A HYBRID HEURISTIC FOR THE CONSTRAINED TWO-DIMENSIONAL NON-GUILLOTINE ORTHOGONAL CUTTING PROBLEM

JOSÉ FERNANDO GONÇALVES AND MAURICIO G. C. RESENDE

ABSTRACT. This paper addresses a constrained two-dimensional (2D) non-guillotine cutting problem, where a fixed set of small rectangles has to be cut from a larger stock rectangle so as to maximize the value of the rectangles cut. The algorithm we propose hybridizes a novel placement procedure with a genetic algorithm based on random keys. We propose also a new fitness function to drive the optimization. The approach is tested on a set of instances taken from the literature and compared with other approaches. The experimental results validate the quality of the solutions and the effectiveness of the proposed algorithm.

1. INTRODUCTION

The constrained two-dimensional (2D) non-guillotine cutting problem addressed in this paper consists of cutting rectangular pieces from a large rectangular sheet of material (stock rectangle) in order to maximize the value of the rectangles cut. We consider the special case in which the items cannot be rotated and must be cut with their edges always parallel to the edges of the large rectangular sheet. The problem is relevant both from a theoretical point of view as well as practical one. It is NP-hard (Garey and Johnson, 1979) and arises in various production processes with applications varying from the home-textile to the glass, steel, wood, and paper industries, where rectangular figures are cut from large rectangular sheets of materials.

Various types of two-dimensional cutting problems have been considered in the literature. Dyckhoff (1990) provided a classification of the various types of cutting problems. Surveys for multidimensional cutting and packing problems are given by Dowsland and Dowsland (1992), Haessler and Sweeney (1991), and Sweeney and Paternoster (1992). Few authors have considered general two-dimensional non-guillotine cutting problems (constrained or unconstrained). The unconstrained non-guillotine cutting problem has been considered by a few authors, namely: Tsai et al. (1988) presented an integer programming approach; Arenales and Morabito (1995) developed an approach based on an AND/OR graph together with a branch and bound search; and Healy et al. (1999) introduced an algorithm based on the identification of the empty space that can be used to cut new items.

Optimal procedures for the constrained two-dimensional non-guillotine cutting problem have been proposed by several authors. Beasley (1985b) proposed a branch and bound algorithm with an upper bound derived from a Lagrangean relaxation of a 0-1 integer linear programming formulation. Scheithauer and Terno (1993) presented an integer programming formulation in which binary variables are used to indicate whether a piece item is cut above/below or to the right/left of another piece. Hadjiconstantinou and Christofides

Date: October 17, 2006.

Key words and phrases. Cutting, packing, two-dimensional packing, two-dimensional cutting, non-guillotine cuting, genetic algorithm.

AT&T Labs Research Technical Report TD-6UNQN6.

(1995) developed a branch and bound algorithm in which the search was limited by using an upper bound based on a Lagrangean relaxation procedure and improved it using subgradient optimization. Computational gains were achieved by applying new reduction tests. Fekete and Schepers (1997a;b;c; 2004a;b) developed a two-level tree search algorithm for solving the d-dimensional knapsack problem. In this algorithm, projections of cut items were made onto both the horizontal and vertical edges of the stock rectangle. These projections were represented by graphs in which the nodes are the cut items and an edge joins two nodes if the projections of the corresponding cut items overlap. By looking at the properties of the graphs, the authors were able to check the feasibility of the corresponding patterns. Amaral and Letchford (2001) presented an upper bound which involved the solution of a large linear program by a column generation algorithm. Boschetti et al. (2002) proposed new upper bounds derived from different relaxations of a new integer programming formulation of the constrained two-dimensional non-guillotine cutting problem. The new formulation is based on the observation that any feasible solution can be represented by two sequences in which each element is the subset of items covering the x and y positions of the master surface, respectively. Caprara and Monaci (2004) compared four new algorithms based on the natural relaxation of the two-dimensional knapsack problem. In the relaxation, the knapsack has a capacity equal to the area of the master surface and the item weights are equal to their areas.

Heuristic procedures for the constrained two-dimensional non-guillotine cutting problem have also been developed by several authors. Lai and Chan (1997b) presented a heuristic based on simulated annealing. They used a problem representation which encodes the order in which pieces should be cut. Lai and Chan (1997a) proposed a heuristic based on an evolutionary strategy approach. They used the same representation as in Lai and Chan (1997b). Their algorithm includes an improvement procedure based on dividing the ordered list into active (cut) and inactive (uncut) pieces and seeing if any (currently) uncut pieces can be cut. Their mutation process involves swapping two pieces in the ordered list. Leung et al. (2001) used the ordered representation of Lai and Chan (1997b) and discussed producing a cutting pattern from it using both the difference process algorithm of Lai and Chan (1997b) and a standard bottom-left algorithm (see Jakobs (1996)). They showed that there exist problems for which the optimal solution cannot be found by these two approaches. They presented a simulated annealing heuristic with a move corresponding to either swapping two pieces in the ordered representation or moving a single piece to a new position in the representation. They also presented a genetic algorithm involving five different crossover operators. In an attempt to alleviate the problem of premature convergence, Leung et al. (2003) developed a hybrid heuristic blending simulated annealing with a genetic algorithm. Beasley (2004) proposes a genetic algorithm based on a non-linear formulation of the problem, where variables indicate if a piece is cut or not and its position on the stock sheet. No placement algorithm is needed since the solutions are lists of variables and show directly the cutting pattern. Alvarez-Valdes et al. (2005) developed a new heuristic based on GRASP (Feo and Resende, 1989; 1995) for the non-guillotine two-dimensional cutting stock problem. The constructive phase explicitly considers the possibility of simultaneously cutting several pieces of the same type and forming a block, as one would do in the pallet loading problem.

In this paper, we present a hybrid heuristic for a two-dimensional non-guillotine cutting problem which combines a random-keys based genetic algorithm with a novel fitness function and a new heuristic placement policy. The remainder of the paper is organized as follows. In Section 2, we define the problem formally, and in Section 3, propose the

new approach, describing the genetic algorithm, the placement strategy, and the fitness function. In Section 4, we present experimental results and in Section 5 make concluding remarks.

2. THE PROBLEM

The two-dimensional non-guillotine cutting problem addressed in this paper is the problem of cutting from a single large planar stock rectangle (W,H), of width W and height H, smaller rectangles (w_i,h_i), $i = 1,\ldots,n$, each of width w_i and height h_i. Each rectangle i has a fixed orientation (i.e. cannot be rotated); must be cut (by infinitely thin cuts) with its edges parallel to the edges of the stock rectangle (i.e. orthogonal cuts); and the number x_i of pieces of each rectangle type that are cut must lie between P_i and Q_i, i.e. $0 \leq P_i \leq x_i \leq Q_i$, for all $i = 1,\ldots,n$. Each rectangle $i = 1,\ldots,n$ has an associated value equal to v_i and the objective is to maximize the total value of the rectangles cut $\sum_{i=1}^{n} v_i x_i$. Without significant loss of generality, it is usual to assume that all dimensions W,H, and (w_i,h_i), $i = 1,\ldots,n$, are integers. To simplify notation, we shall use $M = \sum_{i=1}^{n} Q_i$ as the maximum number of pieces that can be cut. According to the newly developed typology by Wäscher et al. (2006), the problem presented in this paper falls into the *output maximization assignment* kind and belongs to the *two-dimensional rectangular SKP* problem type.

Depending on the values of P_i and Q_i, we can distinguish the following three types of problems:

(1) In the *unconstrained* type, we have $P_i = 0$, $Q_i = \lfloor (W \times H)/(w_i \times h_i) \rfloor$, for $i = 1,\ldots,n$, which is a trivial bound;
(2) In the *constrained* type, we have: for all $i = 1,\ldots,n$, $P_i = 0$ and $\exists\ j \in \{1,\ldots,n\}$ such that $Q_j < \lfloor (W \times H)/(w_j \times h_j) \rfloor$;
(3) In the *doubly constrained* type, we have: $\exists\ i \in \{1,\ldots,n\}$ such that $P_i > 0$ and $\exists\ j \in \{1,\ldots,n\}$ such that $Q_j < \lfloor (W \times H)/(w_j \times h_j) \rfloor$.

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

$$
\begin{aligned}
&\max \sum_{i=1}^{n} v_i x_i + \sum_{i=1}^{n} v_i P_i \\
&\text{subject to} \\
&\sum_{i=1}^{n} (w_i \cdot h_i)\, x_i \leq W \cdot H - \sum_{i=1}^{n} (w_i \cdot h_i)\, P_i \\
&x_i \leq Q_i - P_i,\ i = 1, \ldots, n, \\
&x_i \geq 0, \text{ integer}, \quad i = 1, \ldots, n.
\end{aligned}
$$

This simple upper bound will be used to evaluate the performance of the hybrid genetic algorithm described in this paper.

3. NEW APPROACH

The new approach proposed in this paper combines a random-keys based genetic algorithm, a new placement strategy, and a novel measure of solution quality that we call *modified total value*. Instead of describing the algorithms directly as cutting problems, we use the equivalent notion of packing, where we are giving rectangles and wish to pack them on the stock rectangle.

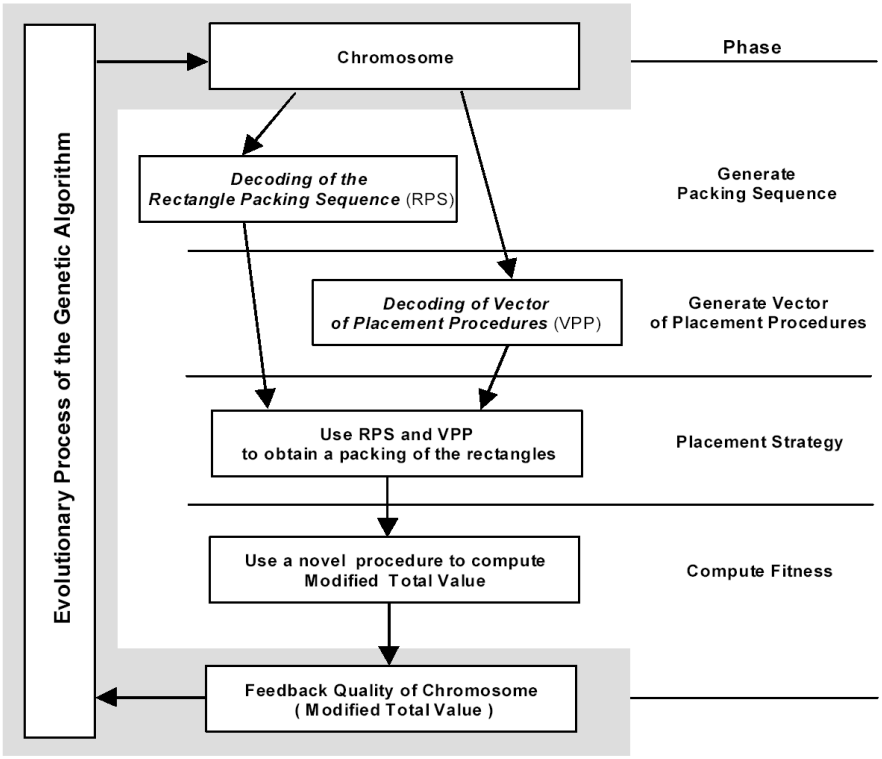

FIGURE 1. Architecture of the new hybrid heuristic.

The role of the genetic algorithm is to evolve the encoded solutions, or *chromosomes*, which represent the rectangle packing sequence and the type of placement procedure used to place each rectangle. For each chromosome, the following four phases are applied:

(1) *Decoding of the rectangle packing sequence*. The first phase is responsible for transforming part of the chromosome supplied by the genetic algorithm into the sequence in which the rectangles are to be packed in the stock rectangle.
(2) *Decoding of the placement procedure*. In the second phase, part of the chromosome supplied by the genetic algorithm is transformed into a vector that indicates which placement procedure is to be used to position each rectangle in the stock rectangle.
(3) *Placement strategy*. The third phase makes use of the rectangle packing sequence defined in the first phase and the vector of placement procedures defined in second phase and constructs a packing of the rectangles.
(4) *Fitness evaluation*. In the final phase, we make use of a novel procedure to compute a modified total value which is used as a fitness measure (quality measure) to feedback to the genetic algorithm.

Figure 1 illustrates the sequence of steps applied to each chromosome generated by the genetic algorithm. The remainder of this section describes in detail the genetic algorithm, the placement strategy, and the fitness function.

```
procedure GENETIC-ALGORITHM
1   Generate initial population P_0;
2   Evaluate population P_0;
3   Initialize generation counter g ← 0;
4   while stopping criteria not satisfied repeat
5       Select some elements from P_g to copy into P_{g+1};
6       Crossover some elements of P_g and put into P_{g+1};
7       Mutate some elements of P_g and put into P_{g+1};
8       Evaluate new population P_{g+1};
9       Increment generation counter: g ← g + 1;
10  end while;
end GENETIC-ALGORITHM;
```

FIGURE 2. Pseudo-code of a standard genetic algorithm.

3.1. **Genetic algorithm.** Genetic algorithms are adaptive methods that are used to solve search and optimization problems (Goldberg, 1989; Beasley et al., 1993). They are based on the genetic process of biological organisms. Over many generations, natural populations evolve according to the principles of natural selection, i.e. *survival of the fittest*, first clearly stated by Charles Darwin (1859), in *The Origin of Species*. By mimicking this process, genetic algorithms, if suitably encoded, are able to *evolve* solutions to real world problems. Before a genetic algorithm can be run, an *encoding* (or *representation*) for the problem must be devised. A *fitness function*, which assigns a figure of merit to each encoded solution, is also required. During the run, parents are *selected* for reproduction and *recombined* to generate offspring (see high-level pseudo-code in Figure 2).

It is assumed that a potential solution to a problem may be represented as a set of parameters. These parameters (known as *genes*) are joined together to form a string of values (*chromosome*). In genetic terminology, the set of parameters represented by a particular chromosome is referred to as an *individual*. The fitness of an individual depends on its chromosome and is evaluated by the fitness function. The individuals, during the reproductive phase, are selected from the population and *recombined*, producing offspring, which comprise the next generation. Parents are randomly selected from the population using a scheme that favors fitter individuals. Having selected two parents, their chromosomes are *recombined*, typically using mechanisms of *crossover*. Mutation is usually applied to some individuals, to guarantee population diversity.

3.1.1. *Chromosome representation and decoding.* The genetic algorithm described in this paper uses a random-keys alphabet comprised of random real numbers between 0 and 1. The evolutionary strategy used is similar to the one proposed by Bean (1994), the main difference occurring in the crossover operator. The important feature of random keys is that all offspring formed by crossover are feasible solutions. This is accomplished by moving much of the feasibility issue into the objective function evaluation. If any random-key vector can be interpreted as a feasible solution, then any crossover vector is also feasible. Through the dynamics of the genetic algorithm, the system learns the relationship between random-key vectors and solutions with good objective function values.

A chromosome represents a solution to the problem and is encoded as a vector of random keys. In a direct representation, a chromosome represents a solution of the original problem, and is called *genotype*, while in an indirect representation it does not, and special

procedures are needed to derive from it a solution called a *phenotype*. In the present context, the direct use of cutting patterns as chromosomes is too complicated to represent and manipulate. In particular, it is difficult to develop corresponding crossover and mutation operations. Instead, solutions are represented indirectly by parameters that are later used by a decoding procedure to obtain a solution. To obtain the solution (phenotype) we use the placement strategy to be described in Section 3.2.

Recall that there are n rectangle types and that at most Q_i rectangles of type i can be packed into the the stock rectangle. In the description of the genetic algorithm, we take a total of $M = \sum_{i=1}^{n} Q_i$ rectangles, i.e. Q_i copies of rectangles $i = 1,\ldots,n$ and, for $j = 1,\ldots,M$, define

$$y_j = \begin{cases} 1 & \text{if rectangle } j \text{ is packed into the stock rectangle,} \\ 0 & \text{otherwise.} \end{cases}$$

Let $R_i = \{j \mid \text{rectangle } j \in \{1,\ldots,M\} \text{ is of type } i\}$. Clearly $x_i = \sum_{j \in R_i} y_j$ and since v_i is the value of any rectangle of type i, then our objective is to

$$\max \sum_{i=1}^{n} \sum_{j \in R_i} v_i y_j.$$

By considering all rectangles individually, we are able to handle the upper bounds $x_i \leq Q_i$, $i = 1,\ldots,n$, implicitly. We later discuss how we deal with the lower bounds $x_i \geq P_i$, for one or more $i \in \{1,\ldots,n\}$, in the case of doubly constrained problems.

Each solution chromosome is made of $2M$ genes, where M is the number of rectangles to be packed, i.e.

$$Chromosome = (\ \underbrace{gene_1,\ldots,gene_M}_{\text{Rectangle Packing Sequence}}\ ,\ \underbrace{gene_{M+1},\ldots,gene_{2M}}_{\text{Vector of Placement Procedures}}\ ).$$

The first M genes are used to obtain the *Rectangle Packing Sequence* (RPS) while the last M genes are used to obtain the *Vector of Placement Procedures* (VPP).

Both the RPS and VPP are used by the placement strategy. The decoding (mapping) of the first M genes of each chromosome into an RPS is accomplished by sorting the genes and packing the rectangles in ascending order (see Figure 3). In the placement strategy we make use of two placement procedures; *BL* and *LB* (see Section 3.2). The decoding (mapping) of the last M genes of each chromosome into a VPP is accomplished using, for $i = 1,\ldots,M$, the following expression:

$$VPP(i) = \begin{cases} BL & \text{if } gene(M+i) \leq \frac{1}{2}, \\ LB & \text{if } \frac{1}{2} < gene(M+i). \end{cases}$$

3.1.2. *Evolutionary strategy.* To breed good solutions, the random-key vector population is operated upon by a genetic algorithm. There are many variations of genetic algorithms obtained by altering the reproduction, crossover, and mutation operators. The reproduction and crossover operators determine which parents will have offspring, and how genetic material is exchanged between the parents to create those offspring. Mutation allows for random alteration of genetic material. Reproduction and crossover operators tend to increase the quality of the populations and force convergence. Mutation opposes convergence and replaces genetic material lost during reproduction and crossover.

The *population is initialized* with random-key vectors whose components are random real numbers uniformly sampled from the interval $[0,1]$. *Reproduction* is accomplished by first copying some of the best individuals from one generation to the next, in what is called an *elitist strategy* (Goldberg, 1989). The advantage of an elitist strategy over

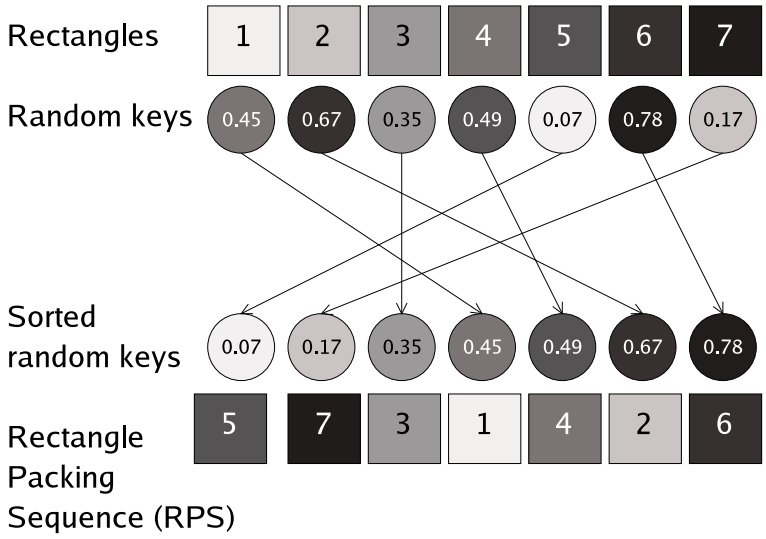

FIGURE 3. Chromosome decoding procedure. Rectangles are sequenced according to the values of their corresponding genes. Genes (random keys) are sorted in increasing order to define the Rectangle Packing Sequence (RPS).

traditional probabilistic reproduction is that the best solution is monotonically improving from one generation to the next. The potential downside is population convergence to a local minimum. This can, however, be overcome by an appropriate amount of mutation as described below.

Parameterized uniform crossovers (Spears and Dejong, 1991) are employed in place of the traditional one-point or two-point crossover. After two parents are chosen, one chosen randomly from the best (*TOP* in Figure 4) and the other chosen randomly from the full old population (including chromosomes copied to the next generation in the elitist pass), at each gene we toss a biased coin to select which parent will contribute the allele. Figure 5 presents an example of the crossover operator. It assumes that a coin toss of heads selects the gene from the first parent, a tails chooses the gene from the second parent, and that the probability of tossing a heads, crossover probability $CProb = 0.7$. In Section 4 we describe how we determine this value empirically. Figure 5 shows one potential crossover outcome.

Rather than using the traditional gene-by-gene mutation with very small probability at each generation, some new members of the population are randomly generated from the same distribution as the initial population (see *BOT* in Figure 4). The purpose of this process is to prevent premature convergence of the population, like in a mutation operator, and leads to a simple statement of convergence. Figure 4 depicts the transitional process between two consecutive generations.

3.2. **Placement strategy.** Rectangles are placed in the stock rectangle, one at a time, in the order defined by the RPS supplied by the genetic algorithm (see Section 3.1.1). While trying to place a rectangle in the stock rectangle we consider only maximal *empty rectangular spaces* (ERSs), i.e ERSs that are not contained in any other ERS. A rectangle is placed in an ERS where it fits according to the placement procedure defined for its placement by the genetic algorithm.

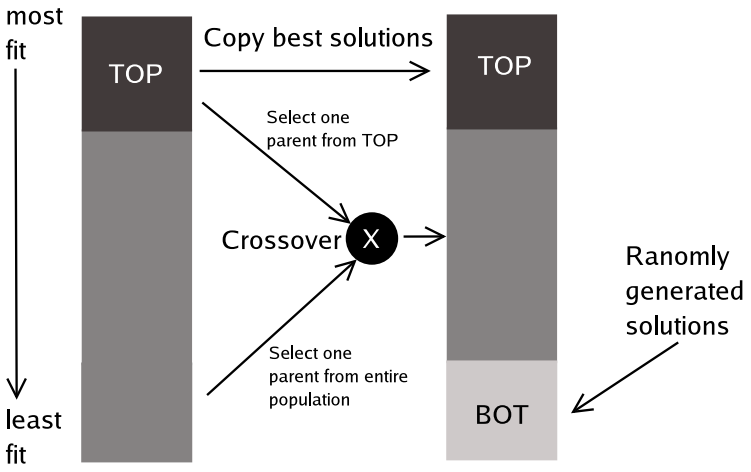

FIGURE 4. Transitional process between consecutive generations. Current population is sorted from best to worst fitness. Top (TOP) individuals from current population are copied unchanged to next population. Bottom (BOT) individuals in next population are randomly generated. The remaining individuals are generated by applying crossover operator to randomly selected individual from TOP individuals of current population and randomly selected individual from the entire current population.

To generate and keep track of the ERSs, we initially considered only a *Bottom-Left* (BL) placement procedure which places a rectangle in an ERS where it fits and which is the closest to the bottom-left corner of the stock rectangle as described in the pseudo-code in Figure 6. However, as observed by Liu and Teng (1999), we noticed that some optimal solutions cannot be constructed by the BL placement procedure. In other words, given an optimal solution to a problem, it is possible that no RPS exists that, combined with the BL placement procedure, produces the given optimal solution. Figure 8 shows an optimal solution for one problem where the BL placement procedure cannot find the optimal solution.

In Figure 9, we present, for the problem in Figure 8, all the solutions obtained by the BL placement procedure for all the possible RPSs starting with rectangle 2. As can be observed, none of those RPSs, when combined with BL placement procedure, produces the optimal solution in Figure 8. To overcome this weakness, we combine the BL placement procedure with a *Left-Bottom* (LB) placement procedure which places a rectangle in an ERS where it fits and which is the closest to left bottom corner of the stock rectangle, as described in the pseudo-code in Figure 7. In summary, our placement strategy uses two placement procedures, the Bottom-Left and the Left-Bottom, to construct a packing of the rectangles. The vector of placement procedures (VPP), supplied by the genetic algorithm, indicates, for each rectangle to be packed, whether it should be placed with the BL or LB procedure. In Figure 10 we present the optimal solution found using $RPS = (2,1,4,3)$ and $VPP = (BL,BL,LB,BL)$.

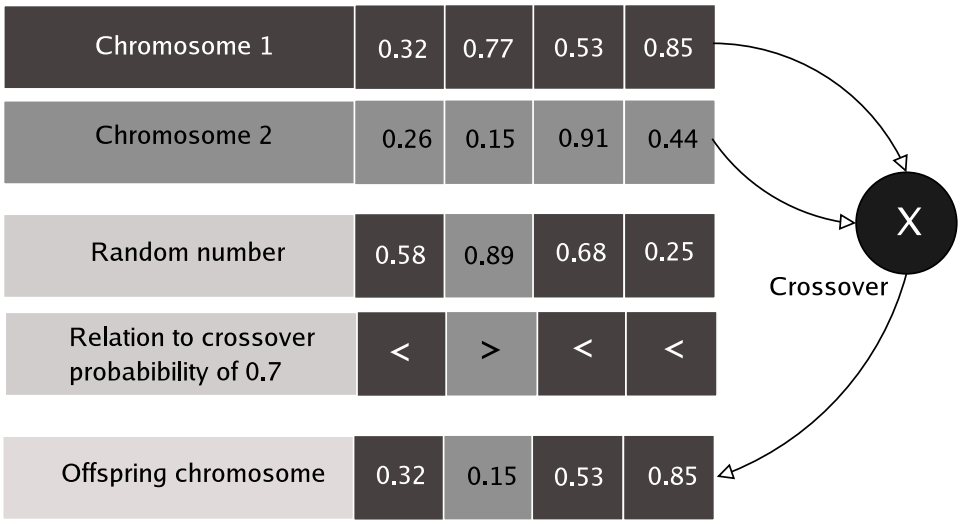

FIGURE 5. Example of parameterized uniform crossover with crossover probability equal to 0.7. For each gene, a random number in the interval $[0,1]$ is generated. With probability 0.7 the offspring inherits the gene of Chromosome 1 and with probability 0.3, it inherits the one of Chromosome 2.

Let M be the number of rectangles to be placed in the stock rectangle and let r_i be the i-th rectangle to be placed as defined by the RPS. Furthermore, let PP_i be the placement procedure that the genetic algorithm assigned for the placement of rectangle r_i. The pseudo-code of the placement procedure is presented in Figure 11.

To generate and keep track of the ERSs, we make use of the *Difference Process* (DP), developed by Lai and Chan (1997b). In short, the DP can be described as follows. Suppose we have two small rectangles to be packed in a stock rectangle as depicted in Figure 12a. Furthermore, assume that we will pack rectangle 1 first and then rectangle 2. At the very beginning, there is only one ERS available (ERS-1) where we can pack rectangle 1 and its bottom left corner is precisely at the origin of the stock rectangle (see Figure 12b). After the first rectangle is packed, usually two new ERSs are generated (unless the height or the width of the first rectangle is the same as that of the stock rectangle (see ERS-2 and ERS-3 in Figure 12b). Every time a new rectangle is packed, it intersects some of the available ERSs, so we need to update the list of available ERSs before we pack the next rectangle. From each existing ERS which intersects with a rectangle nontrivially, it is easy to see that at most three new empty rectangular spaces are generated. From the newly generated ERSs, we check if any one is entirely contained in another, and, in that case, remove it. This is the so-called *elimination process* in Lai and Chan (1997b). In this way, we update the list of available ERSs whenever a rectangle is packed (see ERS-2.1, ERS-3.1 and ERS-3.2 in Figure 12c).

```
procedure BL(r_i)
1   Let r_i be the i-th rectangle to be packed in the stock rectangle;
2   Let n_ERS be the number of available ERSs;
3   Initialize L ← W;
4   Initialize B ← H;
5   for k = 1,...,n_ERS do
6       Let x(ERS_k) be the x coordinate of the the bottom left
·       corner of ERS_k;
7       Let y(ERS_k) be the y coordinate of the the bottom left
·       corner of ERS_k;
8       if r_i fits in ERS_k then
9           if x(ERS_k) ≤ L then
10              L ← x(ERS_k);
11          else if x(ERS_k) = L and y(ERS_k) ≤ B then
12              B ← y(ERS_k) ;
13          end if
14      end if
15  end for
16  Place rectangle r_i at coordinates (L,B);
end BL;
```

FIGURE 6. Pseudo-code of the Bottom-Left (BL) placement procedure.

3.3. **Modified total value fitness function.** A natural fitness function (measure of quality) for this type of problem is the *total value* given by:

$$\textit{Total value} = \sum_{i=1}^{n} v_i x_i,$$

where x_i is the number of rectangular pieces of type i to be cut in a solution and v_i is the value of each rectangular piece of type i. This measure, however, is not ideal because it does not capture well the potential for improvement of a solution. Consider, for example, *Packing-1* and *Packing-2* depicted in Figure 13. Assuming that the value of each rectangular piece is equal to its area, then both packings have a *Total value* of $(3\times 1+4\times 1+1\times 2+2\times 1)=11$. However, *Packing-2* has all the unused area concentrated in one contiguous space, while for *Packing-1* the unused area is made up of three non-adjacent areas. Any rectangle that can be packed in *Packing-1* can also be packed in *Packing-2*. However, the opposite is not true. For example, we cannot pack a 3×1 or a 4×1 rectangle in *Packing-1*, whereas we can in *Packing-2*. Therefore, it is easy to conclude that *Packing-2* has a higher potential than *Packing-1* to have its total value increased.

To be able to capture the improvement potential of different packings which have the same total value, we propose a new fitness measure that we call *modified total value*. The basic idea consists in adding to the total value of a packing a small value proportional to the largest empty rectangular space remaining in the packing, i.e.,

$$\textit{Modified Total Value} = \textit{Total Value} + \beta \times \textit{Area of Largest ERS} \times \textit{Minimum Value Per Unit Area},$$

```
procedure LB(r_i)
1   Let r_i be the i-th rectangle to be packed in the stock rectangle;
2   Let n_ERS be the number of available ERSs;
3   Initialize L ← W;
4   Initialize B ← H;
5   for k = 1,...,n_ERS do
6       Let x(ERS_k) be the x coordinate of the the bottom left
·       corner of ERS_k;
7       Let y(ERS_k) be the y coordinate of the the bottom left
·       corner of ERS_k;
8       if r_i fits in ERS_k then
9           if y(ERS_k) ≤ B then
10              B ← y(ERS_k);
11          else if y(ERS_k) = B and x(ERS_k) ≤ L then
12              L ← x(ERS_k) ;
13          end if
14      end if
15  end for
16  Place rectangle r_i at coordinates (L,B);
end LB;
```

FIGURE 7. Pseudo-code of the Left-Bottom (LB) placement procedure.

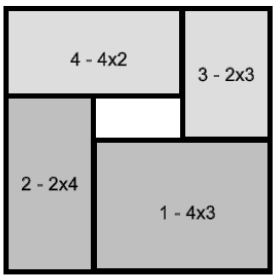

FIGURE 8. Example of an instance with a 6×6 stock rectangle and single copies of 4×3, 2×4, 2×3, and 4×2 rectangles to be packed. The optimal solution is shown in the figure. For this instance, the Bottom-Left (BL) placement heuristic cannot produce the optimal solution.

where β is a parameter that depends on the problem data. The parameter β should be chosen carefully because the modified total value must be able to accommodate the following two objectives:

(1) For solutions with the same real total value, we want the modified total value to assign better fitness values to those solutions that have contiguous empty spaces (i.e. have a few large empty spaces rather than many small empty spaces).
(2) We do not want to have empty spaces in the final solutions obtained by the overall algorithm since the empty spaces do not increase the real total value.

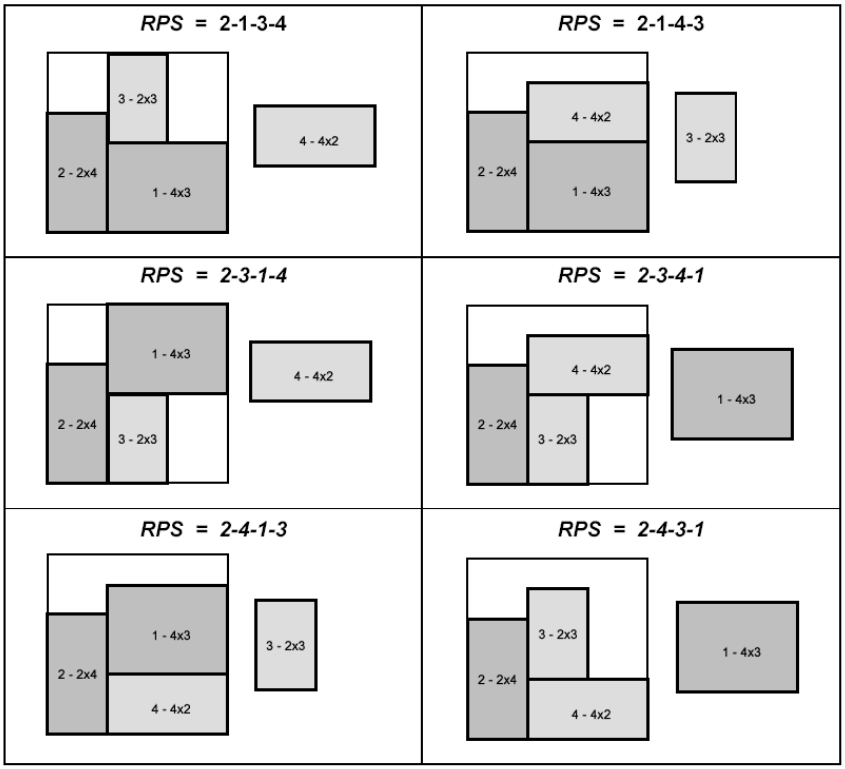

FIGURE 9. Solutions found by the Bottom-Left (BL) placement procedure for the problem in Figure 8 for all Rectangle Packing Sequences with rectangle 2 in the first position. Similar sub-optimal solutions are produced when rectangles 1, 3, and 4 are fixed in the first position.

Very small values of β will not promote the solutions with large empty spaces, while large values of β will cause the genetic algorithm to simply search for solutions with large empty spaces. To avoid having solutions with large empty spaces, we require that the increase in total value caused by the term

$$\beta \times \textit{Area of Largest ERS} \times \textit{Minimum Value Per Unit Area}$$

should be significantly less than the increase in total value caused by the packing of the smallest value rectangle, i.e.

$$\beta \times \textit{Area of Largest ERS} \times \textit{Minimum Value Per Unit Area} << \textit{Minimum Value Rectangle}.$$

Note that, in the worst case, we might have

$$\textit{Area of Largest ERS} = \textit{Area of Stock Rectangle}$$

which is equivalent to having

$$\beta << \frac{\textit{Minimum Value Rectangle}}{\textit{Area of Stock Rectangle} \times \textit{Minimum Value Per Unit Area}}.$$

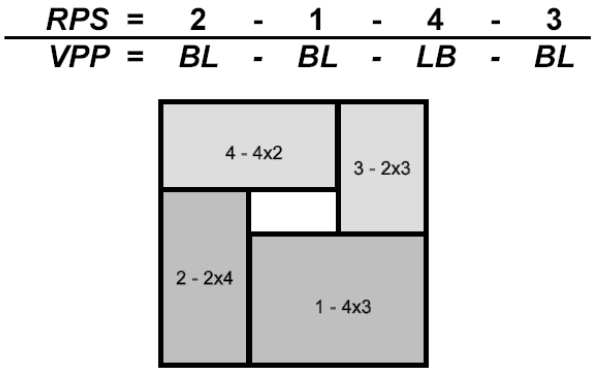

FIGURE 10. Optimal solution, for the problem in Figure 8, found by combining the Bottom-Left (BL) and the Left-Bottom (LB) placement procedures. First, rectangle 2 (of size 2×4) is placed with BL. Then, rectangle 1 (of size 4×3) is placed with BL. This is followed by rectangle 4 (of size 4×2) which is placed with LB. Finally, rectangle 3 (of size 2×3) is placed with BL.

```
procedure PLACEMENT
1   for i = 1,...,M do
2       if PP_i = BL then
3           Let ERS_i^* be the ERS, in the list of available ERSs,
·           in which rectangle r_i is placed when the Bottom-Left
·           placement heuristic is applied;
4       end if
5       else if PP_i = LB then
6           Let ERS_i^* be the ERS, in the list of available ERSs,
·           in which rectangle r_i is placed when the Left-Bottom
·           placement heuristic is applied;
7       end if
8       if ERS_i ≠ ∅ then
9           Place r_i at bottom left corner of ERS_i^*;
10          Update list of available ERSs using the DP process
·           of Lai and Chan (1997b);
11      end if
end PLACEMENT;
```

FIGURE 11. Pseudo-code of the placement procedure of the hybrid heuristic.

We chose to compute β using the expression

$$\beta = K \times \frac{\textit{Minimum Value Rectangle}}{\textit{Area of Stock Rectangle} \times \textit{Minimum Value Per Unit Area}},$$

where $K << 1$ is a constant. Note that this expression also satisfies the previous expression. To determine the appropriate value of K, we experimented with values of K varying

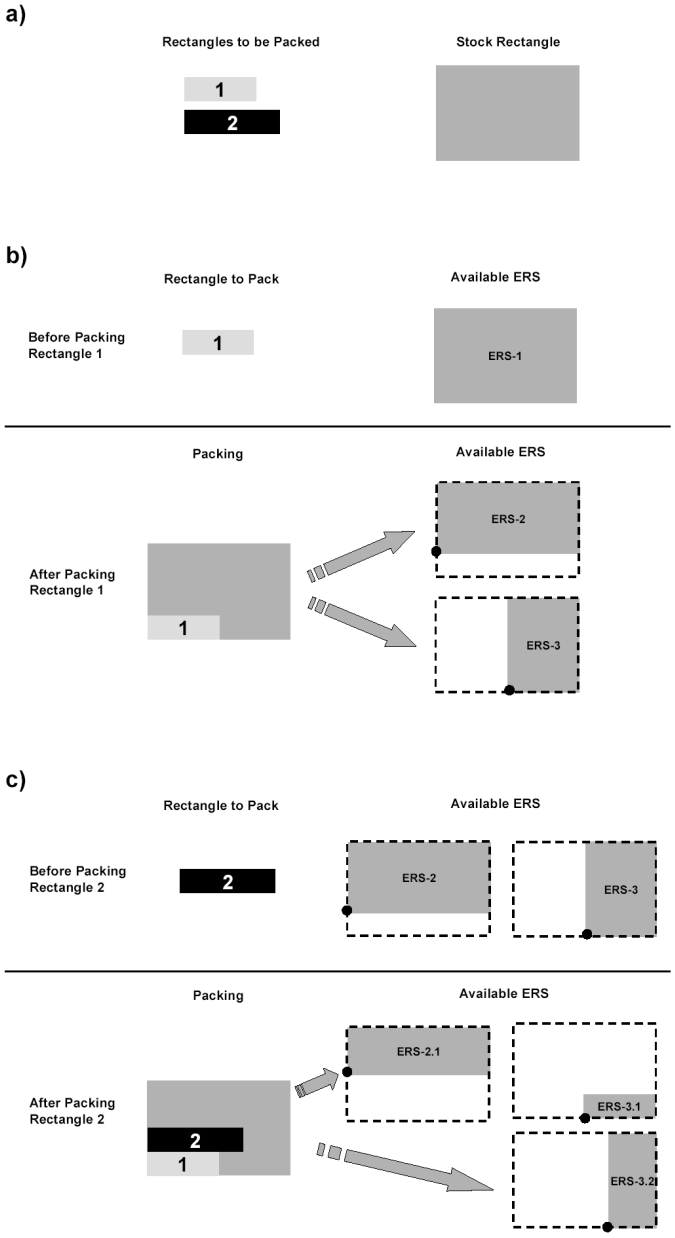

FIGURE 12. Example of Difference Process (DP). In (a) two rectangles (Rectangle-1 and Rectangle-2) are to be packed into the Stock-Rectangle. The stock rectangle is the Empty Rectangular Space (ERS) ERS-1. In (b), after Rectangle-1 is packed into the bottom left corner of ERS-1, two ERSs, ERS-2 and ERS-3 are produced. In (c), Rectangle-2, next to be packed, only fits in ERS-2 and is therefore packed in the bottom left corner of ERS-2. After ERS-2 is packed in ERS-2, two ERSs, ERS-2.1 and ERS-2.2 (not shown) result from ERS-2, and three ERSs, ERS-3.1, ERS-3.2, and ERS-3.3 (not shown) result from ERS-3. Since ERS-2.2 is contained in ERS-3.2, it is eliminated. Likewise, since ERS-3.3 is contained in ERS-2.1, it is eliminated.

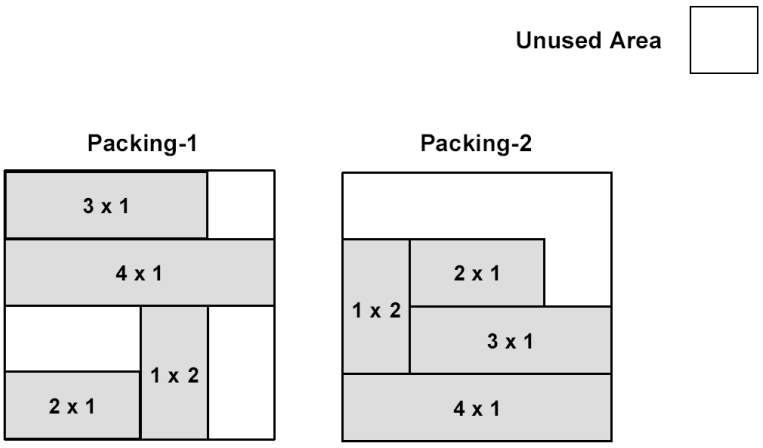

FIGURE 13. Two packings with different improvement potentials.

between 0.01 and 0.10 in steps of 0.01 and concluded that 0.03 was an appropriate value. We can now fully define the modified total value (MTV) as

$$\textit{Modified Total Value} = \textit{Total Value} + 0.03 \times \frac{\textit{Minimum Value Rectangle} \times \textit{Area of Largest ERS} \times \textit{Minimum Value Per Unit Area}}{\textit{Area of Stock Rectangle} \times \textit{Minimum Value Per Unit Area}}$$

or equivalently

$$\textit{Modified Total Value} = \textit{Total Value} + 0.03 \times \frac{\textit{Minimum Value Rectangle} \times \textit{Area of Largest ERS}}{\textit{Area of Stock Rectangle}}.$$

The *Modified Total Value* for *Packing-1* and *Packing-2* of Figure 13 are, respectively,

$$MTV_1 = \left(11 + 0.03 \times \frac{2 \times 2 \times 1}{16}\right) = 11.0075$$

and

$$MTV_2 = \left(11 + 0.03 \times \frac{2 \times 4 \times 1}{16}\right) = 11.015.$$

As desired, *Packing-2* has a higher MTV than *Packing-1*. The use of the modified total value as the fitness function was responsible for significantly improving the total values obtained in the computational experiments.

As presented above, our algorithm does not handle doubly constrained problems since lower bounds are not enforced. To enable the algorithm to handle these problems, we made one further modification to the fitness function. The lower bounds P_i for each type of piece i are treated indirectly through the use of a penalty term in the fitness function. If any lower bound P_i is not satisfied in a solution, then a penalty parameter PF is subtracted from the modified total value. All the computational tests were performed using $PF = 10^{10}$.

TABLE 1. Four sets of test problems used in the experimental evaluation of the hybrid heuristic.

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

4. EXPERIMENTAL RESULTS

In this section we report results obtained on a set of experiments conducted to evaluate the performance of the hybrid heuristic (HH) proposed in this paper. We compare the HH with the following three recently proposed heuristics, which present the best computational results to date:

- *PH* – A population heuristic, proposed by Beasley (2004), where a population of solutions to the problem is progressively evolved. The heuristic uses a nonlinear formulation of the problem.
- *GA* – Proposed by Hadjiconstantinou and Iori (2006), this genetic algorithm uses an elitist theory, immigration, online heuristics, and tailored crossover operators.
- *GRASP* – A greedy randomized adaptive search procedure proposed by Alvarez-Valdes et al. (2005).

The effectiveness of HH is evaluated by solving four sets of test problems taken from the literature and summarized in Table 1. These problem sets are:

(1) A set of 38 problems taken from the literature:
   - 12 from Beasley (1985b): `ngcut01` – `ngcut12`,
   - Four from Hadjiconstantinou and Christofides (1995): `hccut03`, `hccut08`, `hccut11`, and `hccut12`,
   - Five from Fekete and Schepers (2004c): `okp01` – `okp05`,
   - Three from Christofides and Whitlock (1977): `cgcut01` – `cgcut03`,
   - 13 from Beasley (1985a): `gcut01` – `gcut12`, and
   - One from Wang (1983): `wang20`.

(2) A set of 630 large problems generated by Beasley (2004), following Fekete and Schepers (2004c). All problems have a stock rectangle of size 100×100. There are three types of instances, depending on the distribution of types of pieces of each class of rectangle as shown in Table 2. For each type, 210 instances are generated as follows. For each value $m = 40, 50, 100, 150, 250, 500, 1000$ of the number of piece types, ten problems are randomly generated with $P_i = 0$ and $Q_i = 1, 3, 4$, for all $i = 1, \ldots, n$. The value assigned to each piece is equal to its area multiplied by an integer randomly chosen from $\{1, 2, 3\}$.

(3) Test problems used by Leung et al. (2003) consisting of:
   - Three instances from Lai and Chan (1997b),
   - Five from Jakobs (1996), and

TABLE 2. Instances in problem set 2 have rectangles from four classes: Class 1, Class 2, Class 3, and Class 4. The table shows the distributions of pieces of each class in problems of Types I, II, and III.

Type	Class 1	Class 2	Class 3	Class 4
I	20%	20%	20%	40%
II	15%	15%	15%	55%
III	10%	10%	10%	70%

- Two from Leung et al. (2003).

In these problems, the value of each piece corresponds to its area, and the objective is to minimize the waste of the stock rectangle. The problems have been generated in such a way that the optimal solution is a cutting pattern with zero waste. We have included this set of problems because Alvarez-Valdes et al. (2005) use it and considered it complementary to the two first sets used by Beasley (2004). The problems of the second set (Types I, II, and III) can be considered selection problems because there are many available pieces and only a small fraction of them will form part of the solution. However, Leung's problems are jigsaw problems. Almost all the available pieces will form part of the solution and the difficulty here is to find their correct position in the cutting pattern. An algorithm working well on both types of problems can be considered as being general purpose.

(4) A set of 21 problems taken from the literature:
- 12 from Beasley (1985a): `ngcut01` – `ngcut12`
- Two from Hadjiconstantinou and Christofides (1995): `hccut03` and `hccut08`
- Five from Fekete and Schepers (2004c): `okp01` – `okp05`
- One from Christofides and Whitlock (1977): `cgcut03`, and
- One from Wang (1983): `wang20`.

These instance were transformed by Beasley (2004) into doubly constrained problems by defining, for some types of rectangles, non-zero lower bounds. All rectangle types $i = 1,\ldots,m$ whose areas (in total) do not exceed one-third of the area of the stock rectangle had their lower bounds P_i set to 1. This was done by initially setting all $P_i = 0$, for $i = 1,\ldots,m$ and, in turn, for each rectangle type $i = 1,\ldots,m$ satisfying

$$\sum_{j=1(j\neq i)}^{m} (l_j \cdot w_j)P_j + l_i \cdot w_i \leq (H \cdot W)/3,$$

the lower bound P_i is set to 1. This set of problems allows us to test the heuristic in the general case of doubly constrained problems.

4.1. **GA configuration.** Configuring genetic algorithms is oftentimes more an art form than a science. In our past experience with genetic algorithms based on the same evolutionary strategy (see Gonçalves and Almeida (2002), Ericsson et al. (2002), Gonçalves and Resende (2004), Gonçalves et al. (2005), Buriol et al. (2005), Buriol et al. (2006), and Gonçalves (2006)), we obtained good results with values of TOP, BOT, and Crossover Probability (CProb) in the intervals shown in Table 3.

TABLE 3. Range of parameters used to configure past implementations of genetic algorithms using the evolutionary strategy used in the hybrid heuristic.

Parameter	Interval
TOP	0.10 – 0.20
BOT	0.15 – 0.30
Crossover Probability (CProb)	0.70 – 0.80

For the population size, we have obtained good results by indexing it to the size of the problem, i.e. use small size populations for small problems and larger populations for larger problems. With this in mind, we conducted a small pilot study, using the three problem instances in Table 4, to obtain a reasonable configuration. We tested all the combinations of the following values:

- $TOP \in \{0.10, 0.15, 0.20\}$;
- $BOT \in \{0.15, 0.20, 0.25, 0.30\}$;
- $CProb \in \{0.70, 0.75, 0.80\}$;
- *Population size* with 2, 5, 10, and 15 times the number of rectangles in the problem instance.

For each of the three instances in Table 4 and the 144 possible configurations, we made three independent runs of the algorithm (with three distinct random number generator seeds) and computed the average total value. The configuration that minimized the sum, over the three problem instances, was $TOP = 15\%$, $BOT = 15\%$, $CProb = 0.7$, and *Population size* $= 10 \times$ the number of rectangles in the problem instance.

To determine the appropriate value of K for the modified total value, we tested the algorithm using values of K between 0.01 and 0.10, in steps of 0.01, on the four problem instances listed in Table 5. These instances were chosen because an initial variant of the genetic algorithm, that used the usual total value fitness measure, was not obtaining good results on them. We made three independent runs of the algorithm, using the best configuration determined previously, and computed the average modified total value. The value of

TABLE 4. Instances used in the pilot study to determine the configuration of the hybrid heuristic.

Problem Source	**Size (number of rectangles)**
Lai and Chan (1997b) – Instance 3	20
Jakobs (1996) – Instance 5	30
Leung et al. (2003) – Instance 1	40

TABLE 5. Test problems used in experiments to determine the appropriate value of K for the computation of the modified total value.

Problem Source	**Size (number of rectangles)**
Jakobs (1996) – Instance 2	30
Leung et al. (2003) – Instance 1	40
Hopper and Turton (2001) – Instance C4-3	49
Hopper and Turton (2001) – Instance C5-1	73

K that maximized the sum, over the four problems instances, of the average modified trim loss was chosen, i.e. $K = 0.03$.

The configuration presented in Table 6 was held constant for all experiments and all problems instances. The computational results presented in the next section demonstrate that this configuration not only provides excellent results in terms of solution quality but also is very robust.

4.2. **Computational results.** Our algorithm (HH) was implemented in MS Visual Basic 6.0 and the computational experiments were carried out on a computer with a 3.2 GHz Pentium CPU with the MS Windows XP operating system.

The complete computational results appear in Tables 7–17. All tests where performed using the configuration summarized in Table 6. The results in Tables 7, 8, 9, 15, 16, and 17 correspond to the best values of five runs our algorithm using five different random number generator seeds. The results in Tables 10, 11, 12, and 14 correspond to a single run of our

TABLE 6. Configuration used on all runs in computational experiments.

Population size	$\min\{10 \times$ number of input rectangles$, 1000\}$
Crossover probability	0.7
TOP	The 15% most fit solutions from the previous generation are copied to the next generation.
BOT	The 15% least fit solutions from the previous generation are replaced with randomly generated solutions.
Fitness	Maximize modified total value using parameter $K = 0.03$.
Stopping criterion	Stop after 300 generations. For some large instances, a limit of 300 seconds was imposed.

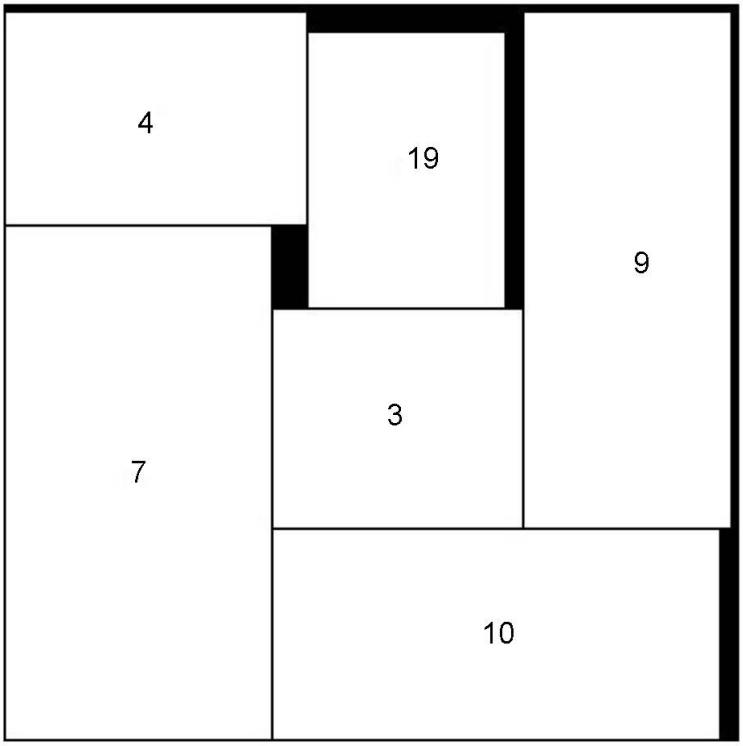

FIGURE 14. Best solution found by HH for problem instance gcut02 using $RPS = \{7,10,3,9,4,19,\ldots\}$ and $VPP = \{BL, BL, BL, BL, LB, BL,\ldots\}$. Total value of packing is 59,798. The genetic algorithm (GA) of Hadjiconstantinou and Iori (2006) found a solution of 59,563.

algorithm. When testing the large instances of Set II, we also limited the computation time to 300 seconds, since the limit of 300 generations was taking too much time.

In terms of computational times we cannot make any fair and meaningful comments since all the other approaches were implemented with different programming languages and tested on computers with different computing power. Instead, we limit ourselves to reporting the average running times for HH.

The first two tables include a direct comparison with the results for PH (Beasley, 2004), GA (Hadjiconstantinou and Iori, 2006), and GRASP (Alvarez-Valdes et al., 2005), in terms of solution quality. The results for the Set I of problems, presented in Table 7 to 9 show that HH finds solutions that are as good, or better, than those found by the other heuristics. Note also that the HH improves the best known solution for problem instance gcut13 by more than 4.5 percent. Figures 14 and 15 illustrate the best solutions found by HH for problem instances gcut02 and gcut13, respectively.

The results for the Set II of large random problems are presented in Tables 10–12. These tables display aggregate results, showing that HH produces overall average deviations from the upper bound for all problem types (Type I, Type 2, and Type 3) that are always lower than those produced by the other heuristics. A close look at the results shows that HH outperforms heuristics PH and GA for all problem types and sizes. However, the GRASP of Alvarez-Valdes et al. (2005) consistently produces better results for problem instances

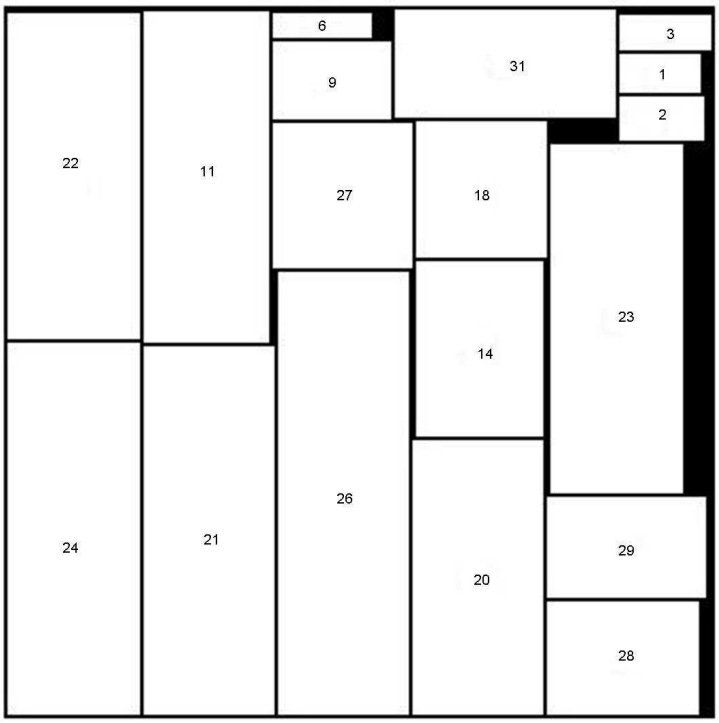

FIGURE 15. Best solution found by HH for problem instance `gcut13` using $RPS = \{24, 21, 26, 20, 28, 29, 22, 11, 27, 9, 6, 14, 18, 31, 23, 2, 1, 3, \ldots\}$ and $VPP = \{BL, BL, BL, BL, BL, BL, LB, LB, LB, LB, LB, LB, LB, BL, BL, BL, BL, \ldots\}$. Total value of packing is 8,654,707, which is a new best known solution (BKS) for the problem instance. The previous BKS was 8,281,060 and was found by the genetic algorithm (GA) of Hadjiconstantinou and Iori (2006).

with $m \geq 250$. This is mainly due to the introduction of the limit in the computation time which reduced the number of HH generations. Table 14 presents the CPU time (in seconds) for each group of problems in Tables 10–12.

Table 15 presents a direct comparison between the GRASP of Alvarez-Valdes et al. (2005) and HH on the problem instances in Set III. HH obtained the optimum value for seven problems, whereas the GRASP obtained the optimum for only the two smallest instances. Overall, HH obtained an average deviation from the optimum of 0.48%, whereas the GRASP obtained a mean deviation of 2.05%, which is more than four times as much.

Finally, Tables 16 and 17 show the results for the doubly-constrained test problems in Set IV, where the heuristics PH (Beasley, 2004), GRASP (Alvarez-Valdes et al., 2005), and HH are contrasted. The upper bound corresponds to the solution of the constrained problem. The problems for which the algorithms do not find solutions are not feasible, but are maintained in the set of test problems and therefore are included in the table. For these problems, HH clearly outperforms the other two both in terms of average deviation from the lower bound (8.11%, 7.36%, *6.63%*) as well as in terms of the number of best solutions obtained out of the 21 problems (14/21, 12/21, *19/21*).

TABLE 7. Computational results for Set I – 38 small problems taken from the literature. Part 1 of 3: problem classes ngcut and hccut. For each instance, the table lists: its dimension ($W \times H$); m, the number of types of rectangles to pack; M, the maximum number of rectangles that can be packed; the optimal or best known solution (BKS); and solutions found by algorithms PH (Beasley, 2004), GA (Hadjiconstantinou and Iori, 2006), GRASP (Alvarez-Valdes et al., 2005), and HH, as well as the average solution time for HH in seconds. Entries in **boldface** indicate best known value or better was obtained.

Instance	Dimension	m	M	BKS	PH	GA	GRASP	HH	HH time
ngcut01	10×10	5	10	164	**164**	**164**	**164**	**164**	1.80
ngcut02	10×10	7	17	230	**230**	**230**	**230**	**230**	4.21
ngcut03	10×10	10	21	247	**247**	**247**	**247**	**247**	5.81
ngcut04	15×10	5	7	268	**268**	**268**	**268**	**268**	1.31
ngcut05	15×10	7	14	358	**358**	**358**	**358**	**358**	3.18
ngcut06	15×10	10	15	289	**289**	**289**	**289**	**289**	4.18
ngcut07	20×20	5	8	430	**430**	**430**	**430**	**430**	1.81
ngcut08	20×20	7	13	834	**834**	**834**	**834**	**834**	3.56
ngcut09	20×20	10	18	924	**924**	**924**	**924**	**924**	5.63
ngcut10	30×30	5	13	1452	**1452**	**1452**	**1452**	**1452**	2.80
ngcut11	30×30	7	15	1688	**1688**	**1688**	**1688**	**1688**	4.28
ngcut12	30×30	10	22	1865	1801	**1865**	**1865**	**1865**	7.47
hccut03	30×30	7	7	1178	**1178**	**1178**	**1178**	**1178**	1.34
hccut08	30×30	15	15	1270	**1270**	**1270**	**1270**	**1270**	3.90
hccut11	40×40	10	10	2517		**2517**		**2517**	2.22
hccut12	40×40	15	15	2949		**2949**		**2949**	4.03
	Number of BKS / Number of problems:				13/14	16/16	14/14	16/16	
	Mean percentage deviation from BKS:				0.25	0.00	0.00	0.00	

TABLE 8. Computational results for Set I – 38 small problems taken from the literature. Part 2 of 3: problem classes `okp` and `cgcut`. For each instance, the table lists: its dimension ($W \times H$); m, the number of types of rectangles to pack; M, the maximum number of rectangles that can be packed; the optimal or best known solution (BKS); and solutions found by algorithms PH (Beasley, 2004), GA (Hadjiconstantinou and Iori, 2006), GRASP (Alvarez-Valdes et al., 2005), and HH, as well as the average solution time for HH in seconds. Entries in **boldface** indicate best known value or better was obtained.

Instance	Dimension	m	M	BKS	PH	GA	GRASP	HH	HH time
okp01	100 × 100	15	50	27718	27486	**27718**	27589	**27718**	32.97
okp02	100 × 100	30	30	22502	21976	**22502**	21976	**22502**	14.09
okp03	100 × 100	30	30	24019	23743	**24019**	23743	**24019**	13.45
okp04	100 × 100	33	61	32893	31269	**32893**	**32893**	**32893**	36.20
okp05	100 × 100	29	97	27923	26332	**27923**	**27923**	**27923**	118.25
cgcut01	15 × 10	7	16	244		**244**		**244**	4.76
cgcut02	40 × 70	10	23	2892		**2892**		**2892**	9.14
cgcut03	40 × 70	20	62	1860	1720	**1860**	**1860**	**1860**	39.98
	Number of BKS / Number of problems:				0/6	8/8	3/6	8/8	
	Mean percentage deviation from BKS:				3.75	0.00	0.66	0.00	

TABLE 9. Computational results for Set I – 38 small problems taken from the literature. Part 3 of 3: problem classes gcut and wang. For each instance, the table lists: its dimension ($W \times H$); m, the number of types of rectangles to pack; M, the maximum number of rectangles that can be packed; the optimal or best known solution (BKS); and solutions found by algorithms PH (Beasley, 2004), GA (Hadjiconstantinou and Iori, 2006), GRASP (Alvarez-Valdes et al., 2005), and HH, as well as the average solution time for HH in seconds. Entries in **boldface** indicate best known value or better was obtained.

Instance	Dimension	m	M	BKS	PH	GA	GRASP	HH	HH time
gcut01	250 × 250	10	10	48368		**48368**		**48368**	1.80
gcut02	250 × 250	20	20	59798		59563		**59798**	5.39
gcut03	250 × 250	30	30	61275		**61275**		**61275**	9.66
gcut04	250 × 250	50	50	61380		**61380**		**61380**	25.78
gcut05	500 × 500	10	10	195582		**195582**		**195582**	2.56
gcut06	500 × 500	20	20	236305		**236305**		**236305**	6.21
gcut07	500 × 500	30	30	240143		**240143**		**240143**	9.70
gcut08	500 × 500	50	50	245758		**245758**		**245758**	23.28
gcut09	1000 × 1000	10	10	939600		**939600**		**939600**	2.29
gcut10	1000 × 1000	20	20	937349		**937349**		**937349**	5.61
gcut11	1000 × 1000	30	30	969709		**969709**		**969709**	10.63
gcut12	1000 × 1000	50	50	979521		**979521**		**979521**	22.22
gcut13	3000 × 3000	32	32	*8281060		8281060		‡**8654707**	22.47
wang20	70 × 40	19	42	2726	**2726**	**2726**	**2726**	**2726**	19.68
	Number of BKS / Number of problems:				1/1	11/12	1/1	12/12	
	Mean percentage deviation from BKS:				0.00	0.36	0.00	0.00	

*Previously best known solution.

‡New best known solution found by HH.

TABLE 10. Computational results for problem Set II – Large random problems of Type I. For each instance, the table lists: m, the number of types of rectangles to be packed; Q, the upper bound on the number of rectangles of a given type that can be packed; and M, the maximum number of rectangles that can be packed, as well as average deviations (in percent) from the upper bounds for solutions found by PH (Beasley, 2004), GA (Hadjiconstantinou and Iori, 2006), GRASP (Alvarez-Valdes et al., 2005), and HH. Total average percentage deviations are shown in the last line of the table. Entries in **boldface** indicate best known value or better was obtained.

m	Q	M	PH	GA	GRASP	HH
40	1	40	6.385	5.346	6.021	**5.300**
	3	120	4.706	3.358	2.788	**2.705**
	4	160	2.983	2.200	1.958	**1.686**
50	1	50	5.192	4.268	4.416	**4.216**
	3	150	2.539	2.199	1.690	**1.503**
	4	200	2.679	1.342	1.133	**0.951**
100	1	100	1.979	1.602	1.467	**1.324**
	3	300	1.214	0.966	**0.511**	0.617
	4	400	1.134	0.997	**0.279**	0.363
150	1	150	1.062	0.859	0.679	**0.636**
	3	450	0.609	0.323	**0.101**	0.145
	4	600	1.105	0.602	**0.152**	0.273
250	1	250	0.829	0.739	0.505	**0.326**
	3	750	0.694	0.460	**0.055**	0.186
	4	1000	0.440	0.138	**0.049**	0.071
500	1	500	0.218	0.170	**0.085**	0.098
	3	1500	0.228	0.175	**0.002**	0.054
	4	2000	0.180	0.146	**0.000**	0.047
1000	1	1000	0.094	0.065	**0.005**	0.007
	3	3000	0.061	0.045	**0.000**	0.009
	4	4000	0.039	0.137	**0.000**	0.020
Average			1.637	1.245	1.043	0.978

TABLE 11. Computational results for problem Set II – Large random problems of Type II. For each instance, the table lists: m, the number of types of rectangles to be packed; Q, the upper bound on the number of rectangles of a given type that can be packed; and M, the maximum number of rectangles that can be packed, as well as average deviations (in percent) from the upper bounds for solutions found by PH (Beasley, 2004), GA (Hadjiconstantinou and Iori, 2006), GRASP (Alvarez-Valdes et al., 2005), and HH. Total average percentage deviations are shown in the last line of the table. Entries in **boldface** indicate best known value or better was obtained.

m	Q	M	PH	GA	GRASP	HH
40	1	40	8.680	6.913	8.172	**6.780**
	3	120	3.071	2.934	2.373	**2.060**
	4	160	3.070	2.772	**2.085**	2.115
50	1	50	5.965	4.845	4.953	**4.401**
	3	150	2.120	1.656	1.315	**1.287**
	4	200	2.776	2.114	**1.236**	1.339
100	1	100	2.390	1.558	1.483	**1.283**
	3	300	1.278	0.87	**0.416**	0.479
	4	400	1.251	0.905	**0.350**	0.498
150	1	150	1.248	0.863	0.737	**0.547**
	3	450	0.520	0.335	0.148	**0.089**
	4	600	0.852	0.567	0.074	0.197
250	1	250	0.952	0.762	0.511	**0.366**
	3	750	0.437	0.306	**0.042**	0.088
	4	1000	0.291	0.378	**0.029**	0.055
500	1	500	0.280	0.246	**0.051**	0.069
	3	1500	0.115	0.113	**0.000**	0.045
	4	2000	0.144	0.171	**0.002**	0.030
1000	1	1000	0.088	0.205	**0.005**	0.009
	3	3000	0.040	0.081	**0.000**	0.002
	4	4000	0.075	0.080	**0.000**	0.004
Average			1.697	1.365	1.142	1.035

TABLE 12. Computational results for problem Set II – Large random problems of Type III. For each instance, the table lists: m, the number of types of rectangles to be packed; Q, the upper bound on the number of rectangles of a given type that can be packed; and M, the maximum number of rectangles that can be packed, as well as average deviations (in percent) from the upper bounds for solutions found by PH (Beasley, 2004), GA (Hadjiconstantinou and Iori, 2006), GRASP (Alvarez-Valdes et al., 2005), and HH. Total average percentage deviations are shown in the last line of the table. Entries in **boldface** indicate best known value or better was obtained.

m	Q	M	PH	GA	GRASP	HH
40	1	40	7.792	6.111	6.717	**5.877**
	3	120	2.849	2.169	1.498	**1.119**
	4	160	3.661	2.217	**1.387**	1.740
50	1	50	5.279	4.575	5.040	**4.354**
	3	150	2.389	1.82	1.501	**1.256**
	4	200	2.421	2.13	**1.177**	1.289
100	1	100	2.419	1.901	1.576	**1.200**
	3	300	1.307	1.14	0.488	**0.483**
	4	400	0.801	0.662	**0.157**	0.172
150	1	150	1.605	1.453	1.250	**0.987**
	3	450	0.666	0.316	**0.160**	0.169
	4	600	0.812	0.618	**0.101**	0.135
250	1	250	0.861	0.735	**0.507**	0.305
	3	750	0.588	0.756	**0.012**	0.048
	4	1000	0.427	0.332	**0.003**	0.028
500	1	500	0.289	0.206	**0.073**	0.019
	3	1500	0.204	0.282	**0.000**	0.026
	4	2000	0.208	0.246	**0.000**	0.013
1000	1	1000	0.086	0.186	**0.002**	0.006
	3	3000	0.098	0.221	**0.000**	0.015
	4	4000	0.089	0.287	**0.000**	0.010
Average			1.660	1.351	1.031	0.917

TABLE 13. Average results for problem Set II – Large random problems of Types I, II, and III. For each instance, the table lists: m, the number of types of rectangles that can be packed; Q, the upper bound on the number of rectangles of a particular type that can be packed; and M, the total number of rectangles that can be packed; as well as average deviations (in percent) from the upper bounds for solutions found by algorithms PH (Beasley, 2004), GA (Hadjiconstantinou and Iori, 2006), GRASP (Alvarez-Valdes et al., 2005), and HH. Averages are computed over all Type I, II, and III results. Entries in **boldface** indicate best known value or better was obtained.

m	Q	M	PH	GA	GRASP	HH
40	1	40	7.62	6.12	6.97	**5.99**
	3	120	3.54	2.82	2.22	**1.96**
	4	160	3.24	2.40	**1.81**	1.85
50	1	50	5.48	4.56	4.80	**4.32**
	3	150	2.35	1.89	1.50	**1.35**
	4	200	2.63	1.86	**1.18**	1.19
100	1	100	2.26	1.69	1.51	**1.27**
	3	300	1.27	0.99	**0.47**	0.53
	4	400	1.06	0.85	**0.26**	0.34
150	1	150	1.31	1.06	0.89	**0.72**
	3	450	0.60	0.32	0.14	**0.13**
	4	600	0.92	0.60	**0.11**	0.20
250	1	250	0.88	0.75	0.51	**0.33**
	3	750	0.57	0.51	**0.04**	0.11
	4	1000	0.39	0.28	**0.03**	0.05
500	1	500	0.26	0.21	0.07	**0.06**
	3	1500	0.18	0.19	**0.00**	0.04
	4	2000	0.18	0.19	**0.00**	0.03
1000	1	1000	0.09	0.15	**0.00**	0.01
	3	3000	0.07	0.12	**0.00**	0.01
	4	4000	0.07	0.17	**0.00**	0.01
Average			1.665	1.320	1.072	0.977

TABLE 14. CPU time (in seconds) for HH on test problems in problem Set II. For each instance, the table lists: m, the number of types of rectangles that can be packed; Q, the upper bound on the number of rectangles of a particular type that can be packed; and M, the total number of rectangles that can be packed; as well as the average CPU times (in seconds) for problems of Types I, II, and III.

m	Q	M	Type I	Type II	Type III
40	1	40	16.54	19.25	18.66
	3	120	102.14	104.39	104.46
	4	160	128.79	132.64	136.40
50	1	50	24.62	25.29	26.39
	3	150	122.89	133.57	131.59
	4	200	158.52	169.29	170.66
100	1	100	84.54	86.99	90.53
	3	300	224.70	226.76	235.46
	4	400	297.58	296.48	298.24
150	1	150	122.06	122.76	125.39
	3	450	*300.00	*300.00	296.02
	4	600	*300.00	*300.00	*300.00
250	1	250	188.30	191.62	195.99
	3	750	*300.00	*300.00	*300.00
	4	1000	*300.00	*300.00	*300.00
500	1	500	*300.00	*300.00	*300.00
	3	1500	*300.00	*300.00	*300.00
	4	2000	*300.00	*300.00	*300.00
1000	1	1000	*300.00	*300.00	*300.00
	3	3000	*300.00	*300.00	*300.00
	4	4000	*300.00	*300.00	*300.00

*A time limit of 300 seconds was imposed.

TABLE 15. Computational results for Set III – Jigsaw-type test problems of Leung et al. (2003). For each problem instance, the table lists: the reference of the problem source; the instance number; the dimension ($W \times H$) of the stock rectangle; m, the number of types of rectangles to be packed; M, the total number of rectangles to be packed; the value of the optimal solution (all but Instance 1 of Lai and Chan (1997b) are zero-waste solutions); the solutions found by GRASP and HH; as well as the HH CPU time. Summary statistics are presented at the bottom of the table. Entries in **boldface** indicate best known value or better was obtained.

Problem source	Instance	Dimension	m	M	Optimum	GRASP	HH	HH time
Lai and Chan (1997b)	1	400×200	10	10	80000	**80000**	**80000**	2.66
	2	400×200	15	15	79000	**79000**	**79000**	5.26
	3	400×400	20	20	160000	154600	**160000**	5.42
Jakobs (1996)	1	70×80	20	20	5600	5447	**5600**	9.50
	2	70×80	25	25	5600	5455	5540	9.95
	3	120×45	25	25	5400	5328	**5400**	15.32
	4	90×45	30	30	4050	3978	**4050**	14.41
	5	65×45	30	30	2925	2871	**2925**	31.60
Leung et al. (2003)	1	150×110	40	40	16500	15856	16172	40.99
	2	160×120	50	50	19200	18628	18860	64.20
					Number of optima / Number of problems:	2/10	7/10	
					Mean percentage deviation from optimum:	2.05	0.48	

TABLE 16. Computational results for Set IV – Doubly constrained problems. Part 1: `ngcut` problems. For each instance, the table lists: the instance name; its dimension ($W \times H$); m, the number of rectangle types to be packed; M, the total number of rectangles to be packed; an upper bound on the optimal solution; the solutions found by heuristics PH (Beasley, 2004), GA (Hadjiconstantinou and Iori, 2006), GRASP (Alvarez-Valdes et al., 2005), and HH; as well as the CPU time (in seconds) for HH. Entries in **boldface** indicate that best value or better was obtained. NFS indicates no feasible solution found. Summary statistics are given at the bottom of the table.

Instance	Dimension	m	M	Upper bound	PH	GRASP	HH	HH time
ngcut01	10×10	5	10	164	**164**	**164**	**164**	1.49
ngcut02	10×10	7	17	230	**225**	**225**	**225**	4.09
ngcut03	10×10	10	21	247	**220**	**220**	**220**	6.52
ngcut04	15×10	5	7	268	**268**	**268**	**268**	1.09
ngcut05	15×10	7	14	358	**301**	**301**	**301**	3.30
ngcut06	15×10	10	15	289	**265**	252	**265**	4.76
ngcut07	20×20	5	8	430	**430**	**430**	**430**	1.93
ngcut08	20×20	7	13	834	**819**	**819**	**819**	4.05
ngcut09	20×20	10	18	924	**924**	**924**	**924**	6.42
ngcut10	30×30	5	13	NFS	NFS	NFS	NFS	3.05
ngcut11	30×30	7	15	1688	1505	**1518**	**1518**	4.62
ngcut12	30×30	10	22	1865	1666	1648	**1672**	8.66
				Number of best / Number of problems:	9 / 12	9 / 12	11 / 12	
				Mean percentage deviation from upper bound:	5.51	5.94	5.41	

TABLE 17. Computational results for Set IV – Doubly constrained problems. Part 2: `hccut`, `okp`, `cgcut`, and `wang` problems. For each instance, the table lists: the instance name; its dimension ($W \times H$); m, the number of rectangle types to be packed; M, the total number of rectangles to be packed; an upper bound on the optimal solution; the solutions found by heuristics PH (Beasley, 2004), GA (Hadjiconstantinou and Iori, 2006), GRASP (Alvarez-Valdes et al., 2005), and HH; as well as the CPU time (in seconds) for HH. Entries in **boldface** indicate that best value or better was obtained. NFS indicates no feasible solution found. Summary statistics are given at the bottom of the table.

Instance	Dimension	m	M	Upper bound	PH	GRASP	HH	HH time
`hccut03`	30×30	7	7	1178	**1178**	**1178**	**1178**	1.63
`hccut08`	30×30	15	15	1270	**1216**	**1216**	**1216**	4.48
`okp01`	100×100	15	50	2726	2499	2700	**2716**	17.61
`okp02`	100×100	30	30	1860	1600	**1720**	**1720**	33.39
`okp03`	100×100	30	30	27718	25373	24869	**25384**	30.14
`okp04`	100×100	33	61	22502	17789	19083	**19526**	13.41
`okp05`	100×100	29	97	NFS	NFS	NFS	NFS	11.72
`cgcut03`	40×70	20	62	32893	27556	27898	**28974**	39.67
`wang20`	70×40	19	42	27923	21997	22011	**22140**	99.03
				Number of best / Number of problems:	2 / 9	3 / 9	8 / 9	
				Mean percentage deviation from upper bound:	11.68	9.32	8.30	

5. CONCLUDING REMARKS

In this paper we addressed a constrained two-dimensional (2D) non-guillotine cutting problem, where a fixed set of small rectangles has to be cut from a larger stock rectangle so as to maximize the value of the rectangles cut. An algorithm which hybridizes a placement strategy with a genetic algorithm based on random keys was proposed. The approach was tested on four sets of instances taken from the literature and compared with other approaches. The experimental results demonstrate the effectiveness and robustness of the proposed heuristic when compared with other heuristics.

ACKNOWLEDGMENTS

The authors would like to thank Francisco Parreño Torres for supplying the upper bounds to the 630 problems of Set II and Manuel Iori for supplying the problem instances `hccut11` and `hccut12`.

This work has been partially supported by funds granted to LIACC/NIAAD through the Programa de Financiamento Plurianual, Fundação para a Ciência e Tecnologia (FCT), Portugal.

REFERENCES

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

A. Amaral and A.N. Letchford. An improved upper bound for the two-dimensional non-guillotine cutting problem. Technical report, Lancaster University, UK, 2001. Available online at `http://www.lancs.ac.uk/staff/letchfoa/ngc.doc`.

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

J.C. Bean. Genetics and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6:154–160, 1994.

D. Beasley, D.R. Bull, and R.R. Martin. An overview of genetic algorithms: Part 1, Fundamentals. *University Computing*, 15:58–69, 1993.

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

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

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

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

L.S. Buriol, M.G.C. Resende, C.C. Ribeiro, and M. Thorup. A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. *Networks*, 46:36–56, 2005.

L.S. Buriol, M.G.C. Resende, and M. Thorup. Survivable IP network design with OSPF routing. *Networks*, 2006. To appear.

A. Caprara and M. Monaci. On the 2-dimensional knapsack problem. *Operations Research Letters*, 32:5–14, 2004.

N. Christofides and C. Whitlock. An algorithm for two dimensional cutting problems. *Operations Research*, 25:31–44, 1977.

C.R. Darwin. *On the origin of species through natural selection*. John Murray, London, 1859.

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

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

M. Ericsson, M.G.C. Resende, and P.M. Pardalos. A genetic algorithm for the weight setting problem in OSPF routing. *J. of Combinatorial Optimization*, 6:299–333, 2002.

S.P. Fekete and J. Schepers. A new exact algorithm for general orthogonal d-dimensional knapsack problems. In *Algorithms - ESA '97*, volume 1284 of *Springer Lecture Notes in Computer Science*, pages 144–156, 1997a.

S.P. Fekete and J. Schepers. On higher-dimensional packing I: Modeling. Technical Report ZPR 97-288, Mathematisches Institut, Universitat zu Köln, 1997b.

S.P. Fekete and J. Schepers. On higher-dimensional packing II: Bounds. Technical Report ZPR97-289, Mathematisches Institut, Universitat zu Köln, 1997c.

S.P. Fekete and J. Schepers. A combinatorial characterization of higher-dimensional orthogonal packing. *Mathematics of Operations Research*, 29:353–368, 2004a.

S.P. Fekete and J. Schepers. A general framework for bounds for higher-dimensional orthogonal packing problems. *Mathematical Methods of Operations Research*, 60:311–329, 2004b.

S.P. Fekete and J. Schepers. An exact algorithm for higher-dimensional orthogonal packing, 2004c. Working paper. Available online at `http://www.math.tu-bs.de/~fekete`.

T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

M.R. Garey and D.S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W.H. Freeman and Company, New York, 1979.

D.E. Goldberg. *Genetic algorithms in search optimization and machine learning*. Addison-Wesley, 1989.

J.F. Gonçalves. A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. *European Journal of Operational Research*, 2006. To appear.

J.F. Gonçalves and J.R. Almeida. A hybrid genetic algorithm for assembly line balancing. *Journal of Heuristics*, 8:629–642, 2002.

J.F. Gonçalves, J.J.M. Mendes, and M.G.C. Resende. A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167:77–95, 2005.

J.F. Gonçalves and M.G.C. Resende. An evolutionary algorithm for manufacturing cell formation. *Computers and Industrial Engineering*, 47:247–273, 2004.

E. Hadjiconstantinou and N. Christofides. An exact algorithm for general, orthogonal, two dimensional knapsack problems. *European Journal of Operations Research*, 83:39–56, 1995.

E. Hadjiconstantinou and M. Iori. A hybrid genetic algorithm for the two-dimensional knapsack problem. *European Journal of Operational Research*, 2006. To appear.

R.W. Haessler and P.E. Sweeney. Cutting stock problems and solution procedures. *European Journal of Operational Research*, 54:141–150, 1991.

P. Healy, M. Creavin, and A. Kuusik. An optimal algorithm for rectangle placement. *Operations Research Letters*, 24:73–80, 1999.

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

S. Jakobs. On genetic algorithms for the packing of polygons. *European Journal of Operational Research*, 88:165–181, 1996.

K.K. Lai and J.W.M. Chan. An evolutionary algorithm for the rectangular cutting stock problem. *International Journal of Industrial Engineering*, 4:130–139, 1997a.

K.K. Lai and J.W.M. Chan. Developing a simulated annealing algorithm for the cutting stock problem. *Computers and Industrial Engineering*, 32:115–127, 1997b.

T.W. Leung, C.K. Chan, and M.D. Troutt. Applications of genetic search and simulated annealing to the two-dimensional non-guillotine cutting stock problem. *Computers and Industrial Engineering*, 40:201–214, 2001.

T.W. Leung, C.K. Chan, and M.D. Troutt. Application of a mixed simulated annealing-genetic algorithm heuristic for the two-dimensional orthogonal packing problem. *European Journal of Operational Research*, 141:241–252, 2003.

D. Liu and H. Teng. An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles. *European Journal of Operational Research*, 112:413–420, 1999.

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

W.M. Spears and K.A. Dejong. On the virtues of parameterized uniform crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, 1991.

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

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

P.Y. Wang. Two algorithms for constrained two-dimensional cutting stock problems. *Operations Research*, 31:573–586, 1983.

G. Wäscher, H. Haussner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 2006. To appear.

(José Fernando Gonçalves) FACULDADE DE ECONOMIA DO PORTO / NIAAD, RUA DR. ROBERTO FRIAS, 4200-464, PORTO, PORTUGAL.

E-mail address: jfgoncal@fep.up.pt

(Mauricio G. C. Resende) ALGORITHMS AND OPTIMIZATION RESEARCH DEPARTMENT, AT&T LABS RESEARCH, 180 PARK AVENUE, ROOM C241, FLORHAM PARK, NJ 07932 USA.

E-mail address: mgcr@research.att.com