A Tabu Search algorithm for two-dimensional non-guillotine cutting problems

R. Alvarez-Valdes [†] , **F. Parreño** [‡] , **J.M. Tamarit** [†] ,

[†] University of Valencia, Department of Statistics and Operations Research, 46100 Burjassot, Valencia, Spain
[‡] University of Castilla-La Mancha. Departamento de Informatica, E.Politecnica Superior, 02071 Albacete, Spain

Abstract

In this paper we study the two-dimensional non-guillotine cutting problem, the problem of cutting rectangular pieces from a large stock rectangle so as to maximize the total value of the pieces cut. The problem has many industrial applications whenever small pieces have to be cut from or packed into a large stock sheet. We propose a tabu search algorithm. Several moves based on reducing and inserting blocks of pieces have been defined. Intensification and diversification procedures, based on long-term memory, have been included. The computational results on large sets of test instances show that the algorithm is very efficient for a wide range of packing and cutting problems.

Keywords: Non-guillotine cutting; heuristics; Tabu Search

1 Introduction

The two-dimensional non-guillotine cutting problem consists of cutting a given finite set of small rectangular pieces from a large stock rectangle of fixed dimensions with maximum profit. The problem appears in many production processes in the textile, paper, steel, glass and wood industries. $R = (L, W)$ is the large stock rectangle of length L and width W. Each piece i has dimensions (l_i, w_i), and value v_i, $i = 1, \ldots, m$. The pieces have fixed orientation and must be cut with their edges parallel to the edges of the stock rectangle (orthogonal cuts). The problem is to cut off the rectangle R into x_i copies of each piece i, such that $0 \leq P_i \leq x_i \leq Q_i$, and the total values of the pieces cut, $\sum_i v_i x_i$ is maximized. We will denote by $M = \sum_i Q_i$ the maximum number of pieces which could be cut.

According to the values of P_i and Q_i we can distinguish three types of problems:

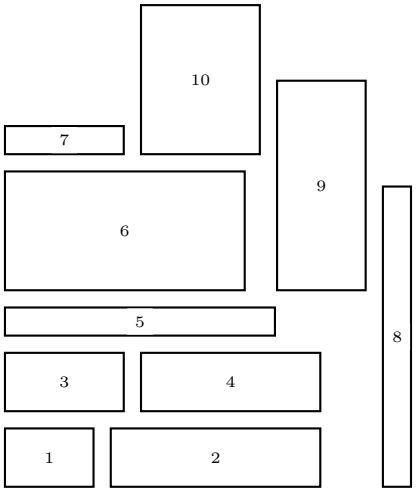
1. *Unconstrained:* $\forall i,\ P_i = 0$, $Q_i = \lfloor L * W / l_i * w_i \rfloor$ (trivial bound).

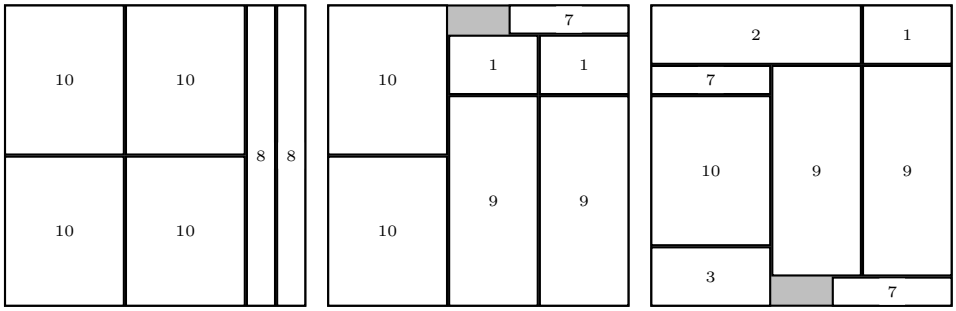
2. *Constrained:* $\forall i,\ P_i = 0;\ \exists i, Q_i < \lfloor L * W/l_i * w_i \rfloor$

3. *Doubly constrained:* $\exists\, i,\ P_i > 0;\ \exists\, j,\ Q_j < \lfloor L * W/l_j * w_j \rfloor$

In Figure 1 we see an example with a stock rectangle of $R = (10, 10)$, and $m = 10$ pieces to be cut. The first solution (Figure 1(b)) is optimal for the unconstrained problem, while the second solution (Figure 1(c)) corresponds to the constrained problem and the third (Figure 1(d)) to the doubly constrained problem, with some $P_i \neq 0$.

Stock rectangle : $R = (10, 10)$					
Piece	l_i	w_i	P_i	Q_i	v_i
1	3	2	1	2	7
2	7	2	1	3	20
3	4	2	1	2	11
4	6	2	0	3	13
5	9	1	0	2	21
6	8	4	0	1	79
7	4	1	1	2	9
8	1	10	0	1	14
9	3	7	0	3	52
10	4	5	0	2	60





(b) Unconstr. Opt: 268 (c) Constr. Opt: 247 (d) Doubly cons. Opt: 220

Figure 1: *Instance 3 from Beasley[4]*

Some authors have considered the unconstrained problem: Tsai et al.[20], Arenales and Morabito[3], Healy et al.[11]. Nevertheless, the constrained problems are more interesting for applications and more research has been devoted to this case. Some exact methods have been proposed by Beasley[4], Scheithauer and Terno[18], Hadjiconstantinou and Christofides[10], Fekete and Schepers[8], and Caprara and Monaci[6].

A simple upper bound for the problem can be obtained by solving the following bounded knapsack problem, where variable x_i represents the number of pieces of type i cut in excess of its lower bound P_i:

$$Max \quad \sum_{i=1}^{m} v_i x_i + \sum_{i=1}^{m} v_i P_i \tag{1}$$

$$s.t.: \quad \sum_{i=1}^{m} (l_i w_i) x_i \leq LW - \sum_{i=1}^{m} P_i (l_i w_i) \tag{2}$$

$$x_i \leq Q_i - P_i, \;\; i = 1, \ldots, m \tag{3}$$

$$x_i \geq 0, \quad integer, \;\; i = 1, \ldots, m. \tag{4}$$

Other bounds, apart from those included in the exact methods mentioned above, have been proposed by Scheithauer and Terno[19], Amaral and Wright[2].

Several heuristic algorithms have been proposed recently. Wu et al.[22] develop a constructive algorithm for the special case in which $P_i = Q_i \; \forall i$. At each step a piece is cut in a corner of the current cutting pattern and the piece to cut is decided according to a fitness evaluation function which estimates the quality of the complete solution that could be obtained starting from the piece being considered. The other proposals are based on metaheuristic procedures, mainly simulated annnealing and genetic algorithms. Lai and Chan[14, 15] use simulated annealing. Each solution is given by an ordered list of pieces and a list is translated into a cutting pattern by a placement algorithm. Instead of the more usual bottom-left algorithm, they propose a difference algorithm in which the piece is placed in the existing empty space which is nearest to the bottom left corner of the stock sheet. They report a limited computational experience on a small set of randomly generated instances and one real problem from a printing company. The Leung et al.[16, 17] algorithms are based on the work by Lai and Chan[14, 15] and by Jakobs[13]. Jakobs[13] develops a genetic algorithm for the related strip packing problem and uses as placement algorithm a bottom-left procedure. They combine both metaheuristics and both placement algorithms and report computational results on a set of randomly generated instances. Beasley [5] develops a genetic algorithm based on a non-linear formulation of the problem, where variables indicate if a piece is cut or not and its position on the stock sheet. Therefore, the solutions are lists of variables and directly show the cutting pattern. No placement algorithm is needed. He presents a complete computational study on a set of standard test problems and on a number of large randomly generated problems. Alvarez-Valdes et al [1] follow a different approach and develop a GRASP algorithm. Their computational tests collect the problems used by Leung et al.[16, 17] and by Beasley[5].

In this paper, we present a Tabu Search algorithm for the two-dimensional non-guillotine cutting problem. We provide computational results obtained on

four sets of test problems: the 21 problems from the literature collected by Beasley [5]; the 630 large random problems also generated by Beasley [5]; 10 problems used by Leung et al. [17], and the 21 problems used by Hopper and Turton[12]. The last set of problems were initially designed for the strip packing problem and have been adapted to the two-dimensional non-guillotine cutting problem with the aim of testing the algorithm on a set of large and difficult zero-waste instances.

2 A constructive algorithm

In this section we briefly describe a constructive algorithm that will be used in the Tabu Search algorithm. More details may be found in [1]. Constructing a solution is an iterative process in which we combine two elements: a list $\mathcal{P}$ of pieces still to be cut, initially the complete list of pieces, and a list $\mathcal{L}$ of empty rectangles in which a piece can be cut, initially containing only the stock rectangle $R = (L, W)$. At each step a rectangle is chosen from $\mathcal{L}$, and from the pieces in $\mathcal{P}$ fitting in it a piece is chosen to be cut. That usually produces new rectangles going into $\mathcal{L}$ and the process goes on until $\mathcal{L} = \emptyset$ or none of the remaining pieces fit into one of the remaining rectangles.

Step 0. Initialization:

$\mathcal{L} = \{R\}$, the set of empty rectangles.
$\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$, the set of pieces still to be cut.
The set $\mathcal{P}$ is initially ordered according to 3 criteria: Order by non-increasing $P_i * l_i * w_i$, giving priority to pieces which must be cut. If there is a tie (for instance, if $P_i = 0, \forall i$), order by non-increasing $v_i/(l_i * w_i)$. If there is a tie (for instance, if $v_i = l_i * w_i, \forall i$), order by non-increasing $l_i * w_i$.

$\mathcal{B} = \emptyset$, the set of pieces cut. Pieces of the same type may appear grouped in rectangular *blocks*.

Step 1. Choosing the rectangle:

Take R^*, the smallest rectangle of $\mathcal{L}$ in which a piece $p_i \in \mathcal{P}$ can fit.
If such R^* does not exist, stop.
Otherwise, go to Step 2.

Step 2. Choosing the piece to cut:

Choose a piece p_i and a quantity $n_i \leq Q_i$, forming block B^* to be cut in R^*.
The piece i is chosen to produce the largest increase in the objective function.
Block B^* is cut in the corner of R^* which is nearest to a corner of the stock rectangle.

Update $\mathcal{P}$, $\mathcal{B}$ and Q_i which indicates the number of pieces still to be cut.
Move block B^* towards the nearest corner of the stock rectangle.

Step 3. Updating the list $\mathcal{L}$:

Add to $\mathcal{L}$ the possible rectangles produced when cutting B^* from R^*.
Take into account the possible changes in $\mathcal{L}$ when moving block B^*.
Merge rectangles to favor cutting new pieces of $\mathcal{P}$.
Go back to Step 1.

Though we keep a list of empty rectangles $\mathcal{L}$, we really have an irregular, polygonal empty space in which the pieces still to be cut can be considered for fitting. One way of adapting our list $\mathcal{L}$ to the flexibility of non-guillotine cutting is to merge some of the rectangles from the list, producing some new rectangles in which the pieces to be cut could fit better.

When we merge 2 rectangles, at most 3 new rectangles may appear, usually one *large* rectangle and 2 *small* ones (see Figure 2). Among the several alternatives for merging we try to select the best, that is, the one in which it is possible to cut the pieces best situated in the ordered list $\mathcal{P}$. With this objective in mind, we impose some conditions:

1. If the order of the best piece which fits into the *large* rectangle is strictly lower than the order of the pieces in the original rectangles, we merge them.

2. If the order of the best piece which fits into the *large* rectangle is equal to the order of the pieces in the original rectangles, we merge them if the area of the large rectangle is bigger than the area of each one of the original rectangles.

3. If the order of the best piece which fits in the *large* rectangle is strictly greater than the order of the pieces in the original rectangles, we do not merge them.

In Figure 2 we see several possible cases. In Figure 2(a) the two original rectangles will always be merged. The new rectangle is larger than them and all the pieces fitting in the original rectangles will fit in it. In Figure 2(b) the new rectangles are not larger than the original ones. These will be merged only if the new central rectangle accommodates a piece of lower order than those fitting in the original rectangles. In Figure 2(c) one of the new rectangles is larger than the original ones and therefore they will be merged unless the best piece fitting in the original vertical rectangle does not fit in the new ones.

At the end of the constructive process, a solution is composed of a list of blocks $\mathcal{B}$, and a list of empty rectangles $\mathcal{L}$, with total value $\sum_i v_i x_i$.

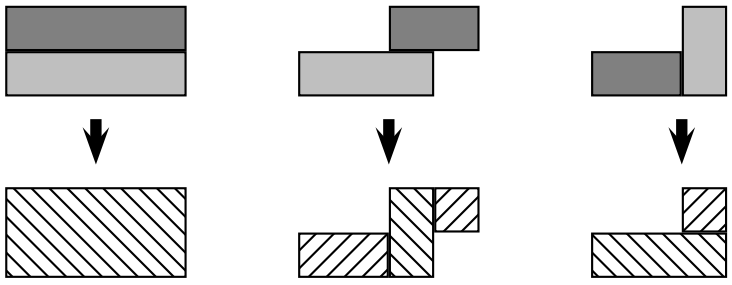


Figure 2: *Merging 2 empty rectangles*

3 A Tabu Search algorithm

Tabu Search is now a well-established optimization algorithm (for an introduction, refer to the book by Glover and Laguna[9]). The basic elements of the algorithm are described in the following subsections.

3.1 Definition of moves

The solution space in which we move is composed of feasible solutions only. In this space we will define several moves to go from one solution to another. An initial solution is obtained by applying the constructive algorithm described in Section 2.

We distinguish two types of moves: block reduction and block insertion. In block reduction, the size of an existing block is reduced, eliminating some of its rows or columns. In block insertion, a new block is added to the solution. For both moves we first present a scheme of the procedures and then a detailed example.

- ***Block reduction***

  Step 0. Initialization:

  $\mathcal{B}$ = the list of blocks of the current solution
  $\mathcal{L}$ = the list of empty rectangles

  Step 1. Choosing the block to reduce

  Take B, one of the blocks of $\mathcal{B}$, with k columns and l rows of pieces p_i.
  Select the number r of columns (rows) to eliminate,

  $$1 \leq r \leq k \quad (1 \leq r \leq l),$$

  keeping the number of pieces in the solution $x_i \geq P_i$.

If $P_i = 0$, the block may disappear completely.

The new waste rectangle R is added to $\mathcal{L}$.

Step 2. Move the remaining blocks to their nearest corners:

The list of waste rectangles $\mathcal{L}$ is updated accordingly.

Step 3. Fill the empty rectangles with new blocks:

Apply the constructive algorithm of Section 2,

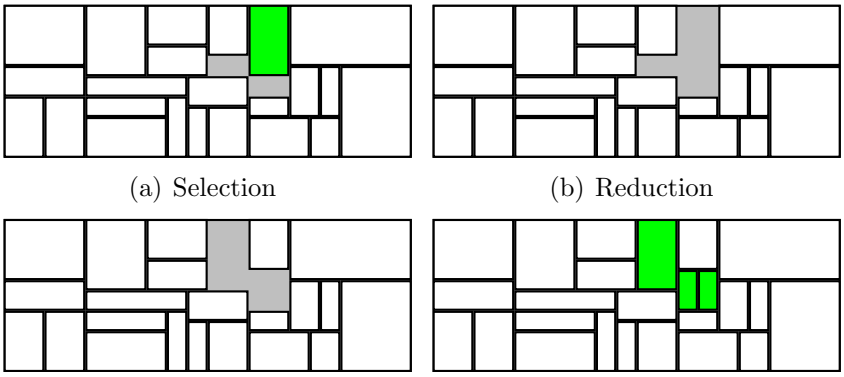
The algorithm starts from the current lists $\mathcal{L}$ and $\mathcal{B}$, and $\mathcal{P}$ contains the pieces which can still be cut and added to the current solution. Before proceeding to the construction process, rectangles in $\mathcal{L}$ are considered for merging, to best accommodate the pieces of $\mathcal{P}$.

When selecting the piece to cut, the piece eliminated at Step 1 is not considered until another piece has been included in the modified solution.

Step 4. Merge the blocks with the same structure:

We try to merge blocks of the same piece with the same length or width if they are adjacent and have a common side, or if one of them can be moved to make them both adjacent to one common side.

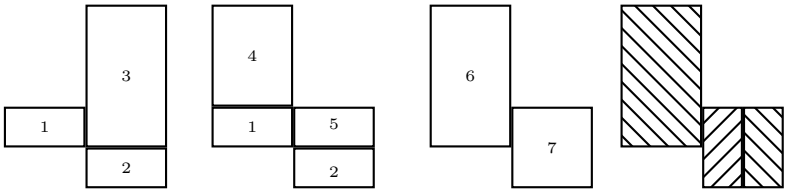
In Figure 3 we see an example of a reduction move on an instance proposed by Jakobs[13] and used later by Leung et al.[17]. The stock rectangle is $R = (120, 45)$, $m = 22$ and $M = 25$ pieces can be cut from it, completely filling it. Figure 3(a) shows a solution with 23 pieces which cannot accommodate two (6x12) pieces. The set $\mathcal{L}$ of empty rectangles is composed of $R_1 = (60, 24, 72, 30)$ and $R_2 = (72, 18, 84, 24)$ (in light grey). In Step 1, a block composed of one piece (12x21) (in dark grey), is selected to be reduced and therefore it disappears from the solution, creating a new empty rectangle $R_3 = (72, 24, 84, 45)$ which is added to $\mathcal{L}$ (Figure 3(b)). In Step 2, a block composed of a piece (12x15) is moved to the top right corner. Therefore, $\mathcal{L} = \{R_1, R_2, R_4, R_5\}$, where $R_4 = (60, 30, 72, 45)$ and $R_5 = (72, 24, 84, 30)$ (Figure 3(c)). In Step 3 the constructive procedure fills the empty rectangles. First, R_1 and R_4 are merged, forming $R_6 = (60, 24, 72, 45)$, and R_2 and R_5 are merged, forming $R_7 = (72, 18, 84, 30)$. Then, R_7 is selected and the two pieces (6x12) are cut in it, completely filling it. Finally, R_6 is taken and the piece initially eliminated is cut in it. The final solution, which is optimal, appears in Figure 3(d).



(a) Selection (b) Reduction

(c) Moving to the corner (d) Fill

Figure 3: *Block reduction. Instance 3 from Jakobs[13]*



(a) Step 1 (b) Step 2 (c) Merge (d) Fill

Figure 4: *Empty rectangles in the reduction move*

- ***Block insertion***

  Step 0. Initialization:

  $\mathcal{B}$ = the list of blocks
  $\mathcal{L}$ = the list of empty rectangles

  Step 1. Choosing the block to insert

  Take p_i, a piece for which $x_i < Q_i$, and consider a block of these pieces with k columns and l rows ($k * l \leq Q_i - x_i$).

  Step 2. Select the position to insert the new block

  Step 3. Remove the pieces of the solution overlapping with the inserted block

  Update $\mathcal{B}$ (some of the original blocks are reduced or eliminated)
  Update $\mathcal{L}$ (some new empty rectangles may appear).

Step 4. Fill the empty rectangles with new blocks

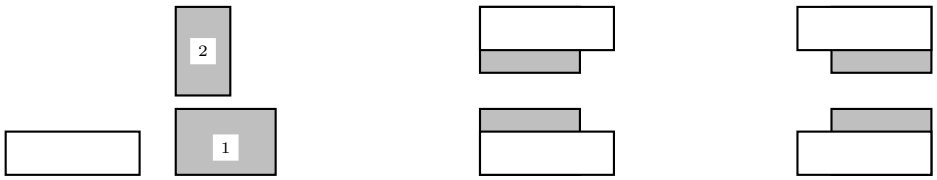
Step 5. Merge the blocks with the same structure:

Steps 4 and 5 are the same as in block reduction.

In Step 3, two strategies have been considered to select the position of the new block. In both cases, the block is placed partially or totally covering one or more empty rectangles.

- For each empty rectangle consider the four alternatives in which a corner of the rectangle is chosen for the corresponding corner of the block. If the dimensions of the block are larger than those of the rectangle, the block may overlap with other blocks or may occupy part of other empty rectangles.
- Select only one empty rectangle, that producing the largest intersection with the block if the bottom left corner of the block were placed in the bottom left corner of the rectangle. For this rectangle, the four alternatives described above are considered.

In Figure 5 we see an example of the second strategy. In Figure 5(a) two empty rectangles, in grey, are considered to accommodate the new block, in white. As the largest common area corresponds to rectangle 1, it is chosen. In Figure 5(b) the four corners of the empty rectangle are considered for situating the corner of the new block.

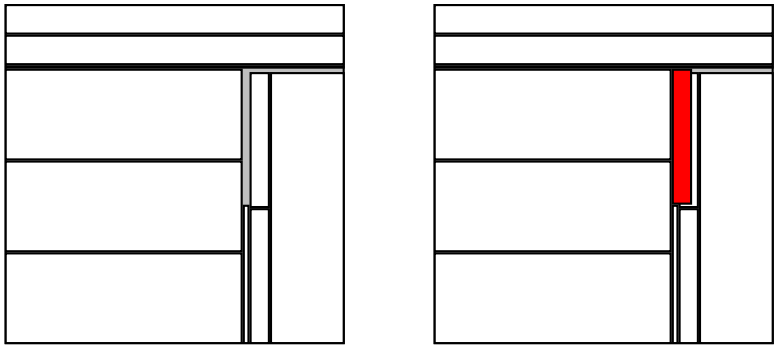


(a) Largest common area (b) Possible positions

Figure 5: *Selecting the position of the new block*

In Figure 6 we see an example of an insertion move on an instance proposed by Fekete and Schepers[8] and later used by Beasley[5]. The stock rectangle is $R = (100, 100)$, $m = 15$ and $M = 50$ pieces can be cut from it. Figure 6(a) shows a solution of value $z = 27539$. The set $\mathcal{L}$ of empty rectangles is composed of $R_1 = (70, 41, 72, 81)$ and $R_2 = (72, 80, 100, 81)$. In Step 1 we select a piece $i = 5$ of dimensions (6x40) with $Q_i = 5$ and only 2 copies in the current solution

and consider a block B^* of one piece. In Step 2 we place B^* over R_1, selecting the top left corner of the rectangle to locate the top left corner of the block. B^* completely covers R_1 and part of R_2, which becomes $R_3 = (76, 80, 100, 81)$. B^* also partially covers an existing block (Figure 6(b)). Therefore, in Step 3, we remove the pieces of the initial solution overlapping with B^*. That produces two new empty rectangles $R_4 = (76, 40, 78, 80)$ and $R_5 = (72, 40, 76, 41)$ (Figure 6(c)). In Step 4, the filling procedure starts form this list $\mathcal{L} = \{R_3, R_4, R_5\}$. First, R_3 and R_4 are merged, producing $R_6 = (76, 40, 78, 81)$ and $R_7 = (78, 80, 100, 81)$. While none of the remaining pieces could fit either in R_3 or in R_4, a piece $i = 13$ of dimensions (2x41) now fits into R_6. The new solution is better than the initial one and has a value $z* = 27718$, optimal for the problem (Figure 6(d)).



(a) Initial (b) Insert

(c) Eliminate overlap (d) Merge and fill

Figure 6: *Block insertion. Instance 1 from Fekete and Schepers[8]*

3.2 Moves to be studied

At each iteration we study all the possible reduction and insertion moves which can be applied to the current solution.

- Reduction:
  1. Take each block of the solution, one at a time, in random order.
  2. Consider all possibilities of reduction in the directions adjacent to empty rectangles.
- Insertion:
  1. Select a piece for which the number of copies in the solution, x_i, is lower than Q_i, one at time, in random order.
  2. Consider all the possible blocks which can be built with this piece.
  3. Consider all the alternatives for placing the block onto an empty rectangle.

3.3 Selection of the move

The objective function consists only of maximizing the value of pieces cut $f(x) = \sum_i v_i x_i$. However, if the moves are evaluated according to that function, there can be many moves with the same evaluation. Therefore, if there are ties, we break them by using a secondary objective function $g(x) = k_1 S + k_2|\mathcal{L}| + k_3 C + k_4 F$.

- *S (Symmetry)*: We try not to explore symmetric solutions but prefer solutions in which empty rectangles are mostly concentrated to the right and to the top of the stock rectangle.

  $S = 1$ if there is no symmetric solution with the wastes more concentrated to the right and to the top. Otherwise, $S = 0$.

- $|\mathcal{L}|$ *(Number of empty rectangles))*. If possible, we prefer solutions in which the number of empty rectangles will be as low as possible.

- *C (Centered and concentrated wastes)*: We prefer solutions in which empty rectangles are centered and concentrated as much as possible, because that will make it easier to merge them and obtain spaces for more pieces. We consider the smallest rectangle ER containing all the empty rectangles and

  $C = 1 - (0.75 * rd + 0.25 * ra)$

  where rd is the distance from the center of ER to the center of the stock rectangle, divided by the distance from the center of the stock rectangle to its bottom left corner, and ra is the area of ER divided by the area of the stock rectangle.

- *F (Feasibility).* In doubly constrained problems, the initial solution may not be feasible. In this case $F = 1$. Otherwise, $F = 0$.

These criteria are added to the secondary function with some weights reflecting their relative importance, according to the results of a preliminary computational experience on a subset of problems. In the current implementation, the weights are:

Criterion	Coefficient	Weight
Symmetry	k_1	5000
Empty rectangles	k_2	-950
Centered empty rectangles	k_3	50
Feasibility	k_4	-50000

3.4 Tabu list

The tabu list contains for each solution the following pair of attributes: the value of the objective function and the smallest rectangle ER containing all its empty rectangles. A move is then tabu if these two attributes of the new solution match one pair of the tabu list.

The tabu list size varies dynamically. After a given number of iterations without improving the solution, the length is randomly chosen from $[0.25*M, 0.75*M]$, where $M = \sum_i Q_i$.

The *aspiration criterion* allows us to move to a tabu solution if it improves the best solution obtained so far.

3.5 Intensification and diversification strategies

The moves we have defined involve a high level of diversification. However, we have included two more diversification strategies:

- Long term memory

  Throughout the search process, we keep in memory the frequency of each piece appearing in the solutions.

  This information is used for both diversification and intensification purposes. When used for diversification, we favor the moves of pieces not appearing very frequently in the solutions, then inducing new pieces to appear. When used for intensification, we consider only pieces corresponding to high quality solutions and then we favor these pieces appearing again in the new solutions.

In a diversification phase, the objective function is modified by subtracting a term that is the sum of the frequencies of the pieces appearing in the solutions:

$f(x) \rightarrow f(x) - \sum_i freq(p_i)$

In an intensification phase, the objective function is modified by adding a term reflecting the frequency of the pieces in the set of elite solutions $\mathcal{E}$

$f(x) \rightarrow f(x) + K \sum_{i \in \mathcal{E}} freq(p_i)$

- Restarting

  According to the secondary objective function, we tend to explore solutions satisfying the symmetry criterion. After a given number of iterations without improving the best known solution, the current solution is changed by performing a horizontal and a vertical symmetry on it. The new solution obtained in that way would be quite different from the recently studied solutions and it is taken as a new starting point for the search.

3.6 Adjusting the bounds of the pieces

Throughout the iterative process we have the best known solution of value v_{best}. We can use this value to adjust P_i of some pieces that must appear if we want to improve the solution, and Q_j of some pieces whose inclusion would not allow us to improve the solution.

- *Increasing lower bounds P_i*

  Let us define $total_{pieces} = \sum_i^m v_i * Q_i$, the total value of the available pieces. If there is a piece i such that $P_i < Q_i$, and $total_{pieces} - (Q_i - P_i) * v_i <= v_{best}$, a solution with the minimum P_i copies of this type of piece cannot improve the best known solution. Any better solution must include more pieces of this type and P_i can be increased. If we compute t as:

$$max\, t: \quad total_{pieces} - t * v_i > v_{best}\,; \quad t \geq 0,\, t \leq Q_i - P_i$$

  Then, $P_i = Q_i - t$. This improved lower bound can be useful in the constructive phase, in which the pieces with $P_i > 0$ are cut first, and in the improvement phase, in which pieces in their lower bounds are not considered to be removed from the current solution.

- *Decreasing upper bounds Q_i*

  Let us denote by $R = \sum_{P_i>0} P_i * l_i * w_i$ the area of the pieces which must appear in any feasible solution, $R_v = \sum_{P_i>0} P_i * v_i$, the value of these pieces, $e_i = v_i/(l_i * w_i)$ the efficiency of piece i and $e_{max} = max\{e_i, i = 1, \ldots, m\}$, the maximum efficiency of the pieces. If there is a piece i, with $Q_i > P_i$ and $e_i < e_{max}$ satisfying:

  $$(Q_i * l_i * w_i * (e_{max} - e_i) \geq e_{max} * (L * W - R) + R_v - v_{best}$$

  any solution with Q_i copies of this piece cannot improve the best known solution. Therefore, at any better solution the number of copies of piece i should be limited to below Q_i. If we compute t as:

  $$max\, t: \quad t * l_i * w_i * (e_{max} - e_i) < (e_{max} * (L * W - R) + R_v - v_{best}$$
  $$t \geq 0,\, t \leq Q_i - P_i$$

  then $Q_i = P_i + t$. This decrease in the upper bound can be useful when constructing and improving solutions in the subsequent iterations. In some cases, Q_i can be set to 0 and the corresponding piece is no longer considered for cutting.

3.7 Path relinking

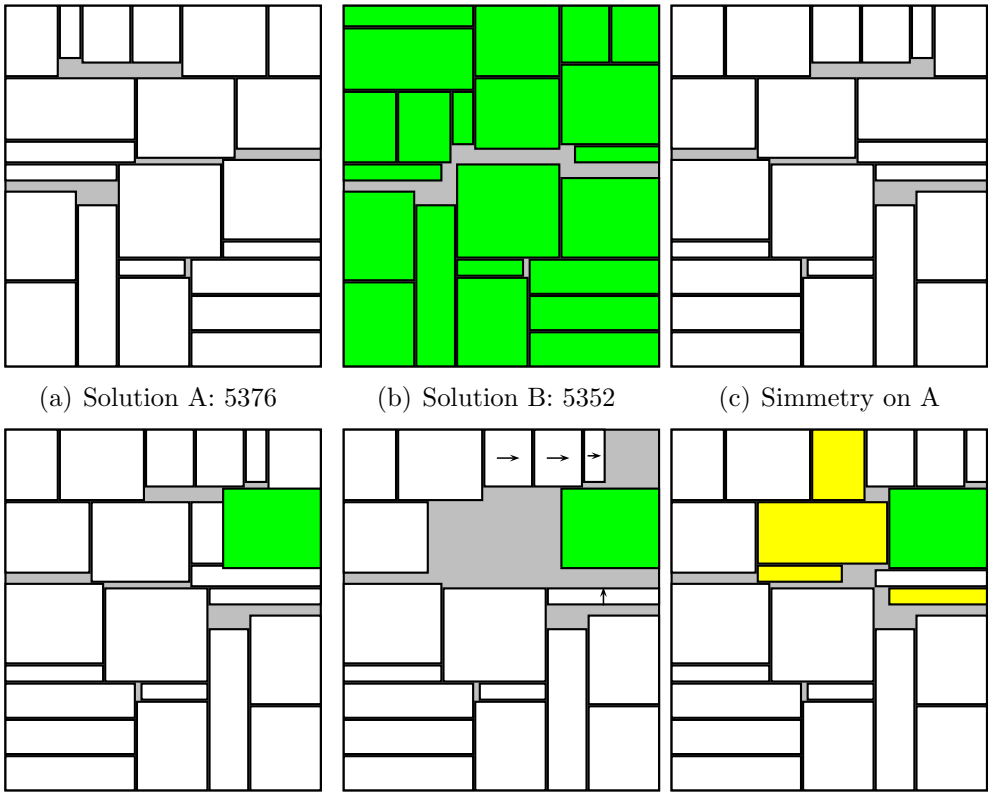
Path Relinking is an approach for integrating intensification and diversification strategies in the context of Tabu Search (Glover and Laguna[9]). This approach generates new solutions by exploring trajectories that connect high quality solutions. Starting from one of these solutions, called the initiating solution, a path is generated in the solution space that leads towards another solution, called the guiding solution. This is done by selecting moves that introduce the attributes of the guiding solution into the new solutions.

Throughout the search process we keep a set of elite solutions, the best solutions found. We now take pairs of elite solutions and use one of them, solution A, as the initiating solution and the other, solution B, as the guiding solution. As the Tabu search algorithm favors solutions with empty rectangles preferably in the upper right part of the stock rectangle, both solutions will tend to have the empty rectangles in this zone. Therefore, we apply a vertical symmetry to the initiating solution before starting the Path Relinking process.

We follow a constructive strategy, inserting the blocks of solution B, one at a time, into solution A. The insertion move follows the procedure described in

Section 3.1. The pieces overlapping with the block are eliminated, the remaining blocks are moved to their nearest corners and the resulting empty rectangles are filled. At the end of the process, we will have reproduced solution B, but along the path new solutions will have been generated. When inserting the blocks of solution B, we first insert the blocks adjacent to the sides of the stock rectangle and then the blocks in the center.

An example of the first step of Path Relinking appears in Figure 7. Solution A with value 5376 is selected as the initiating solution (Figure 7(a)), and solution B with value 5352 as the guiding solution (Figure 7(b)). The vertical symmetry applied to solution A produces the solution of Figure 7(c). Then a block of B is inserted on it (Figure 7(d)). The pieces overlapping with it are deleted as well as one piece of the same type of the piece being inserted so that Q_i is not exceeded. The remaining blocks are moved to their nearest corners, as indicated by the arrows in Figure 7(e). Finally the empty spaces are merged and filled with new pieces, producing the solution in Figure 7(f). This solution is different from solutions A and B and has a value of 5395.



(a) Solution A: 5376 (b) Solution B: 5352 (c) Simmetry on A

(d) Insert a block of B (e) Eliminate overlapping (f) Merge and fill

Figure 7: *Path Relinking. Instance 2 from Jakobs[13]*

4 Computational experience

4.1 Test problems

We have used several sets of test problems:

1. A set of 21 problems from the literature: 15 from Beasley [4], 2 from Hadjiconstaninou and Christofides[10], 1 from Wang[21], 1 from Christofides and Whitlock[7], 5 from Fekete and Schepers[8]. For all of them the optimal solutions are known. They have also been solved by Beasley [5].

2. A set of 630 large problems generated by Beasley[5], following the work by Fekete and Schepers[8]. All the problems have a stock rectangle of size $(100, 100)$. For each value of m, number of piece types (m =40, 50, 100, 150, 250, 500, 1000), 10 problems are randomly generated with $P_i = 0$, $Q_i = Q^*, \forall i = 1, \ldots, m$ where $Q^* = 1; 3; 4$. These 630 instances are divided into 3 types, according to the percentages of the types of pieces of each class:

Class	*Description*	*Length*	*Width*
1	Short and wide	[1,50]	[75,100]
2	Long and narrow	[75,100]	[1,50]
3	Large	[50,100]	[50,100]
4	Small	[1,50]	[1,50]

Type	Percentages of pieces of each class			
	1	2	3	4
1	20	20	20	40
2	15	15	15	55
3	10	10	10	70

   The value assigned to each piece is equal to its area multiplied by an integer randomly chosen from $\{1, 2, 3\}$.

3. The 21 test problems mentioned first were transformed by Beasley[5] into doubly constrained problems by defining some lower bounds P_i. Specifically, for each type of piece from $i = 1, \ldots, m$ satisfying:

   $\sum_{j=1, j\neq i}^{m} (l_j w_j) P_j + l_i w_i \leq (LW)/3$, the lower bound P_i is set to 1.

   This set of problems would allow us to test the algorithm in the general case of doubly constrained problems.

4. Finally, we have included the test problems used by Leung et al.[17], consisting of 3 instances from Lai and Chan[14], 5 from Jakobs[13], and 2 from Leung et al.[17]. We have also included 21 larger instances from Hopper and Turton [12]. There are unweighted problems in which the value of each piece corresponds to its area and the objective is to minimize the waste of the stock rectangle. The problems have been generated in such a way that the optimal solution is a cutting pattern with zero waste.

We have included the Leung et al.[17] and Hopper and Turton[12] sets of problems because they have characteristics which can be considered complementary to the two first sets used by Beasley, as can be seen in Table 1, in which we show the ratios of total pieces available to be cut to the upper bound of pieces fitting into the stock rectangles. We can see that the problems of the second set, Types I, II and III, can be considered *selection* problems because there are many available pieces and only a small fraction of them will make part of the solution. However, the Leung et al. and Hopper and Turton problems are *jigsaw* problems. All available pieces will make part of the solution and the difficulty here is to find their correct position in the cutting pattern. An algorithm working well on both types of problems can be considered a general purpose algorithm.

Sets of problems	Averages	
	Total value of pieces/ Upper bound of value	Total area of pieces/ Upper bound of area
Literature problems	3,13	3,61
Type I	123,69	185,60
Type II	101,69	152,71
Type III	79,67	119,20
Zero-waste problems	1,00	1,00

Table 1: *Test problems – Characteristics.*

4.2 Implementation

Our algorithm has been coded in $C++$ and run on a $PentiumIII$ at $800\,Mhz$. After 100 iterations without improving the best known solution, the length of the tabu list is changed. After 400 iterations without improvement we do a diversification phase based on long term frequencies over 100 iterations or until an improved solution is found. We then recover the original objective function and continue the search. After 400 iterations without improvement we do an intensification phase with $K = 100$ over 100 iterations or until an improved solution is found. We recover the original objective function but if the solution has not been not improved, instead of continuing the search from it we do a restarting move and proceed from the new solution.

In Section 4.1 we have described three types of problems: constrained selection problems, constrained jigsaw problems and doubly constrained problems. Some of the strategies described in Section 3 are more adequate for some of these types of problems. In Step 2 of the Block insertion move, if we have a selection problem, we use the second strategy, identifying the empty rectangle with the largest common area and only studying the possible positions of the new block on it. If we have a jigsaw problem, we study all the empty rectangles as possible placements for the block. The algorithm automatically detects the type of problem and applies the adequate strategy.

The algorithm runs until it reaches the optimal solution, if known, or the corresponding upper bound, or until a limit of 1500 iterations. The strategy of stopping at the optimal solution or the upper bound has been used by Beasley[5] and we have adopted it in order to compare our results with those obtained by him. The limit of 1500 has been set to keep the running times similar to those of the GRASP algorithm used by Alvarez-Valdes et al.[1].

Throughout the search we keep a set of 10 elite solutions upon which the Path Relinking procedure will act. However, as the solutions provided by Tabu Search were so good, Path Relinking could not improve the initial solutions and this procedure has not been included in the final implementation.

4.3 Computational results

The computational results appear in Tables 2, 3 and 4. The first two tables include a direct comparison with Beasley's results[5] and with the GRASP algorithm results[1] in terms of solution quality. The computing times cannot be directly compared with Beasley's time. Beasley coded his algorithm in FORTRAN and used a Silicon Graphics O2 workstation (R10000 chip, 225MHz, 128 MB). An approximate comparison ($http://www.spec.org$) indicates that his computer is twice as fast as ours. On Table 2 we see that our Tabu Search algorithm optimally solves all the problems in very short computing times, clearly outperforming the other two algorithms in terms of quality and running times. For the large problems in Table 3 the optimal solutions are not known and the comparisons are made with knapsack upper bounds. Table 3 shows that the Tabu Search algorithm again obtains better results on every type of problem, except for $m = 50$, $Q^* = 1$ in which GRASP is slightly better. The computing times are much shorter than those of Beasley's algorithm, though they are larger than those required by the GRASP procedure. Both algorithms are based on similar ideas. The more elaborated Tabu Search algorithm obtains better solutions but needs longer processing times.

The adjustment of lower bounds does not have significant effects on the performance of the algorithms, but the adjustment of upper bounds does have a dramatic effect, especially in these large random problems in which there are im-

portant differences in the efficiency of the pieces. For instance, for problems with $m = 1000$ types of pieces, more than 60% of the pieces are discarded as soon as good solutions are found. That increases significantly the speed of GRASP and Tabu Search algorithms which use these adjustments.

The direct comparison with Leung et al.[17] is not possible, though their 19 test instances are a subset of those appearing in Table 4. On the one hand, they do not give CPU times. On the other hand, they propose two versions of their algorithm, each of them with several mutation rates, and give minimum and mean waste in 15 runs of 30000 iterations. The best that can be said is that our average distance to optimum is slightly better than the average distances of their algorithms and quite similar to the best distances they obtain after 15 runs. We can also point out, as Beasley [5] illustrates, that there are some optimal cutting patterns that cannot be obtained by the Leung et al.[17] procedure, a situation that does not arise with our procedure. In Table 4 we again compare the GRASP and the Tabu Search algorithms. Tabu Search clearly outperforms GRASP in terms of the quality of the solution, with quite similar computing times.

Finally, Table 5 shows the results of the algorithms on the set of doubly constrained test problems. The upper bound corresponds to the solution of the constrained problem. The problems for which the algorithms do not find solutions are not feasible, but they are maintained in the set of test problems and therefore are included in the table. The Tabu Search algorithm obtains the best result for each instance of the set but its computing times are slightly longer.

5 Conclusions

We have developed a new heuristic algorithm based on Tabu Search techniques for the non-guillotine two-dimensional cutting stock problem. Two moves have been proposed, based on the reduction and insertion of blocks of pieces. The efficiency of the moves is based on a *merge and fill* strategy that accommodates the empty rectangles to the pieces still to be cut. Some intensification and diversification strategies, based on long-term memory, have also been included.

The computational results show that these ideas work well for the constrained and doubly constrained test problems proposed by Beasley [5]. For the Leung et al.[17] and Hopper and Turton[12] zero-waste problems the results are also good and the proposed algorithm can be considered to work consistently well for a wide range of cutting problems.

Source of problem	I	Problem size (L,W)	m	M	Beasley's solution	GRASP solution	TABU solution	Optimal solution	CPU time (seconds) Beasley	GRASP	TABU
Beasley [4]	1	(10, 10)	5	10	164	164	164	164	0,02	0,00	0,06
	2	(10, 10)	7	17	230	230	230	230	0,16	0,00	0,00
	3	(10, 10)	10	21	247	247	247	247	0,53	0,00	0,00
	4	(15, 10)	5	7	268	268	268	268	0,01	0,00	0,00
	5	(15, 10)	7	14	358	358	358	358	0,11	0,00	0,00
	6	(15, 10)	10	15	289	289	289	289	0,43	0,00	0,00
	7	(20, 20)	5	8	430	430	430	430	0,01	0,00	0,00
	8	(20, 20)	7	13	834	834	834	834	3,25	0,77	0,16
	9	(20, 20)	10	18	924	924	924	924	2,18	0,00	0,05
	10	(30, 30)	5	13	1452	1452	1452	1452	0,03	0,00	0,00
	11	(30, 30)	7	15	1688	1688	1688	1688	0,60	0,05	0,00
	12	(30, 30)	10	22	1801	1865	1865	1865	3,48	0,05	0,06
Hadjiconstantinou	3	(30, 30)	7	7	1178	1178	1178	1178	0,03	0,00	0,00
and Christofides [10]	11	(30, 30)	15	15	1270	1270	1270	1270	0,04	0,00	0,00
Wang[21]		(70, 40)	19	42	2721	2726	2726	2726	6,86	0,77	0,11
Christofides and Whitlock[7]	3	(40, 70)	20	62	1720	1860	1860	1860	8,63	0,39	0,06
Fekete and Scheppers [8]	1	(100, 100)	15	50	27486	27589	27718	27718	19,71	2,31	0,05
	2	(100, 100)	30	30	21976	21976	22502	22502	13,19	4,17	2,14
	3	(100, 100)	30	30	23743	23743	24019	24019	11,46	3,68	3,40
	4	(100, 100)	33	61	31269	32893	32893	32893	32,08	0,00	0,66
	5	(100, 100)	29	97	26332	27923	27923	27923	83,44	0,00	0,00
Mean percentage of deviation from optimum					1,21%	0,19%	0,00%		8,87	0,58	0,32
Number of optimal solutions (out of 21)					13	18	21				

Table 2: *Computational results – Problems from literature*

Mean percentages of deviation from knapsack upper bound

m	Q*	M	Beasley's solution	GRASP solution	TABU solution	CPU time (seconds) Beasley	GRASP	TABU
40	1	40	7,77	6,97	6,55	13,57	2,33	10,97
	3	120	3,54	2,22	1,95	47,43	6,62	14,20
	4	160	3,24	1,81	1,65	63,30	4,44	18,26
50	1	50	5,48	4,80	4,85	14,60	4,71	15,49
	3	150	2,35	1,50	1,27	59,27	7,05	22,50
	4	200	2,63	1,18	0,96	80,07	5,34	18,19
100	1	100	2,26	1,51	1,50	27,20	5,36	38,79
	3	300	1,27	0,47	0,31	119,47	9,41	32,11
	4	400	1,06	0,26	0,18	175,10	6,99	19,67
150	1	150	1,31	0,89	0,84	40,60	5,53	54,90
	3	450	0,60	0,14	0,07	190,53	11,71	31,76
	4	600	0,92	0,11	0,05	323,83	6,75	19,87
250	1	250	0,88	0,51	0,45	76,70	5,27	90,07
	3	750	0,57	0,04	0,01	439,47	13,89	13,70
	4	1000	0,39	0,03	0,00	693,67	6,65	4,50
500	1	500	0,26	0,05	0,03	203,10	3,24	86,17
	3	1500	0,18	0,00	0,00	1210,80	12,24	1,10
	4	2000	0,18	0,00	0,00	1790,83	1,15	0,84
1000	1	1000	0,09	0,00	0,00	667,23	1,01	7,80
	3	3000	0,07	0,00	0,00	3318,47	6,53	1,54
	4	4000	0,07	0,00	0,00	4840,57	0,29	1,19
Type I			1,64	1,04	0,95	558,11	5,13	19,61
Type 2			1,70	1,14	1,06	668,41	5,90	23,84
Type 3			1,66	1,03	0,94	830,02	7,28	32,56
All			1,67	1,07	0,98	685,51	5,91	25,34

Table 3: *Computational results– Large random problems.*

Source of problem	I	Problem size (L,W)	m	M	GRASP solution	TABU solution	Optimal solution	CPU time GRASP	CPU time TABU
Lai and Chan[14]	1	(400,200)	9	10	80000	80000	80000	0,00	0,00
	2	(400,200)	7	15	79000	79000	79000	0,00	0,02
	3	(400,400)	5	20	154600	160000	160000	4,12	0,38
Jakobs[13]	1	(70,80)	14	20	5447	5600	5600	10,16	1,89
	2	(70,80)	16	25	5455	5540	5600	15,44	16,88
	3	(120,45)	22	25	5328	5400	5400	12,57	0,42
	4	(90,45)	16	30	3978	4050	4050	10,28	1,97
	5	(65,45)	18	30	2871	2925	2925	14,94	1,53
Leung et al.[17]	1	(150,110)	40	40	15856	16280	16500	90,52	52,36
	2	(160,120)	50	50	18628	19044	19200	132,26	63,95
Hopper and Turton[12]	1-1	(20,20)	16	16	400	400	400	0,94	0,42
	1-2	(20,20)	17	17	386	400	400	9,28	4,23
	1-3	(20,20)	16	16	400	400	400	0,06	0,95
	2-1	(40,15)	25	25	590	600	600	19,44	0,44
	2-2	(40,15)	25	25	597	600	600	17,36	4,16
	2-3	(40,15)	25	25	600	600	600	0,71	0,00
	3-1	(60,30)	28	28	1765	1800	1800	26,80	4,91
	3-2	(60,30)	29	29	1755	1800	1800	37,35	10,11
	3-3	(60,30)	28	28	1774	1800	1800	30,92	5,52
	4-1	(60,60)	49	49	3528	3580	3600	102,05	45,27
	4-2	(60,60)	49	49	3524	3564	3600	110,79	68,59
	4-3	(60,60)	49	49	3544	3580	3600	94,41	51,11
	5-1	(60,90)	73	73	5308	5342	5400	212,07	135,97
	5-2	(60,90)	73	73	5313	5361	5400	231,56	96,80
	5-3	(60,90)	73	73	5312	5375	5400	231,24	82,06
	6-1	(80,120)	97	97	9470	9548	9600	480,44	240,39
	6-2	(80,120)	97	97	9453	9448	9600	465,49	399,86
	6-3	(80,120)	97	97	9450	9565	9600	478,02	206,78
	7-1	(160,240)	196	196	37661	38026	38400	3760,14	3054,38
	7-2	(160,240)	197	197	37939	38145	38400	2841,96	1990,70
	7-3	(160,240)	196	196	37745	37867	38400	3700,99	5615,75
Mean percentage of deviation from optimum					1,68%	0,42%		423,95	392,19
Number of optimal solutions (out of 31)					5	16			

Table 4: *Computational results – Zero-waste problems*

Source of problem	I	Problem size (L,W)	m	M	Beasley's solution	GRASP solution	TABU solution	Upper bound	CPU time (seconds) Beasley	GRASP	TABU
Beasley [4]	1	(10, 10)	5	10	164	164	164	164	0,02	0,00	0,00
	2	(10, 10)	7	17	225	225	225	230	5,53	0,71	1,70
	3	(10, 10)	10	21	220	220	220	247	7,85	1,21	2,26
	4	(15, 10)	5	7	268	268	268	268	0,01	0,00	0,00
	5	(15, 10)	7	14	301	301	301	358	5,05	0,72	1,48
	6	(15, 10)	10	15	265	252	265	289	6,81	1,81	1,59
	7	(20, 20)	5	8	430	430	430	430	0,01	0,00	0,00
	8	(20, 20)	7	13	819	819	819	834	6,54	1,32	1,76
	9	(20, 20)	10	18	924	924	924	924	5,64	0,00	0,00
	10	(30, 30)	5	13	n/f	n/f	n/f	n/f	2,38	0,22	0,94
	11	(30, 30)	7	15	1505	1518	1518	1688	2,96	1,59	2,52
	12	(30, 30)	10	22	1666	1648	1672	1865	3,78	1,65	3,73
Hadjiconstantinou	3	(30, 30)	7	7	1178	1178	1178	1178	0,25	0,00	0,00
and Christofides [10]	11	(30, 30)	15	15	1216	1216	1216	1270	2,60	2,08	3,18
Wang[21]		(70, 40)	19	42	2499	2700	2716	2726	6,36	1,48	6,16
Christofides and Whitlock [7]	3	(40, 70)	20	62	1600	1720	1720	1860	6,81	0,88	5,27
Fekete and Scheppers [8]	1	(100, 100)	15	50	25373	24869	25384	27718	11,86	3,73	25,27
	2	(100, 100)	30	30	17789	19083	19657	22502	5,80	3,02	18,35
	3	(100, 100)	30	30	n/f	n/f	n/f	n/f	4,03	0,66	12,41
	4	(100, 100)	33	61	27556	27898	28974	32893	20,42	2,80	37,46
	5	(100, 100)	29	97	21997	22011	22011	27923	18,41	3,30	61,90
Mean percentage of deviation from upper bound					8,11%	7,36%	6,62%		5.86	1.29	8.86

n/f: No feasible solution found

Table 5: *Computational results – Doubly constrained problems*

Acknowledgements

This work has been partially supported by Project PBC-02-002, Consejeria de Ciencia y Tecnologia, JCCM, the Spanish Ministry of Science and Technology DPI2002-02553, and the Valencian Science and Technology Agency, GRUPOS03/174.

References

[1] R. Alvarez-Valdes, F. Parreño, J.M. Tamarit, A GRASP algorithm for constrained two-dimensional non-guillotine cutting problems, Journal of Operational Research Society (2004), in press.

[2] A. Amaral, A. Letchford, An improved upper bound for the two-dimensional non-guillotine cutting problem, Working paper available from the second author at Department of Management Science, Management School, Lancaster University, Lancaster LA1 4YW, England, 2003.

[3] M. Arenales, R. Morabito, An AND/OR-graph approach to the solution of two-dimensional non-guillotine cutting problems, European Journal of Operational Research 84 (1995) 599-617.

[4] J.E. Beasley, An exact two-dimensional non-guillotine cutting tree search procedure, Operations Research 33 (1985) 49-64.

[5] J.E. Beasley, A population heuristic for constrained two-dimensional non-guillotine cutting, European Journal of Operational Research 156 (2004) 601-627.

[6] A. Caprara, M. Monaci, On the two-dimensional Knapsack problem, Operations Research Letters 32 (2004) 5-14.

[7] N. Christofides, C. Whitlock, An algorithm for two-dimensional cutting problems, Operations Research 25 (1977) 30-44.

[8] S.P. Fekete, J. Schepers, On more-dimensional packing III: Exact Algorithms, Report 97290 available from the first author at Department of Mathematics, Technical University of Berlin, Germany (1997), revised (2000).

[9] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Boston, 1997.

[10] E. Hadjiconstantinou, N. Christofides, An exact algorithm for general, orthogonal, two-dimensional knapsack problems, European Journal of Operational Research 83 (1995) 39-56.

[11] P. Healy, M. Creavin, A. Kuusik, An optimal algorithm for placement rectangle, Operations Research Letters 24 (1999) 73-80.

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

[13] S. Jakobs, On genetic algorithms for the packing of polygons, European Journal of Operational Research 88 (1996) 165-181.

[14] K.K. Lai, J.W.M. Chan, Developing a simulated annealing algorithm for the cutting stock problem, Computers and Industrial Engineering 32 (1997) 115-127.

[15] K.K. Lai, J.W.M. Chan, A evolutionary algorithm for the rectangular cutting stock problem, International Journal of Industrial Engineering 4 (1997) 130-139.

[16] T.W. Leung, C.H. Yung, M.D. Troutt, Applications of genetic search and simulated annealing to the two-dimensional non-guillotine cutting stock problem, Computers and Industrial Engineering 40 (2001) 201-214.

[17] T.W. Leung, C.H. Yung, 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 (2003) 530-542.

[18] G. Scheithauer, J. Terno, Modeling of packing problems, Optimization 28 (1993) 63-84.

[19] G. Scheithauer, LP-based bounds for the Container and Multi-Container Loading Problem, International Transactions in Operations Research 6 (1999) 199-213.

[20] R.D. Tsai, E.M. Malstrom, H.D. Meeks, A two-dimensional palletizing procedure for warehouse loading operations, IIE Transactions 20 (1988) 418-425.

[21] P.Y. Wang, Two algorithms for constrained two-dimensional cutting stock problems, Operations Research 31 (1983) 573-586.

[22] Y.L. Wu, W. Huang, S.C. Lau, C.K. Wong, G.H. Young, An effective quasi-human based heuristic for solving rectangle packing problem, European Journal of Operational Research 141 (2002) 341-358.