Heuristics for Two-Dimensional Knapsack and Cutting Stock Problems with Items of Irregular Shape

Aline M. Del Valle[a], Thiago Alves de Queiroz[b,*], Flávio K. Miyazawa[a], Eduardo C. Xavier[a]

[a] *Institute of Computing, IC, UNICAMP, 13084-971, Campinas - SP, Brazil.*
[b] *Department of Mathematics, DM, UFG/CAC, 75704-020, Catalão - GO, Brazil.*

Abstract

In this paper, the two-dimensional cutting/packing problem with items that correspond to simple polygons that may contain holes are studied in which we propose algorithms based on No-Fit polygon computation. We present a GRASP based heuristic for the 0/1 version of the Knapsack Problem, and another heuristic for the unconstrained version of the Knapsack Problem. This last heuristic is divided in two steps: first it packs items in rectangles and then use the rectangles as items to be packed into the bin. We also solve the Cutting Stock problem with items of irregular shape, by combining this last heuristic with a column generation algorithm. The algorithms proposed found optimal solutions for several of the tested instances within a reasonable runtime. For some instances, the algorithms obtained solutions with occupancy rates above 90% with relatively fast execution time.

Keywords: two-dimensional irregular shape cutting and packing problems, GRASP, column generation.

1. Introduction

In this paper we investigate cutting and packing problems with items of irregular shape. These problems have several practical applications in garment manufacturing, sheet metal-cutting, furniture making, and shoes manufacturing (Gomes & Oliveira, 2002; Bennell & Oliveira, 2009).

There are several papers that deal with the two-dimensional (2D) version of cutting and packing problems for rectangular shapes. However, packing problems with items of irregular shape have not received as much attention. This probably happened because the lack of good approaches to detect feasible regions to pack items with complex shape. Recently, some new approaches were proposed and used to check/generate feasible regions to pack items of irregular shape. One of the most successful approaches is based on no-fit polygon

*Corresponding author. Tel.: +55 64 34415316. Fax: +55 64 34415320.
Email addresses: amdvalle@gmail.com (Aline M. Del Valle), th.al.qz@pq.cnpq.br (Thiago Alves de Queiroz), fkm@ic.unicamp.br (Flávio K. Miyazawa), eduardo@ic.unicamp.br (Eduardo C. Xavier)

constructions (Burke et al., 2007), but other approaches has also been developed such as the use of phi-functions (Chernov et al., 2010).

In the problems studied in this paper, we assume that we have a list I of two-dimensional items of irregular shape to be packed. The items are represented by simple polygons, with oriented edges. Each item $i \in I$ is defined by a list of points $(p_1, \ldots, p_{x_i})$, where there are edges between p_j and p_{j+1} for $j = 1, \ldots, x_i - 1$ and finally between p_{x_i} and p_1. Some of these items may contain holes, and then, for each hole we consider a list of points that represents it. In some instances, items can be rotated and in this case we have a set of allowable rotations in degrees, $\Theta = \{\theta_1, \ldots, \theta_m\}$. The bins are rectangles with width W and length L. A feasible placement of items in a bin consists in a placement where no two items overlap and all of them fit inside the bin.

The problems of interest in this paper are the following (special cases of these problems are already $\mathcal{NP}$-hard (Garey & Johnson, 1979)):

Two-dimensional 0/1 Knapsack Problem (2KP-(0/1)): An input instance consists of a rectangular bin B and a list I of n items of irregular shape. Each item i has value v_i, for $i = 1, \ldots, n$. We consider that the value of each item corresponds to the items area. The aim is to decide how to cut a subset of items in I to maximize the total value of the items produced, i.e., maximize the total area of items packed. In this problem, each item $i \in I$ may be produced at most once.

Two-dimensional Unconstrained Knapsack Problem (2KP): In this problem we have the same input as in the 2KP-(0/1) problem. But now one item of I can be cut several times from the bin. The aim is the same, but many copies of one item can be produced (again some items may not occur), hence the term *unconstrained*.

Two-dimensional Cutting Stock Problem (2CS): We are given an unlimited quantity of rectangular bins B, and a list I of n items of irregular shape, each item i with demand d_i for $i = 1, \ldots, n$. The aim is to decide how to pack all items considering their demand using the minimum number of bins.

According to Bennell & Oliveira (2008) these problems are also referred to in the literature as *nesting problems*, that are problems where more than one item of irregular shape must be placed in a configuration to optimize some objective function.

Albano & Sapuppo (1980) presented a nesting algorithm combined with the no-fit polygons that aims to reduce the geometric complexity of the nesting process. The no-fit polygon has become a popular and efficient option for dealing with the geometry of packing problems with irregular shapes. The purpose of no-fit polygons is to present regions of possible place-

ment of items such that they do not overlap. Adamowicz & Albano (1976) introduced the term no-fit polygon. Oliveira et al. (2000) also use no-fit polygons functions to solve nesting problems.

Daniels & Milenkovic (1997) proposed exact and approximation algorithms for nesting problems restricted to the case with at most seven different items, in which the items and the bin have convex shape.

Martins (2000) presented a genetic algorithm combined with the bottom-left-fill approach for a strip packing version of the irregular nesting problem.

Burke et al. (2007) presented an overview of several techniques for the no-fit polygon computation, and then, propose a complete and robust no-fit polygon algorithm. Their approach is based on a robust orbital method, such that one item orbits above another item to generate the no-fit-polygon. A recent survey covering the core geometric methodologies employed in cutting and packing problems with items of irregular shape, was done by Bennell & Oliveira (2008).

Martins & Tsuzuki (2010) proposed a heuristic based on simulated annealing for the problem of minimizing the waste while packing a set of items of irregular shape in a bin, in which rotation of the items are allowed. Other approaches especially meta-heuristics can be found in Egeblad et al. (2007); Lee et al. (2008).

In this paper we propose the following algorithms for the considered problems with items of irregular shape:

- For the 2KP-(0/1) we propose a GRASP heuristic in which for each initial greedy-randomized solution generated, a local search is performed to improve this solution. The initial solution is generated by packing item by item according to some specific order. In the local search we try to rearrange, remove/include new items while maximizing the value of items packed.

- For the 2KP we propose a strategy that works in two phases. First we pack items in rectangles such that these rectangles have high occupancy rate. Then we pack these generated rectangles optimally using an exact dynamic programming approach developed by Cintra et al. (2008) with the use of *reduced raster points* of Scheithauer (1997).

- For the 2CS we use a column generation approach as in Cintra et al. (2008). So to generate columns we use the previous proposed heuristic for the 2KP.

This paper is organized as follows. In Section 2, we discuss the 2KP-(0/1). We outline

how the no-fit polygon is computed and the GRASP based heuristic developed to solve this problem. Sections 3 and 4 present the algorithm to solve the 2KP and outlines the column generation approach for the 2CS. In Section 5, experiments are reported to show the efficiency of the proposed algorithms. Finally in Section 6 some concluding remarks are presented.

2. 2KP-(0/1)

In this section we present the GRASP heuristic for the 2KP-(0/1). First we discuss the use of No-Fit Polygon (NFP) to get feasible positioning of items in the generated packing. Finally we describe how our heuristic works.

2.1. The No-Fit Polygon Generation

There are some different approaches on how to decide feasible (no overlap) positioning of two given items of irregular shape. Some of these approaches include direct trigonometry, the phi-function and the no-fit polygon (NFP) computation. According to Bennell & Oliveira (2008), the NFP is one of the most promising strategies to find feasible packing. In our work we use the NFP to decide feasible positions when packing a given item in some existent packing.

Given two polygons, one of them is fixed, lets say polygon A, while the other is the orbital polygon, say B. Given a reference point of B, we aim to build another polygon orbiting B along A, in such a way that B is always touching A, but they never overlap. The movement of the reference point while orbiting B along A, generates another polygon called NFP(A, B). The NFP(A, B) is another polygon, that combines the two items A and B, in such a way that: (i) when the reference point of B is positioned in the interior of the NFP, then A and B overlaps; (ii) when the reference point of B is positioned in the boundary of the NFP, then A and B are touching and, (iii) when the reference point of B is positioned outside the NFP, then A and B are separated.

In this paper we implemented the robust approach developed by Burke et al. (2007) to generate NFPs. We call this algorithm by NoFitP. This algorithm is divided in two phases. In the first phase, the item B slides around the item A to create the outer path of the no-fit polygon of these two items. The second phase consists in finding the remaining paths (internal holes) of the no-fit polygon that were not found in the first phase. Details about the algorithm NoFitP can be found in Burke et al. (2007).

2.2. The Extended Search Algorithm

Given a packing P of some items, we want to pack a new item in P such that this new packing is possible, and it is the best new packing according to some quality measure. For that, we adapted the extended search algorithm proposed by Adamowicz & Albano (1976) to compute the best position inside the bin to pack a new item. In our adaptation, we want to find a point where to pack the new item, such that the area of the rectangular and convex closures of the entire new packing is minimized. The dimensions of the bin must also be considered. We denominate such algorithm by ExSEARCH.

Given two polygons, some fixed polygon A (that corresponds to the union of items already packed) and an orbital polygon B (a new item to be packed), the extended search of Adamowicz & Albano (1976) proceeds as follows: For each vertex of the NFP(A, B), find the position, along the edge that leaves this vertex, that minimizes the area of the rectangular closure of A and B. To find such position, the algorithm considers some special points along the edge, called the span points (details in Adamowicz & Albano (1976)). In our adaptation, among all position points that minimizes the area of the rectangular closure, we consider as the best point, the one that minimizes the area of the convex closure of A and B when packed in this position.

2.3. Packing Single Items

Now we describe the algorithm that uses the algorithms NoFitP and ExSEARCH with the aim to pack an item in a bin that may already contain items. This algorithm is referred to as PackS and presented in Algorithm 1.

For a given instance, we first pre-compute the NFP$(i, \theta_i, j, \theta_j)$ for each pair of items i (rotated by θ_i degrees) and j (rotated by θ_j degrees), and for each of their allowable pair of rotations θ_i and θ_j. When packing a new item i in some given packing P, consisting of items $P = \{x_1, \ldots, x_k\}$, we compute the union U of the items in P and then compute the NFP(U, i). Another alternative is to compute an approximation of NFP(U, i) by computing the union of pairs of pre-computed NFPs

$$\cup_{j=1}^{k} \text{NFP}(x_j, i).$$

This second approach was much faster in our experiments, and provided better results, so it is used in the PackS algorithm.

The algorithm PackS has as input an item i to be packed, the set of allowable rotations Θ that this item may assume in which $|\Theta| = m$, the dimensions of the bin (L, W), and a (partial) packing P of items. It returns false if the item i can not be packed inside the bin

of dimensions (L, W). Otherwise, it returns the point (x, y) where the item i can be packed and the rotation θ_i considered for such packing.

Algorithm 1: Algorithm PackS.

Input : item i; set of rotations Θ for i; dimensions of the bin (L, W); packing P.

Output : point (x, y) and angle θ to pack item i inside bin.

```
1.1  flag ← false;
1.2  areaR ← 0;
1.3  foreach θ_i ∈ Θ do
1.4      flag' ← false;
1.5      nfp ← { };
1.6      foreach item (j, θ_j) ∈ P do
1.7          nfp ← nfp ∪ NoFitP(i, j, θ_i, θ_j);
1.8      (x', y', flag') ← ExSEARCH(nfp, i, θ_i, L, W);
1.9      if flag' then
1.10         Let areaR' be the area of the rectangular closure for P ∪ {i, (x', y'), θ_i};
1.11         if areaR' < areaR OR areaR = 0 then
1.12             (x, y, θ) ← (x', y', θ_i);
1.13             areaR ← areaR';   flag ← true;
1.14 if flag then
1.15     return {true, (x, y), θ};
1.16 return {false, (0, 0), 0};
```

Given a current packing P, the idea of the algorithm is to pack item i in P, in each of its possible rotations. Among these new generated packing, the one with smallest rectangular closure area is returned.

In the algorithm PackS, the variable *flag* and *flag*$'$ indicate whether item i can be packed inside the bin (which is checked by the algorithm ExSEARCH given the bin dimensions and items already packed). The variable *nfp* corresponds to the union of the NFPs of i and each packed item j. Thus *nfp* has feasible positions where to pack item i given the current items packed. This variable is computed using pre-computed NFPs. The ExSEARCH algorithm finds the best position (as explained in section 2.2) where to pack item i considering the current packing. It returns in *flag*$'$ (line 1.8) if there is a feasible packing, and if so, it returns in x' and y' the best position where to pack i.

For each possible rotation θ_i of item i, in the main loop (line 1.3), the algorithm computes the best packing of i given the current packing P. If this packing is feasible (line 1.9), then the algorithm computes the rectangular closure area, and updates the best found solution

so far if necessary (lines 1-11 – 1.13). At the end, the algorithm returns the best packing, if a feasible one exists, or returns false.

2.4. GRASP Heuristic for the 2KP-(0/1)

Now, we present a GRASP heuristic to solve the 2KP-(0/1). We consider that the value of each item corresponds to its area. The first step is to generate an initial solution (using a greedy-randomized algorithm). Then, a local search algorithm that aims to improve the initial solution is repeatedly applied.

The greedy-randomized algorithm used to generate the initial solution constructs a solution in which the items are packed one by one. This algorithm is described in Algorithm 2 and it is called by IniSOL.

Algorithm 2: IniSOL.

Input : set of items I; set of rotations Θ for items in I; dimensions of the bin (L, W); percentage p of items to be chosen at random.

Output : solution (packing) P; set $I_1 \subseteq I$ with the items that are in P; set $I_2 = I \setminus I_1$ with the items that are not in P.

2.1 $P \leftarrow \{\ \}$; $I_1 \leftarrow \{\ \}$; $I_2 \leftarrow \{\ \}$;

2.2 **while** $I \neq \{\ \}$ **do**

2.3 $I' \leftarrow$ choose p% of the items from I at random;

2.4 **foreach** $i \in I'$ **do**

2.5 $(\mathit{flag}, x_i^*, y_i^*, \theta_i^*) \leftarrow \text{PackS}(i, \Theta, L, W, P)$;

2.6 **if** $\mathit{flag} = \mathit{true}$ **then**

2.7 Compute the cost to add i into S;

2.8 **else**

2.9 $I' \leftarrow I' - \{i\}$; $I \leftarrow I - \{i\}$; $I_2 \leftarrow I_2 \cup \{i\}$;

2.10 **if** $I' \neq \{\ \}$ **then**

2.11 Select $i \in I'$ of smallest cost with its rotation θ_i^* and packing position (x_i^*, y_i^*);

2.12 $I \leftarrow I - \{i\}$; $I_1 \leftarrow I_1 \cup \{i\}$; $P \leftarrow P \cup \{i, \theta_i^*, (x_i^*, y_i^*)\}$;

2.13 **return** (P, I_1, I_2);

In each iteration of lines 2.2–2.12, the algorithm IniSOL chooses a random subset of items to generate a candidate set of items I'. For each item $i \in I'$, this item is packed together with previously packed items P, in an optimal position according to algorithm PackS (loop of line 2.4). Among the items in I' that have a feasible packing with P, the algorithm selects the item that generates the best cost solution. Items that have a feasible packing are the ones that stay in list I' at line 2.10. The cost is related to the total area of items packed in

the bin (denoted by T) and with the area of the rectangular closure of these items (denoted by AR). The best cost solution is the one with largest $\frac{T}{AR}$ value.

The item i that generates the best new packing is, then, packed with P (lines 2.11–2.12) and the algorithm repeats the process trying to pack some new item. Notice that items that cannot be packed in the current solution are removed from the list I and included in list I_2 (line 2.9).

The entire process is repeated while there are items in I to be evaluated. Also notice that items without a feasible packing (stored in I_2) are not evaluated anymore. Items in I' that were not chosen as a best packing, can be evaluated later.

The solution P returned by IniSOL is the first solution for the GRASP based heuristic. The next step is to improve such solution using a local search, denoted here by LocSEARCH (see Algorithm 3). We assume that a solution P consists of a sequence of packed items $P = ((i_1, \theta_1), \ldots, (i_s, \theta_s))$ with their corresponding orientations (these items belongs to set $I_1 \subseteq I$). The solution is obtained by applying algorithm ExSEARCH, on each item in this order. A *neighbor* solution of a given solution P is obtained by applying changes into P. The changes can be insertions of new items into P (items in the set I_2), or permutations of items into P. The local search procedure finishes when a solution can not be improved.

A solution P_1 is considered better than another solution P_2 if the total area of items in P_1 is larger than the total area of items in P_2. If the items area are the same, then P_1 is better than P_2 if it is more compact, i.e., its rectangular closure area is smaller.

Algorithm 3: LocSEARCH.

Input : initial solution P; set I_1 containing items that are in P; set I_2 containing items that are not in P; dimensions of the bin (L, W); number of iterations x; probabilities (q_1, q_2, q_3) for each type of neighbor.

Output : solution P possibly improved.

```
3.1 while P was improved do
3.2     for i ← 1 to x do
3.3         Choose a type of neighborhood operation in {t1, t2, t3} at random considering
            probabilities (q1, q2, q3);
3.4         Compute a neighbor of the solution P considering type ti, and sets I1 and I2;
3.5     if there is one neighbor P' of P that improves the solution then
3.6         P ← P';
3.7 return P;
```

On each iteration of the $w$hile loop at line 3.1, the algorithm generates x neighbors of the current solution P using the sets I_1 and I_2. The neighbors are obtained by insertions and

permutations of items in P. Three different types of neighborhood operators are considered:

- t_1 : A neighbor is obtained by permuting two items of P. This new order of the items may produce a more compact packing leaving space for some item $j \in I_2$ to be packed later. Even if item j can not be packed in I_1, the new solution (neighbor) may be better than the current solution. The new solution is better if its rectangular closure is smaller than the current solution. This operator is chosen with probability q_1;
- t_2 : In this case, a neighbor is obtained by exchanging a packed item $i \in I_1$ with an item $j \in I_2$. The exchange occurs only if the area of the item j is greater than the area of item i. After the change, if all the items in I_1 can be packed, this new solution is better than the current one, since its occupied area is greater than the area of the current solution. This operator is chosen with probability q_2;
- t_3 : A neighbor is obtained by inserting an item $j \in I_2$ into I_1. If all the items can be packed, this solution is clearly better than the current one. This operator is chosen with probability q_3.

For all neighborhood operators presented, the insertions and permutations of items occurs only in the last p_q fraction of items in the tuple $P = ((i_1, \theta_1), \ldots, (i_s, \theta_s))$. We use this strategy since repacking all items is very time-consuming, so keeping an initial partial packing, and computing the packing only for the last p_q of items is faster. Each neighbor type t_i is selected by the algorithm LocSEARCH at random. But notice that the probability to chose type t_1 (q_1), should be greater than the other probabilities, since we should try to improve the given packing creating new space, for then try to include other items in the solution. Notice that an optimal packing is obtained when all items in I are packed, (I_2 becomes empty). In this case the algorithm LocSEARCH should halt.

We present in Algorithm 4 the GRASP heuristic to solve the 2KP-(0/1). This heuristic uses the algorithm IniSOL to generate initial solutions and the algorithm LocSEARCH to improve the initial solutions, if possible.

Remember that the goal in the 2KP-(0/1) is to maximize the value of the items that are packed. Since the value of each item corresponds to its area, the best solution corresponds to the one that has largest total items area. If two solutions have equal values, then the GRASP heuristic considers as the best solution the one with the smaller rectangular closure area.

Algorithm 4: GRASP algorithm to solve the 2KP-(0/1).

Input : set of items I; set of rotations Θ for items in I; dimensions of the bin (L, W); maximum number of iterations *mIter*; number of neighbors to generate x; percentage p_q of items to be chosen at random; probabilities (q_1, q_2, q_3) for each type of neighbor.

Output : solution P.

4.1 $P \leftarrow \{\ \}$;
4.2 **for** $i \leftarrow 1$ **to** *mIter* **do**
4.3 $\quad (P', I_1, I_2) \leftarrow \text{IniSOL}(I, \Theta, L, W)$;
4.4 $\quad P' \leftarrow \text{LocSEARCH}(P', I_1, I_2, L, W, x, q_1, q_2, q_3, p_q)$;
4.5 $\quad$ **if** P' *is better than* P **then**
4.6 $\quad\quad P \leftarrow P'$;
4.7 **return** P;

3. Algorithm for the 2KP

In this section we present an algorithm to solve the 2KP. This algorithm first packs subsets of items of irregular shapes into small rectangles with a high occupancy ratio (given by items area over rectangular closure area), and then, it uses an optimum exact dynamic programming algorithm proposed by Cintra et al. (2008) to solve the 2KP for rectangular items. Here, we also assume that the value of each item corresponds to its area.

The algorithm that is used to pack subsets of irregular items into small rectangles is denoted by GenRET and it is presented in the Algorithm 5.

The idea is to select items at random and then pack them one by one, using algorithm PackS to find the best position for each item, and check for each generated packing, if its occupancy ratio is at least a threshold value α. If so, then this packing is saved in a set S as one rectangle with dimensions that corresponds to the rectangular closure of the packing. And the value of this rectangle corresponds to the value of the items packed on it. The algorithm GenRET repeats this process until a maximum number of iterations is reached or a maximum number of rectangles is created. Notice that on each iteration of the main loop (line 5.3) a rectangle *rtg* is generated from the set I' given, which has $p\%$ of the items from I chosen at random.

It is possible that after the execution of algorithm GenRET, several rectangles are generated, but several items do not appear in any one of these generated rectangles. In order to guarantee that each item appears in at least one rectangle, we execute again algorithm GenRET with the set of unpacked items, and with smaller values α of occupancy ratio, since

Algorithm 5: GenRET.

Input : set of items I; set of rotations Θ for items in I; minimum occupancy ratio α; maximum allowable dimensions (L, W) for a rectangle; maximum number $\max_i$ of iterations; maximum number $\max_r$ of rectangles to be generated; percentage $p\%$ of items to be chosen at random.

Output : set of rectangles S whose occupancy ratio is at least α.

```
5.1  S ← { };
5.2  j, l ← 0;
5.3  while j < max_i AND l ≤ max_r do
5.4      j ← j + 1;
5.5      rtg ← { };
5.6      I' ← choose p% of items from I at random;
5.7      foreach i ∈ I' do
5.8          (flag, x, y, θ) ← PackS(i, Θ, L, W, rtg) ;
5.9          if flag ≠ false then
5.10             Pack item i into rtg on point (x, y) and at rotation θ;
5.11             occupp ←(sum of the area of items packed in rtg) / (area of the rectangular closure
                 of rtg);
5.12             if occup ≥ α then
5.13                 S ← S ∪ {rtg};
5.14                 l ← l + 1;
5.15 return S;
```

rectangles that has such items were not generated possible because these items do not induce rectangles with high occupancy ratio.

In Algorithm 6, denominated by PackAllRET, we presented an algorithm that tries to generate rectangles with high occupancy ratio, but that guarantees that each item appears in at least one generated rectangle. The algorithm PackAllRET first packs items into rectangles with high occupancy ratio. If there are items that were not packed in any of the generated rectangles, then the algorithm repeats the generation of rectangles using a smaller value of occupancy ratio threshold α. The value of α is decremented on each iteration by a small value (ϵ), so new rectangles may be generated with unpacked items. The value of ϵ is chosen to allow all the items in I to be packed.

After generating rectangles containing items of irregular shape, our strategy is to use the exact algorithm developed by Cintra et al. (2008) to pack rectangles into the bin. The exact algorithm of Cintra et al. (2008) consists in solving the recurrence formula proposed

Algorithm 6: PackAllRET.

Input : set of items I; set of rotations Θ for items in I; initial occupancy ratio α_0; maximum allowable dimensions (L, W) for a rectangle; maximum number $\max_i$ of iterations; maximum number $\max_r$ of rectangles to be generated; percentage $p\%$ of items to be chosen at random; value ϵ used to decrement the occupancy ratio.

Output : set of rectangles S with different occupancy rates.

6.1 $S \leftarrow \{\ \}$;

6.2 $\alpha \leftarrow \alpha_0$;

6.3 **while** $\alpha > 0$ **AND** *there are items not packed yet* **do**

6.4 $\quad I' \leftarrow$ items of I that are not packed in any rectangle in S;

6.5 $\quad S' \leftarrow \text{GenRET}(I', \Theta, \alpha, L, W, \max_i, \max_r, p\%)$;

6.6 $\quad S \leftarrow S \cup S'$;

6.7 $\quad \alpha \leftarrow \alpha - \epsilon$;

6.8 **return** S;

by Beasley (1985) considering guillotine cutting patterns generated over *discretization points* (see Herz (1972)). We use the same algorithm of Cintra et al. (2008), but instead of using discretization points we use the so-called *reduced raster points* (raster points) (see Scheithauer (1997)). The use of raster points reduces much the running time of algorithms (see for instance Queiroz et al. (2012)). We denote this algorithm by DP2KP.

Now, we can describe the algorithm that solves the 2KP, which we denote by Solve2KP:

i) Generate a set S of rectangular items using algorithm PackAllRET;

ii) Create an instance I for the 2KP for rectangular items by considering the rectangles in S. Each rectangle $r \in S$ has value v_r, that is equal to the total area of items of irregular shape that are packed in it;

iii) Execute algorithm DP2KP for instance I using raster points as discretization points;

iv) Return the solution computed by DP2KP as the final solution for the 2KP.

Note that in the final solution, each rectangle represents a subset of items (polygons) packed in it.

4. The Column Generation Heuristic for the 2CS

In this section, we consider the Two-dimensional Cutting Stock problem with items of irregular shape, denoted by 2CS. We use a column generation algorithm to solve such problem where the columns are generated by the algorithm Solve2KP.

For a given instance of the problem, let $\mathcal{P}$ be the set of all cutting patterns and let $|\mathcal{P}| = m$ denote its size. The linear program formulation uses a matrix P with dimensions $n \times m$ where each column represents a cutting pattern, and each row corresponds to an item. Each column $p = (p_1, \ldots, p_n)$, corresponding to a pattern, has on line i the value p_i, that corresponds to the number of times item i is packed in this pattern. There is a vector $d = (d_1, \ldots, d_n)$, that corresponds to the demand of each item. We use the integer variable x_j, for each pattern $j \in \mathcal{P}$, to indicate how many times the pattern j is to be used in the solution. The following linear program is a relaxation of the equivalent integer formulation:

$$\begin{aligned} \textit{minimize} \quad & \sum_{j \in \mathcal{P}} x_j \\ \textit{subject to} \quad & Px \geq d \\ & x_j \geq 0, \quad \forall\, j \in \mathcal{P}. \end{aligned} \tag{1}$$

We use the algorithm CG^p presented in Cintra et al. (2008) to solve the 2CS. The main idea of such algorithm is to solve the linear formulation (1) using the column generation approach proposed by Gilmore & Gomory (1961, 1963). Given a fractional solution $\hat{x}$, the algorithm round down the fractional variables. It uses $\lfloor \hat{x} \rfloor$ as an integer solution, but probably some demand of items are not completely fulfilled. These items with remaining demand are then considered as a residual instance, that is then solved again with the same method, and using a primal heuristic. The primal heuristic M-HFF is similar to the HFF algorithm to get cutting patterns. As the heuristic M-HFF can not handle items with irregular shape, in this heuristic we consider each item as the smallest rectangle in which the item can be packed.

During the column generation process, the algorithm must generate a new column to be inserted into the basis. To this end, the dual value y_i is computed for each item i. The new column (pattern) must satisfies the condition $\sum_{i=1}^{n} y_i z_i > 1$, so z_i is the number of times that item i appear in such pattern. Clearly, we can use any algorithm developed for the 2KP to compute these patterns, but we need that items value correspond to their dual value y_i. The subroutine used to generate columns corresponds to the algorithm Solve2KP with a simple modification on algorithm GenRET.

We first compute an upper bound for an optimal solution of the 2KP. Let i be the item with maximum v_i/a_i ratio. We use as an upper bound U the value of the area of the bin multiplied by v_i/a_i. The value $V = U/(L \cdot W)$ corresponds to an optimum value by unit area of the bin.

The change on algorithm GenRET consists to generate rectangles with items value at least α times $V \cdot A$, where A is the area of the rectangle closure of the items being considered.

5. Computational Results

All algorithms were implemented in $C/C++$ and we also used the library CGAL (Computational Geometry Algorithms Library) available at the url: `http://www.cgal.org/`, to do some geometrical operations. We also used the COIN-OR CLP solver (Coin-OR CLP) in the column generation algorithm to solve systems of linear equations.

The tests were performed on a computer with processor Intel® Core^{TM} 2 Quad 2.4 GHz, 4 GB of RAM, and *Linux* operating system.

There are few works for packing problems with items of irregular shape (Wascher et al., 2007), and specially we could not find any related work for the 2CS when items have irregular shape. For the 2KP-(0/1) we could find some related works, but none of them provided information about running times to solve the tested instances. The only work that provided running times, and that it is related to the 2KP-(0/1) is the one of Martins & Tsuzuki (2010). In their work, they presented Simulated Annealing (SA) Heuristics to solve the rotational placement of multiple items inside a container of fixed dimensions, and evaluated their algorithms on some simple instances. In some of their instances, the recipient is non-rectangular and items may be rotated at an arbitrary angle. To compare their algorithm with ours, we considered the Martins and Tsuzuki's instances that had rectangular recipient. In our algorithm, we limited to eight the number of possible rotation angles of items.

Also, to test our algorithms, we adapted some other known instances for packing problems with one open dimension. The generated instances were adapted from the Two-Dimensional Irregular Strip Packing problem Gomes & Oliveira (2006), and they can be found at the url: `http://www.fe.up.pt/esicup/`. In these instances, the following information is available: the quantity of items, where each item is represented by a polygon; the set of allowable rotations for these items; and, the length of the strip. In the problems considered in this work, the bins have fixed rectangular dimensions. So we adapted the instances from Gomes & Oliveira (2006), where the length of the bin is the same length available and the width of the bin corresponds to the height obtained by Gomes & Oliveira (2006) when solving the strip packing version of the problem. Since the 2KP and 2CS assume that the recipient have integer dimensions, we considered the ceil of the dimensions of the bin in this case. Besides that, it was necessary to generate demand for the instances of the 2CS. For each item we choose at random a demand value in the interval $[1,\ 100]$. The instances used in this work are available upon request or at the url: `http://www.loco.ic.unicamp.br/instances/irregular2d/`.

Table 1 presents some information about the instances used in this work. A total of 19 instances were considered for computational tests. Each row of this table has the following

information: instance name (Name); quantity of items (n); length of the bin (L); width of the bin (W); and, set of allowable rotations for the items (Rotations); The last four instances of the table are the ones from Martins & Tsuzuki (2010), and the remaining instances are the ones adapted from Gomes & Oliveira (2006).

Table 1: Informations about the instances under consideration.

Name	n	L	W	Rotations
FU	12	38	34	(0, 90, 180)
JACKOBS1	25	40	13	(0, 90, 180)
JACKOBS2	25	70	28.2	(0, 90, 180)
SHAPES0	43	40	63	(0)
SHAPES1	43	40	59	(0, 180)
SHAPES2	28	15	27.3	(0, 180)
DIGHE1	16	100	138.13	(0)
DIGHE2	10	100	134.05	(0)
ALBANO	24	4900	10122.63	(0, 180)
DAGLI	30	60	65.6	(0, 180)
MAO	20	2550	2058.6	(0, 90, 180)
MARQUES	24	104	83.6	(0, 90, 180)
SHIRTS	99	40	63.13	(0, 180)
SWIM	48	5752	6568	(0, 180)
TROUSERS	64	79	245.75	(0, 180)
LARGER FIRST FAILS PUZZLE	5	400	400	(0, 45, 90, 135, 180, 225, 270, 315)
BOTTOM LEFT FAILS PUZZLE	5	280	280	(0, 45, 90, 135, 180, 225, 270, 315)
SMALL PUZZLE	4	100	100	(0, 45, 90, 135, 180, 225, 270, 315)
TANGRAM	7	800	800	(0, 45, 90, 135, 180, 225, 270, 315)

The GRASP heuristic as well as the algorithms used as subroutines depend on several parameters which need to be well-defined. Likewise, the algorithms Solve2KP and Solve2CS (and their subroutines) depend on various parameters too.

In Table 2 we summarize the values for all those parameters considering all the algorithms previously discussed. It is worth mentioning that such values were chosen after an extensive execution of tests considering the instances presented in Table 1.

In Table 3 we present the results of the SA heuristic of Martins & Tsuzuki (2010) and the results of the GRASP heuristic for the 2KP-(0/1). In Martins & Tsuzuki (2010), the SA heuristics were executed in a 2.21 GHz Phenom 9550 processor, but no information about main memory is provided. So our comparisons must be taken carefully. In Martins & Tsuzuki (2010) the SA heuristic was executed 30 times. So in this experiment we also executed the

Table 2: Parameters required by the algorithms.

Parameter	Value considered	Algorithm that uses such parameter
p	10%	GRASP
x	25	GRASP
(q_1, q_2, q_3)	(70, 20, 10)	GRASP
p_q	60%	LocSEARCH
$mIte$	15	GRASP
pu	80%	Solve2KP, Solve2CS
B	90%	Solve2KP, Solve2CS
m	15	Solve2KP, Solve2CS
r	two times the quantity of items n	Solve2KP, Solve2CS
ϵ	10%	Solve2KP, Solve2CS
(L_R, W_R)	same values considered for (L, W)	Solve2KP, Solve2CS

GRASP heuristic 30 times for each instance. Since the number of items is small, we used $p_q = 100\%$, for this experiment.

The columns of Table 3 are: instance name (Name), percentage of runs that converged to the optimum solution (P_{conv}) for SA and GRASP, time spent by the SA heuristic to find the best solution (T_{min}) in seconds, time in seconds spend by the GRASP heuristic in the fastest run (Best time), including the time needed to generate the NFPs, and average time spent by all GRASP runs (Average time). In the work of Martins & Tsuzuki (2010), it is considered different heuristics. The values presented in each line of Table 3 correspond to their heuristic that had the best value of P_{conv} and, in case of tie, the one with smallest T_{min} value. So the results in this table does not correspond to a single heuristic of Martins & Tsuzuki (2010), since for each instance, we considered their best heuristic.

Table 3: Performance of the GRASP heuristic for the 2KP-(0/1) with instances of Martins & Tsuzuki (2010).

	SA Heuristic		GRASP		
Name	P_conv (%)	T_min (s)	P_conv (%)	Best Time (s)	Average Time (s)
BOTTOM LEFT FAILS PUZZLE	-	-	0	108.83	112.31
LARGER FIRST FAILS PUZZLE	93.5	212.6	100	4.67	9.46
SMALL PUZZLE	100	153.6	100	5.93	7.33
TANGRAM	77.4	2614.8	86.67	8.63	45.19

We can see from Table 3 that the GRASP heuristic found optimum solutions for most of the runs with reasonable small time. For these instances, the convergence rates to the

optimum solutions of the GRASP heuristic are always larger than the SA heuristic. The only instance for which no optimal solution was found is the Bottom Left Fails Puzzle, and in Martins & Tsuzuki (2010), it is reported values for such solution only for the translational case (when no rotations are needed).

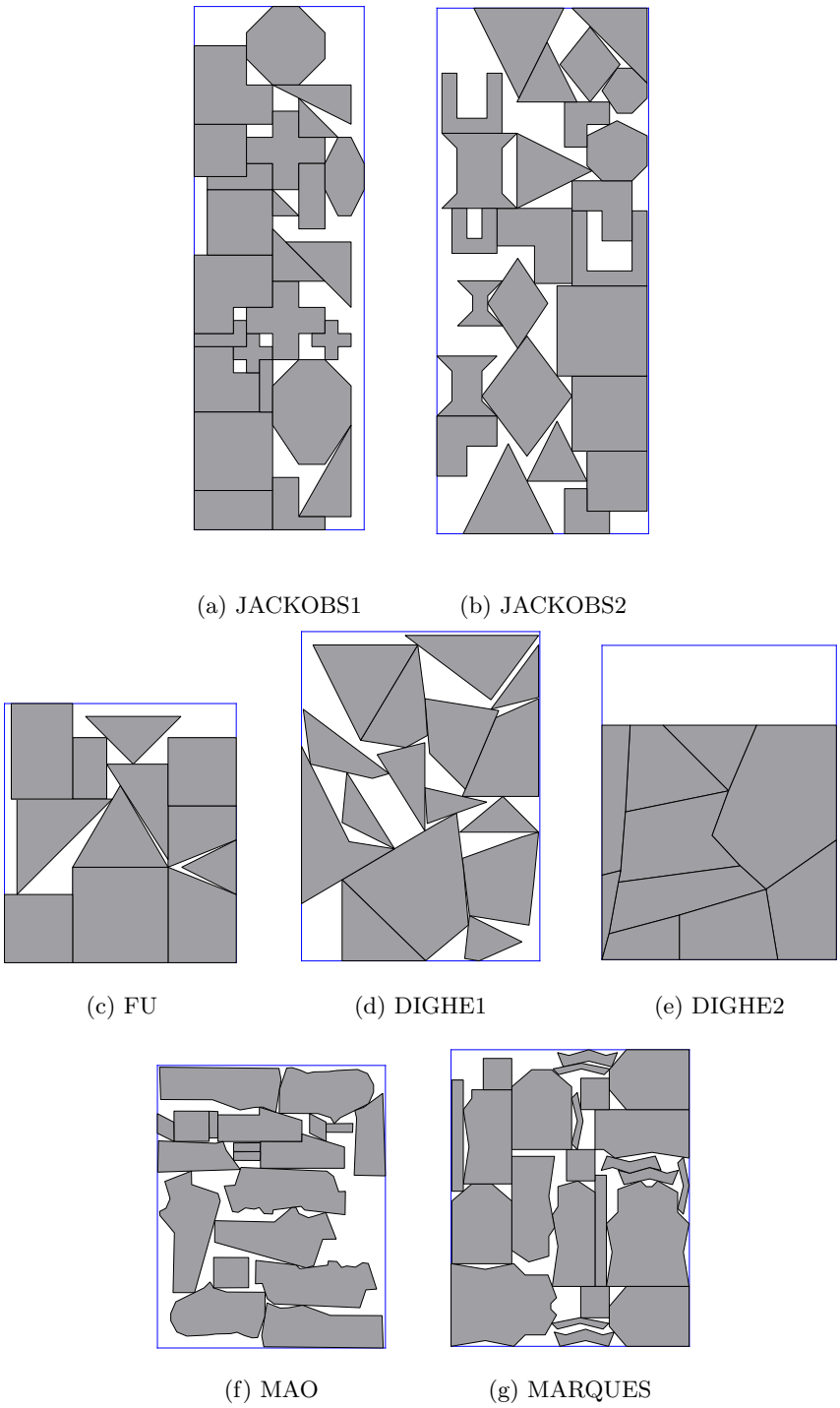
Table 4 presents the results of the GRASP heuristic for the 2KP-(0/1) considering the remaining instances. The following information is presented in this table: instance name (Name); the time spent (in seconds) to compute all the no-fit polygons (Time of NFP); the quantity of items packed in the solution (Quantity of Items); the area of the bin occupied by items (Occupied Area); the time spent (in seconds) to solve the respective instance (Time of GRASP); and, the total time spent: Time of NFP + Time of GRASP. It is worth noting that the GRASP heuristic was executed 15 times for each instance, and all values are the average of these 15 executions.

Table 4: Performance of the GRASP heuristic for the 2KP-(0/1).

Name	Time of NFP (s)	Quantity of Items	Occupied Area	Time of GRASP (s)	Total Time (s)
FU	0.26	12	0.8382	21.79	22.05
JACKOBS1	59.49	25	0.7538	8.30	67.79
JACKOBS2	53.80	25	0.6844	565.71	619.51
SHAPES0	51.32	41	0.6016	1552.46	1603.78
SHAPES1	202.56	41	0.6424	3891.60	4094.16
SHAPES2	17.90	26	0.7289	1048.12	1066.02
DIGHE1	0.12	16	0.7240	10.68	10.80
DIGHE2	0.15	10	0.7460	0.07	0.22
ALBANO	14.40	23	0.8038	961.59	975.99
DAGLI	26.16	29	0.7586	1106.40	1132.56
MAO	102.91	20	0.7160	120.42	223.33
MARQUES	57.50	24	0.8274	217.77	275.27
SHIRTS	208.49	96	0.7702	14317.13	14525.62
SWIM	1088.21	46	0.6427	39781.43	40869.64
TROUSERS	48.22	62	0.7866	5796.59	5844.81

Observing Table 4 the heuristic found optimal solutions for 7 instances (figure 1 shows such optimal solutions), since it packed all items. Considering all instances, the average occupied area is 73.50%. The worst result was obtained with instance SHAPES0, with occupied area of 60.16%. On average, the computation of the NFPs required 128.77 seconds, while 4626.67 seconds were spent during the execution of the heuristic.

The results in Table 4 shows that the heuristic computed solutions with good occupancy



(a) JACKOBS1 (b) JACKOBS2

(c) FU (d) DIGHE1 (e) DIGHE2

(f) MAO (g) MARQUES

Figure 1: Optimal packings found by the GRASP heuristic.

rates. It is worth noting that the considered instances have a large number of items with completely irregular shape and they are considered hard to solve.

Table 5 presents the results of the computational experiments for the 2KP. For each instance, the algorithm Solve2KP was executed 10 times.

In each line of Table 5 we have: instance name (Name); the best solution (Best Occupied Area); the average value solution (Avg. Occupied Area); the average time spent (in seconds)

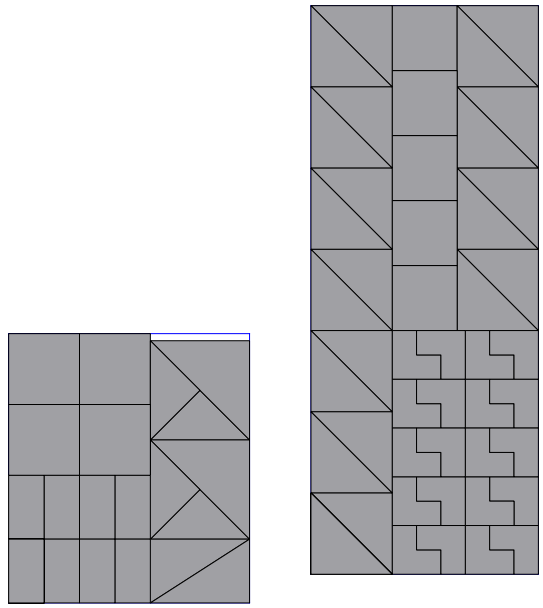
by algorithm Solve2KP to solve the respective instance (Avg. Time); and, the time spent (in seconds) to compute the no-fit polygons (Time of NFP).

Table 5: Performance of the algorithm Solve2KP for the 2KP.

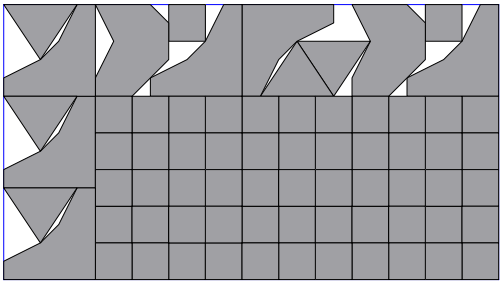
Name	Best Occupied Area	Avg. Occupied Area	Avg. Time (s)	Time of NFP (s)
FU	0.9892	0.9892	5.01	0.26
JACKOBS1	1.0000	0.9870	49.60	59.49
JACKOBS2	1.0000	0.9851	66.08	53.80
SHAPES0	0.6190	0.5873	134.43	51.32
SHAPES1	0.6949	0.6893	248.15	202.56
SHAPES2	0.9284	0.9183	49.37	10.48
DIGHE1	0.7631	0.6909	7.62	0.12
DIGHE2	0.7791	0.7657	3.14	0.15
ALBANO	0.9653	0.9653	37.93	14.40
DAGLI	0.9256	0.9196	50.99	26.16
MAO	0.9812	0.9644	71.53	102.91
MARQUES	0.9606	0.9515	43.76	57.50
SHIRTS	1.0000	1.0000	508.92	208.49
SWIM	0.7590	0.7272	2206.42	1088.21
TROUSERS	1.0000	0.9986	193.54	48.22

From Table 5, we can see that the instances with several rectangular items or few items of irregular shape allow the algorithm PackAllRET to generate solutions with high occupancy rates. The instances with occupancy rates greater than 90% were: FU, JACKOBS1, JACKOBS2, SHAPES2, ALBANO, DAGLI, MAO, MARQUES, SHIRTS and TROUSERS, that corresponds to 10 out of 15 instances. On the other hand, the instances SHAPES0, SHAPES1, DIGHE1, DIGHE2 and SWIM have several items with shape very irregular, and we can see that the average occupancy rates decrease accordingly. But it is worth noting that even in these harder instances, the algorithm obtained good occupancy rates. Figures 2 and 3 show some solutions found by the algorithm Solve2KP.

Finally, we present results for the 2CS. In Table 6 we present solutions computed by the algorithm Solve2CS. We used as a lower bound for the optimal solution, the total area of the items divided by the area of one bin. More formally the lower bound LB is equal to $\lceil \frac{\sum_{i=1}^{n} d_i * area(i)}{area(bin)} \rceil$. The rows in this table contain the following information: instance name (Name); solution value computed by algorithm Solve2CS (Solution); the lower bound (LB); the difference (in percentage) on number of bins computed by Solve2CS and LB; the time spent in seconds (Time); the number of columns generated (Columns).



(a) FU (b) JACKOBS2

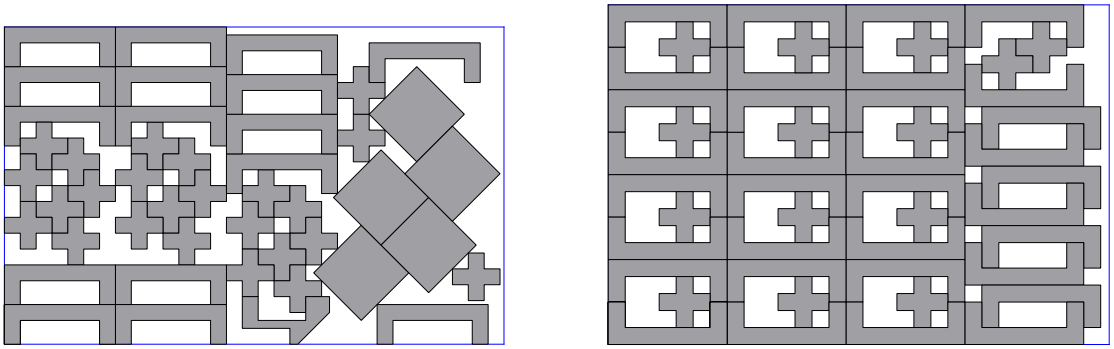


(c) SHAPES2

Figure 2: Solutions computed by Solve2KP for instances that contains rectangular items.

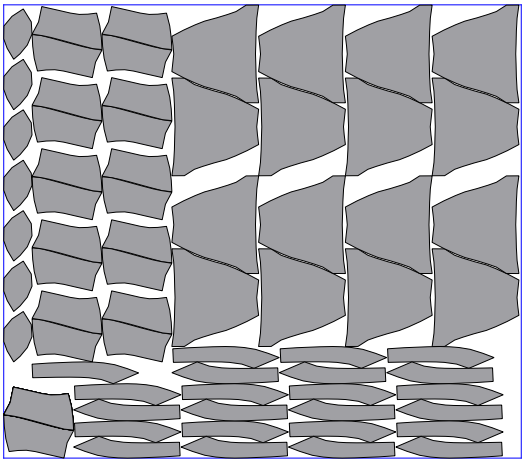
From Table 6 we can see that, on average, the value of the solutions returned by the algorithm is 46.899% larger than the lower bound, and that in the worst case, for instance DIGHE1, it is 90.625% larger. We believe that the results of the algorithm are much closer to the optimal solutions and these differences are due to the weakness of the lower bound. Some very hard instances took $1,195,677.01$ seconds ($\approx$ 14 days) of CPU processing (see instance SHIRTS). The CPU time spent, on average, is $138,462.01$ seconds. The number of columns generated is 1614.13, on average. Notice that working with items of irregular shape makes the problem harder than when dealing with rectangular items. That is why some instances took a large time to be solved.

It is worth to mention that all the instances were executed 15 times, except SHIRTS (1 time), SWIM (10 times) and TROUSERS (5 times) due to the high CPU time required by them. All the results presented on Table 6 are the average values of all executions. Such results show that the column generation heuristic returned good solutions for the Cutting



(a) SHAPES0

(b) SHAPES1



(c) SWIM

Figure 3: Solutions computed by Solve2KP for instances without any rectangular items.

Stock problem with items of irregular shape. However, it requires high CPU time when solving instances with several items of completely irregular shape.

6. Concluding Remarks

In this paper we presented heuristics to 2D cutting/packing problems where items have irregular shape. We presented heuristics for the 0/1 Knapsack, Unconstrained Knapsack and Cutting Stock problems.

To find feasible positions where to pack items, we implemented the robust algorithm to compute no-fit polygons proposed by Burke et al. (2007). We joined this algorithm with an extended version of the search algorithm proposed by Adamowicz & Albano (1976) to find good arrangement of the items in the packing.

For the 0/1 Knapsack problem, we presented a GRASP heuristic that shows to be efficient in comparison to other methods of the literature. This heuristic found optimal solutions for 7 out of the 15 instances within a reasonable CPU time.

Table 6: Results obtained for the 2CS.

Name	Solution	LB	Difference (%)	Time (s)	Columns
FU	74	56	24.324%	228.93	150
JACKOBS1	50	38	31.579%	6,726.73	516
JACKOBS2	47	30	56.667%	6,897.00	622
SHAPES0	55	30	83.333%	26,701.14	1711
SHAPES1	56	32	75.000%	59,601.60	1962
SHAPES2	63	50	26.000%	9,413.40	809
DIGHE1	61	32	90.625%	252.30	245
DIGHE2	46	29	58.621%	45.18	94
ALBANO	84	65	29.231%	6,750.17	660
DAGLI	58	43	34.884%	9,304.73	885
MAO	49	33	48.485%	7,073.23	489
MARQUES	53	44	20.455%	8,298.07	617
SHIRTS	45	37	21.622%	1,195,677.01	8586
SWIM	60	34	76.471%	466,819.10	2476
TROUSERS	53	42	26.190%	273,141.60	4390

For the Unconstrained Knapsack problem, the proposed approach obtained, for 66.67% of the instances, solutions that occupies at least 90% of the bin area spending little CPU time.

For the Cutting Stock problem the column generation heuristic showed to be applicable, not only for the case with rectangular items (as presented by Cintra et al. (2008)) but also for the case with items of irregular shape. On average, the solutions returned by the algorithm have value 46.899% larger than the lower bound, but we used a weak upper bound in these comparisons. As mentioned, the only drawback was the required CPU time that can be considered high for instances with items of very irregular shape.

Acknowledgements.

This research was partly supported by CAPES, CNPq (grant 483604/2011-9) and FAPESP.

References

Adamowicz, M., & Albano, A. (1976). Nesting two-dimensional shapes in rectangular modules. *Computer-Aided Design*, *8*, 27–33.

Albano, A., & Sapuppo, G. (1980). Optimal allocation of two-dimensional irregular shapes using heuristic search methods. *IEEE Trans. Syste., Man, Cybern., SMC-10*, *5*, 242–248.

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

Bennell, J. A., & Oliveira, J. F. (2008). The geometry of nesting problems: A tutorial. *Journal of the Operational Research Society*, *184*, 397–415.

Bennell, J. A., & Oliveira, J. F. (2009). A tutorial in irregular shape packing problems. *Journal of the Operational Research Society*, *60*, 93–105.

Burke, E. K., Hellier, R., Kendall, G., & Whitwell, G. (2007). Complete and robust no-fit polygon generation for the irregular stock cutting problem. *Journal of the Operational Research Society*, *179*, 27–49.

Chernov, N., Stoyan, Y. G., & Romanova, T. (2010). Mathematical model and efficient algorithms for object packing problem. *Comput. Geom.*, *43*, 535–553.

Cintra, G. F., Miyazawa, F. K., Wakabayashi, Y., & Xavier, E. C. (2008). Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation. *European Journal of Operational Research*, *191*, 59–83.

Coin-OR CLP (). Linear programming solver: An open source code for solving linear programming problems. `http://www.coin-or.org/Clp/index.html`.

Daniels, K. M., & Milenkovic, V. J. (1997). Multiple translational containment. part 1: An approximation algorithm. *Algorithmica*, *19 (Special Issue on Computational Geometry in Manufacturing)*, 148–182.

Egeblad, J., Nielsen, B. K., & Odgaard, A. (2007). Fast neighborhood search for two- and three-dimensional nesting problems. *European Journal of Operational Research*, *183*, 1249–1266.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. Freeman, San Francisco.

Gilmore, P., & Gomory, R. (1961). A linear programming approach to the cutting stock problem. *Operations Research*, *9*, 849–859.

Gilmore, P., & Gomory, R. (1963). A linear programming approach to the cutting stock problem - part II. *Operations Research*, *11*, 863–888.

Gomes, A. M., & Oliveira, J. F. (2002). A 2-exchange heuristic for nesting problems. *European Journal of Operational Research*, *141*, 359–370.

Gomes, A. M., & Oliveira, J. F. (2006). Solving irregular strip packing problems by hybridising simulated annealing and linear programming. *Journal of the Operational Research Society*, *171*, 811–829.

Herz, J. C. (1972). A recursive computational procedure for two-dimensional stock-cutting. *IBM Journal of Research Development*, (pp. 462–469).

Lee, W. C., Ma, H., & Cheng, B. H. (2008). A heuristic for nesting problems of irregular shapes. *Computer Aided Design*, *40*, 625–633.

Martins, T. C. (2000). *Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods*. Ph.D. thesis University of Wales, Cardiff, UK.

Martins, T. C., & Tsuzuki, M. S. G. (2010). Simulated annealing applied to the irregular rotational placement of shapes over containers with fixed dimensions. *Expert Systems with Applications*, *37*, 1955–1972.

Oliveira, J. F., Gomes, A. M., & Ferreira, J. S. (2000). Topos – a new constructive algorithm for nesting problems. *OR Spectrum*, *22*, 263–284.

Queiroz, T. A., Miyazawa, F., Wakabayashi, Y., & Xavier, E. (2012). Algorithms for 3d guillotine cutting problems: Unbounded knapsack, cutting stock and strip packing. *Computers & Operations Research*, *39*, 200–212.

Scheithauer, G. (1997). Equivalence and dominance for problems of optimal packing of rectangles. *Ricerca Operativa*, *27*, 3–34.

Wascher, G., Hausner, H., & Schumann, H. (2007). An improved typology of cutting and packing problems. *Journal of the Operational Research Society*, *183*, 1109–1130.