**	32738.22	32810.68
APT23	3494778	0.71	**3546535**	25.49	**3559028**	13606.86	13635.08
APT24	3898694	0.68	**3948037**	73.03	**3956498**	36299.42	36375.59
APT25	3485589	0.70	**3507615**	34.62	3507615	14851.38	14888.59
APT26	2639964	0.50	**2683689**	32.43	2683689	13092.85	13127.84
APT27	2382270	0.56	**2438174**	18.42	**2443696**	6770.92	6791.78
APT28	3969356	0.56	**4065011**	79.07	4065011	39197.62	39279.83
APT29	3619071	0.70	**3652858**	61.30	3652858	28104.65	28169.23

Table 4: Results obtained by a heuristic version of the Recursive Partitioning Approach for instances GCUT14–GCUT17 and APT10–APT29. The search depth limit was set to $N = 1$ in the five-block phase and the complete conic combinations sets were used in all the phases of the method.

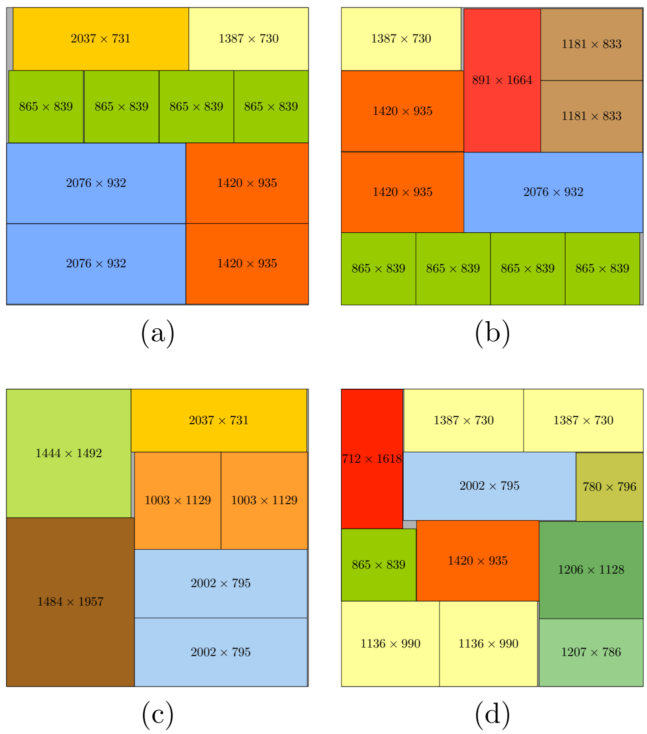

Figure 5: Graphical representations of solutions found by the Recursive Partitioning Algorithm for instance UU10. (a) Optimal two-stage pattern with value 11,929,561. (b) Optimal guillotine pattern with value 11,955,852. (c) First-order guillotine pattern with value 11,995,637. (d) L-Algorithm pattern with value 12,001,291.

5 Concluding remarks

While a large number of studies in the literature have considered staged and non-staged two-dimensional guillotine cutting problems, much less studies have considered two-dimensional non-guillotine cutting problems (constrained and unconstrained), and only a few of them have proposed exact methods to generate non-guillotine patterns. Moreover, most of the approaches (exact and heuristic) for non-guillotine cutting (or packing) were developed for the constrained problem, which can be more interesting for certain practical applications with relatively low demands of the ordered items. However, part of these methods may not perform well when solving the unconstrained problem. On the other hand, the unconstrained problem is particularly interesting for cutting stock applications with large-scale production and weakly heterogeneous items, in which the problem plays the role of a column generation procedure.

This study presented a Recursive Partitioning Approach to generate unconstrained two-dimensional non-guillotine cutting (or packing) patterns. Since we were unable to find a counterexample for which the approach fails, we conjecture that it always finds an optimal unconstrained non-guillotine cutting. The approach was able to find the optimal solution of a large number of moderate-sized instances known in the literature. To cope with large instances, we combined the approach with simple heuristics to reduce its computational efforts. For moderate-sized instances, both the five-block and the L-Algorithm phases of the approach seem to be promising alternatives for obtaining reasonably good or optimal non-guillotine solutions under affordable computer runtimes, whereas for larger instances, the guillotine or the five-block phase may be preferable, depending on the definition of an acceptable time limit. An interesting perspective for future research is to extend the Recursive Partitioning Approach to deal with constrained two-dimensional non-guillotine cutting.

The computer implementation of the algorithms as well as the data sets used in our experiments and the solutions found are publicly available for benchmarking purposes at [68].

References

[1] R. Alvarez-Valdés, A. Parajón, and J. M. Tamarit. A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems. *Computers and Operations Research*, 29:925–947, 2002.

[2] R. Alvarez-Valdés, F. Parreño, and J. M. Tamarit. A GRASP algorithm for constrained two-dimensional non-guillotine cutting problems. *Journal of the Operational Research Society*, 56:414–425, 2005.

[3] R. Alvarez-Valdés, F. Parreño, and J. M. Tamarit. A tabu search algorithm for a two-dimensional non-guillotine cutting problem. *European Journal of Operational Research*, 183:1167–1182, 2007.

[4] M. Arenales and R. Morabito. An and/or-graph approach to the solution of 2-dimensional non-guillotine cutting problems. *European Journal of Operational Research*, 84:599–617, 1995.

[5] M. Arenales, R. Morabito, and H. Yanasse. Cutting and packing problems (Special Issue). *Pesquisa Operacional*, 19(2), 1999.

[6] R. Baldacci and M. A. Boschetti. A cutting-plane approach for the two-dimensional orthogonal non-guillotine cutting problem. *European Journal of Operational Research*, 183:1136–1149, 2007.

[7] J. E. Beasley. Algorithms for unconstrained two-dimensional guillotine cutting. *Journal of the Operational Research Society*, 36:297–306, 1985.

[8] J. E. Beasley. An exact two-dimensional non-guillotine cutting tree-search procedure. *Operations Research*, 33:49–64, 1985.

[9] J. E. Beasley. A population heuristic for constrained two-dimensional non-guillotine cutting. *European Journal of Operational Research*, 156:601–627, 2004.

[10] E. G. Birgin and R. D. Lobato. Orthogonal packing of rectangles within isotropic convex regions. submitted.

[11] 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*, 61:306–320, 2010.

[12] E. G. Birgin, J. M. Martínez, W. F. Mascarenhas, and D. P. Ronconi. Method of sentinels for packing items within arbitrary convex regions. *Journal of the Operational Research Society*, 57:735–746, 2006.

[13] E. G. Birgin, J. M. Martínez, F. H. Nishihara, and D. P. Ronconi. Orthogonal packing of rectangular items within arbitrary convex regions by nonlinear optimization. *Computers and Operations Research*, 33:3535–3548, 2006.

[14] 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*, 56:1448–1451, 2005.

[15] M. Biró and E. Boros. Network flows and nonguillotine cutting patterns. *European Journal of Operational Research*, 16:215–221, 1984.

[16] E. Bischoff and G. Wäescher. Cutting and packing (Special Issue). *European Journal of Operational Research*, 84(3), 1995.

[17] A. Bortfeldt and T. Winter. A genetic algorithm for the two-dimensional knapsack problem with rectangular pieces. *International Transactions in Operational Research*, 16:685–713, 2009.

[18] M. A. Boschetti, A. Mingozzi, and E. Hadjiconstantinou. New upper bounds for the two-dimensional orthogonal non-guillotine cutting stock problem. *IMA Journal of Management Mathematics*, 13:95–119, 2002.

[19] A. Caprara and M. Monaci. On the two-dimensional knapsack problem. *Operations Research Letters*, 32:5–14, 2004.

[20] C. S. Chen, S. M. Lee, and Q. S. Shen. An analytical model for the container loading problem. *European Journal of Operational Research*, 80:68–76, 1995.

[21] D. Chen and W. Huang. A new heuristic algorithm for constrained rectangle-packing problem. *Asia-Pacific Journal of Operational Research*, 24:463–478, 2007.

[22] 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*, 191:61–85, 2008.

[23] Y. Cui, Z. Wang, and J. Li. Exact and heuristic algorithms for staged cutting problems. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 219:201–207, 2005.

[24] Y. Cui and X. Zhang. Two-stage general block patterns for the two-dimensional cutting problem. *Computers and Operations Research*, 34:2882–2893, 2007.

[25] K. A. Dowsland. Some experiments with simulated annealing techniques for packing problems. *European Journal of Operational Research*, 68:389–399, 1993.

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

[27] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44:145–159, 1990.

[28] H. Dyckhoff, G. Scheithauer, and J. Terno. Cutting and packing. In F. Maffioli M. Amico and S. Martello, editors, *Annoted Bibliographies in Combinatorial Optimization*, Wiley Interscience Series in Discrete Mathematics, chapter 22, pages 393–414. John Wiley and Sons, New York, 1997.

[29] H. Dyckhoff and G. Wäscher. Cutting and packing (Special Issue). *European Journal of Operational Research*, 44(2), 1990.

[30] J. Egeblad and D. Pisinger. Heuristic approaches for the two- and three-dimensional knapsack packing problem. *Computers and Operations Research*, 36:1026–1049, 2009.

[31] D. Fayard, M. Hifi, and V. Zissimopoulos. An efficient approach for large-scale two-dimensional guillotine cutting stock problems. *Journal of the Operational Research Society*, 49:1270–1277, 1998.

[32] S. P. Fekete, J. Schepers, and J. C. van der Veen. An exact algorithm for higher-dimensional orthogonal packing. *Operations Research*, 55:569–587, 2007.

[33] Y. G. G and M. K. Kang. A new upper bound for unconstrained two-dimensional cutting and packing. *Journal of the Operational Research Society*, 53:587–591, 2002.

[34] P. C. Gilmore and R. E. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13:94–120, 1965.

[35] J. F. Gonçalves. A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. *European Journal of Operational Research*, 183:1212–1229, 2007.

[36] E. Hadjiconstantinou and N. Christofides. An exact algorithm for general, orthogonal, 2-dimensional knapsack-problems. *European Journal of Operational Research*, 83:39–56, 1995.

[37] J. C. Herz. Recursive computational procedure for two-dimensional stock cutting. *IBM Journal of Research and Development*, 16:462–469, 1972.

[38] M. Hifi. The DH/KD algorithm: a hybrid approach for unconstrained two-dimensional cutting problems. *European Journal of Operational Research*, 97:41–52, 1997.

[39] M. Hifi. Exact algorithms for large-scale unconstrained two and three staged cutting problems. *Computational Optimization and Applications*, 18:63–88, 2001.

[40] M. Hifi. Cutting and packing problems (Special Issue). *Studia Informatica Universalis*, 2(1), 2002.

[41] M. Hifi and V. Zissimopoulos. A recursive exact algorithm for weighted two-dimensional cutting. *European Journal of Operational Research*, 91:553–564, 1996.

[42] M. Hifi and V. Zissimopoulos. Une amélioration de l'algorithme récursif de Herz pour le problème de découpe à deux dimensions. *RAIRO*, 30:111–125, 1996.

[43] E. Hopper and B. C. H. Turton. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research*, 128:34–57, 2001.

[44] W. Huang and D. Chen. An efficient heuristic algorithm for rectangle-packing problem. *Simulation Modelling Practice and Theory*, 15:1356–1365, 2007.

[45] T. W. Leung, C. K. Chan, and M. D. Troutt. Application of a mixed simulated annealing-genetic algorithm heuristic for the two-dimensional orthogonal packing problem. *European Journal of Operational Research*, 145:530–542, 2003.

[46] 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*, 54:777–789, 2003.

[47] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141:241–252, 2002.

[48] S. Martello. Knapsack, Packing and Cutting – Part I: One Dimensional Knapsack Problems (Special Issue). *INFOR*, 32(3), 1994.

[49] S. Martello. Knapsack, Packing and Cutting – Part II: Multidimensional Knapsack and Cutting Stock Problems (Special Issue). *INFOR*, 32(4), 1994.

[50] W. F. Mascarenhas and E. G. Birgin. Using sentinels to detect intersections of convex and nonconvex polygons. *Computational and Applied Mathematics*. To appear.

[51] R. Morabito, M. Arenales, and V. F. Arcaro. An and-or-graph approach for two-dimensional cutting problems. *European Journal of Operational Research*, 58:263–271, 1992.

[52] R. Morabito, M. N. Arenales, and H. H. Yanasse. Cutting, packing and related problems (Special Issue). *International Transactions in Operational Research*, 16(6), 2009.

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

[54] 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*, 50:876–876, 1999.

[55] J. F. Oliveira and G. Wäescher. Cutting and packing (feature cluster). *European Journal of Operational Research*, 183(3), 2007.

[56] M. Padberg. Packing small boxes into a big box. *Mathematical Methods of Operations Research*, 52:1–21, 2000.

[57] V. Parada, L. Pradenas, M. Solar, and R. Palma. A hybrid algorithm for the non-guillotine cutting problem. *Annals of Operations Research*, 117:151–163, 2002.

[58] G. Scheithauer. LP-based bounds for the container and multi-container loading problem. *International Transactions in Operational Research*, 6:199–213, 1999.

[59] G. Scheithauer and J. Terno. Modeling of packing problems. *Optimization*, 28:63–84, 1993.

[60] G. Scheithauer and J. Terno. The G4-heuristic for the pallet loading problem. *Journal of the Operational Research Society*, 47:511–522, 1996.

[61] A. Soke and Z. Bingul. Hybrid genetic algorithm and simulated annealing for two-dimensional non-guillotine rectangular packing problems. *Engineering Applications of Artificial Intelligence*, 19:557–567, 2006.

[62] P.E. Sweeney and E. R. Paternoster. Cutting and packing problems - a categorized, application-orientated research bibliography. *Journal of the Operational Research Society*, 43:691–706, 1992.

[63] R.D. Tsai, E.E. Malstrom, and W. Kuo. 3-dimensional palletization of mixed box sizes. *IIE Transactions*, 25:64–75, 1993.

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

[65] P. Y. Wang and G. Wäscher. Cutting and packing (Special Issue). *European Journal of Operational Research*, 141(2), 2002.

[66] L. Wei, D. Zhang, and Q. Chen. A least wasted first heuristic algorithm for the rectangular packing problem. *Computers and Operations Research*, 36:1608–1614, 2009.

[67] http://www.fe.up.pt/esicup/.

[68] http://www.ime.usp.br/∼egbirgin/packing/.