Reactive GRASP for the Strip Packing Problem

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

[†]University of Valencia, Department of Statistics and Operations Research, Burjassot, Valencia, Spain
[‡]University of Castilla-La Mancha. Department of Computer Science, Albacete, Spain

Abstract

This paper presents a greedy randomized adaptive search procedure (GRASP) for the strip packing problem, which is the problem of placing a set of rectangular pieces into a strip of a given width and infinite length so as to minimize the required length. 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 which have been previously reported, first to select the best alternatives and then to compare the efficiency of our algorithm with other procedures. The results show that the GRASP algorithm outperforms recently reported metaheuristics.

Keywords: Strip packing; non-guillotine cutting; heuristics; GRASP

1 Introduction

The *Two-Dimensional Strip Packing Problem* (2SP) consists of finding the best way of placing a given set of n rectangular pieces $i = 1, .., n$ of given heights and widths (h_i, w_i), without overlapping, into a strip of width W and infinite height so as to minimize the required height H. We assume that the pieces have a fixed orientation. An example proposed by Bengtsson[6] appears in Figure 1, in which 80 pieces have to be packed into a strip of width $W = 40$. The problem has many real-world applications in the paper, cardboard, glass and metal industries, whenever the stock to be cut comes in large rolls which can be considered to be of infinite length.

The problem is NP-hard in the strong sense because the strongly NP-hard one-dimensional bin-packing problem can easily be transformed into the 2SP problem. Therefore most research effort has been focused on developing heuristic algorithms for this problem. Nevertheless, Martello et al.[28] have recently proposed a branch

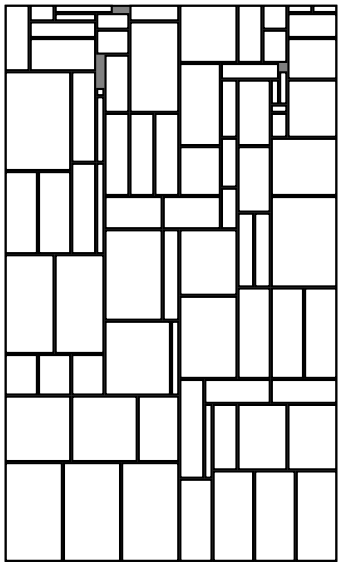


Figure 1: *Instance* beng07*, Table 10*

and bound algorithm which solves instances of up to 200 pieces. Simple lower bounds based on geometric considerations and an extremely good bound obtained from a relaxation of the problem are described and used in the exact procedure. Fekete and Schepers[15] have developed a framework for the exact solution of multi-dimensional packing problems. For the special case of perfect packings, Lesh et al.[23] have proposed another exact algorithm.

Many heuristic algorithms have been developed, ranging from simple constructive algorithms to complex metaheuristic procedures. The most used constructive approach is the Bottom-Left (BL) algorithm, proposed by Baker et al.[2] in 1980, later improved by Liu and Teng[26], and the Bottom-Left-Fill (BLF)algorithm by Chazelle[11]. In both procedures a list of rectangles is given and the rectangles are placed in turn into the strip as far down and to the left as possible. Many metaheuristic procedures use these algorithms, modifying at each iteration the list of rectangles to be provided to BL or BLF. Iori et al.[21] propose a tabu search algorithm, a genetic algorithm and a hybrid of both procedures. Genetic algorithms have also been developed by Jakobs[22], Gomez and De la Fuente[17], Liu and Teng[26], Leo and Wallace[25] and Yeung and Tang[34]. Lesh et al.[24] also use a version of the BL algorithm and modify the list of rectangles randomly according to a special probability distribution.

A completely different approach is proposed by the constructive algorithm Best-Fit (BF) of Burke et al.[9]. The order in which the rectangles are placed into the strip depends on the layout of the partial solution, and the rectangle fitting best into this layout is selected. Later, Burke et al.[10] improve their heuristic

by adding a metaheuristic phase in which Tabu Search, Simulated Annealing and Genetic Algorithms are used on the last part of the solution obtained by BF. Other metaheuristic algorithms working directly on the solution layouts are the genetic algorithm by Bortfeld[8] and the simulated annealing algorithm by Dowsland[14]. Following a completely different line, Zhang et al.[35] have recently proposed a fast recursive heuristic.

In this paper, we present a reactive GRASP algorithm for the strip packing problem and provide computational results obtained on several sets of test problems. This approach seems to be particularly adequate for the strip packing problem and, to the best of our knowledge, has only been used by Beltrán et al.[5]. In the constructive phase of GRASP, the algorithm selects the most promising rectangle according to the structure of the partial solution, in a similar way of to the BF algorithm, avoiding the rigidity of static lists. However, as no criterion will always provide the best rectangle to pack, the selection process is randomized. Later, the improving phase tries to correct some wrong decisions inherent to the randomized construction process. The remainder of the paper is organized as follows. In Section 2 we describe a constructive algorithm. Section 3 contains the GRASP algorithm and Section 4 describes the computational results. Finally, in Section 5 we draw some conclusions.

2 A constructive algorithm

The n pieces to pack are grouped into m types of pieces, of dimensions $(h_i, w_i), i = 1, .., m$, and we have to pack Q_i copies of each type i, with $\sum_{i=1}^{m} Q_i = n$.

We follow an iterative process in which we combine two elements: a list $\mathcal{P}$ of types of pieces still to be packed, initially the complete list of pieces, and a list $\mathcal{L}$ of empty rectangles of infinite height in which a piece can be packed, initially containing only the strip S of width W. The rectangles will be denoted by the pair $\{w_i, l_i\}$, where w_i is the width and l_i is the base level. At each step, we first choose a rectangle from $\mathcal{L}$ and then we choose the piece to be packed from among the pieces in $\mathcal{P}$ fitting into it. That usually produces new rectangles going into $\mathcal{L}$ and the process goes on until all the pieces have been packed.

- *Step 0: Initialization*

  $\mathcal{L} = \{S\}$, the set of empty rectangles.

  $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$, the set of piece types still to be packed,

  ordered by non-increasing w_i. Ties are broken by non-increasing h_i.

  Q_i is the number of pieces of type i to be packed.

  $\mathcal{C} = \emptyset$, the set of piece types i whose Q_i copies are already packed.

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

  Take from $\mathcal{L}$ a rectangle R^* of dimensions $\{w^*, l^*\}$,

  such that $l^* = min\{\, l_i \mid \{w_i, l_i\} \in \mathcal{L}\}$.

  Ties on rectangle lengths are broken by minimum w_i.

  If none of the pieces still to be packed can fit into R^*:

  - R^* is modified by lifting its bottom side to the minimal level l_i of its adjacent rectangles and then it is merged with the rectangle(s) of minimum level l_i.
  - That leaves a closed rectangle below which is considered waste.
  - Update the list of rectangles $\mathcal{L}$
  - Go back to select a new R^*

  Otherwise, go to Step 2

- *Step 2: Choosing the piece to pack*

  Once a rectangle R^* has been chosen, we consider the pieces i of $\mathcal{P}$ fitting into R^* in order to choose which one to pack. For each of these pieces i we compute $m_i = max\{m \in \mathcal{Z}^+ \mid m * w_i \leq w^*, m \leq Q_i\}$, the number of copies of piece i fitting into the width of R^*, and consider all possible blocks formed by $1, 2, .., m_i$ adjacent copies as alternatives to fill R^*. The height of a block composed of k copies of piece i is h_i and its width $k * w_i$. All these alternatives form the set $\mathcal{P}^*$. For the sake of simplicity, all the elements of $\mathcal{P}^*$ will be called *pieces* from this point on.

  Several criteria have been considered to select the piece to pack:

  1. $Max\{w_j\}$: Piece j with maximum width w_j, breaking ties by non-increasing h_j.
  2. $Max\{w_j + 0.1 * h_j\}$
  3. $Max\{w_j + 0.5 * h_j\}$
  4. *Best profile* : The piece j whose height h_j is the most similar to the difference between the base of R^* and the base of one of the adjacent rectangles. This criterion tries to leave as smooth as possible a profile after packing the piece.

  The first three criteria are based on the width, trying to fill the bottom of rectangle R^* as much as possible. Each one of them gives a different importance to the height of the pieces. The fourth criterion tries to maintain

a profile of the current solution which is as smooth as possible, avoiding peaks and troughs. However, all these criteria may delay the packing of tall pieces which will cause large increases in the required height H at the end of the process. In order to avoid this situation, we complement these criteria by computing a double estimation of the effect of not packing the tallest remaining piece.

When we select a piece, according to the chosen criterion, before packing it we check whether it is the highest remaining piece fitting in R^*. If that is the case, we place the piece into the strip. Otherwise, we do a double computation:

- We put the tallest piece j into the strip and see if that piece increases the current required height H. If it does, we determine the empty area, E, defined by the new height and compare it with the area of the pieces still to be packed, M, plus an estimation of the unavoidable waste involved in the process: $U = (W * LB - A)/4$, where LB is a lower bound on the required length and A is the total area of the pieces. If $E > M + U$, the tallest piece j is selected for packing and m_j copies of it are packed into R^*. Otherwise, we compute the second estimation.
- We put the selected piece into the strip and then we put the tallest piece j into one of the other rectangles of $\mathcal{L}$ or on top of the selected piece, wherever it produces the minimum required length. We repeat the argument of the first estimation, decide if the tallest piece j is preferred for packing and then pack m_j copies of it.

Figure 2 shows an example of this double estimation on the instance *ngcut11* whose data appear in Table 1, with $n = 15$ and $W = 30$. A partial solution with 6 pieces in white and a current required length of $H = 28$ is given in (a). If we use the first criterion, the widest remaining piece, piece 1 in dark gray, would be selected. However, the tallest piece is piece 2, with $h_2 = 29$. In the first estimation, the new required height is 52. If we put this piece in, the empty area in (b) is $E = 690$, while the area of the remaining pieces $M = 630$ and $U = (30 * 50 - 1483)/4 = 4.25$. Therefore, one copy of piece 2 is placed in the strip, as appears in (c), because $Q_2 = 1$. In the next step, the widest piece fitting in R^* is piece 4, but the tallest piece is piece 7. In the first estimation, in (d), if we put piece 7 in, the current required height is not increased and then piece 7 is not selected. However, in the second estimation, in (e), if we put piece 4 in first, we would have four rectangles in $\mathcal{L}$ and piece 7 would only fit on top of piece 4. The required length would

increase to $H = 61$, and $E = 605$, $M = 275$, $U = 4.25$. Therefore, piece 7 is selected at this step and, as $Q_7 = 1$, one copy of it is packed. The final solution appears in (h) with $H = 52$, which is the optimal solution.

$n = 15$, $W = 30$			
Piece	w_i	h_i	Q_i
1	23	3	3
2	5	29	1
3	2	21	3
4	11	17	3
5	7	14	2
6	5	8	2
7	8	21	1

Table 1: *Instance* ngcut11

- *Step 3: Choosing a position in R^* to pack the piece*

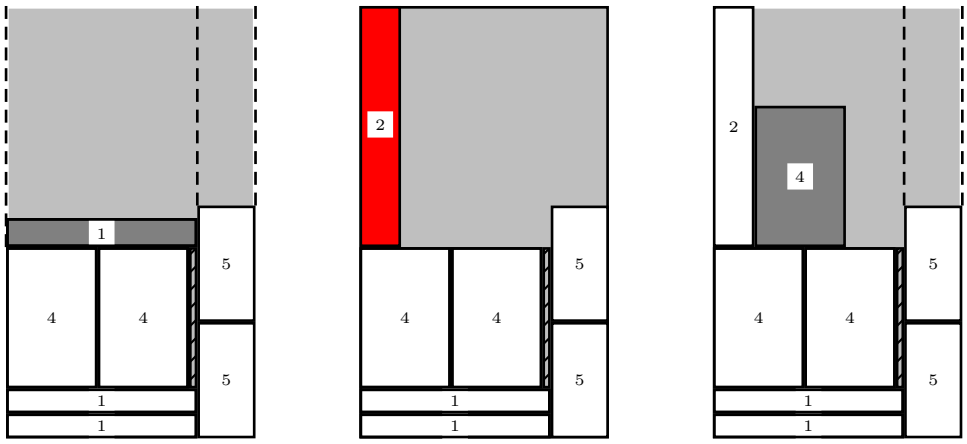
  Usually the piece to be packed does not completely fill rectangle R^*. Therefore we have to decide its position inside R^*. Obviously, the piece will be at the bottom of R^*, but its position on the left or right hand side of R^* has to be determined. Let us denote by R_l and R_r the rectangles of $\mathcal{L}$ adjacent to R^* on its left and on its right.

  1. If $R_l = \emptyset$ $(R_r = \emptyset)$, the piece goes on the left (right) hand side of R^*. If $R_l = R_r = \emptyset$, the piece is placed on the right hand side.
  2. Otherwise, we take into account l_l and l_r, the levels of R_l and R_r :
     - If $l^* + h_i = l_l$ $(l^* + h_i = l_r)$, the piece is placed on the left (right) hand side.
     - If $l_l = l_r$, the piece is placed as near as possible to one side of the strip.
     - Otherwise, the piece is put adjacent to the rectangle with the maximum base level.

- *Step 4: Updating the lists*

  Make $Q_i = Q_i - k$, where k is the number of copies of the piece i forming the block chosen to be packed. If $Q_i = 0$, remove piece type i from the list $\mathcal{P}$ and add it to $\mathcal{C}$.

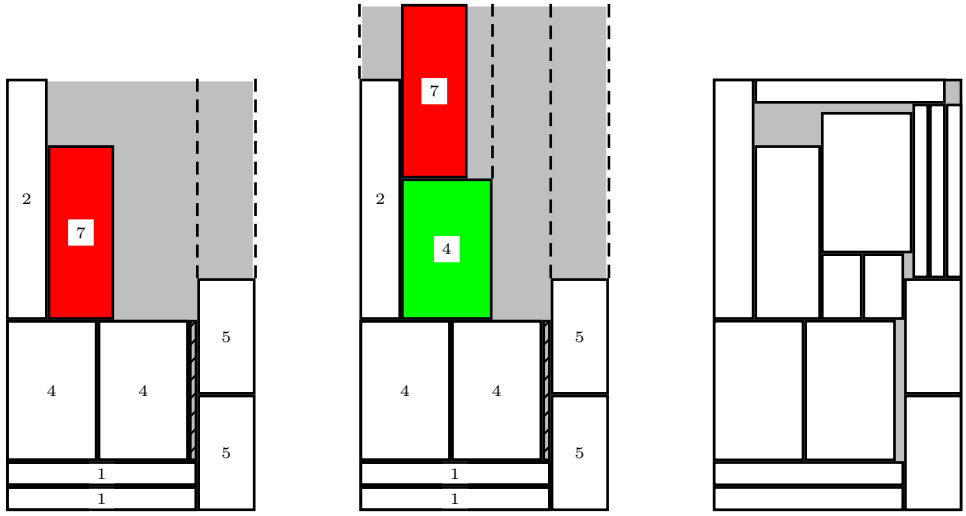
  Add the new rectangles to $\mathcal{L}$. Merge two rectangles if they are adjacent and are at the same level.



(a) Partial solution + widest piece 3

(b) First estimation for highest piece 4

(c) Next step: widest piece 10



(d) First estimation for highest piece 15

(e) Second estimation: Pieces 10 and 15

(f) Complete solution

Figure 2: *Estimating the effect of the highest piece*

3 A GRASP algorithm

The GRASP algorithm was developed by Feo and Resende [16] to solve hard combinatorial problems. For an updated introduction, refer to Resende and Ribeiro[32]. 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 is computed, which is dynamically adapted as the partial solution is built. 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 a result of the randomization by others, producing an overall better 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 pack. We denote by s_i the score of piece $i \in \mathcal{P}^*$, according to the criterion we are using at Step 2 of the constructive algorithm, for instance, if we use the first criterion: $s_i = w_i$. Let $s_{max} = max\{s_i \mid i \in \mathcal{P}^*\}$ and let δ be a parameter to be determined ($0 < \delta < 1$). We have considered four alternatives:

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

Using one of these randomization procedures on one of the selection criteria described above, a piece is chosen to be packed at each step of the constructive procedure. Nevertheless, the estimations of the effect of the tallest piece are also taken into account. These estimations are also randomized by using a parameter γ. If $E > \gamma * (M + U)$, the tallest piece j is selected and a number of copies randomly chosen between 1 and m_j is packed. At each iteration the parameter γ is randomly chosen in the interval $(0.9, 1.6)$. A preliminary computational study showed that the term $M + U$ tends to underestimate the total area which will

be required by the remaining pieces and therefore the value of γ should oscillate above value 1, though we also allow it to be slightly lower than 1.

3.2 Determining the parameter δ

A preliminary computational experience showed that no value of δ always produced the best results. Therefore, we considered several strategies basically consisting 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 one of these 5 values in turn: 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 [30], δ is taken at random from a set of discrete values. Initially, all values have the same probability of being chosen. In the iterative process we keep the value of the solutions obtained for each value of δ. After a certain number of iterations the probabilities are modified. Those corresponding to values of δ which have produced good solutions are increased and, conversely, those corresponding to values producing low quality solutions are decreased. The procedure is described in Figure 3, following Delorme et al.[13]. The value of α is fixed at 10, as in [30].

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 four alternatives:

I) From the initial solution of height H, we define a closed stock sheet of width W and height $H-1$ and remove the last $k\%$ pieces from the solution. We then have a sheet with some pieces packed into it and a list of pieces still to be packed. We can apply the constructive algorithm developed by Alvarez-Valdes et al.[1] for the non-guillotine cutting stock problem. If that algorithm succeeds in getting a solution with all the pieces packed into the

Initialization:

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

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

$n_{\delta^*} = 0$, 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 $mod(numIter, 200) == 0$ (every 200 iterations):

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

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

}

Figure 3: *Reactive Grasp*

sheet, that solution is an improved solution for the original problem. Figure 4 shows an example of this method on instance *ngcut02* in Table 10. As an algorithm for cutting a closed sheet has been used, the pieces tend to be placed near the corners of the sheet, but it is easy to move them down to obtain an adequate solution for the strip.

II) The pieces defining the maximum height H are removed from the solution and placed on some of the waste rectangles at lower levels of the strip. If the piece exceeds the dimensions of the waste rectangle, the pieces overlapping it are deleted. The deleted pieces are packed again using the deterministic constructive procedure. In Figure 5 we see an example of this method. One piece is removed from the solution and placed on a waste rectangle. The two pieces overlapping it are then removed and packed again, producing an improved solution.

III) The third method consists of eliminating the last $k\%$ pieces of the solution (for instance, the last 20%), as proposed by Beltran et al.[5], and filling the empty space with the deterministic constructive algorithm (see Figure 6). The chosen percentage has to guarantee that the required length for packing the remaining pieces is strictly lower than the original one. If this is not the case, we keep removing pieces until this condition is satisfied.

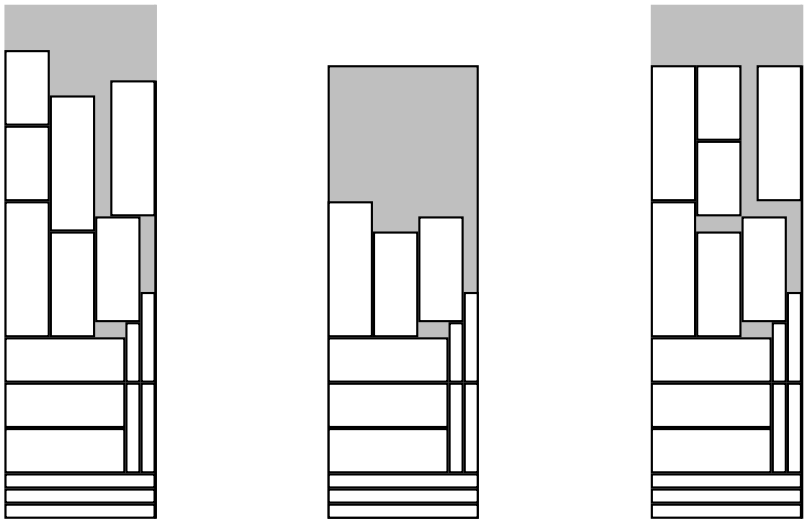
This method is based on the fact that wrong selections in the final stages of the randomized constructive process may produce large increases in the final required height H and therefore significant reductions may be obtained by doing the final part of that process again. A similar strategy is followed by Burke et al.[10]. They use their BF algorithm for packing $n - k$ pieces and then apply some metaheuristic procedures to pack the remaining k pieces.

IV) The fourth method is similar to the third one, but in this case all the pieces with their upper side exceeding a height λH are removed, with $0 < \lambda < 1$ (see Figure 7, with $\lambda = 0.9$).

4 Computational results

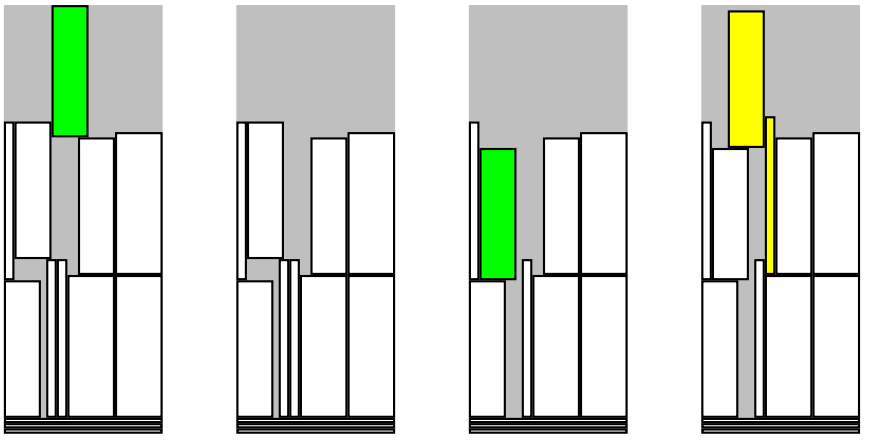
4.1 Test problems

There have been several sets of test problems proposed in the literature in recent years. Some of them have been generated by repeatedly slicing an initial rectangle into small pieces. They have the advantage of having a known optimal solution and, moreover, the generation procedure can be controlled to produce problems with different characteristics. Among this first group we have used:



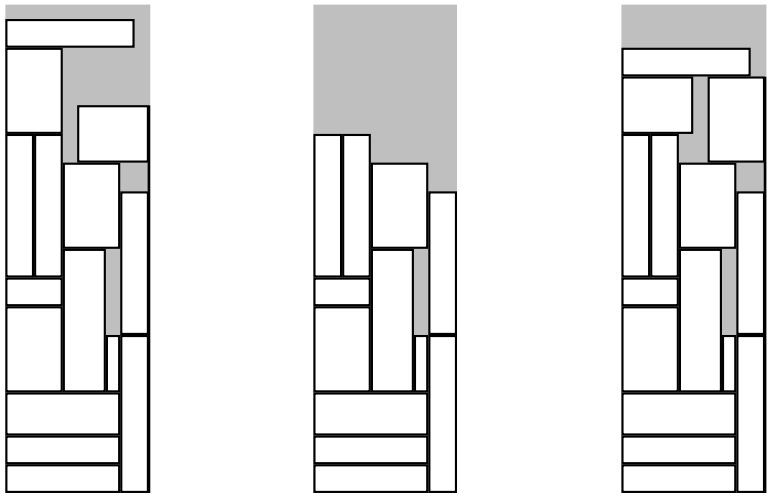
(a) Initial solution (b) Reducing and closing (c) Filling

Figure 4: *Improvement method I. Instance ngcut02, Table 10*



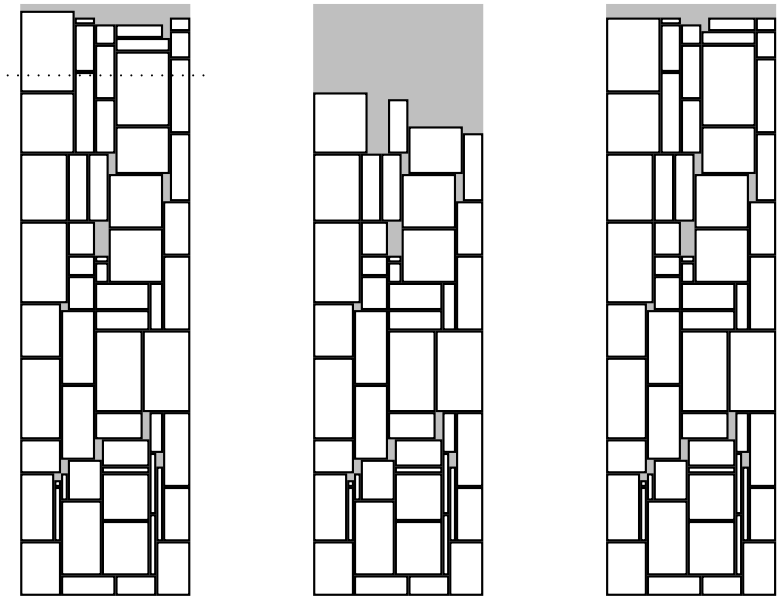
(a) Selecting (b) Removing (c) Packing (d) Completing

Figure 5: *Improvement method II. Instance ngcut10, Table 10*



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

Figure 6: *Improvement method III. Instance ngcut06, Table 10*



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

Figure 7: *Improvement method IV. Instance beng03, Table 10*

1. A set of 21 problems by Hopper and Turton[19], with sizes ranging from 16 to 197 pieces, denoted as *ht*.

2. A set of 70 problems proposed by Hopper[18]. Subset T contains 35 guillotine instances and subset N 35 non-guillotine instances, with a number of pieces ranging from 17 to 199, denoted as *hopper*.

3. A set of instances provided by Wang and Valenzuela[33]. They describe a generation procedure and use it to generate two types of problems: *nice*, with pieces similar in shape and size, and *path*, with extreme (pathological) variations of shape and size among pieces, ranging from 25 to 1000 pieces.

4. A set of instances generated by Burke et al.[9], with sizes from 10 to 3152, denoted as *N1* to *N13*.

5. An instance proposed by Ramesh Babu and Ramesh Babu[31] with 15 pieces.

6. A set of large instances generated by Pinto and Oliveira[29], with sizes from 50 to 15000 pieces, denoted as *pinto*.

A second group of test instances corresponds to problems adapted from other cutting and packing problems. In this case, not all optimal solutions are known and in general the optimal packing involves some parts of the strip remaining unused. However, they are interesting because in real life situations we cannot expect the required pieces to completely fill the given strip. Therefore, it is very interesting to test the algorithms of this type of problem, because algorithms working well on zero-waste problems do not always also perform well on this other type of problem.

We have used several sets of these test problems:

1. A set of 38 problems from the literature, some of them built for the strip packing problem and others adapted from the bin packing problem: 3 instances by Christofides and Whitlock[12], denoted as *cgcut*, 10 instances by Bengtsson [6], denoted as *beng*, and 25 instances by Beasley[3, 4], denoted as *gcut* and *ngcut*.

2. A set of 500 instances, 10 classes of instances adapted from bin packing, each of them composed of 50 problems, 4 of them generated by Martello et al. [27] and the remaining 6 by Berkey and Wang [7], denoted as *berkey*.

4.2 Choosing the best strategies

In order to choose the best strategies for defining the GRASP algorithm, we have done a limited computational study using some of the problem sets described above. The test problems for this preliminary study have been classified into three sets. The first set, denoted by *Literature*, contains the 38 *gcut*, *ngcut*, *cgcut* and *beng* problems; the second set contains the 500 *berkey* instances; the third set contains the 91 *ht* and *hopper* instances.

Table 2 compares the results obtained by the constructive algorithm of Section 2, when the piece to pack is selected according to every one of the four strategies considered at Step 2. For the first two sets, the table reports the average percentage deviation from the lower bound by Iori et al.[21]; for the third set, the percentage deviations from the known optimal solution. Among the three first criteria, based on the width of the pieces, none of them seemed to obtain the best results consistently, indicating that there is not a fixed value to be assigned to the relative weight of the piece height. Therefore we decided to use the criterion: $Max\{w_j + \kappa * h_j\}$, where κ is taken at random from the interval $(0.01, 0.75)$.

	Average percentage deviation from bound		
	Literature	*berkey*	*ht and hopper*
$Max\{w_j\}$	7,10	4,81	9,71
$Max\{w_j + 0.1 * h_j\}$	7,64	4,82	9,18
$Max\{w_j + 0.5 * h_j\}$	6,47	5,39	8,27
Best profile	13,27	10,10	13,66

Table 2: *Selection of the piece*

Table 3 compares the four randomization procedures described in Section 3.1, when the constructive phase of the GRASP algorithm is run for 1000 iterations, with no improvement phase. The parameter δ used in the first three strategies has been set to 0.5. This table has the same structure as Table 2. The randomization procedure *RCL-Value Min* seems to be the best alternative.

	Average percentage deviation from bound		
	Literature	*berkey*	*ht and hopper*
RCL-Value Min	4,23	3,65	4,80
RCL-Value	5,54	5,28	6,44
RCL-Percentage	4,12	5,28	5,98
Biased	5,00	5,22	6,95

Table 3: *Randomization procedures*

Table 4 compares the results obtained by the randomized constructive phase of the GRASP algorithm when using the alternatives for choosing δ described in

Section 3.2. The algorithm runs 1000 iterations and we use *RCL-Value Min* as a randomization procedure, with no improvement phase. Again Table 4 has the same structure as previous tables. It can be seen that Reactive GRASP obtains slightly better results than the deterministic selection from 0.5 to 0.9.

	Average percentage deviation from bound		
	Literature	*berkey*	*ht and hopper*
Random in [0.4, 0.9]	3,72	2,31	2,86
Random in [0.25, 0.75]	3,46	2,86	3,35
Deterministic from 0.5 to 0.9	3,29	2,18	2,94
Fixed to 0.9	3,77	2,19	3,35
Reactive Grasp	3,22	2,18	2,95

Table 4: *Study of parameter δ*

Table 5 compares the four improvement methods described in Section 3.3. In Methods III and IV, after removing some pieces, the solution is completed by using the constructive phase of the algorithm, but we modify the value of parameter κ in the selection criterion: $Max\{w_j + \kappa' * h_j\}$ and use $\kappa' = 0.75 - \kappa$, that is we complement the original value of κ with respect to the upper value 0.75. The improvement phase is only called if the solution of the constructive phase $H \leq H_{best} + 0.25(H_{worst} - H_{best})$, where H_{worst} and H_{best} correspond to the worst and best solutions obtained in the process. Methods III and IV produce virtually the same results and we decided to use Method III.

	Average percentage deviation from bound		
	Literature	*berkey*	*ht and hopper*
Method I	3,31	2,16	2,96
Method II	3,04	2,17	2,86
Method III	3,04	2,10	2,65
Method IV	3,06	2,10	2,66

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: $Max\{w_j + \kappa * h_j\}$
- Randomization procedure: *RCL-Value Min*
- Selection of δ: Reactive GRASP

- Improvement phase: Method III

The algorithm was coded in *C++* and run on a Pentium 4 Mobile at 2 Ghz. In order to compare it with other algorithms, the stopping criterion was defined as a time limit of 60 CPU seconds.

4.3.1 Results on zero-waste problems

Tables 6, 7, 8, 9 show the results obtained on problems for which optimal solutions pack all pieces into a rectangle, completely filling the strip of width W up to the optimal height H. At each table, the results of our GRASP algorithm are compared with previously reported results.

Table 6 presents the results on the 21 *ht*, which have been used by many authors. The instances are classified into 7 classes, according to their size. The first five columns describe the characteristics of each instance. Column 6 corresponds to the solutions obtained by the hybrid algorithm by Iori et al.[21]. They allow the algorithm to run for 300 seconds on a Pentium III at 800 Mhz. Columns 7 and 8 contain the results obtained by the BLD* algorithm by Lesh et al.[24]. They run their algorithm on a Pentium at 2000 Mhz. with several time limits, from which we have taken the results after 60 seconds and 3600 seconds. Columns 9 to 12 show the results obtained by the Best-Fit algorithm [9] and its enhancements adding Tabu Search (TS), Simulated Annealing (SA) and a genetic algorithm (GA) [10]. They run their algorithms on a Pentium IV at 2 Ghz. Best-Fit is very fast and it is run just once, while its combinations with metaheuristics are run 10 times, with a time limit of 60 seconds per run, and the best solution is reported. Columns 13 and 14 show the average and best solutions obtained by Bortfeld's genetic algorithm, running 10 times for each instance. Bortfeld reports an average time per run of 159 seconds on a Pentium at 2 Ghz. Finally, the last two columns show the average and best results obtained by our GRASP algorithm over 10 runs. The average percentages from optimum are calculated as $(sol - opt)/opt$ because this is the way in which some of the results are reported. The table shows that our algorithm obtains the best results on each class.

Table 7 shows the results obtained on the 70 *hopper* instances. The first set corresponds to guillotine patterns, while the second corresponds to non-guillotine patterns. Within each set, the instances are classified into 7 classes, according to their size. Only Lesh et al.[24] have used some of these problems and the comparison only includes their results with time limits of 60 and 3600 CPU seconds. It can be observed that the percentage deviation from the optimal solution tends to decrease with increasing problem sizes. For the same strip dimensions, many smaller pieces are easier to pack than fewer larger ones. The table also shows that the GRASP algorithm outperforms the BLD* algorithm on each instance class.

Class	*Instance*	*Problem size*		*Opt.*	*Iori*	*Lesh solutions*		*Burke solutions*				*Bortfeld solutions*		*GRASP solutions*	
		W	*n*		*solution*	*60 s*	*3600 s*	*Best-Fit*	*BF+TS*	*BF+SA*	*BF+GA*	*Average*	*Best*	*Average*	*Best*
C1	*01*	*20*	*16*	*20*	20			21	20	20	20			20	20
	02	*20*	*17*	*20*	21			22	21	20	21			20	20
	03	*20*	*16*	*20*	20			24	20	20	20			20	20
Average percentage deviation from optimum					*1.59*			*10.17*	*1.59*	*0.00*	*1.59*	*1.59*	*1.59*	*0.00*	*0.00*
C2	*04*	*40*	*25*	*15*	15			16	16	16	16			15	15
	05	*40*	*25*	*15*	16			16	16	16	16			15	15
	06	*40*	*25*	*15*	15			16	16	16	16			15	15
Average percentage deviation from optimum					*2.08*			*6.25*	*6.25*	*6.25*	*6.25*	*3.33*	*2.08*	*0.00*	*0.00*
C3	*07*	*60*	*28*	*30*	31			32	31	31	31			30	30
	08	*60*	*29*	*30*	31			34	32	31	32			31	31
	09	*60*	*28*	*30*	30			33	31	31	31			30	30
Average percentage deviation from optimum					*2.15*			*9.04*	*4.23*	*3.23*	*4.23*	*3.16*	*3.16*	*1.08*	*1.08*
C4	*10*	*60*	*49*	*60*	64			63	62	61	62			61	61
	11	*60*	*49*	*60*	63			62	62	61	62			61	61
	12	*60*	*49*	*60*	62			62	61	61	62			61	61
Average percentage deviation from optimum					*4.75*			*3.74*	*2.70*	*1.64*	*3.23*	*3.52*	*2.70*	*1.64*	*1.64*
C5	*13*	*60*	*73*	*90*	94	92	92	93	92	91	92			91	91
	14	*60*	*73*	*90*	93	92	92	92	92	91	92			91	91
	15	*60*	*73*	*90*	94	92	92	93	92	92	92			91	91
Average percentage deviation from optimum					*3.91*	*2.17*	*2.17*	*2.88*	*2.17*	*1.46*	*2.17*	*2.03*	*1.46*	*1.10*	*1.10*
C6	*16*	*80*	*97*	*120*	126	123	122	123	122	122	122			121.9	121
	17	*80*	*97*	*120*	124	123	122	122	121	121	121			121.9	121
	18	*80*	*97*	*120*	124	123	122	124	122	122	122			121.9	121
Average percentage deviation from optimum					*3.74*	*2.44*	*1.64*	*2.43*	*1.37*	*1.37*	*1.37*	*1.72*	*1.64*	*1.56*	*0.83*
C7	*19*	*160*	*196*	*240*				246	245	244	245			244	244
	20	*160*	*197*	*240*				244	244	244	244			242.9	242
	21	*160*	*196*	*240*				245	245	245	245			243	243
Average percentage deviation from optimum								*2.04*	*1.91*	*1.77*	*1.91*	*1.52*	*1.23*	*1.36*	*1.23*
Mean deviation from optimum (all problems)								*5.22*	*2.89*	*2.24*	*2.96*	*2.56*	*2.10*	*0.96*	*0.84*
Mean deviation from optimum (subset Iori)					*3.04*			*5.75*	*3.05*	*2.32*	*3.14*	*2.41*	*1.98*	*0.90*	*0.77*
Mean deviation from optimum (subset Lesh)					*3.82*	*2.65*	*1.91*	*2.65*	*1.77*	*1.41*	*1.77*	*1.87*	*1.55*	*1.33*	*0.96*
Number of optimal solutions					*5/21*				*2/21*	*3/21*	*2/21*			*8/21*	*8/21*

Table 6: *Comparison on Hopper and Turton[19] instances*

Percentage deviation from optimal solutions							
Type	*Class*	*Size of problem*		*GRASP*		*Lesh solutions*	
		W	*n*	*Mean*	*Best*	*60 s*	*3600 s*
Guillotine	*1*	200	17	0.0	0.0		
	2	200	25	3.2	3.2		
	3	200	29	3.7	3.7		
	4	200	49	3.0	2.7		
	5	200	73	2.4	2.2		
	6	200	97	2.1	1.9		
	7	200	199	1.5	1.4		
Average distance from optimum				*2.3*	*2.2*		
Non Guillotine	*1*	200	17	0.9	0.9	6.0	4.5
	2	200	25	3.3	3.3	6.4	4.7
	3	200	29	3.6	3.6	6.0	4.6
	4	200	49	3.0	2.8	5.1	3.9
	5	200	73	2.6	2.3	4.6	4.0
	6	200	97	2.2	2.1	4.0	3.0
	7	200	197	1.3	1.2	2.3	1.8
Average distance from optimum				*2.4*	*2.3*	*4.9*	*3.8*
Overall				*2.3*	*2.2*		

Table 7: *Computational results – Zero-waste instances by Hopper[18]*

Table 8 shows the results obtained on the set of problems used by Burke et al.[9, 10]. Columns 1 to 5 show the source and the characteristics of each instance. Columns 6 to 9 show the results obtained by the Best-Fit algorithm [9] and its improvements when it is combined with Tabu Search (TS), Simulated Annealing (SA) and genetic algorithm (GA)[10]. Columns 10 and 11 show the average and best results of 10 runs of the GRASP algorithm. The comparison on Wang and Valenzuela [33] instances is only approximate, because Burke et al.[10] use the original floating point data while we use integer data obtained by multiplying the original data by 10 and rounding to the nearest integer. It can also be noted that Burke et al.[9, 10] allow the pieces to be rotated, while our algorithm does not. The table shows that our GRASP algorithm performs very well on these instances, outperforming even the BF+SA algorithm, which seems to be the best combination.

Table 9 shows the results of the GRASP algorithm on *pinto* instances, which are specially interesting because they include some very large instances of up to 15000 pieces. We have run our algorithm on these instances to check if it could handle such very large instances efficiently and we compare our results with the results of the Bottom-Left algorithm reported in [29]. The table shows that within the time limit of 60 CPU seconds our algorithm is able to find the optimal solution for the largest instances and near-optimal solutions for the smaller ones. That confirms the comment in Table 7: instances with many small pieces are easier that those with few large pieces.

Source	*Instance*	*Problem size*		*Opt*	*Burke solutions*				*GRASP solutions*	
		W	*n*		*Best-Fit*	*BF+TS*	*BF+SA*	*BF+GA*	*Mean*	*Best*
Wang and	*Nice25*	*100*	*25*	100	108.0	106.9	104.0	108.4	103.9	103.7
Valenzuela[33]	*Nice50*	*100*	*50*	100	109.7	106.6	104.4	106.9	104.7	104.6
	Nice100	*100*	*100*	100	107.9	105.2	105.0	105.9	104.5	104.0
	Nice200	*100*	*200*	100	106.9	104.2	104.7	104.9	103.8	103.6
	Nice500	*100*	*500*	100	103.4	103.2	103.5	103.5	102.4	102.2
	Nice1000	*100*	*1000*	100	103.8	103.8	103.8	103.8	102.3	102.2
	Path25	*100*	*25*	100	110.2	105.8	103.1	108.2	104.2	104.2
	Path50	*100*	*50*	100	113.6	103.7	103.4	103.7	101.9	101.8
	Path100	*100*	*100*	100	106.7	104.7	103.0	104.4	102.7	102.6
	Path200	*100*	*200*	100	104.1	103.5	103.4	103.8	102.3	102.0
	Path500	*100*	*500*	100	103.8	103.1	103.5	103.4	103.2	103.1
	Path1000	*100*	*1000*	100	103.1	102.8	102.9	102.9	102.7	102.5
Burke	*N1*	*40*	*10*	40	45	40	40	40	40	40
et al.[9, 10]	*N2*	*30*	*20*	50	53	50	50	50	50	50
	N3	*30*	*30*	50	52	51	51	52	51	51
	N4	*80*	*40*	80	86	83	82	83	81	81
	N5	*100*	*50*	100	105	103	103	104	102	102
	N6	*50*	*60*	100	102	102	102	102	101	101
	N7	*80*	*70*	100	108	105	104	104	101	101
	N8	*100*	*80*	80	83	82	82	82	81	81
	N9	*50*	*100*	150	152	152	152	152	151	151
	N10	*70*	*200*	150	152	152	152	152	151	151
	N11	*70*	*200*	150	153	153	153	153	151	151
	N12	*100*	*500*	300	306	306	306	306	303.2	303
	N13	*640*	*3152*	960	964	964	964	964	963	963
Ramesh	*P1*	*1000*	*50*	375	400	400	400	400	375	375
Babu[31]										
Mean percentage deviation from optimum					*5.53%*	*3.29%*	*2.86%*	*3.61%*	*1.94%*	*1.86%*

Table 8: *Computational results – Problems from literature*

Instance	*Problem size*		*Optimum*	*Bottom-Left*	*GRASP solutions*	
	W	*n*		*solution*	*Mean*	*Best*
50cx	*400*	*50*	600	674	617	617
100cx	*400*	*100*	600	679	617.7	617
500cx	*400*	*500*	600	692	605.3	605
1000cx	*400*	*1000*	600	690	602.9	602
5000cx	*400*	*5000*	600	687	600	600
10000cx	*400*	*10000*	600	681	600	600
15000cx	*400*	*15000*	600	660	600	600

Table 9: *Computational results – Problems by Pinto and Oliveira[29]*

4.3.2 Results on general problems

Tables 10 and 11 contain the computational results on the sets of more general problems for which the optimal solution includes some wasted areas of the strip. For these instances the lower bound based on the total area of the pieces may be very loose and some other bounds have to be used. Martello et al.[28] developed a very good lower bound based on solving a relaxed problem, the One-Dimensional Contiguous Bin Packing Problem. The bounds appearing in Tables 10 and 11 have been provided by Iori et al.[21], using the Martello et al.[28] procedure. Table 10 presents the results on a first subset of these problems, containing *gcut*, *ngcut*, *cgcut* and *beng* instances. It includes the results reported by Iori et al.[21] and by Lesh et al.[24]. For the small *ngcut* instances the solutions by Iori et al.[21] and GRASP coincide. In fact, both provide all the optimal solutions, as reported by Martello et al.[28]. For *gcut* instances, we see that all algorithms get the optimal solutions for instances *gcut01* and *gcut03*. For the remaining instances, except *gcut05* in which all algorithms coincide, GRASP outperforms the other algorithms. For the remaining instances in the table, GRASP also obtains better results than the hybrid algorithm by Iori et al[21].

Table 11 contains the average results obtained on the 500 *berkey* instances. The table compares the results of the Iori et al.[21] hybrid algorithm, the Lesh et al.[24] BLD* algorithm and the Bortfeld[8] genetic algorithm with our GRASP algorithm. The overall results of the last line of the table clearly show that GRASP outperforms the other algorithms. The results are not homogeneous among classes. For classes 1, 7 and 9, all algorithms get solutions which are quite near to the lower bounds and therefore there are no significant differences between them. For the other classes, the distances to the lower bounds increase as do the differences between algorithms. In all these cases, the GRASP algorithm clearly obtains the best solutions.

5 Conclusions

We have developed a new heuristic algorithm based on GRASP techniques for the strip packing problem. We have followed the basic scheme of GRASP, because from the beginning we felt that this scheme was particularly well suited to this problem, but we have also included some new features that are partly responsible for the good results obtained. When we select the piece to pack, we consider not only individual pieces, but also blocks of pieces of the same type for packing together. That adds some flexibility to the process. Also, we have added a double estimation to foresee the future effect of the tallest pieces on the final solution. This aspect is very important for Strip Packing. The basic

Source of problem	*Instance*	*Size of problem*		*LB*	*Iori*	*Lesh solutions*		*GRASP solutions*	
		W	*n*		*solution*	*60 s*	*3600 s*	*Average*	*Minimum*
Beasley [4]	*01*	*10*	*10*	*23*	23			23	23
ngcut	*02*	*10*	*17*	*30*	30			30	30
	03	*10*	*21*	*28*	28			28	28
	04	*10*	*7*	*20*	20			20	20
	05	*10*	*14*	*36*	36			36	36
	06	*10*	*15*	*29*	31			31	31
	07	*20*	*8*	*20*	20			20	20
	08	*20*	*13*	*32*	33			33	33
	09	*20*	*18*	*49*	50			50	50
	10	*30*	*13*	*80*	80			80	80
	11	*30*	*15*	*50*	52			52	52
	12	*30*	*22*	*87*	87			87	87
Beasley [3]	*01*	*250*	*10*	*1016*	1016	1016	1016	1016	1016
gcut	*02*	*250*	*20*	*1133*	1207	1211	1195	1191	1191
	03	*250*	*30*	*1803*	1803	1803	1803	1803	1803
	04	*250*	*50*	*2934*	3130	3072	3054	3002	3002
	05	*500*	*10*	*1172*	1273	1273	1273	1273	1273
	06	*500*	*20*	*2514*	2675	2682	2656	2627	2627
	07	*500*	*30*	*4641*	4758	4795	4754	4693	4693
	08	*500*	*50*	*5703*	6240	6181	6081	5912.2	5908
	09	*1000*	*10*	*2022*				2256	2256
	10	*1000*	*20*	*5356*				6393	6393
	11	*1000*	*30*	*6537*				7736	7736
	12	*1000*	*50*	*12522*				13172	13172
	13	*3000*	*32*	*4772*				5009.5	5009
Christofides and	*01*	*10*	*16*	*23*	23			23	23
Whitlock[12]	*02*	*70*	*23*	*63*	65			65	65
cgcut	*03*	*70*	*62*	*636*	676			661	661
Bengtsson [6]	*01*	*25*	*20*	*30*	31			30	30
beng	*02*	*25*	*40*	*57*	58			57	57
	03	*25*	*60*	*84*	86			84	84
	04	*25*	*80*	*107*	110			107	107
	05	*25*	*100*	*134*	136			134	134
	06	*40*	*40*	*36*	37			36	36
	07	*40*	*80*	*67*	69			67	67
	08	*40*	*120*	*101*				101	101
	09	*40*	*160*	*126*				126	126
	10	*40*	*200*	*156*				156	156
Mean percentage deviation from bound (all problems)								*2.84%*	*2.84%*
Mean percentage deviation from bound (subset Iori)					*2.77%*			*1.62%*	*1.61%*
Mean percentage deviation from bound (subset Lesh)					*5.02%*	*4.82%*	*4.11%*	*3.17%*	*3.16%*
Number of proven optimal solutions (matching LB)					*11/30*	*2/8*	*2/8*	*18/38*	*18/38*

Table 10: *Computational results– Adapted packing instances from literature.*

	Size of problem		LB	Iori	Lesh solutions		Bortfled solutions		GRASP solutions	
	W	n		solution	60 s	3600 s	Average	Best	Average	Best
Class 1	*10*	*20*	60.3	61.2	61.5	61.3	62.0	61.6	61.3	61.3
	10	*40*	121.6	121.9	122.1	122.0	122.3	122.0	121.9	121.9
	10	*60*	187.4	189.0	189.1	188.9	189.1	189.0	188.7	188.6
	10	*80*	262.2	262.8	262.9	262.8	262.9	262.8	262.9	262.8
	10	*100*	304.4	305.5	305.9	305.5	305.2	305.0	305.6	305.5
Average deviation from bound				*0.64%*	*0.81%*	*0.68%*	*0.97%*	*0.75%*	*0.65%*	*0.63%*
Class 2	*30*	*20*	19.7	19.9	20.0	19.8	20.5	20.5	19.8	19.8
	30	*40*	39.1	40.0	39.5	39.1	39.5	39.1	39.1	39.1
	30	*60*	60.1	61.6	61.0	60.6	60.5	60.1	60.2	60.1
	30	*80*	83.2	84.7	84.0	83.6	83.4	83.3	83.2	83.2
	30	*100*	100.5	101.8	101.1	100.8	100.7	100.7	100.5	100.5
Average deviation from bound				*1.78%*	*1.12%*	*0.42%*	*1.24%*	*0.88%*	*0.14%*	*0.10%*
Class 3	*40*	*20*	157.4	164.5	164.6	164.2	167.3	166.7	163.5	163.5
	40	*40*	328.8	338.5	336.6	334.8	336.9	335.4	333.8	333.8
	40	*60*	500.0	517.0	513.0	510.1	511.1	509.8	506.6	506.4
	40	*80*	701.7	719.2	716.9	713.0	714.5	712.5	710.0	709.8
	40	*100*	832.7	848.3	847.8	844.1	844.4	842.6	840.2	839.7
Average deviation from bound				*3.05%*	*2.71%*	*2.23%*	*2.84%*	*2.52%*	*1.77%*	*1.73%*
Class 4	*100*	*20*	61.4	65.6	64.5	63.9	66.7	66.3	63.4	63.4
	100	*40*	123.9	131.5	129.6	128.1	127.9	127.1	126.3	126.2
	100	*60*	193	202.2	201.0	199.9	197.6	196.6	196.7	196.3
	100	*80*	267.2	278.9	278.8	276.6	273.6	272.2	272.2	271.8
	100	*100*	322	332.5	334.6	332.1	328.6	327.3	327.3	327.0
Average deviation from bound				*5.08%*	*4.41%*	*3.54%*	*3.74%*	*3.19%*	*2.13%*	*2.02%*
Class 5	*100*	*20*	512.2	536.8	536.6	534.8	537.8	536.6	533.9	533.9
	100	*40*	1053.8	1084.7	1084.2	1076.7	1082.7	1081.4	1074.7	1074.4
	100	*60*	1614.0	1666.2	1656.9	1651.1	1654.1	1650.8	1645.9	1645.2
	100	*80*	2268.4	2307.9	2303.2	2297.2	2303.0	2299.5	2290.4	2289.6
	100	*100*	2617.4	2697.2	2680.8	2669.5	2672.5	2666.9	2652.0	2649.0
Average deviation from bound				*3.15%*	*2.85%*	*2.43%*	*2.77%*	*2.59%*	*2.10%*	*2.05%*
Class 6	*300*	*20*	159.9	174.9	172.5	170.3	179.8	179.1	167.3	167.2
	300	*40*	323.5	345.8	343.8	339.2	338.9	337.0	333.6	333.1
	300	*60*	505.1	532.0	536.6	529.7	522.9	519.8	520.6	519.9
	300	*80*	699.7	732.2	743.9	733.9	724.2	719.4	718.9	717.3
	300	*100*	843.8	875.2	890.6	882.1	872.4	868.1	865.4	864.1
Average deviation from bound				*5.99%*	*6.45%*	*5.13%*	*5.52%*	*4.96%*	*3.22%*	*3.08%*
Class 7	*100*	*20*	490.4	501.9	501.9	501.9	503.1	502.7	501.9	501.9
	100	*40*	1049.7	1059.9	1059.4	1059.0	1060.8	1059.4	1059.0	1059.0
	100	*60*	1515.9	1529.6	1530.4	1529.7	1530.6	1529.7	1529.6	1529.6
	100	*80*	2206.1	2224.1	2223.7	2222.2	2225.9	2222.9	2222.2	2222.1
	100	*100*	2627	2646.5	2648.4	2646.5	2650.2	2648.8	2645.2	2644.1
Average deviation from bound				*1.16%*	*1.17%*	*1.12%*	*1.27%*	*1.19%*	*1.11%*	*1.10%*
Class 8	*100*	*20*	434.6	470.9	466.0	461.6	467.4	465.9	458.0	458.0
	100	*40*	922.0	979.4	978.6	967.8	962.0	956.2	954.4	953.0
	100	*60*	1360.9	1436.7	1437.5	1425.1	1407.0	1398.9	1405.9	1402.5
	100	*80*	1909.3	2015.3	2014.9	1992.0	1976.1	1967.3	1973.6	1967.5
	100	*100*	2362.8	2483.7	2491.4	2466.7	2432.4	2422.3	2439.5	2433.7
Average deviation from bound				*6.16%*	*5.99%*	*4.93%*	*4.34%*	*3.85%*	*3.76%*	*3.57%*
Class 9	*100*	*20*	1106.8	1106.8	1106.8	1106.8	1107.0	1106.8	1106.8	1106.8
	100	*40*	2189.2	2190.6	2190.9	2190.6	2191.7	2191.2	2190.6	2190.6
	100	*60*	3410.4	3410.4	3410.4	3410.4	3417.5	3417.5	3410.4	3410.4
	100	*80*	4578.6	4588.1	4588.1	4588.1	4588.1	4588.1	4588.1	4588.1
	100	*100*	5430.5	5434.9	5434.9	5434.9	5434.9	5434.9	5434.9	5434.9
Average deviation from bound				*0.07%*	*0.07%*	*0.07%*	*0.13%*	*0.12%*	*0.07%*	*0.07%*
Class 10	*100*	*20*	337.8	353.5	352.0	351.1	355.3	354.2	350.5	350.5
	100	*40*	642.8	673.7	670.1	665.7	666.8	664.7	664.5	664.2
	100	*60*	911.1	953.0	946.9	940.1	935.3	932.6	935.5	934.3
	100	*80*	1177.6	1234.7	1226.1	1217.8	1211. 2	1207.4	1209.7	1208.1
	100	*100*	1476.5	1542.3	1536.0	1525.3	1512.4	1507.8	1515.1	1512.1
Average deviation from bound				*4.67%*	*4.11%*	*3.48%*	*3.25%*	*3.05%*	*3.03%*	*2.93%*
Overall				*3.17%*	*2.97%*	*2.40%*	*2.61%*	*2.31%*	*1.80%*	*1.73%*

Table 11: *Computational results – Random instances proposed by Martello and Vigo [27] and Berkey and Wang[7].*

Bottom-Left algorithm does not include any such strategy and it is very common to observe that its solutions show high spikes at the top that decrease the quality of solutions. Burke et al.[9] takes it into account by adding a postprocessing phase to their Best-Fit algorithm in which the pieces defining the final height H of the solution are considered for rotation, thus eliminating the spikes.

Besides these concrete aspects, we have also explored several randomization procedures and several improvement moves to determine the best algorithmic structure for this problem. We do not think that this search for the best strategies and the best values of the involved parameters could be considered a disadvantage of the method. A preliminary study was conducted over a restricted set of test instances and then the complete algorithm was tested on all the test instances, obtaining good results on all of them, at least matching and frequently improving the best reported results. Therefore, the algorithm can be used as a black box, without further parameter tuning, and the results show that it performs well on a wide range of problems. However, the modular structure of the algorithm can also be used to modify some parts and adjust it, if necessary, to problems with special characteristics, different from those tested in this paper.

Finally, we would like to note that the algorithm is quite flexible and could be adapted to accommodate other conditions or constraints, such as the possibility of rotating the pieces.

Acknowledgements

This work has been partially supported by the Spanish Ministry of Science and Technology, DPI2005-04796, and by Project PBI-05-022, Consejeria de Ciencia y Tecnologia, Junta de Comunidades de Castilla-La Mancha.

References

[1] Alvarez-Valdes, R., Parreño, F., Tamarit, J.M. (2005) A GRASP algorithm for constrained two-dimensional non-guillotine cutting problems, *Journal of the Operational Research Society* **56**, 414-425.

[2] Baker, B.S., Coffman, E.G., Rivest, R.L. (1980) Orthogonal packing in two dimensions, *SIAM Journal on Computing* **9**, 846-855.

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

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

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

[6] Bengtsson, B. E.(1982) Packing rectangular pieces - a heuristic approach, *The Computer Journal* **25**, 353-357.

[7] Berkey, J. O., Wang, P. Y. (1987) Two dimensional finite bin packing algorithms, *Journal of the Operational Research Society* **38**, 423-429.

[8] Bortfeld, A.(2005) A genetic algorithm fot the two-dimensional strip packing problem with rectangular pieces, *European Journal of Operational Research*, in press.

[9] Burke, E.K., Kendall, G., Whitwell, G. (2004) A new placement heuristic for the orthogonal stock-cutting problem, *Operations Research* **52**, 655-671.

[10] Burke, E.K., Kendall, G., Whitwell, G. (2006) Metaheuristic enhancements of the Best-Fit heuristic for the orthogonal stock cutting problem, submitted to *INFORMS Journal on Computing*(2nd revision).

[11] Chazelle, B. (1983) The Bottom-Left bin packing heuristic: an efficient implementation, *IEEE Transactions on Computers* **32**, 697-707.

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

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

[14] Dowsland, K.A.(1993) Some experiments with simulated annealing techniques for packing problems, *European Journal of Operational Research* **68**, 389-399.

[15] Fekete, S.P., Schepers, J. (2004) An exact algorithm for higher-dimensional orthogonal packing, *submitted to: Operations Research*

[16] Feo, T., Resende, M.G.C. (1989) A probabilistic heuristic for a computationally difficult set covering problem, *Operations Research Letters* **8**, 67-71.

[17] Gómez, A., De la Fuente, D. (2000) Resolution of strip-packing problems with genetic algorithms, *Journal of the Operational Research Society* **51**, 1289-1295.

[18] Hopper, E.(2000) Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods, *PhD thesis Cardiff University, School of Engineering.*

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

[20] Hopper, E., Turton, C.H.(2001) A review of the application of meta-heuristic algorithms to 2D strip packing problems, *Artificial Intelligence Review*, **16**, 257-300.

[21] Iori, M., Martello, S., Monaci, M. (2003) Metaheuristic algorithms for the strip packing problem, in: P.M. Pardalos and V. Korotkith, editors, *Optimization and Industry: New Frontiers, Kluwer Academic Publishers*, 159-179.

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

[23] Lesh, N., Marks, J., Mitzenmacher, M.(2004) Exhaustive approaches to 2D rectangular perfect packings, *Information Processing Letters*, **90**, 7-14.

[24] Lesh, N., Marks, J., McMahon, A., Mitzenmacher, M. (2005) New heuristic and interactive approaches to 2D rectangular strip packing, *ACM Journal of Experimental Algorithmics*, **10**, 1-18.

[25] Leo, H.W., Wallace, K.S.(2004) Strip-packing using hybrid genetic approach, *Engineering Applications of Artificial Intelligence*, **17**, 169-177.

[26] Liu, D., Teng, H.(1999) An improved BL-algorithm for genetic algorithm of the ortogonal packing of rectangles, *European Journal of Operational Research* **112** , 413-420.

[27] Martello, S., Vigo, D. (1998) Exact solution of the two-dimensional finite bin packing problem, *Management Science* **44**, 388-399.

[28] Martello, S., Monaci, M., Vigo, D. (2003) An exact approach to the strip packing problem, *INFORMS Journal on Computing* **15 (3)**, 310-319.

[29] Pinto, E., Oliveira, J.F. (2005) Algorithm based on graphs for the non-guillotinable two-dimensional packing problem, 2nd ESICUP Meeting, Southampton.

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

[31] Ramesh Babu, A., Ramesh Babu, N. (1999) Effective nesting of rectangular parts in multiple rectangular sheets using genetic and heuristic algorithms, *International Journal of Production Research* **37**, 1625-1643.

[32] Resende, M.G.C, 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.

[33] Wang, P.Y., Valenzuela, C.L. (2001) Data set generation for rectangular placement problems, *European Journal of Operational Research* **134**, 378-391.

[34] Yeung, L.H.W., Tang, W.K.S (2004) Strip-packing using hybrid genetic approach, *Engineering Applications of Artificial Intelligence* **17**, 169-277.

[35] Zhang, D., Kang, Y., Deng, A. (2006) A new heuristic recursive algorithm for the strip rectangular packing problem, *Computers & Operations Research* **33**, 2209-2217.