A GRASP algorithm for constrained 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

This paper presents a greedy randomized adaptive search procedure (GRASP) for the constrained two-dimensional non-guillotine cutting problem, the problem of cutting the rectangular pieces from a large rectangle so as to maximize the value of the pieces cut. We investigate several strategies for the constructive and improvement phases and several choices for critical search parameters. We perform extensive computational experiments with well known instances previously reported, first to select the best alternatives and then to compare the efficiency of our algorithm with other procedures.

Keywords: Non-guillotine cutting; heuristics; GRASP

1 Introduction

The constrained two-dimensional non-guillotine cutting problem consists of cutting pieces of dimensions (l_i, w_i), $i = 1, \ldots, m$, from a large stock rectangle of dimensions (L, W). Each piece has a fixed orientation and must be cut with its edges parallel to the edges of the stock rectangle. The number of pieces of each type i that are cut must be between the limits P_i, Q_i, with $0 \leq P_i \leq Q_i$. Each piece has an associated value v_i and the objective is to maximize the total values of the pieces cut.

In accordance 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:* $\exists\, i,\ P_i > 0;\ \exists\, j,\ Q_j < \lfloor L * W/l_j * w_j \rfloor$

We define the *efficiency* of a piece i, as $e_i = v_i/(l_i * w_i)$. According to this efficiency, we distinguish two types of problems:

1. *Unweighted:* $e_i = 1, \forall i$. The value of each piece is equal to its area.

2. *Weighted:* $e_i \neq 1$, at least for one i. Some pieces have a value which does not correspond to their surface, reflecting other aspects such as their shape or their relative importance for customers.

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

Some authors have considered the unconstrained problem: Tsai et al.[22], Arenales and Morabito[2], Healy et al.[12]. Nevertheless, the constrained problems are more interesting for applications and more research has been devoted to these. Some exact methods have been proposed by Beasley[3], Scheithauer and Terno[20], Hadjiconstantinou and Christofides[11], Fekete and Schepers[9], 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 to be cut in excess of its lower bound P_i:

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

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

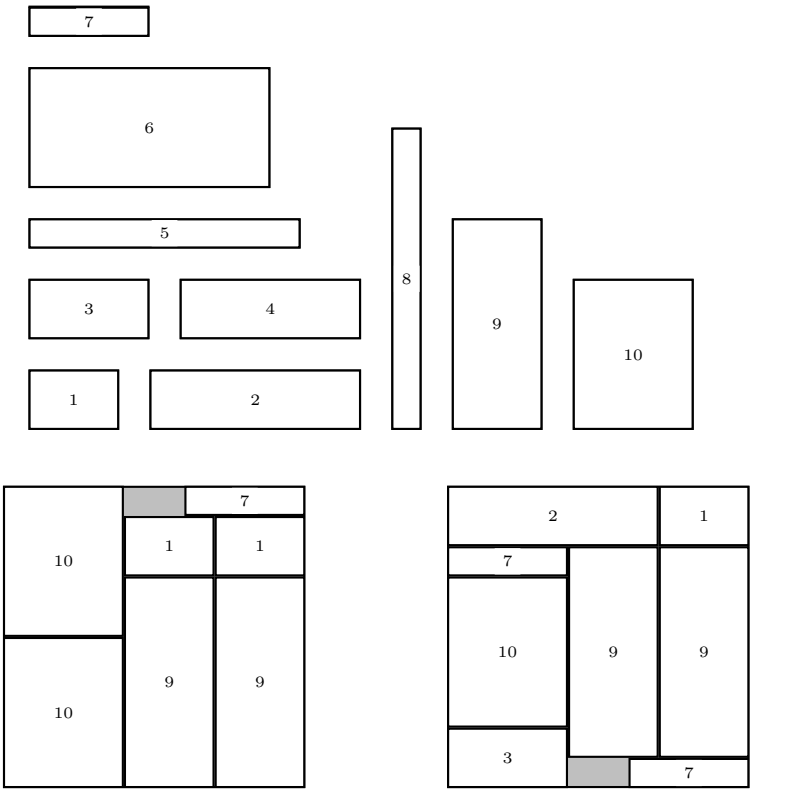
$$x_i \leq Q_i - P_i, \quad i = 1, \ldots, m \tag{3}$$

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

Other bounds, apart from those included in the exact methods mentioned above, have been proposed by Scheithauer and Terno[21], Amaral and Wright[1].

Several heuristic algorithms have been proposed recently. Wu et al.[24] develop a constructive algorithm for the special case where $P_i = Q_i\ \forall i$, in which at each step a piece is cut in a corner of the current cutting pattern, and the piece to be cut is decided according to a fitness evaluation function which estimates the quality of the solution that would be obtained if the piece were to be cut. Lai and Chan[14], [15], and Leung el al.[16], [17] propose simulated annealing and genetic algorithms

Stock rectangle : $L = 10$, $W = 10$						
Piece	l_i	w_i	P_i	Q_i	v_i	e_i
1	3	2	1	2	7	1,166
2	7	2	1	3	20	1,428
3	4	2	1	2	11	1,375
4	6	2	0	3	13	1,083
5	9	1	0	2	21	2,333
6	8	4	0	1	79	2,468
7	4	1	1	2	9	2,25
8	1	10	0	1	14	1,4
9	3	7	0	3	52	2,476
10	4	5	0	2	60	3



(b) Constrained. Optimum: 247 (c) Doubly const. Optimum: 220

Figure 1: *Instance 3, Table 6*

in which each solution is given by an ordered list of pieces, and the list is translated into a cutting pattern by a placement algorithm, either the bottom-left algorithm or the difference algorithm [14]. Combinations and changes of solutions are made on the ordered lists. Finally, Beasley [4] develops a genetic algorithm based on a non-linear formulation of the problem. He presents a complete computational study on a set of standard test problems and on a number of large, randomly generated problems.

In this paper, we present a new GRASP algorithm for the two-dimensional non-guillotine cutting problem. We provide computational results obtained on three sets of test problems: the 21 problems from the literature collected by Beasley [4]; the 630 large random problems also generated by Beasley [4]; and 11 problems used by Leung et al. [17].

2 A constructive algorithm

We follow 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. We present first a scheme of the algorithm and then a detailed description of its components.

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.
$\mathcal{C} = \emptyset$, the set of pieces cut.

Step 1. Choosing the rectangle:

Take R^*, the smallest rectangle of $\mathcal{L}$ in which a piece of $\mathcal{P}$ 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 the block b^*, to be cut in R^*.
Choose a position in R^* to cut b^*.
Update $\mathcal{P}$, $\mathcal{C}$ 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.

Now we give a more detailed description of the algorithm steps:

1. *Step 0: Initialization*

   The list $\mathcal{P}$ of pieces to be cut is initially ordered according to 3 criteria:

   (a) Order by $P_i * l_i * w_i$, giving priority to pieces which must be cut.

   (b) If there is a tie in (a) (for instance, if $P_i = 0, \forall i$), order by e_i

   (c) If there is a tie in (b) (for instance, if $e_i = 1, \forall i$), order by $l_i * w_i$.

2. *Step 1: Choosing the rectangle in $\mathcal{L}$*

   We take the smallest rectangle of $\mathcal{L}$, breaking the ties by the nearest distance to a corner of the stock rectangle. The reason behind this decision is to try to satisfy the demand for small pieces with small rectangles, leaving large rectangles for large pieces. If we take a large rectangle at the beginning and use it to cut a small piece, the empty rectangles resulting from the cutting may be useless for the large pieces still to be cut.

3. *Step 2: Choosing the pieces to cut*

   (a) *Choose a piece p_i and a quantity $n_i \leq Q_i$, forming the block b^**

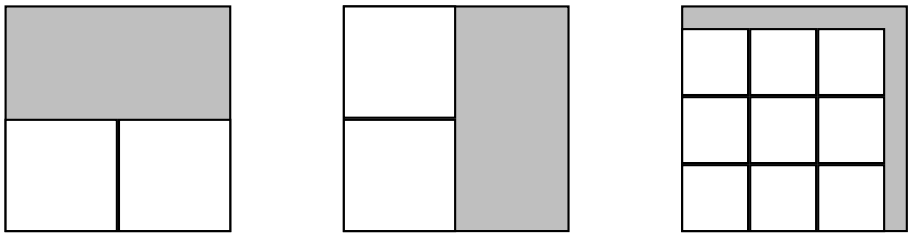
   Once a rectangle R^* has been chosen, we consider the pieces i of $\mathcal{P}$ fitting in R^* in order to choose which one to cut.

   If $Q_i > 1$, we consider the possibility of cutting a *block*, several copies of the piece arranged in rows and columns, such that the number of pieces in the block does not exceed Q_i. Figure 2 shows examples of this situation.

   Two criteria have been considered to select the piece:

   i. The first piece in the ordered list $\mathcal{P}$.
   This is the most reasonable way to choose the piece, according to the order described above.

   ii. The piece producing the largest increase in the objective function.
   This is a more greedy selection in which the rectangle is filled with the block producing the largest benefit, irrespective of its efficiency.



(a) Piece 5x5, Q=2 (b) Piece 5x5, Q=2 (c) Piece 3x3, Q=9

Figure 2: *Several alternatives for cutting pieces. Rectangle 10 × 10*

(b) *Choose a position in R^* to cut b^**

Usually the block b^* to be cut does not completely fill rectangle R^* (see Figure 2). The block is cut in the corner of R^* which is nearest to a corner of the stock rectangle.

(c) *Update $\mathcal{P}$, $\mathcal{C}$ and Q_i*

Update $\mathcal{C}$ with the type i and the number n_i of pieces cut.

Make $Q_i = Q_i - n_i$

If $Q_i = 0$, remove piece i from the list $\mathcal{P}$

(d) *Move block b^* towards the nearest corner of the stock rectangle*

The block b^* is moved to the nearest corner of the stock rectangle. If that is the case, it goes outside the rectangle R^* in which it was cut and comes totally or partially into another empty rectangle if it is located nearer to the corner of the stock rectangle. In that way, the empty regions of the current cutting pattern are concentrated towards the center and can be more easily merged at Step 3.

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

(a) *Add to $\mathcal{L}$ the new rectangles*

Unless block b^* fits exactly in rectangle R^* and it does not move, cutting and moving b^* produce new empty rectangles which are added to the list $\mathcal{L}$.

(b) *Merge rectangles to favor cutting new pieces of $\mathcal{P}$*

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 3). 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:

i. 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.

ii. 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.

iii. 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.

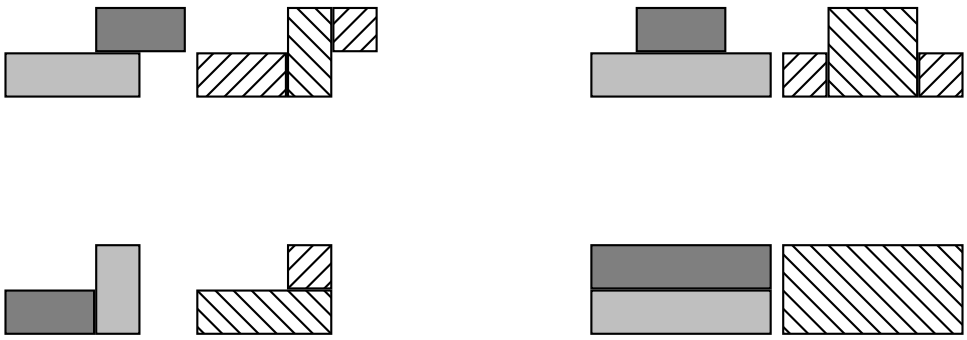


Figure 3: *Merging 2 empty rectangles*

We finish the description of the constructive algorithm by applying it to the constrained instance Ap3 of Figure 1 with $P_i = 0\ \forall i$. We denote each rectangle by a quadruple (x_1, y_1, x_2, y_2), where (x_1, y_1) are the coordinates of its lower left corner and (x_2, y_2) the upper right corner. The process can be followed in Figure 4.

Initially, $\mathcal{L} = \{R\} = \{(0, 0, 10, 10)\}$ and $\mathcal{P} = \{10, 9, 6, 5, 7, 2, 8, 3, 1, 4\}$, in order of efficiency. We consider the first piece of the list, piece 10, with $Q_{10} = 2$

and try to cut it twice from the first rectangle of $\mathcal{L}$. We then cut the block composed of 2 copies of piece 10 in the lower left corner of the rectangle. We update $\mathcal{P} = \{9, 6, 5, 7, 2, 8, 3, 1, 4\}$ and $\mathcal{L} = \{(4, 0, 10, 10)\}$. We try to cut piece 9 with $Q_9 = 3$, but only 2 copies of it fit into the rectangle. We cut the block at the lower right corner. We update $\mathcal{P} = \{9, 6, 5, 7, 2, 8, 3, 1, 4\}$, $Q_9 = 1$, $\mathcal{L} = \{(4, 7, 10, 10)\}$. Pieces 9, 6, 5 are considered in turn but they do not fit into the rectangle and piece 7 with $Q_7 = 2$ is taken. A block with 2 copies is cut at the bottom right corner. We update $\mathcal{P} = \{9, 6, 5, 2, 8, 3, 1, 4\}$, $\mathcal{L} = \{(4, 7, 6, 10), (6, 7, 10, 8)\}$. None of the pieces fit into these rectangles. Before finishing the process, we try to merge the rectangles in an attempt to produce some other shapes which could accommodate the pieces. In this case, the only alternative would be to produce the pair of rectangles (4,7,10,8) and (5,7,6,10) but it would not allow us to cut any new piece and the process ends.

Figure 4: *Instance 3, Table 6. Selecting most efficient piece*

3 A GRASP algorithm

The GRASP algorithm was developed by Feo and Resende [10] to solve hard combinatorial problems. For an updated introduction, refer to Resende and

Ribeiro[19]. GRASP is an iterative procedure combining a constructive phase and an improvement phase. In the constructive phase a solution is built step by step, adding elements to a partial solution. In order to choose the element to be added, a greedy function, which is dynamically adapted as the partial solution is built, is computed. However, the selection of the element is not deterministic but subjected to a randomization process. In that way when we repeat the process we can obtain different solutions. After each constructive phase, the improvement phase, usually consisting of a simple local search, tries to substitute some elements of the solution which are there as the result of the randomization, by others, thereby producing a better overall solution.

3.1 The constructive phase

In our algorithm the constructive phase corresponds to the constructive algorithm described in Section 2, introducing randomization procedures when selecting the piece to cut. Let s_i be the score of piece i on the selection criterion we are using and $s_{max} = max\{s_i | i \in \mathcal{P}\}$, and let δ be a parameter to be determined $(0 < \delta < 1)$. We have considered three alternatives:

1. Select piece i at random in set $S = \{j | s_j \geq \delta s_{max}\}$
   (S is commonly called a *Restricted Set of Candidates*).
2. Select piece i at random from among the best $100\,(1-\delta)\%$ of the pieces, irrespective of their score.
3. Select piece i from among the whole set $\mathcal{P}$ but with probability proportional to its score s_i $(p_i = s_i / \Sigma s_j)$

3.2 Determining the parameter δ

A preliminary computational experience showed that no value of δ always produced the best results. We therefore considered several strategies which basically consisted of changing the value of δ randomly or systematically along the iterations. These strategies were:

1. At each iteration, choose δ at random from the interval $[0.4, 0.9]$
2. At each iteration, choose δ at random from the interval $[0.25, 0.75]$
3. At each iteration δ takes in turn one of these 5 values: 0.4,0.5,0.6,0.7,0.8,0.9.
4. $\delta = 0.75$

5. *Reactive GRASP*

   In Reactive GRASP, proposed by Prais and Ribeiro [18], δ is initially taken at random from a set of discrete values, but after a certain number of iterations the relative quality of the solutions obtained with each value of δ is taken into account and the probability of values consistently producing better solutions is increased. The procedure is described in Figure 5, following Delorme et al.[8]. The value of α is fixed to 10, as in [18].

3.3 Improvement phase

Each solution built at the constructive phase is the starting point for a local search procedure in which we try to improve the solution. We have studied three alternatives:

I) We take a block adjacent to an empty rectangle and consider reducing or completely eliminating that block. The remaining pieces are then moved to the corners, the empty rectangles are merged and the resulting list of empty rectangles is filled by applying the constructive algorithm (Figure 6). If the resulting solution improves the initial solution, the move is made and the process goes on studying a new block. Note that we only consider reducing a block while it does not violate the lower bounds P_i on the number of pieces to be cut.

II) This second procedure is a simplification of method I in which pieces are not moved to the corners and the new empty rectangles are only merged with existing adjacent empty rectangles (Figure 7).

III) The third method consists of eliminating the final $k\%$ blocks of the solution (for instance, the last 10%) and filling the empty space with the deterministic constructive algorithm, as proposed by Beltran et al.[5]. Once the final pieces have been removed from the solution, the remaining pieces are moved to the corners, the empty rectangles are merged and the constructive algorithm is then applied (see Figure 8, in which the numbers in the pieces reflect the order in which they were included in the cutting pattern).

3.4 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.

Initialization:

$\mathcal{D} = \{0.1, 0.2, \ldots, 0.9\}$, set of possible values for δ

$S_{best} = 0; \quad S_{worst} = \infty$

$n_{\delta^*} = 0$, number of iterations with δ^*, $\forall\, \delta^* \in \mathcal{D}$.

$Sum_{\delta^*} = 0$, sum of values of solutions obtained with δ^*.

$P(\delta = \delta^*) = p_{\,\delta^*} = 1/|\mathcal{D}|\,, \forall\, \delta^* \in \mathcal{D}$

$numIter = 0$

While $(numIter < maxIter)$

{

Choose δ^* from $\mathcal{D}$ with probability $p_{\,\delta}$.

$n_{\delta^*} = n_{\delta^*} + 1$

$numIter = numIter + 1$

Apply Constructive Phase with δ^*obtaining solution S

Apply Improvement Phase obtaining solution S'

If $S' > S_{best}$ then $S_{best} = S'$.

If $S' < S_{worst}$ then $S_{worst} = S'$

$Sum_{\delta^*} = Sum_{\delta^*} + S'$

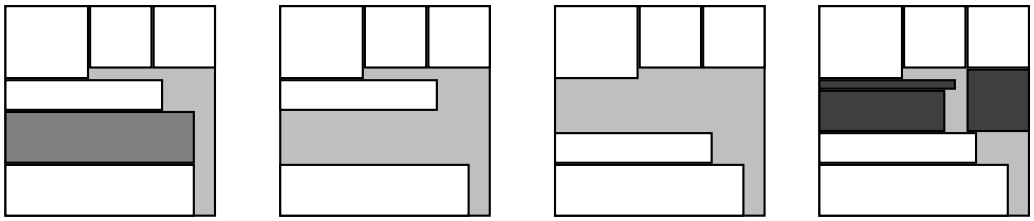
If $mod(numIter, 100) == 0$ (every 100 iterations):

$$eval_\delta = \left(\frac{mean_\delta - S_{worst}}{S_{best} - S_{worst}}\right)^\alpha \quad \forall \delta \in \mathcal{D}$$

$$p_{\,\delta} = \frac{eval_\delta}{\left(\sum_{\delta' \in \mathcal{D}} eval_{\delta'}\right)} \quad \forall \delta \in \mathcal{D}$$

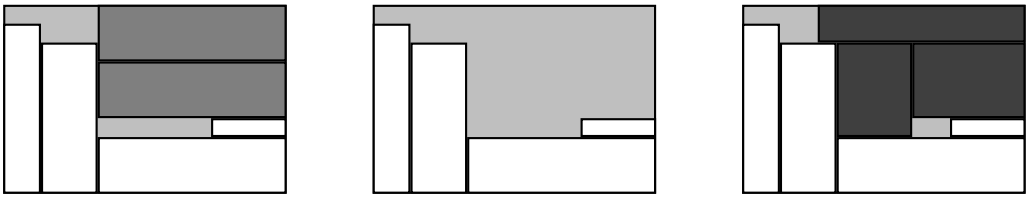
}

Figure 5: *Reactive Grasp*



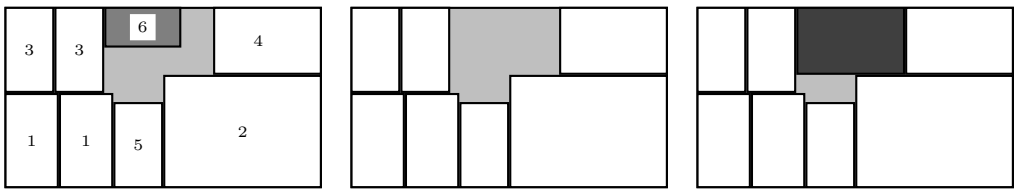
(a) Selecting (b) Reducing (c) Moving (d) Filling

Figure 6: *Improvement method I. Instance 8, Table 6*



(a) Selecting (b) Reducing (c) Filling

Figure 7: *Improvement method II. Instance 6, Table 6*



(a) Selecting (b) Removing (c) Filling

Figure 8: *Improvement method III. Instance 15, Table 6*

- *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:

$$max\, t: \quad total_{pieces} - t * v_i > v_{best}\,; \quad t \geq 0,\, t \leq Q_i - P_i \tag{5}$$

  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 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:

$$(Q_i * l_i * w_i * (e_{max} - e_i) \geq e_{max} * (L * W - R) + R_v - v_{best} \tag{6}$$

  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:

$$max\, t: \quad t*l_i*w_i*(e_{max}-e_i) < (e_{max}*(L*W-R)+R_v-v_{best} \quad t \geq 0,\, t \leq Q_i-P_i \tag{7}$$

  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.

4 Computational results

4.1 Test problems

We have used several sets of test problems:

1. A set of 21 problems from the literature: 15 from Beasley [3], 2 from Hadjiconstaninou and Christofides[11], 1 from Wang[23], 1 from Christofides and Whitlock[7], 5 from Fekete and Schepers[9]. For all of them the optimal solutions are known. They have also been solved by Beasley [4].

2. A set of 630 large problems generated by Beasley[4], following the work by Fekete and Schepers[9]. All the problems have a stock rectangle of size $(100, 100)$. For each value of m, the 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[4] 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]. 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] set of problems because it has characteristics which can be considered as 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 form part of the solution. However, Leung's problems are *jigsaw* problems. Almost all the available pieces will form 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 as 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
Leung	1,01	1,01

Table 1: *Test problems – Characteristics.*

4.2 Choosing the best strategies

We have used sets 1, 2, and 4 of the test problems described in the previous subsection, which will be denoted as *Literature*, *Large* and *Leung et al.* respectively.

4.2.1 Selection of the piece to cut

In the constructive phase (*Step 2*), we have considered 2 criteria for choosing the piece to cut:

- *Efficiency*: Choose the most efficient piece.
- *Value:* Choose the piece which will produce the largest increase in the value of the objective function.

The results appear in Table 2. Apart from the Leung et al. problems in which the results obviously coincide because the efficiency of all pieces is 1, the results for the *efficiency* criterion are better than those for the *value* criterion.

	Literature		Large		Leung et al.	
	% Mean deviation from opt.	Number of optimal solutions	% Mean deviation from bound	Number of optimal solutions	% Mean deviation from opt.	Number of optimal solutions
Efficiency	8,850	6	2,929	28	8,080	1
Value	10,172	7	3,837	28	8,080	1

Table 2: *Selection of the piece*

4.2.2 Randomization procedures

In this study we have kept both criteria of selection of the piece to cut because it is possible that a criterion which is not the best when used in a deterministic way may work better in a randomized structure.

We have considered 3 alternatives to randomize the selection of the piece:

1. *RCL-Value*: Select piece i at random in set $S = \{j | s_j \geq \delta s_{max}\}$
2. *RCL-Percentage*: Select piece i at random from among the best $100\,\delta\%$ of the pieces.
3. *Biased*: Select piece i from among the whole set $\mathcal{P}$ but with probability proportional to its score s_i ($p_i = s_i / \Sigma s_j$).

We have also considered the possibility of randomizing the selection of the rectangle from which to cut the pieces at each step. Instead of taking the smallest rectangle of the list $\mathcal{L}$, we take it at random. The results of these alternatives appear in Table 3, where δ=0.5 and $NumIter = 1000$.

4.2.3 Choosing the parameter δ

We have studied several alternatives to choose parameter δ.

- At each iteration, choose δ at random from the interval $[0.4, 0.9]$
- At each iteration, choose δ at random from the interval $[0.25, 0.75]$
- At each iteration δ takes in turn one of these 5 values: 0.4,0.5,0.6,0.7,0.8,0.9.
- $\delta = 0.75$
- *Reactive GRASP*

The results appear in Table 4 in which it can be seen that Reactive GRASP obtains slightly better results than the random selection in $[0.25, 0.75]$.

		Literature		Large		Leung et al.	
		% Mean deviation from opt.	Number of optimal solutions	% Mean deviation from bound	Number of optimal solutions	% Mean deviation from opt.	Number of optimal solutions
Taking smallest rectangle							
Efficiency	RCL-Value	1,594	13	2,649	14	5,809	0
	RCL-Percentage	0,782	13	1,987	19	2,880	0
	Biased	0,795	11	2,533	37	5,296	3
Value	RCL-Value	0,874	12	1,368	142	4,154	1
	RCL-Percentage	0,835	15	3,076	26	3,430	2
	Biased	0,800	14	1,900	47	4,460	2
Taking rectangle at random							
Efficiency	RCL-Value	1,212	13	2,492	13	6,282	1
	RCL-Percentage	0,823	11	1,882	11	5,748	1
	Biased	1,027	15	2,445	15	3,727	1
Value	RCL-Value	0,597	14	1,223	157	3,999	2
	RCL-Percentage	1,163	15	2,992	16	3,561	2
	Biased	0,591	15	1,678	58	3,949	2

Table 3: *Randomization procedures*

	Literature		Large		Leung et al.	
	% Mean deviation from opt.	Number of optimal solutions	% Mean deviation from bound	Number of optimal solutions	% Mean deviation from opt.	Number optimal solutions
Random in [0.4,0,9]	0,428	15	1,195	177	3,207	2
Random in [0.25,0.75]	0,334	15	1,166	179	3,618	2
Deterministic from 0.4 to 0.9	0,498	14	1,254	162	3,019	3
Fixed to 0,75	1,647	11	1,658	168	3,214	2
Reactive GRASP	0,216	17	1,194	194	3,061	2

Table 4: *Study of parameter δ*

4.2.4 Improvement phase

We have tried three improvement methods:

- Method I, based on a complex move.
- Method II, based on a simplified move.
- Method III, removing the last 10% of pieces and filling the empty space.

Note that Method I requires much more time than the other two methods. Therefore, the iteration limit for them is higher in order to compare the results over similar times. The results appear in Table 5.

	Literature		Large		Leung et al.	
	% Mean deviation from opt.	Number of optimal solution	% Mean deviation from bound	Number of optimal solutions	% Mean deviation from opt.	Number optimal solutions
Random in [0.25,0.75]						
Method I	0,279	15	1,098	194	2,801	2
Method II	0,205	17	1,055	220	2,235	2
Method III	0,205	17	1,054	224	2,173	2
Reactive GRASP						
Method I	0,239	17	1,118	197	2,247	2
Method II	0,230	17	1,077	217	2,077	2
Method III	0,188	18	1,072	210	2,054	2

Table 5: *Improvement methods*

4.3 Complete computational results

As a consequence of the results obtained in the previous subsection, the complete GRASP algorithm will use the following strategies:

- Selection of the piece: Largest increase in the objective function.
- Selection of the rectangle: Random.
- Randomization procedure: Restricted Candidate List.
- Selection of δ: Reactive GRASP.
- Improvement phase: Method III.
- NumIter: 10000 iterations.

The complete computational results appear in Tables 6, 7 and 8. The first two Tables include a direct comparison with Beasley's [4] results in terms of the quality of the solutions. The computing times, however, cannot be directly compared. Our algorithm has been coded in *C++* and run on a *Pentium III* at *800 Mhz*, while 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. We can say therefore that our algorithm improves Beasley's results on every type of problem with much shorter computing times. Note that in both cases the algorithms stop when the optimal solution is discovered or after an iteration limit is reached (10000 iterations in our algorithm, 75000 children in Beasley's genetic algorithm).

A direct comparison with Leung et al.[17] is not possible. 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 they 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 similar to theirs. We can also point out, as Beasley [4] 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.

Finally, Table 9 shows the results of the GRASP algorithm 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 adjustment of lower bounds does not have significant effects on the performance of the algorithms, but the adjustment of upper bounds has a dramatic effect, especially in the large random problems of the second test set in which there are important differences in the efficiencies 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.

5 Conclusions

We have developed a new heuristic algorithm based on GRASP techniques for the non-guillotine two-dimensional cutting stock problem. The constructive phase explicitly considers the possibility of simultaneously cutting several pieces of the same type, forming a *block*, an idea taken from the the Pallet Loading Problem. In this constructive phase the algorithm maintains a list of empty rectangles $\mathcal{L}$, a procedure commonly used in guillotine cutting problems and which is adapted

Source of problem	I	Size (L,W)	m	M	Constructive with efficiency	Constructive randomized	GRASP	Beasley's solution	Optimal solution	CPU time (seconds) GRASP	Beasley
Beasley [3]	1	(10, 10)	5	10	146	164	164	164	164	0,00	0,02
	2	(10, 10)	7	17	213	230	230	230	230	0,00	0,16
	3	(10, 10)	10	21	220	247	247	247	247	0,00	0,53
	4	(15, 10)	5	7	268	268	268	268	268	0,00	0,01
	5	(15, 10)	7	14	358	358	358	358	358	0,00	0,11
	6	(15, 10)	10	15	268	289	289	289	289	0,00	0,43
	7	(20, 20)	5	8	430	430	430	430	430	0,00	0,01
	8	(20, 20)	7	13	753	831	834	834	834	0,77	3,25
	9	(20, 20)	10	18	863	924	924	924	924	0,00	2,18
	10	(30, 30)	5	13	1452	1452	1452	1452	1452	0,00	0,03
	11	(30, 30)	7	15	1524	1688	1688	1688	1688	0,05	0,6
	12	(30, 30)	10	22	1389	1865	1865	1801	1865	0,05	3,48
Hadjiconstantinou	1	(30, 30)	7	7	1178	1178	1178	1178	1178	0,00	0,03
and Christofides [11]	2	(30, 30)	15	15	1270	1270	1270	1270	1270	0,00	0,04
Wang[23]	1	(70, 40)	19	42	2277	2726	2726	2721	2726	0,77	6,86
Christofides [7]	1	(40, 70)	20	62	1560	1860	1860	1720	1860	0,39	8,63
and Whitlock											
Fekete and Scheppers [9]	1	(100, 100)	15	50	18384	27589	27589	27486	27718	2,31	19,71
	2	(100, 100)	30	30	19790	21976	21976	21976	22502	4,17	13,19
	3	(100, 100)	30	30	23282	23743	23743	23743	24019	3,68	11,46
	4	(100, 100)	33	61	30197	32893	32893	31269	32893	0,00	32,08
	5	(100, 100)	29	97	25650	27923	27923	26332	27923	0,00	83,44
Mean percentage of deviation from optimum					8,85%	0,22%	0,19%	1,21%		0,58	8,87
Number of optimal solutions					6	17	18	13			

Table 6: *Computational results – Problems from literature*

Mean percentage of deviation from knapsack upper bound								
m	Q*	M	Constructive with efficiency	Constructive randomized	GRASP	Beasley's solution	CPU time (seconds) GRASP	Beasley
40	1	40	10,81	7,27	6,97	7,77	2,33	13,57
	3	120	5,47	2,55	2,22	3,54	6,62	47,43
	4	160	4,38	1,99	1,81	3,24	4,44	63,30
50	1	50	8,90	4,91	4,80	5,48	4,71	14,60
	3	150	3,69	1,69	1,50	2,35	7,05	59,27
	4	200	4,04	1,41	1,18	2,63	5,34	80,07
100	1	100	4,44	1,75	1,51	2,26	5,36	27,20
	3	300	2,24	0,60	0,47	1,27	9,41	119,47
	4	400	1,96	0,34	0,26	1,06	6,99	175,10
150	1	150	3,66	1,13	0,89	1,31	5,53	40,60
	3	450	1,61	0,21	0,14	0,60	11,71	190,53
	4	600	1,48	0,22	0,11	0,92	6,75	323,83
250	1	250	2,41	0,66	0,51	0,88	5,27	76,70
	3	750	1,07	0,12	0,04	0,57	13,89	439,47
	4	1000	1,01	0,04	0,03	0,39	6,65	693,67
500	1	500	0,91	0,09	0,05	0,26	3,24	203,10
	3	1500	0,62	0,01	0,00	0,18	12,24	1210,80
	4	2000	0,60	0,01	0,00	0,18	1,15	1790,83
1000	1	1000	0,85	0,03	0,00	0,09	1,01	667,23
	3	3000	0,74	0,00	0,00	0,07	6,53	3318,47
	4	4000	0,51	0,00	0,00	0,07	0,29	4840,57
Type 1			3,01	1,15	1,04	1,64	5,13	558,11
Type 2			2,88	1,25	1,14	1,70	5,90	668,41
Type 3			2,90	1,18	1,03	1,66	7,28	830,02
All			2,93	1,19	1,07	1,67	5,91	685,51

Table 7: *Computational results– Large random problems.*

Source of problem	I	Size of problem (L,W)	m	M	Constructive with efficiency	Constructive randomized	GRASP	Optimal solution	CPU time GRASP
Lai and Chan[14]	1	(400,200)	9	10	80000	80000	80000	80000	0,00
	2	(400,200)	7	15	75000	79000	79000	79000	0,00
	3	(400,400)	5	20	143500	149800	154600	160000	4,12
Jakobs[13]	1	(70,80)	14	20	4695	5400	5447	5600	10,16
	2	(70,80)	16	25	5055	5295	5455	5600	15,44
	3	(120,45)	22	25	4968	5310	5328	5400	12,57
	4	(90,45)	16	30	3717	3978	3978	4050	10,28
	5	(65,45)	18	30	2649	2844	2871	2925	14,94
Leung et al.[17]	1	(150,110)	40	40	15400	15668	15856	16500	90,52
	2	(160,120)	50	50	17816	18440	18628	19200	132,26
Mean percentage of deviation from optimum					8,08	3,06	2,05		29,03

Table 8: *Computational results – Leung et al. problems*

Source of problem	I	Size of problem (L,W)	m	M	Constructive with efficiency	Constructive randomized	GRASP	Beasley's solution	Upper bound	CPU time (seconds) GRASP	CPU time (seconds) Beasley
Beasley [3]	1	(10, 10)	5	10	156	164	164	164	164	0,00	0,02
	2	(10, 10)	7	17	n/f	225	225	225	230	0,71	5,53
	3	(10, 10)	10	21	176	220	220	220	247	1,21	7,85
	4	(15, 10)	5	7	268	268	268	268	268	0,00	0,01
	5	(15, 10)	7	14	301	301	301	301	358	0,72	5,05
	6	(15, 10)	10	15	229	249	252	265	289	1,81	6,81
	7	(20, 20)	5	8	430	430	430	430	430	0,00	0,01
	8	(20, 20)	7	13	712	819	819	819	834	1,32	6,54
	9	(20, 20)	10	18	552	924	924	924	924	0,00	5,64
	10	(30, 30)	5	13	n/f	n/f	n/f	n/f	n/f	0,22	2,38
	11	(30, 30)	7	15	1132	1518	1518	1505	1688	1,59	2,96
	12	(30, 30)	10	22	1443	1648	1648	1666	1865	1,65	3,78
Hadjiconstantinou	1	(30, 30)	7	7	1178	1178	1178	1178	1178	0,00	0,25
and Christofides [11]	2	(30, 30)	15	15	1216	1216	1216	1216	1270	2,08	2,6
Wang[23]	1	(70, 40)	19		2180	2587	2700	2499	2726	1,48	6,36
Christofides [7]	1	(40, 70)	20	62	1340	1720	1720	1600	1860	0,88	6,81
and Whitlock											
Fekete and Scheppers [9]	1	(100, 100)	15	50	n/f	24869	24869	25373	27718	3,73	11,86
	2	(100, 100)	30	30	n/f	18078	19083	17789	22502	3,02	5,8
	3	(100, 100)	30	30	n/f	n/f	n/f	n/f	n/f	0,66	4,03
	4	(100, 100)	33	61	25973	27665	27898	27556	32893	2,80	20,42
	5	(100, 100)	29	97	n/f	21899	22011	21997	27923	3,30	18,41
Mean percentage of deviation from optimum						7,93	7,36	8,11			

n/f: No feasible solution found

Table 9: *Computational results – Doubly constrained problems*

here at each step, merging the rectangles in the most convenient way to favor cutting the best pieces. The improvement phase corrects some of the decisions of the constructive phase, allowing the whole procedure to obtain a high quality solution in very short computing times.

The computational results show that these ideas work well for Beasley's [4] constrained and doubly constrained test problems. For Leung et al.[17] problems the results are also good and the proposed algorithm can be considered to work consistently well for a wide range of cutting problems.

References

[1] Amaral, A. and Letchford, A (2003) 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.

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

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

[4] Beasley, J.E. (2003) A population heuristic for constrained two-dimensional non-guillotine cutting, *European Journal of Operational Research, in press.*

[5] Beltrán, J.C., Calderón, J.E., Cabrera, R.J. and Moreno, J.M. (2002) Procedimientos constructivos adaptativos (GRASP) para el problema del empaquetado bidimensional, *Revista Iberoamericana de Inteligencia Artificial*, **15**, 26-33.

[6] Caprara, A. and Monaci, M. (2004) On the two-dimensional Knapsack problem, *Operations Research Letters* **32 (1)**, 5-14.

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

[8] Delorme, X. and Gandibleux, X. and Rodriguez, J. (2003) GRASP for set packing problems, *European Journal of Operational Research* **153 (3)**, 564-580.

[9] Fekete, S. P. and Schepers, J. (1997) On more-dimensional packing III: Exact Algorithms, *submitted to: Operations Research*

[10] Feo, T. and Resende, M.G.C. (1989) A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem, *Operations Research Letters* **8**, 67-71.

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

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

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

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

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

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

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

[18] Prais, M. and Ribeiro, C.C. (2000) Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment, *INFORMS Journal on Computing* **12**, 164-176.

[19] Resende, M.G.C and Ribeiro, C.C. (2003) Greedy Randomized Adaptive Search Procedures, in *Handbook of Metaheuristics*, F.Glover and G.Kochenberger, Eds., Kluwer Academic Publishers, pp. 219-249.

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

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

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

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

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