PERGAMON

Computers & Operations Research 29 (2002) 925–947

computers & operations research

www.elsevier.com/locate/dsw

A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems

Ramón Alvarez-Valdés[a,*], Antonio Parajón[b], José Manuel Tamarit[a]

[a]*Department of Statistics and Operations Research, University of Valencia, Doctor Moliner 50, 46100 Burjassot Valencia, Spain*

[b]*Department of Mathematics, National Autonomous University of Nicaragua, Nicaragua*

Received 1 October 1999; received in revised form 1 March 2000

Abstract

In this paper we develop several heuristic algorithms for the two-dimensional cutting problem (TDC) in which a single stock sheet has to be cut into a set of small pieces, while maximising the value of the pieces cut. They can be considered to be general purpose algorithms because they solve the four versions of the TDC: weighted and unweighted, constrained and unconstrained. We begin by proposing two constructive procedures based on simple bounds obtained by solving one-dimensional knapsack problems. We then use these constructive algorithms as building blocks for more complex procedures. We have developed a greedy randomised adaptive search procedure (GRASP) which is very fast and obtains good results for both constrained and unconstrained problems. We have also developed a more complex tabu search algorithm that obtains high quality results in moderate computing times. Finally, we have implemented a path relinking procedure to improve the final results of the above algorithms. For the computational results we have used the set of large-scale test problems collected and generated by Fayard et al. (J. Oper. Res. Soc. 49 (1998) 1270).

Scope and purpose

The two-dimensional cutting problem (TDC) consists of cutting a single rectangular stock sheet into a set of small rectangular pieces of given sizes and values to maximise the total value of the pieces cut. This problem has a wide range of commercial and industrial applications, whenever a sheet of wood, glass, paper or metal has to be cut. The TDC problem can also be considered as a subproblem of more general cutting problems, involving several available stock sheets of different sizes. In this paper we develop several heuristic algorithms for solving TDC problems. The computational results show that they produce high-quality results in short computing times.

Keywords: Cutting stock problem; Heuristics; Knapsack problem; GRASP; Tabu search; Path relinking

* Corresponding author. Tel.: + 34-96-3864308; fax: + 34-96-3864735.
E-mail address: ramon.alvarez@uv.es (R. Alvarez-Valdés).

0305-0548/02/$ - see front matter
PII: S0305-0548(00)00095-2

1. Introduction

The two-dimensional cutting problem (TDC) consists of cutting a given finite set of small rectangular pieces from a large stock rectangle of fixed dimensions with maximum profit. This problem has a wide range of commercial and industrial applications. It appears in cutting wood plates to make furniture and cardboard to make boxes, as well as in production of glass, metal sheets, for example. The TDC problem can also be considered as a simple version or subproblem of the general cutting problem in which a set of orders for different pieces has to be met, cutting them from a given set of stock sheets of different sizes.

An instance of the TDC problem can be given by the quadruple (R, S, v, b). $R = (L, W)$ is the large stock rectangle of length L and width W. $S = \{(l_1, w_1), (l_2, w_2), \ldots, (l_m, w_m)\}$ is the set of small pieces of length l_i and width w_i. Each piece i has a value v_i and a demand b_i. The problem is to cut off the rectangle R into x_i copies of each piece i, such that $0 \leqslant x_i \leqslant b_i$, $i = 1, \ldots, m$, and the total utility $\sum_i v_i x_i$ is maximised.

A sequence of cuts of R into rectangular pieces is a *cutting pattern*. The produced pieces not belonging to S is the waste. In this paper we impose two additional constraints to the cutting patterns that usually appear in real-life applications:

(a) The pieces have *fixed orientation*, that is, pieces of dimensions (a, b) and (b, a) are not the same.
(b) Only *guillotine cuts* are allowed. A guillotine cut goes from one edge of the rectangle to the opposite edge.

Following the classification of Fayard et al. [1] for the TDC problem, four versions can be distinguished:

1. *The unconstrained unweighted version* (UU_TDC): an instance is defined by the quadruple (R, S, s, ∞). The value v_i of each piece i is equal to its surface $s_i = l_i w_i$. The objective function maximises the total occupied area, which is equivalent to minimising the waste.
2. *The unconstrained weighted version* (UW_TDC): the instance is represented by (R, S, v, ∞), where the values of the pieces are independent from their areas. The objective is to maximise the total value of the pieces cut.
3. *The constrained unweighted version* (CU_TDC): the instance is described by (R, S, s, b), where each b_i limits the number of pieces of type i to be cut and, at least, some of b_i are strictly lower than $\lfloor L/l_i \rfloor \lfloor W/w_i \rfloor$, the maximum number of pieces i that could be obtained from rectangle R.
4. *The constrained weighted version* (CW_TDC): the instance is defined by (R, S, v, b). It is the most general case.

Many authors have studied TDC problems. For small and medium size instances some exact algorithms have been proposed (see, for instance, Beasley [2] and Hifi and Zissimopoulos [3] for the unconstrained case and Christofides and Whitlock [4], Viswanathan and Bagchi [5], Christofides and Hadjiconstantinou [6], Hifi [7] and Hifi and Zissimopoulos [8] for the constrained case).

For large problems many heuristic algorithms have been developed. For the unconstrained case Morabito et al. [9] used depth-first search and hill climbing strategies, producing algorithm DH. Fayard and Zissimopoulos [10] designed an algorithm based on solving a series of one-dimensional knapsack problems using dynamic programming (algorithm KD). Finally, Hifi [11] proposed a combined DH/KD algorithm, obtaining very good results for large size problems.

More recently, Fayard et al. [1] have extended the ideas of Fayard and Zissimopoulos [10] to develop a general purpose algorithm for the four classes of TDC problems. Morabito and Arenales [12] extended their previous algorithm to the constrained case, also obtaining good results.

A completely different approach, for unweighted problems only, constrained or not, is that of Wang [13], who generates partial cutting patterns by successive horizontal and vertical combinations of rectangles. In order not to create a too long list of rectangles, only those with a waste percentage below a given value are kept. In order to get high-quality results we need to fix high values of accepted waste percentages and then computing times tend to be long if an optimal solution is required. However, the algorithm may be modified to obtain fast heuristic solutions.

In this paper we propose several heuristic algorithms to deal with the four versions of the TDC problem. The next section describes simple constructive procedures that will be used as a step in more elaborated algorithms. In the following sections we propose a fast GRASP algorithm, and a more complex tabu search procedure. These metaheuristic algorithms have not been used for TDC problems to date. The performance of algorithms is tested on a set of large problem instances and the results are compared with the best known so far. Finally, we draw some conclusions.

2. A constructive heuristic algorithm

We develop a cutting process in which, for each (sub)rectangle to be cut, we consider the consequences of cutting each possible piece at the bottom left-hand corner of the rectangle.

2.1. Upper bound BK_1

If we have a rectangle $R_k = (L_k, W_k)$ and a piece (l_i, w_i) to be cut in the bottom left-hand corner, we will have one of these two situations: if the first guillotine cut is vertical, besides piece i we obtain two rectangles $R_1^i = (l_i, W_k - w_i)$ and $R_2^i = (L_k - l_i, W_k)$ (Fig. 1a). If the first guillotine cut is horizontal, then besides piece i we obtain $R_3^i = (L_k - l_i, w_i)$ and $R_4^i = (L_k, W_k - w_i)$ (Fig. 1b).

An estimate of the value that we would get from each of these rectangles $R = (L, W)$ is obtained by solving the following knapsack problem that will give us an upper bound (see [3]):

$$
\begin{aligned}
BK_1(R) \quad = \quad & \text{Max} \quad \sum_{i \in S^*} v_i x_i \quad \text{where } S^* = \{i / l_i \leqslant L, w_i \leqslant W_i\} \\
& \text{s.t.} \quad \sum_{i \in S^*} s_i x_i \leqslant LW \\
& \qquad 0 \leqslant x_i \leqslant \min\{b_i - n_i, \lfloor L/l_i \rfloor \lfloor W/w_i \rfloor\} \quad i = 1, \ldots, m,
\end{aligned}
$$

where n_i is the number of pieces of type i already cut, including the piece being considered.

The problem $BK_1(R)$ is approximately solved by using the fast greedy algorithm proposed by Martello and Toth [14]:

Step 0. Initialization

Let $z = 0$, the total value of pieces cut,

$c = L\ W$, the current right-hand side of the constraint.

The set S^* of pieces is ordered by non-increasing v_i/s_i.

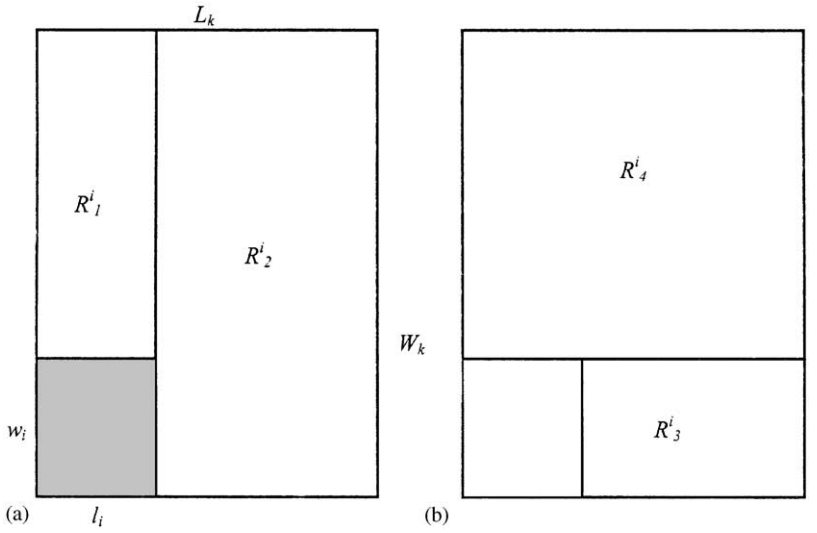

Fig. 1.

Let $j = 1$, index of the piece currently considered.
Let $z^* = 0$, the value obtained if only one type of pieces is cut and j^* the corresponding index

Step 1.

$u_j = \min\{b_j - n_j, \lfloor L/l_j \rfloor \lfloor W/w_j \rfloor\}$,
$x_j = \min\{u_j, \lfloor c/s_j \rfloor\}$,
$z = z + v_j x_j$,
$c = c - s_j x_j$.
If $v_j u_j > z^*$, then $z^* = v_j u_j$ and $j^* = j$.

Step 2: If $j > m$, then $j = j + 1$. Go to Step 1. Else,

Step 3: If $z^* > z$, then $z = z^*$; for $j = 1$ to m, $x_j = 0$; $x_{j^*} = u_{j^*}$.

Based on that estimate we decide on the piece to be cut until no more pieces can be obtained from the remaining rectangles, which are then considered waste.

The idea of solving two knapsack problems for regions R^i_1 and R^i_2 in Fig. 1a, or regions R^i_3 and R^i_4 in Fig. 1b, has already been introduced by Hifi and Ouafi [14] for obtaining lower bounds at each node of a branch and bound algorithm. They solve exactly the knapsack problem for the largest region by using dynamic programming and add the value of the best homogeneous solution for the smallest region, that is, the best solution involving only one type of piece.

2.2. Upper bound BK_2

Suppose we are using the greedy algorithm of Martello and Toth [15] to cut rectangle $R = (L, W)$, and in Step 1 the first piece has been cut x_1 times. When we consider the second piece,

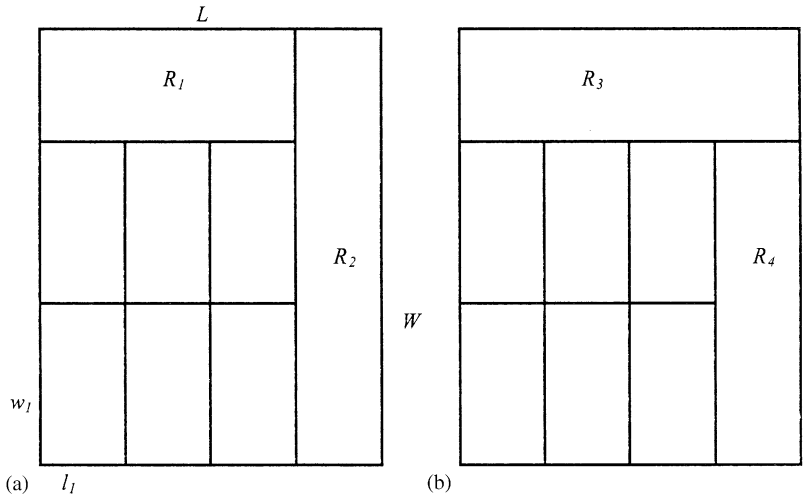

Fig. 2.

the right-hand side of the constraint will be $LW - x_1 l_1 w_1$ and we will try to cut as many pieces as possible without exceeding this limit. However, after cutting piece 1 the situation may be that of Fig. 2 (taking for the example $x_1 = 6$). We can take into account the way the pieces have been cut. We determine n_l and n_w, lower bounds on the number of times the piece 1 has been cut along the length and width of the rectangle, respectively. These values are such that $n_l n_w = x_1$ for unconstrained problems, but $n_l n_w$ may be less than x_1 for constrained problems. Therefore, the situation in the example is $n_l = 3$, $n_w = 2$. The remaining space will be R_1 and R_2, or R_3 and R_4, where $R_1 = (n_l l_1, W - n_w w_1)$, $R_2 = (L - n_l l_1, W)$, $R_3 = (L, W - n_w w_1)$ and $R_4 = (L - n_l l_1, W - n_w w_1)$. Therefore, for pieces $i = 2, 3, \ldots, m$ their upper bound will be

$$x_i \leqslant \min\{b_i - n_i, \max\{\lfloor L_1/l_i \rfloor \lfloor W_1/w_i \rfloor + \lfloor L_2/l_i \rfloor \lfloor W_2/w_i \rfloor, \lfloor L_3/l_i \rfloor \lfloor W_3/w_i \rfloor + \lfloor L_4/l_i \rfloor \lfloor W_4/w_i \rfloor\}\},$$

where L_j, W_j are the dimensions of rectangle R_j, $j = 1, \ldots, 4$.

In this second upper bound we again use the algorithm of Martello and Toth [15], but from $j = 2$ on, the upper bound on each piece will therefore be that of the above expression. This modification will have a significant effect on the bounds of large pieces and a smaller effect on the bounds of small pieces.

2.3. Constructive algorithm

The complete algorithm is described as follows:

Step 0: Initialization.
Let $L = \{R\}$ the set of rectangles still to be cut
Let $P = \emptyset$ the set of pieces already cut

Let $v_T = 0$ the total value of the pieces cut.
The set S of pieces is ordered by nonincreasing $r_i = v_i/s_i$. Ties are broken by nonincreasing v_i.
For each piece i, $n_i = 0$, the number of pieces cut in the process.

Step 1: While $L \neq \emptyset$,
Take the smallest rectangle of L, $R_k = (L_k, W_k)$ and set $L = L - \{R_k\}$.
Step 2: For each piece i, $i = 1, \ldots, m$,
If $l_i \leqslant L_k$, $w_i \leqslant W_k$, $n_i < b_i$, calculate the estimates:
$e_1 = BK_1(R_1^i)$, $R_1^i = (l_i, W_k - w_i)$,
$e_2 = BK_1(R_2^i)$, $R_2^i = (L_k - l_i, W_k)$,
$h_1 = BK_1(R_3^i)$, $R_3^i = (L_k - l_i, w_i)$,
$h_2 = BK_1(R_4^i)$, $R_4^i = (L_k, W_k - w_i)$.
The upper bound for piece i is: $B_i = v_i + \max\{e_1 + e_2, h_1 + h_2\}$,
Else, $B_i = 0$.
Step 3: For the piece j, such that $B_j = \max\{B_i,\ i = 1, \ldots, m\}$
If $B_j > 0$:
Cut piece j at the bottom left corner of R_k and $P = P \cup \{j\}$
$v_T = v_T + v_j$
If $e_1 + e_2 \geqslant h_1 + h_2$, then
$L = L \cup \{R_1^j\} \cup \{R_2^j\}$.
Else, $L = L \cup \{R_3^j\} \cup \{R_4^j\}$.
If $B_j = 0$, rectangle R_k is waste.
Go to Step 1.

Note that if in Step 2 instead of BK_1 we use upper bound BK_2, we have another heuristic algorithm.

3. A GRASP algorithm

The GRASP (greedy adaptive search procedure) was developed in the late 1980s by Feo and Resende for solving set covering problems [16]. An updated introduction may be found in [17]. Each GRASP iteration consists of a constructive phase and an exchange procedure. The construction phase is iterative because the solution is built considering one element at a time, greedy because the selection of the element to add is guided by a greedy function, and adaptive because the greedy function takes into account the elements previously chosen. The exchange phase is usually a local search procedure.

In our case, the construction phase follows the constructive algorithm in the previous section, but in Step 3 instead of selecting piece j of maximum bound B_j, the piece is randomly selected from the set of pieces $S_{\max} = \{i \,|\, B_i \geqslant \delta B_j\}$, where δ is an acceptance parameter.

The improvement phase focuses on each waste rectangle and merges it, if possible, with an adjacent piece in order to create a new rectangle that could be cut with greater value. In this process we consider that two rectangles are adjacent if they have a common side. In a first pass (from Fig. 3a to b), shadowed waste rectangles R_1 and R_2 merge with adjacent pieces while waste

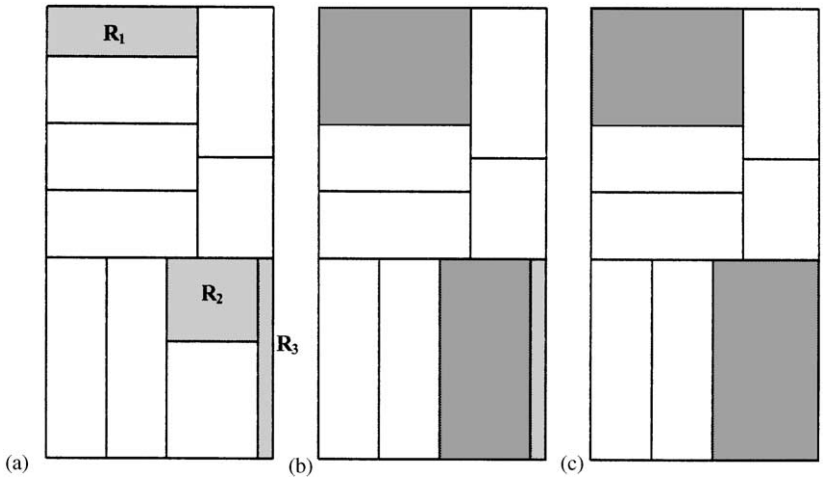

Fig. 3.

rectangle R_3 cannot be merged. In a second pass (from Fig. 3b to c), waste rectangle R_3 is merged with the recently created adjacent rectangle.

The new rectangles are cut following one of the constructive procedures described above, producing a new complete solution.

The GRASP algorithm repeats these construction and improving phases until a stop criterion is satisfied. This stop criterion is typically a given number of iterations without improving the best-known solution.

4. A tabu search algorithm

Tabu search is by now a well-established meta-heuristic for optimization (see Glover and Laguna [18] for an introduction). The basic elements of the algorithm are described as follows.

4.1. Definition of move

The first step in designing a Tabu Search procedure is the definition of a move that maintains the guillotine cut propriety. We say that two rectangles are *neighbours* if they share part of a side. This part may be a complete side (corresponding to the definition of adjacency in the previous section), only a part or even just a vertex.

The move is defined as follows. We take a couple of neighbouring rectangles R_i and R_j (Fig. 4a) and define the common region as the smallest rectangle containing both of them (Fig. 4b). We then determine the guillotine cuts nearest to that region and containing it (Fig. 4c). The new rectangle defined by these cuts is emptied and cut again following the GRASP procedure in Section 3. This means that, instead of cutting it once following the constructive procedure, we perform several iterations of GRASP constructive and improving phases.

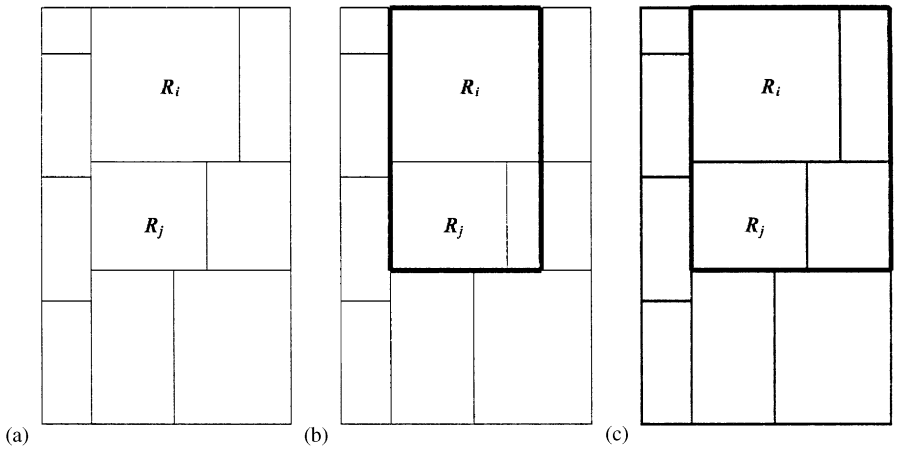

Fig. 4.

4.2. Selection of move

At each iteration we select the move to make following three steps. First, a rectangle is selected at random from the list of all rectangles, both pieces and waste, from the current solution. Second, we consider all the rectangles which are neighbours of it, one at a time. Finally, for each pair of rectangles, we follow the steps explained above in the definition of move to obtain a new rectangle to be cut. If this rectangle does not contain any waste, we do not consider it for cutting, because it is impossible in unweighted problems and highly unlikely in weighted ones to improve the current solution. If the rectangle contains some waste, it is emptied and cut again. In this context, each iteration of the *GRASP* procedure may be seen as the exploration of different possible moves, because each *GRASP* iteration may produce a different final cutting pattern.

The move to be made is selected as the alternative producing the maximum increase in the solution value of the current solution.

4.3. Tabu list

For each move we keep in the tabu list the dimensions and position of the new rectangle that has been cut again (coordinates of its bottom left and upper right vertices) and the type of piece cut at its bottom left-hand corner. Hence, in a later iteration, the same rectangle can be considered for cutting but the same cutting pattern cannot be used.

The length of the list changes dynamically, after a given number of iterations, without improving the best known solution. If m^* is the square root of the number of pieces, the length is randomly chosen in the interval [0.5 m*, 1.5 m*].

4.4. Basic tabu search procedure

Let S be the current solution with an associate set of rectangles R (pieces and waste), a set of guillotine cuts C and value v_S.

Let S^* be the best known solution with value v^*

Initialization: Set $S^* = S = SI$, the initial solution obtained by using GRASP algorithm
Set $niter = 0$.
Iteration: While ($niter < maxiter$) {

Step 1: Select at random a rectangle $R_i \in R$
Let $S_i = \emptyset$ the partial solution to be built and $v_i = 0$ its value
Step 2: Let $N_i = \{R_j \in R \mid R_j \text{ is neighbor of } R_i\}$
For each $R_j \in N_i$: (Fig. 4a)
1. Build R_{ij}, the smallest rectangle containing R_i and R_j (Fig. 4b)
2. Determine P_{ij}, the smallest rectangle such that
   - $R_{ij} \subseteq P_{ij}$
   - each side of P_{ij} is included in a cut of C (Fig. 4c)
3. If P_{ij} does not contain any waste, go back and take the next R_j.
4. For each rectangle $R_k \in R$ such that $R_k \subset P_{ij}$:
   $R = R - \{R_k\}$,
   $v_S = v_S - v_k$.
5. For $n = 1$ to $niter$:
   - Solve the cutting problem of P_{ij} by using GRASP algorithm
   - Let v_n the value of solution obtained S_n
   - If the move is not tabu and $v_n + v_S > v_i$, or $v_n + v_S > v^*$
     - $v_i = v_S + v_n$,
     - $S_i = S_n$,
     - $P_i^* = P_{ij}$.
6. Recover the original R and v_S and take the next $R_j \in N_i$.

Step 3: Update the current solution S by substituting the rectangles of P_i^* by those associated to S_i and $v_S = v_i$
If $v_i > v^*$, update the best known solution S^*
Update the tabu list
$niter = niter + 1$;
}

4.5. Intensification and diversification strategies

The definition of move involves an important degree of diversification in the search procedure. If the selected rectangles are placed towards the upper right-hand corner, a local change may happen, but if they are placed near the bottom left-hand corner, then possibly most of the original large rectangle will be cut again. Therefore, the move may dramatically change the structure of the solution. The random selection of the first rectangle to be considered for the move ensures that in the long term we will have some influence moves in which the solution changes almost completely.

However, the evaluation procedure does not change and it is based on a one-dimensional bound that may lead to erroneous decisions. Therefore, we have designed a change in the evaluation function that may be used for intensification and diversification purposes.

In the search process, we keep a set of n elite solutions, i.e., the set of the n best solutions obtained so far. If f_{ik} is the number of times piece i appears in solution k, and M is the total number of pieces contained in the n solutions, we can define a modified value of piece i:

$$p_i = v_i\left(1 + \beta\left(\sum_k f_{ik}/M\right)\right).$$

If parameter $\beta > 0$, then the value of the pieces appearing more often in the solutions is increased and their presence is therefore favoured in an intensification strategy. On the contrary, if $\beta < 0$, then the pieces with higher frequencies will have lower values and other pieces will have more possibilities of being included in the solution.

These modified values are used in the computation of bounds BK_1 and BK_2 which are the decision procedures for selecting the piece to cut.

5. Path relinking

If we have a set of *reference solutions*, typically a set of high-quality solutions, path relinking generates new solutions by exploring the paths that connect them. Starting from one of these solutions, called the *initiating solution*, a series of moves define a path in the neighbourhood space that leads towards the other solutions, called *guiding solutions*.

This technique is called path relinking because in tabu search any two solutions are linked by a series of moves performed during the search [18]. Now, these solutions are linked again but following a different path in which the moves are not selected according to the best values of the objective function, but looking for those moves leading more directly to the guiding solutions.

Though path relinking comes from tabu search, it can be used whenever a set of reference solutions is available. In our case, we can apply path relinking to the set of high-quality solutions obtained by the *GRASP* algorithm in Section 3 and the tabu search algorithm in Section 4.

We start from an extreme case of an initiating solution, a null solution in which we have the original rectangle without any cuts. We take as a guiding solution one of the elite solutions previously obtained by any algorithm. We follow a constructive strategy by introducing the guillotine cuts defined by the guiding solution one at a time. Each time we add a new guillotine cut from the list of cuts of the guiding solution, it cuts one rectangle of the current solution and two new rectangles appear. At the end of the process we will repeat the guiding solution, but in the meantime at each step we take the current list of rectangles and apply a *GRASP* algorithm to cut each of them and obtain a complete solution, in which part of the cutting structure of the guiding solution is enforced.

5.1. Path relinking procedure

Let $\boldsymbol{S}$ be the set of reference solutions
Let S^* be the best known solution with value v^* and list of pieces P^*
Let us denote by R_0 the initial rectangle to be cut
For each $S \in \boldsymbol{S}$, repeat the following procedure:

Initialization: Let C be the ordered list of guillotine cuts of S.
Set $L = \{R_0\}$, the list of the rectangles still to be cut.
Set $P = \emptyset$, the set of pieces already cut
Set $v = 0$, the value of the built solution

Step 1: Take the next cut $c \in C$
This cut divides some rectangle $R_i \in L$ into two new rectangles $R_{i'}$, $R_{i''}$.
Set $L = L \cup \{R_{i'}\} \cup \{R_{i''}\} - \{R_i\}$.
Remove from P the pieces included in R_i.
Set $v = v - \sum_{j \in Ri} v_j$.
Step 2: Solve the cutting problem of rectangles $R_{i'}$ and $R_{i''}$ by GRASP,
obtaining pieces $P_{i'}$ and $P_{i''}$ and values $v_{i'}$ and $v_{i''}$
Set $P = P \cup P_{i'} \cup P_{i''}$
Step 3: If $v_{i'} + v_{i''} + v > v^*$, update S^*
If all cuts in C have been taken, go to the next reference solution $S \in \boldsymbol{S}$
Else, go to Step 1.

Example. Consider the example, taken from [5], with $L = 5$, $W = 3$, $m = 2$ and

i	l_i	w_i	v_i	b_i
1	2	2	25	2
2	3	1	10	5

and suppose we have a reference solution in Fig. 5, in which we have cut one piece of type 1 and three pieces of type 2, with a total value of 55. The sequence of guillotine cuts is marked outside the rectangle.

We start the process of building a new solution by considering the initial rectangle $R_0 = (L, W)$ and applying the first cut of the reference solution, obtaining Fig. 6a with two regions $R_{1'}$ and $R_{1''}$. When we cut these two regions by using *GRASP*, we can obtain the solutions of Fig. 6b. The value of the new global solution is 60, which improves the reference solution.

Next, we impose the second cut of the reference solution and obtain Fig. 7a, in which $R_{1''}$ has been divided into $R_{2'}$ and $R_{2''}$. $R_{1'}$ was already solved, and we solve now $R_{2'}$ and $R_{2''}$, obtaining

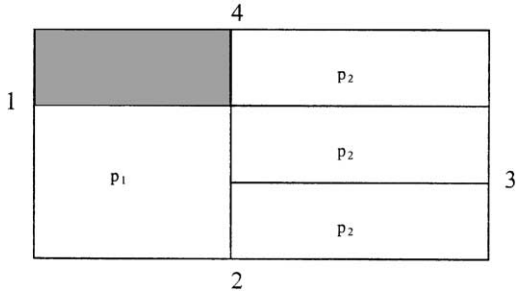

Fig. 5.

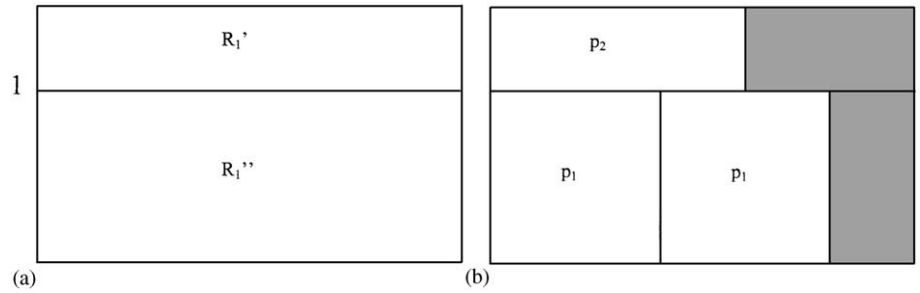

Fig. 6.

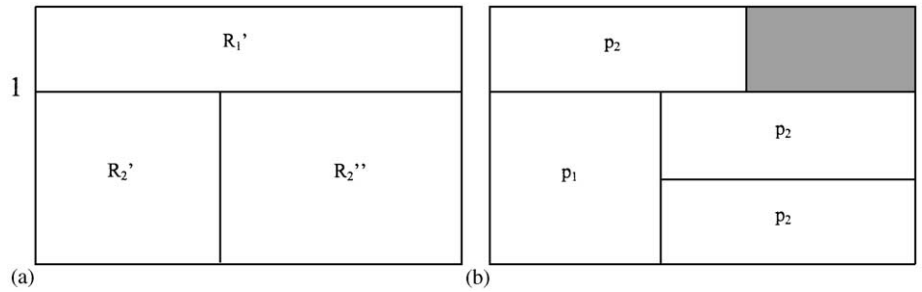

Fig. 7.

Fig. 7b. Proceeding in the same way, we will impose cuts 3 and 4. The final solution will be the same reference solution, but in the building process we have found an improved solution.

6. Computational results

The algorithms were coded in C++ and run on a personal computer with a Pentium II processor at 350 MHz. This section describes the test problems and the values of parameters and strategies used in each algorithm. All test problems are publicly available from ftp://www.univ-paris1.fr/pub/CERMSEM/hifi/2Dcutting.

6.1. The test problems of Fayard et al. [1]

6.1.1. Test problems

We have first used the 60 test problems of Fayard et al. [1] for which the optimal solutions are known. Some of them are taken from previous work which has been published and the rest are large-scale problems randomly generated, covering the four versions of the TDC problem.

For the unconstrained unweighted case (UU_TDC) there are 19 instances. *H* is given by Herz [19], HZ1 appears in Hifi and Zissimopoulos [20], M1–M5 in Morabito et al. [9] and *B* is the largest problem proposed by Beasley [2]. UU1–UU11 are random instances generated as follows:

the dimensions of the initial sheet are uniformly taken in the interval [500, 4000], the dimensions of the pieces in the intervals $[0.01L, 0.7W]$, and the number of pieces in the interval [25, 60].

For the unconstrained weighted case (UW_TDC) there are 12 instances. HZ2 is taken from Hifi and Zissimopoulos [3] and the others UW1–UW11 are generated as before, with values v_i for the pieces taken in the interval [100, 1000].

For the constrained unweighted case (CU_TDC) there are 14 instances. OF1 and OF2 are taken from Oliveira and Ferreira [21] and W from Wang [13]. CU1–CU11 are generated as follows: the dimensions of the sheet in the interval [100, 1000], the dimensions of pieces in the intervals $[0.1L, 0.7W]$ and the number of pieces from [25, 60]. The upper bounds b_i are equal to $\max\{1, \min\{10, \text{random}(\gamma)\}\}$, where $\gamma = \lfloor L/l_i \rfloor \lfloor W/w_i \rfloor$, $i = 1, \ldots, m$.

For the constrained weighted case (CW_TDC) there are 15 instances. CH1 and CH2 are taken from Christofides and Whitlock [4] and TH1 and TH2 from Tschöke and Holthöfer [22]. Instances CW1–CW11 are generated as CU, with values v_i taken from the interval [100, 1000].

6.1.2. Constructive algorithm

The constructive algorithm in Section 2 involves the decision about which bound to use. In a preliminary experiment we have solved the test problems with both bounds. A summary of the results appear in Table 1, where in *CONS1* the constructive algorithm uses BK_1 and in *CONS2* uses BK_2. Over the 60 problems, *CONS1* obtains better solutions than *CONS2* 18 times, there are 5 ties and on 37 occasions *CONS2* obtains better solutions than *CONS1*. There is no relation between the algorithm producing the best solution and the type of problem being solved, constrained or unconstrained, weighted or unweighted. As the algorithms are very fast, we therefore propose using both bounds, building two solutions and keeping the best of them. The results of this combined strategy appear in row *CONS12* in Table 1 where for all the problem instances, the unconstrained and the constrained instances, we show the number of optimal solutions obtained by each algorithm, the average ratio and the minimum ratio of the solutions with respect to the optimum. The values obtained for *CONS12* on each test problem appear in column 3 of Tables 3 and 4 under the heading *CONS*.

6.1.3. GRASP algorithm

The implementation of the *GRASP* algorithm involves two decisions: the way of choosing the next piece to be cut, and the number of iterations to do. The piece to be cut is randomly selected

Table 1
Computational results of constructive algorithms

	All problem instances			Unconstrained instances			Constrained instances		
	Optima	Average ratio	Minimum ratio	Optima	Average ratio	Minimum ratio	Optima	Average ratio	Minimum ratio
CONS1	5	0.9444	0.8156	3	0.9403	0.8156	2	0.9487	0.8249
CONS2	14	0.9630	0.8601	10	0.9750	0.8995	4	0.9501	0.8601
CONS12	15	0.9719	0.8658	10	0.9796	0.8995	5	0.9637	0.8658

Table 2
Computational results of *GRASP* algorithms

	All problem instances			Unconstrained instances			Constrained instances		
	Optima	Average ratio	Minimum ratio	Optima	Average ratio	Minimum ratio	Optima	Average ratio	Minimum ratio
GRASP1/100	28	0.9895	0.9091	14	0.9904	0.9486	14	0.9886	0.9091
GRASP2/100	22	0.9911	0.9346	14	0.9928	0.9604	8	0.9892	0.9346
GRASP12/50	32	0.9951	0.9234	18	0.9969	0.9806	14	0.9932	0.9234

from the set of pieces $S_{\max} = \{i/B_i \geqslant \delta B_{\max}\}$. In order to find the right value of δ we have explored three strategies:

- only high values of δ: δ varies among 0.99, 0.975, 0.95, 0.90,
- a more extended range of values of δ: δ varies among 0.95, 0.90, 0.75, 0.66,
- a different strategy with $\delta = 0$, in which all the pieces may be chosen, but with a probability proportional to their bound.

The best results were obtained in the first case and these values of δ are used.

The stopping criterion is set to 100 iterations without improving the best-known solution.

In a preliminary experiment we have used both bounds BK_1 and BK_2 in the constructive and improvement phases. A summary of the results appears in Table 2, where *GRASP1*/100 uses BK_1 and *GRASP2*/100 uses BK_2. The results are quite complementary. In 22 problems *GRASP1*/100 obtained better solutions than *GRASP2*/100 and 20 times *GRASP2*/100 performed better than *GRASP1*/100, with 18 ties. Therefore, we propose using both bounds, but in order to maintain speed, we change the stopping criterion for each bound to 50 iterations without improving the best known solution. The results of the combined algorithm appear in row *GRASP12*/50 of Table 2 and in column 4 of Tables 3 and 4 under the heading *GRASP*.

6.1.4. Tabu search algorithm

The stopping criterion for the tabu search algorithm was set to a maximum number of iterations of 500. This limit allows us to keep the average computing times below 1 min of CPU.

The length of the tabu list varies dynamically. If m^* is the square root of the number of pieces, the length is randomly chosen in the interval [0.5 m*, 1.5 m*]. The change of tabu list is performed after 50 iterations without improving the best current solution.

After 100 iterations without improving we begin the diversification strategy with values of β which start at $\beta = -1$ and decrease 0.5 every 50 iterations without improving up to a lowest value of $\beta = -3$. When the diversification strategy has finished, the remaining iterations are used in an intensification strategy, starting with $\beta = 1$ and increasing it by 0.5 every 50 iterations without improving.

Each move of the tabu search process performs 10 embedded *GRASP* iterations using BK_1. The use of BK_2 produces similar results and it is computationally slower.

The results appear in column 7 of Tables 3 and 4 under the heading *TS500*.

Table 3
Computational results on Fayard et al. [1] unconstrained problem instances

Instance	Optimum	*CONS*	*GRASP*	*GR/PR*	*TS500*	*FHZ*
H	12348	11922	12348[a]	12348[a]	12348[a]	12192
HZ1	5226	5226[a]	5226[a]	5226[a]	5226[a]	5226[a]
M1	15024	15024[a]	15024[a]	15024[a]	15024[a]	15024[a]
M2	73176	72564	72564	72564	73176[a]	72564
M3	142817	134179	142735	142817[a]	142817[a]	142817[a]
M4	265768	265768[a]	265768[a]	265768[a]	265768[a]	265768[a]
M5	577882	558303	577882[a]	577882[a]	577882[a]	577882[a]
B	8997780	8913068	8913068	8980610	8997780[a]	8997780[a]
UU1	242919	237602	242201	242919[a]	242919[a]	241260
UU2	595288	554409	595288[a]	595288[a]	595288[a]	595288[a]
UU3	1072764	1008401	1056974	1065051	1072764[a]	1072764[a]
UU4	1179050	1162180	1179050[a]	1179050[a]	1179050[a]	1178295
UU5	1868999	1800588	1867816	1868985	1868985	1868985
UU6	2950760	2944368	2950760[a]	2950760[a]	2950760[a]	2950760[a]
UU7	2930654	2924000	2924000	2924000	2930654[a]	2930654[a]
UU8	3959352	3930800	3945648	3959171	3959352[a]	3959352[a]
UU9	6100692	6100692[a]	6100692[a]	6100692[a]	6100692[a]	6100692[a]
UU10	11955852	11723195	11928699	11955852[a]	11955852[a]	11955852[a]
UU11	13157811	12721274	12949529	12987419	13091814	13141175
HZ2	8226	8148	8226[a]	8226[a]	8226[a]	8046
UW1	6036	6036[a]	6036[a]	6036[a]	6036[a]	6036[a]
UW2	8468	8384	8468[a]	8468[a]	8468[a]	8468[a]
UW3	6302	6226	6226	6226	6226	6226
UW4	8326	8326[a]	8326[a]	8326[a]	8326[a]	8326[a]
UW5	7780	7780[a]	7780[a]	7780[a]	7780[a]	7780[a]
UW6	6615	5950	6487	6548	6615[a]	6615[a]
UW7	10464	10464[a]	10464[a]	10464[a]	10464[a]	10464[a]
UW8	7692	7046	7692[a]	7692[a]	7692[a]	7692[a]
UW9	7038	7038[a]	7038[a]	7038[a]	7038[a]	7038[a]
UW10	7507	7507[a]	7507[a]	7507[a]	7461	7507[a]
UW11	15747	15680	15680	15747[a]	15747[a]	15747[a]

[a]Denotes the optimum.

6.1.5. Path relinking

We have applied path relinking to a set of 10 reference solutions obtained by the *GRASP* algorithm *GRASP12*/50. The results appear in column 5 of Tables 3 and 4 under the heading *GR*/*PR*, showing an improvement in the solutions for unconstrained and constrained test problems.

When we use path relinking after the tabu algorithm mentioned in the above subsection *TS500*, with diversification and intensification phases, we do not get any improvement.

In all cases path relinking uses BK_1 in the constructive phase of embedded GRASP. The use of BK_2 does not improve the overall performance of the procedure.

Table 4
Computational results on Fayard et al. [1] constrained problem instances

Instance	Optimum	*CONS*	*GRASP*	*GR/PR*	*TS500*	*FHZ*
OF1	2737	2737[a]	2737[a]	2737[a]	2737[a]	2713
OF2	2690	2565	2690[a]	2690[a]	2690[a]	2586
W	2721	2614	2721[a]	2721[a]	2721[a]	2721[a]
CU1	12330	11826	12312	12312	12330[a]	12312
CU2	26100	25934	26100[a]	26100[a]	26100[a]	25806
CU3	16723	15964	16652	16652	16679	16608
CU4	99495	96739	99264	99264	99366	98190
CU5	173364	172160	173364[a]	173364[a]	173364[a]	171651
CU6	158572	156295	158572[a]	158572[a]	158572[a]	158572[a]
CU7	247150	232416	247150[a]	247150[a]	247150[a]	246860
CU8	433331	432198	432714	432714	432714	432198
CU9	657055	631510	651597	657055[a]	657055[a]	657055[a]
CU10	773772	759159	767580	770659	773485	764696
CU11	924696	897151	909898	914399	922161	913387
CHW1	2892	2625	2892[a]	2892[a]	2892[a]	2731
CHW2	1860	1860[a]	1860[a]	1860[a]	1860[a]	1740
TH1	4620	4320	4500	4580	4620[a]	4620[a]
TH2	9700	9011	9525	9560	9700[a]	9529
CW1	6402	6402[a]	6402[a]	6402[a]	6402[a]	6402[a]
CW2	5354	5150	5333	5333	5354[a]	5354[a]
CW3	5689	5689[a]	5689[a]	5689[a]	5689[a]	5148
CW4	6175	6031	6158	6165	6170	6168
CW5	11659	11580	11580	11580	11644	11550
CW6	12923	12907	12907	12923[a]	12923[a]	12403
CW7	9898	9484	9687	9687	9898[a]	9484
CW8	4605	4140	4605[a]	4605[a]	4605[a]	4504
CW9	10748	10748[a]	10748[a]	10748[a]	10748[a]	10748[a]
CW10	6515	5866	6016	6515[a]	6515[a]	6116
CW11	6321	5473	6321[a]	6321[a]	6321[a]	6084

[a]Denotes the optimum.

6.1.6. Summary and comparison of results

The computational results of Tables 3 and 4 are summarised in Table 5.

The constructive algorithm, based only on one-dimensional knapsack bounds, produces good quality results on average, 97.19% from optimum, but occasionally it can obtain poor results on some problems, with a worst-case of 86.58%. It can be considered a useful starting point for more complex procedures.

The *GRASP* algorithm is very fast (6.38 s on average with a largest value of 21.15 s) and produces good results, with 32 optimal solutions and an overall percentage of 99.51% with respect to the optimum. Moreover, the results are good for both types of problems, unconstrained problems for which it gets an average of 99.69% and constrained problems, with an average of 99.32%. Therefore, this algorithm appears to be a good choice when a fast and reliable solution is needed.

Table 5
Summary of computational results on Fayard et al. [1] instances

	All problem instances			Unconstrained instances			Constrained instances		
	Optima (of 60)	Average ratio	Minimum ratio	Optima (of 31)	Average ratio	Minimum ratio	Optima (of 29)	Average ratio	Minimum ratio
CONS	15	0.9719	0.8658	10	0.9796	0.8995	5	0.9637	0.8658
GRASP	32	0.9951	0.9346	18	0.9969	0.9806	14	0.9932	0.9346
GR/PR	39	0.9977	0.9787	22	0.9982	0.9871	17	0.9972	0.9787
TS500	49	0.9994	0.9879	27	0.9993	0.9879	22	0.9996	0.9973
FHZ	30	0.9895	0.9049	23	0.9979	0.9781	7	0.9806	0.9049

The results of *GRASP* algorithm are improved by adding a path relinking phase. The computation times increase (10.73 s on average, with a largest value of 57.13 s) and the quality is higher, with an overall percentage of 99.77% and a worst case of 97.87%. These results are better than those obtained by directly using *GRASP* with a larger number of iterations. More concretely, if we use *GRASP* with 100 iterations for each bound, in an average computing time of 11.72 s the average percentage with respect to the optimum is 99.68% and the number of optimal solutions obtained is 36. Therefore, it seems that using path relinking diversifies the search and allows us to get better solutions.

The tabu search algorithm *TS500*, with diversification and intensification phases, needs longer computing times (59.04 s on average, with a largest value of 171.68 s) but it obtains very high-quality results even in the worst cases. Fortynine out of 60 optimal solutions are attained and the overall percentage with respect to the optimum is 99.94%, with a worst case of 98.79%. These good results are similar for both unconstrained, with a percentage of 99.93% and 27 optima out of 31 instances, and constrained problems, with a percentage of 99.96% and 22 optima out of 29 instances.

Comparing the results of our tabu search algorithm with those obtained by the algorithm *FHZ* (Fayard et al. [1]), we see that the results are similar for unconstrained problems, for which the lower bound that they propose works extremely well. However, our results are clearly better for constrained problems, maintaining the high percentages and the high number of optimal solutions. Therefore, our tabu search algorithm appears to be a good choice for problems in which a high-quality solution is needed, even though it takes longer than GRASP algorithms. A direct comparison with the computing times of *FHZ* algorithm cannot be made, because it was run on a different computer (Sparc-Server20, module 712p). On that machine, the *FHZ* algorithm needed an average time of 58.8 s, with a maximum of 461.1 s.

6.2. *The test problems of Hifi* [11]

We have also compared our algorithms with algorithm DH/KD, developed by Hifi [11] for unconstrained problems.

Table 6
Computational results on Hifi [11] unconstrained instances

Instance	Optimum	*CONS*	*GRASP*	*GRAS/PR*	*TABU500*	*DH/KD*
H	12348	11922	12348[a]	12348[a]	12348[a]	12348[a]
M1	15024	15024[a]	15024[a]	15024[a]	15024[a]	15024[a]
M2	73176	72564	72564	72564	73176[a]	73176[a]
M3	142817	134179	142735	142817[a]	142817[a]	142817[a]
M4	265768	265768[a]	265768[a]	265768[a]	265768[a]	265768[a]
M5	577882	558303	577882[a]	577882[a]	577882[a]	577882[a]
B	8997780	8913068	8913068	8980610	8997780[a]	8997780[a]
U1	22370130	21404746	21823635	22117255	22317702	22370130[a]
U2	20232224	19511607	19967604	19986452	20168944	20192444
U3	48142840	45452673	46888786	47432733	47916436	48142836
W1	35159	34496	35159[a]	35159[a]	35159[a]	35159[a]
W2	162867	154096	159682	161424	161424	162867[a]
W3	234108	234108[a]	234108[a]	234108[a]	234108[a]	234108[a]
C1	249	248	249[a]	249[a]	249[a]	249[a]
C2	3076	2971	3076[a]	3076[a]	3076[a]	3076[a]
C3	2240	2240[a]	2240[a]	2240[a]	2240[a]	2240[a]

[a] Denotes the optimum.

Table 7
Summary of computational results on Hifi [11] unconstrained instances

	Results			Times	
	Optima (out of 16)	Average ratio	Minimum ratio	Average	Maximum
CONS	4	0.9755	0.9395	0.05	0.29
GRASP	9	0.9937	0.9740	3.29	15.81
GRASP/PR	10	0.9964	0.9852	9.95	47.89
TABU500	12	0.9988	0.9911	26.19	120.34
DH/KD	14	0.9999	0.9980		

In Table 6, problem H is taken from Hertz [19], M1–M5 from Morabito et al. [9], and B from Beasley [2]. Problems U1–U3 and W1–W3 are six randomly generated instances and C1–C3 are taken from Christofides and Whitlock [4]. Some of these problems have already been used in the previous section, but are included again to make a complete comparison with DH/KD algorithm.

The results of our algorithms and DH/KD appear in Table 6. In some cases the results of DH/KD are different from those published in [11] and have been provided directly by the author.

A summary of the results appears in Table 7. The performance of our tabu search algorithm is slightly inferior to that of algorithm DH/KD which has been considered as the best existing heuristic algorithm for unconstrained problems.

6.3. The test problems of Cung et al. [23]

Cung et al. [23] have recently developed a branch and bound algorithm for constrained two-dimensional problems. In order to test its performance, they have used 27 instances of medium size and nine large-scale problems.

The 27 medium-size instances include weighted and unweighted problems. For the weighted version they consider 10 instances (see Table 8). Four instances, P2, P3, A1, A2, are taken from [7].

Table 8
Computational results on Cung et al. [23] constrained problems

Instance	Optimum	*CONS*	*GRASP*	*GRASP/PR*	*TABU500*
P2	2892	2625	2892[a]	2892[a]	2892[a]
P3	1860	1860[a]	1860[a]	1860[a]	1860[a]
A1	2020	1800	1900	2020[a]	2020[a]
A2	2505	2285	2395	2435	2455
STS2	4620	4320	4500	4580	4620[a]
STS4	9700	9011	9525	9560	9700[a]
CHL1	8671	8400	8484	8518	8660
CHL2	2326	2326[a]	2326[a]	2326[a]	2326[a]
CHL3	5283	5283[a]	5283[a]	5283[a]	5283[a]
CHL4	8998	8998[a]	8998[a]	8998[a]	8998[a]
P2s	2778	2473	2740	2778[a]	2778[a]
P3s	2721	2614	2721[a]	2721[a]	2721[a]
A1s	2950	2909	2950[a]	2950[a]	2950[a]
A2s	3535	3312	3535[a]	3535[a]	3535[a]
STS2s	4653	4511	4653[a]	4653[a]	4653[a]
STS4s	9770	9335	9583	9642	9770[a]
CHL1s	13099	12864	13014	13040	13099[a]
CHL2s	3279	3162	3231	3239	3279[a]
CHL3s	7402	7402[a]	7402[a]	7402[a]	7402[a]
CHL4s	13932	13932[a]	13932[a]	13932[a]	13932[a]
A3	5451	5081	5410	5410	5436
A4	6179	5645	6052	6052	6179[a]
A5	12985	12662	12832	12832	12929
H	11586	10779	11391	11391	11586[a]
CHL5	390	335	390[a]	390[a]	390[a]
CHL6	16869	16448	16643	16723	16869[a]
CHL7	16881	16338	16583	16814	16838
Hchl1	11303	10096	10829	11005	11089
Hchl2	9954	9553	9691	9840	9918
Hchl3s	12215	11638	12159	12209	12208
Hchl4s	11994	11311	11621	11739	11967
Hchl5s	45361	43831	44346	44616	45223
Hchl6s	61040	58373	60403	60708	61002
Hchl7s	63112	59701	62547	62806	62802
Hchl8s	911	858	904	904	904
Hchl9	5240	4840	5240[a]	5240[a]	5240[a]

[a] Denotes the optimum.

Table 9
Summary of computational results on Cung et al. [23] constrained instances

	Results			Times	
	Optima (out of 36)	Average ratio	Minimum ratio	Average	Maximum
CONS	6	0.9507	0.8590	0.05	0.17
GRASP	13	0.9871	0.9406	2.66	10.27
GRASP/PR	15	0.9927	0.9721	6.09	21.81
TABU500	23	0.9980	0.9800	45.04	172.47

Table 10
Computational results on unconstrained random problems

Instance	Optimum	*CONS*	*GRASP*	*GRASP/PR*	*TABU500*
APT10	3589703	3445765	3482995	3526455	3585450
APT11	4188915	4020581	4097432	4133554	4148798
APT12	5156065	4858283	5057137	5062397	5137069
APT13	3498302	3305548	3409771	3477010	3483722
APT14	4463550	4283559	4369811	4415090	4463550
APT15	6047188	5830797	5923859	5973430	5997899
APT16	7566719	7198379	7356243	7472065	7513717
APT17	4535302	4282976	4445425	4467820	4512417
APT18	5825956	5730322	5730322	5779344	5759831
APT19	6826674	6435029	6637529	6735402	6763810
APT20	5545818	5418829	5521885	5521885	5521885
APT21	3484406	3311882	3465637	3470409	3484406[a]
APT22	4145317	3922596	4047968	4079956	4116075
APT23	3546535	3469159	3480665	3516524	3535623
APT24	3948037	3889970	3923776	3923776	3939485
APT25	3507615	3373350	3433221	3462778	3500380
APT26	2683689	2548943	2607788	2632076	2656729
APT27	2438174	2316597	2399928	2410836	2435046
APT28	4065011	3821688	3963766	3997097	4065011[a]
APT29	3652858	3616335	3649004	3652858[a]	3652858[a]

[a] Denotes the optimum.

Two other instances are taken from [22]. CHL1 to CHL4 are randomly generated. For the unweighted version of the problem they consider 17 instances. P2s, P3s, A1s, A2s, STS2s, STS4s and CHL1s to CHL4s represent exactly the weighted instances for which the value of each piece is represented by its area. A3, A4, A5 and H are taken from [7], and CHL5 to CHL7 are randomly generated. There are also nine large and more complex instances, Hchl1 to Hchl9, which are randomly generated.

For the random instances, the dimensions l_i and w_i of pieces to cut are taken uniformly from the intervals $[0.1L, 0.75L]$ and $[0.1W, 0.75W]$, respectively. The weight associated to a piece i is

computed by $c_i = \lceil \rho l_i w_i \rceil$, where $\rho = 1$ for the unweighted case and $\rho \in [0.25, 0.75]$ for the weighted case. The constrains b_i have been chosen such that $b_i = \min\{\rho_1, \rho_2\}$, where $\rho_1 = \lfloor L/l_i \rfloor \lfloor W/w_i \rfloor$ and ρ_2 is a number randomly generated in the interval [1,10].

The results of our algorithms appear in Table 8. The optimal solutions are taken from [23]. We do not know of any heuristic algorithms tested on these problems.

A summary of the results appears in Table 9. Note that the performance of the tabu search algorithm is remarkably good for these constrained instances. More concretely, on the last nine large-scale complex problems the tabu search algorithm gets an average ratio of 0.9954 with a worst result of 0.9811.

6.4. Some new randomly generated problems

We have also generated a set of large-scale test problems, covering the four versions of the problem, constrained and unconstrained, weighted and unweighted. For unconstrained problems, the number of pieces is taken from the interval [30, 60], the dimensions of the stock L, W, from [1500, 3000] and the dimensions of the pieces, l_i, w_i, from the intervals $[0.05L, 0.4L]$ and $[0.05W, 0.4W]$, respectively. For the unweighted problems, APT10 to APT19, the value of each piece is its area. For the weighted instances, APT20 to APT29, the value $c_i = \lceil \rho l_i w_i \rceil$, where ρ is taken from [0.25, 0.75].

For the constrained problems, the number of pieces is taken uniformly from the interval [25, 60]. L and W are taken from [100, 1000], l_i from $[0.05L, 0.4L]$ and w_i from $[0.05W, 0.4W]$. The

Table 11
Computational results on constrained random problems

Instance	Optimum	*CONS*	*GRASP*	*GRASP/PR*	*TABU500*
APT30	140904	136569	138863	138863	140144
APT31	822393 +	780406	801767	804476	814081
APT32	38068	36391	37786	37810	38030
APT33	236611 +	229464	231178	231367	234920
APT34	360084 +	343408	353822	353822	360084
APT35	620797 +	601158	607864	612997	620700
APT36	130744	127333	129634	130156	130338
APT37	387276	361030	379329	379329	381966
APT38	261154 +	252664	252605	252886	259380
APT39	268750	257879	263076	263471	267168
APT40	66362 +	62134	65002	65002	66362
APT41	206542 +	195578	199349	200826	206542
APT42	33435 +	32035	32644	32714	33435
APT43	214651 +	199756	206627	206627	214651
APT44	73410 +	68848	72261	72906	73410
APT45	74691 +	72967	73210	73231	74691
APT46	149911 +	144558	147340	147387	149911
APT47	148764 +	139766	148638	148638	148764
APT48	166927 +	154036	163484	166115	166927
APT49	215728 +	200513	209472	211570	215728

Table 12
Summary of computational results on randomly generated instances

	Results			Times	
	Optima (out of 25)	Average ratio	Minimum ratio	Average	Maximum
CONS	0	0.9589	0.9322	0.4	0.7
GRASP	0	0.9820	0.9703	23.7	50.6
GRASP/PR	1	0.9883	0.9795	63.7	131.8
TABU500	4	0.9953	0.9863	210.9	450.4

bounds $b_i = \min\{\rho_1, \rho_2\}$, where $\rho_1 = \lfloor L/l_i \rfloor \lfloor W/w_i \rfloor$ and ρ_2 is a number randomly generated in the interval [1,10]. Problems APT30 to APT39 are unweighted, with piece values equal to their areas, and problems APT40 to APT49 are weighted, with values defined as for the unconstrained case.

The computational results appear in Tables 10 and 11. The column under the heading of *Optimum* contains in some cases only the best known solutions, without proof of optimality. These cases have been marked with a " + " sign. The optimal solutions of unconstrained problems have been obtained by allowing long runs of Beasley's algorithm [2]. The optimal solutions of constrained unweighted problems have been obtained by using Wang's algorithm [13] with an iterative tuning of parameters.

A summary of the results appears in Table 12. The results of the first part of the table involve only those problems for which the optimal solution is known. The computing times have been calculated for the whole set of 40 instances.

7. Conclusions

We have developed *GRASP* and tabu search general purpose algorithms for solving large-scale two-dimensional problems. The computational results show that these meta-heuristic procedures obtain high-quality results for these problems. If fast solutions are needed, the *GRASP* algorithm should be used, coupled if possible with a path relinking procedure. If high quality solutions are preferred to speed, the more complex tabu search algorithm should be chosen.

References

[1] Fayard D, Hifi M, Zissimopoulos V. An efficient approach for large-scale two-dimensional guillotine cutting stock problems. Journal of the Operational Research Society 1998;49:1270–7.
[2] Beasley JE. Algorithms for unconstrained two-dimensional guillotine cutting. Journal of the Operational Research Society 1985;36:297–306.
[3] Hifi M, Zissimopoulos V. A recursive exact algorithm for weighted two-dimensional cutting. European Journal of Operational Research 1996;91:553–64.

[4] Christofides N, Whitlock C. An algorithm for two-dimensional cutting problems. Operations Research 1977;25:30–44.
[5] Viswanathan KV, Bagchi A. Best-first search methods for constrained two-dimensional cutting stock problems. Operations Research 1993;41:768–76.
[6] Christofides N, Hadjiconstantinou E. An exact algorithm for orthogonal 2-D cutting problems using guillotine cuts. European Journal of Operational Research 1995;83:21–38.
[7] Hifi M. An improvement of Viswanathan and Bagchi's exact algorithm for cutting stock problems. Computers & Operations Research 1997;24:727–36.
[8] Hifi M, Zissimopoulos V. Constrained two-dimensional cutting: an improvement of Chrisfofides and Whitlock's exact algorithm. Journal of Operational Research Society 1997;48:324–31.
[9] Morabito RN, Arenales MN, Arcaro VF. An and-or-graph approach for two-dimensional cutting problems. European Journal of Operational Research 1992;58:263–71.
[10] Fayard D, Zissimopoulos V. An approximation algorithm for solving unconstrained two-dimensional knapsack problems. European Journal of Operational Research 1995;84:618–32.
[11] Hifi M. The DH/KD algorithm: a hybrid approach for unconstrained two-dimensional cutting problems. European Journal of Operational Research 1997;97:41–52.
[12] Morabito RN, Arenales MN. Staged and constrained two-dimensional guillotine cutting problems: an AND/OR-graph approach. European Journal of Operational Research 1996;94:548–60.
[13] Wang PY. Two algorithms for constrained two-dimensional cutting stock problems. Operations Research 1983;31:573–86.
[14] Hifi H, Ouafi R. Best-first search and dynamic programming methods for cutting problems: the cases of one or more stock plates. Computers & Industrial Engineering 1997;32:187–205.
[15] Martello S, Toth P. Knapsack problems. Algorithms and computer implementations. Chichester: Wiley, 1990.
[16] Feo T, Resende MGC. A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters 1989;8:67–71.
[17] Feo T, Resende MGC. Greedy randomized adaptive search procedures. Journal of Global Oprimization 1995;2:1–27.
[18] Glover F, Laguna M. Tabu search. Boston: Kluwer Academic Publishers, 1997.
[19] Herz J. A recursive computing procedure for two-dimensional stock cutting. IBM Journal of Research Development 1972;16:462–9.
[20] Hifi M, Zissimopoulos V. Une amelioration de l'algorithme recursif de Herz por le probleme de decoupe a deux dimensions. RAIRO 1996;30:111–25.
[21] Oliveira JF, Ferreira JS. An improved version of Wang's algorithm for two-dimensional cutting problems. European Journal of Operational Research 1990;44:256–66.
[22] Tschöke S, Holthöfer N. A new parallel approach to the constrained two-dimensional cutting stock problem. Technical Report, University of Padeborn, D.C.S. 33095 Padeborn. Germany, 1996.
[23] Cung VD, Hifi M, Le Cun B. Constrained two-dimensional cutting stock problems. A best-first branch-and-bound algorithm. International Transactions in Operational Research 2000;7:185–210.

Ramon Alvarez-Valdes is Professor in the Statistics and Operations Research Department at the University of Valencia, Spain. He received a Ph. D. in Mathematics from the University of Valencia. His main area of research is combinatorial optimization.

Antonio Parajon is Professor in the Department of Mathematics, National Autonomous University of Nicaragua. Currently he is working on his Ph. D. at the Department of Statistics and Operations Research, University of Valencia, developing algorithms for combinatorial optimization problems.

Jose M. Tamarit is Professor in the Department of Statistics and Operations Research at the University of Valencia. He received a Ph. D. in Mathematics from the University of Valencia. His main area of research is combinatorial optimization.