Irregular Packing Problems: Heuristic Algorithms and Evaluation of NFP generators

Tiago Silveira[a], Eduardo Candido Xavier[b]

[a]*Federal University of Alfenas, Minas Gerais, MG, Brazil (e-mail: tiago@bcc.unifal-mg.edu.br).*
[b]*State University of Campinas, São Paulo, SP Brazil (e-mail: ecx@ic.unicamp.br).*

Abstract

This paper presents a new heuristic algorithm based on meta-heuristic Genetic Algorithm for the 2D Irregular Knapsack problem. In simple terms, the proposed heuristic takes a list of items to be packed and builds a solution, taking into account the order of items on the list and some geometric characteristics of the packaged item with the partial solution. Furthermore, we implement an algorithm to the 2D Irregular Strip Packing problem, based on proposed heuristics packing, to assist the construction of solutions, such that the partial solution can be compacted during the packing. Additionally, we analysed the performance of three algorithms to generate the geometric structures called "*No-Fit Polygons*" (NFPs), which can be used for the verification of overlap between items in the solving of packing problems with irregular items. We performed computational experiments to compare the proposed heuristics with other heuristics from the literature. The proposed algorithm obtained better results in many cases, indicating that the Genetic algorithm is a suitable choice when dealing with packing problems with irregular items.

Keywords: 2D Irregular Packing Problems, Genetic Algorithms, No-Fit Polygon, Combinatorial Optimization, Computational Geometry, Knapsack, Strip Packing.

1. Introduction

Cutting & Packing problems are classical problems in the Operations Research field that have a great applicability in many practical problems.

In an informal way, cutting problems relate to the division of larger units (objects) into smaller units (items), and packing problems relate to the grouping of smaller units (items) into larger units (objects or containers). In general, the purpose of these problems is to maximize the use of space in which the items are contained, avoiding waste of materials (raw materials). Among the problems that can be modeled as cutting and packing problems are optimization of layouts in pieces of wood, textiles, paper and sheets of metal, of plastic and glass etc.

As described by Dyckhoff (1990), both problems have the same logical structure, which allow them to similar formulations, providing the same resolution strategies. Therefore, we focus on packing problems.

1.1. Packing problems

Based on the typology of Wäscher et al. (2007), the 2D packing problems considered in this work can be defined as follows:

`Knapsack problem`. Given a set I of items, where each item $i \in I$ presents a value $v(i)$ associated, and the set R formed by a single rectangular element r (with constant height and width, and greater than zero), we seek to find a subset of I that can be packaged into r such that the sum of the values $v(i)$ of the items in r is maximized, being satisfied the restrictions of the packing problems. For the problem considered in this paper, the $v(i)$ function is the area of the item i.

`Strip Packing problem`. Given a set I of items and a single rectangular element $r \in R$ of height h, a constant greater than or equal to zero, and width w, a non-negative variable, each item $i \in I$ must be packaged in r in order to minimize the value of w, being satisfied the restrictions of the packing problems. In this case, given a solution to the problem, the value of w is the difference of the largest and lowest x coordinates among the vertices of all items of that solution.

By itself, the one-dimensional version of many packing problems belong to the class of NP-hard problems (Garey and Johnson (1979)). The treatment of these becomes even more complex when we use polygons with holes or irregular items.

In this work, our interest is solving the Knapsack problem with irregular items, considering a limited set of rotations $\theta = \{\theta_1, \theta_2, \ldots, \theta_k\}, \forall\ k \in \mathbb{N}$ for each instance I. Each item $i \in I$ is represented by a simple polygon, composed of a list of coordinates of points $((x_1, y_1), (x_2, y_2), \ \ldots\ , (x_n, y_n))$ in the plane (representing the vertices of the polygon), in which each of its consecutive pairs (including the pair $((x_n, y_n), (x_1, y_1))$) represents a directed edge in a counterclockwise direction. For the edges of boundary of the polygon, the left side of these edges represents the inside of the item. For the edges that represent the boundary of the holes of a item (if any), the left side of these edges represents the outside of the item. For the case of Strip Packing problem, we use this same structure. However, the interest in this problem is only to assist the construction of the Knapsack solutions.

1.2. Literature review

Some of the earliest works to address two-dimensional packing problems with irregular items were proposed around the 1970s when Adamowicz and Albano (1976) described a technique for grouping of 2D irregular items in rectangular containers, using an algorithm based on the geometric structure No-Fit Polygon

to check the overlap between the items. From this, the literature begins to exhibit works that focus on geometric aspects of these problems, as well as the optimisation techniques to develop new approaches for obtaining solutions. An interesting description of these aspects for solving of positioning problems of irregular items can be found in Bennell and Oliveira (2008, 2009); Dowsland and Dowsland (1995).

In the case of packing problems with irregular items addressed in this paper, the literature presents several strategies for generating solutions. Martins and Tsuzuki (2010) proposed a new packing approach for 2D irregular items considering a number of rotations and a single container with fixed dimensions, in order to minimise the loss of use of the container after the packing. Similarly, Del Valle et al. (2012) propose a heuristic based on meta-heuristic GRASP for 0-1 Irregular Knapsack problem, and a heuristic for the unrestricted version of this. Based on the resolution of that problem solved by Del Valle et al. (2012), Mundim and de Queiroz (2012) presented a hybrid heuristic which combines GRASP with the heuristic Simulated Annealing to search for better solutions.

For Strip Packing problem, we highlight the works that are based on the use of meta-heuristics in order to guide the heuristics of packing items in the building of solutions. Gomes and Oliveira (2006) developed a system based on linear programming with a local search guided by meta-heuristic Simulated Annealing. The works Burke et al. (2006, 2007, 2010) focused on two important parts of the problem: the development of techniques of verification of overlap between irregular items, and the development of good packing heuristics, which are assisted by meta-heuristics Tabu Search and Hill Climbing during the construction of solutions. Some of the better results for this problem are presented in Leung et al. (2012), where a nonlinear optimization model with a local search guided by the meta-heuristic Tabu Search was proposed. Featuring quality results close to the results of Leung et al. (2012), Sato et al. (2012) proposed a new constructive heuristic, which is directed by the Simulated Annealing. Finally, Bennell and Song (2010) proposed a new constructive approach using the heuristic developed by Oliveira et al. (2000).

In relation to other packing problems, Terashima-Marín et al. (2010) presented an approach based on meta-heuristic Genetic Algorithm for creating hyper-heuristics for the Irregular Bin Packing problem, considering instances with regular (rectangular shaped) and irregular (convex shaped) items. There are other works that treat only the variation with regular items of the Bin Packing problem, like Berkey and Wang (1987), which presented new heuristics to the problem with the adaptation of different heuristics packing; and Martello and Vigo (1998), in which was proposed an exact heuristic branch-and-bound.

Despite the singularity of each heuristic, it is important to highlight that all cited works use efficient structures for verification of overlap between the items of a packing during the construction of their solutions, being the NFPs used in most of them.

1.3. Contributions

There are two main contributions of this work. The first, the implementation and comparison of three algorithms of the literature for construction of No-Fit Polygons. The second, the development of a new packing heuristic for the Irregular Knapsack problem, plus an adaptation of this to the use an algorithm to Irregular Strip Packing problem.

For the generation of No-Fit Polygons, two different geometric approaches are used: the first based on basic geometric operations; and the second based on the theory of the Minkowski sum. We implemented three algorithms to generate the No-Fit Polygons. The first is based on the orbital method, while others use different approaches of the Minkowski sum.

With respect to the packing problems, we propose a new heuristic based on meta-heuristic Genetic Algorithm, generating the solution through a constructive approach: given a list of items to pack, the algorithm takes into account the initial order of these items and positions them in the container, considering some of their geometric characteristics with the partial solution. In the context of local search of the algorithm, operators are used to change the order of items in the list, in order to explore the solution space. In addition to this heuristic, we extend their algorithm with one that solves the Irregular Strip Packing problem. In this case, the extended algorithm uses the algorithm of the Strip Packing problem as a compactor of items of a container when the initial heuristic finds the full container.

This work is divided as follows: the section 2 presents the investigated techniques for generating No-Fit Polygon, and an experimental evaluation of them; the section 3 describes a heuristic for the packing problem considered, as well as its adaptation with the heuristic for solving the Strip Packing problem; the section 4 presents the results of the proposed heuristics; and, finally, section 5 presents the conclusions about the work and the future works.

2. No-Fit Polygon generators

The No-Fit Polygon (NFP) is a geometric structure that defines the positions in which two simple polygons can be positioned without any overlap between them. It is widely used in packing problems with irregular items because is simple to implement and efficient for the verification of overlap, and also enables the development of heuristics that can take advantage of these structures in a packing (Bennell and Oliveira (2008)).

Generated the NFP of the item A to the item B, represented by NFP_{AB}, the verification of overlap between the two items is reduced to a problem of determining whether the reference point of B is contained within NFP_{AB} (an operation of complexity $O(n)$, where n is the number of edges of the NFP_{AB}, which is the complexity of point location problem for a simple polygon). Determined the location of the point, it is trivial to decide whether there is overlap between the items: if the point is contained within the respective NFP, there is overlap. Otherwise, the position generates no overlap between them, so that A

and B are disjoint (reference point of B outside of NFP_{AB}) or are touching by the boundary (reference point of B on one of the edges of the NFP_{AB}).

Despite this simplicity, the choice of an algorithm for generating the NFPs is not a simple task. Since a NFP generation algorithm can use different geometric operations according to the geometric characteristics of item (convexity, holes, etc.), its complexity may also be different. Therefore, we introduce three algorithms that build NFPs based on two distinct geometric approaches. In this case, the aim is to perform an empirical analysis of the implemented NFP generation techniques.

2.1. Geometric approach: orbital method

Proposed by Mahadevan (1985), the orbital model uses the sliding of the polygons and elementary trigonometry as follows: given two simple polygons A and B, a point (usually a vertex) of each of them is selected, being called reference points; the polygon A is translated within a coordinate space so that your reference point remain at the origin, and this polygon remains fixed throughout the process; and the polygon B must be moved around all edges of A, maintaining contact between edges of both polygons (without any overlap), and, while this is done, trace each edge of the NFP, based on the path taken through the reference point of B.

The implemented algorithm for this approach of creation of NFPs is proposed by Burke et al. (2007). The features of this algorithm is its ability to generate the NFP for any simple polygon, with holes or not. Another important point refers to the ability to find edges of the NFP called exact slides or simple exact points, which are regions where, once properly positioned, the base polygons of the NFP fit precisely, maintaining contact for at least two edges.

2.2. Geometric approach: Minkowski sum

The Minkowski Sum (MS), based on the theory of sets and integral geometry, is one of the elementary operations of the image processing area, forming part of the so-called mathematical morphology. More specifically, corresponds to the dilation operation.

Let A and B be two sets of points in space $\mathbb{R}^d$. The MS of A with B is defined as (1):

$$A \oplus B = \{a + b \mid a \in A, b \in B\}, \tag{1}$$

where a and b are vectors corresponding to each point of A and B, respectively, and $a + b$ is the vector sum of a and b.

Since the interest is to use the theory of MS in the process of verifying overlap between simple polygons, the interpretation for the generation of NFP from MS is: the MS of the points of A with the symmetric set $(-B)$ which is the set in each vector of these points (vector starting at the origin point) can move B so that it overlaps the set A. Note that, regardless of the position of A and B in the plane, the sets generated by operations $A \oplus (-B)$ will be congruent[1] (in this case, the geometric concept of congruence applies to the sets generated

using only the translation), and A and B_p will overlap when the result of the SM contains, necessarily, the point of origin of the plane. This comes from the fact that if A and B_p overlap, then there exists $x \in A \cap B_p$, and to compute the SM of A and $-B_p$, will add x and $-x$.

Transforming the sets A and B in polygons, this operation should be treated differently, because such objects are formed by infinite points. Thus, Cunninghame-Green (1989); Ghosh (1991, 1993) showed a way to calculate the MS by taking into consideration only the edges of the polygons. Similarly, is necessary the use of the symmetrical set $(-B)$ to obtain the corresponding NFP, which in this case corresponds to the inversion of the direction of each of its edges (changing between the start and end points of the edge), but without changing its initial topological orientation (connection order between the edges). Representatively, for the polygon

$$B = (\overrightarrow{b_1} \rightarrow \overrightarrow{b_2} \rightarrow \cdots \rightarrow \overrightarrow{b_n}),$$

we have the representation of its symmetrical as:

$$-B = (\overleftarrow{b_1} \rightarrow \overleftarrow{b_2} \rightarrow \cdots \rightarrow \overleftarrow{b_n}),$$

where "$\rightarrow$" represents the sequence of the topological order of the edges, and "$\overrightarrow{b_x}$" and "$\overleftarrow{b_x}$" represent the orientation (normal or reverse) of the edge x of B, considering the direction of the edges of B.

Based on the theory of MS, we implemented two techniques to generate NFPs from polygons:

`Decomposition`. Works with the vertices set of the simple polygons. To generate NFP with this technique, the idea is to decompose simple nonconvex polygons into simple convex polygons, compute the MS among the pairs of polygon of the decomposition and unite them: let A and B be two simple polygons, we obtain the convex polygons $A_1, \ldots, A_k$ and $B_1, \ldots, B_\ell$ such that $\bigcup_{i=1}^{k} A_i = A$ and $\bigcup_{j=1}^{\ell} B_j = B$. With these sets, we calculate the MS $S_{ij} = A_i \oplus -B_j$ for each pair, obtaining m convex polygons. Finally, the NFP_{AB} is obtained by the union of these m polygons, i.e., $\text{NFP}_{AB} = A \oplus -B = \bigcup_{ij} S_{ij}$. Thus, the time complexity of this method is dependent of the complexity of the algorithm used for convex decomposition of simple polygons.

Based on this technique, we implemented an algorithm based on method of MS of the geometric library CGAL (2012), making use of the convex decomposition proposed by Chazelle and Dobkin (1985), of time complexity $O(n^2)$, where n is the number of vertices of the polygon. The characteristic of the implemented algorithm is the generation of NFPs only to instances formed by

[1] Two sets of geometric points are said to be congruent if, and only if one can be transformed into the other by a combination of translations and/or rotations and/or reflections.

simple polygons without holes, which is a restriction imposed by the CGAL library in their algorithm for calculating the MS. Another constraint refers to the inability to find exact slides and exact points of the NFP, which are eliminated by the library during the geometric operations of polygons union.

`Boundary Addition`. Despite being formalized by Ghosh (1991), Cunninghame-Green (1989) have already presented the using of the ideas of the boundary addition for the generation of the NFP of convex polygons. The Boundary Addition (BA) works with the edges of the polygons. For convex polygons, the technique boils down to obtain the set of edges of the polygons A and $-B$, and following the angular order of each, concatenate them to obtain the NFP. For simple polygons, Ghosh (1991, 1993) show how to compute the MS using its edges, but requiring a more elaborate set of steps.

In this case, we implemented the algorithm proposed by Bennell and Song (2008) to calculate the NFP, which works as follows: given the polygons A and B, take the symmetric $(-B)$ and divide its sequence of edges in convex portions. Following the order of division of these portions, concatenate them with the edges of the polygon A, according to the angular order, while crosses each edge of A. Then, detect the edges that forming the NFP boundary, based on the resulting segments of the concatenation and in the intersections between them. The algorithm presented by Bennell and Song (2008) has time complexity $O(m^2n^2 \log m^2n^2)$, where m and n are the number of edges of A and B, respectively. We emphasize that this algorithm is able to generate the exact slides and exact points of the NFP, but considers only simple polygons without hole.

2.3. Results of the NPF generators

We present the results obtained by each NFP generator algorithm, considering several distinct instances composed by simple polygons without hole, obtained from the ESICUP base (http://paginas.fe.up.pt/∼esicup/ (2013)). Note that when there are repeated instances (i.e., instances with different names, but with the same polygons, as blaz1 and shapes2, and shapes0 and shapes1), only one of these was used in the performing experiments.

All of the experiments conducted throughout this paper have been performed on a PC with a Xeon 2.4GHz CPU, 8GB RAM, Linux Operating System. The algorithms have been implemented in C++, with the aid of geometric library CGAL (2012) for geometric manipulation.

The conducted experiment was done only once. For this, we consider only distinct items of each instance, and each of them has been rotated by the angles set 0 °, 90 °, 180 ° and 270 °, resulting in four distinct polygons. For each instance, we compute the NFP between all pairs of distinct items (including an item with itself), using the three algorithms of the previous section. We denote by Gen1 the proposal of Burke et al. (2006), based on orbital method; by Gen2 the implementation using the geometric library CGAL, based on the MS via decomposition; and by Gen3 the proposal described by Bennell and Song (2008), based on MS via boundary addition.

Table 1: NFPs generator times – original instances.

Name	Tot	Dstc	V. avg	NFPs	Gen1(sec)	Gen2(sec)	Gen3(sec)
albano	24	8	7.25	1024	6.92	3.40	**2.05**
blaz2	20	4	7.50	256	1.44	0.45	**0.32**
dagli	30	10	6.20	1600	8.45	2.42	**1.79**
dighe1	16	16	3.88	4096	6.86	1.50	**0.59**
dighe2	10	10	4.70	1600	3.52	0.90	**0.36**
fu	12	11	3.55	1936	2.97	0.45	**0.24**
han	23	20	7.35	6400	66.87	**17.72**	21.70
jakobs1	25	22	5.45	7744	30.89	**5.44**	11.82
jakobs2	25	22	5.41	7744	36.48	10.31	**10.30**
mao	20	9	9.22	1296	19.74	7.35	**5.39**
marques	24	8	7.13	1024	6.55	3.15	**2.16**
poly1a	15	15	4.60	3600	8.04	3.16	**1.00**
poly2b	30	30	4.93	14400	37.76	13.26	**4.78**
poly3b	45	45	4.93	32400	81.63	29.53	**10.28**
poly4b	60	60	4.93	57600	143.06	49.86	**17.40**
poly5b	75	75	4.84	90000	216.37	76.18	**26.01**
shapes1	43	4	8.75	256	4.45	**1.00**	4.17
shapes2	28	7	6.29	784	3.17	1.16	**0.56**
shirts	99	8	6.63	1024	5.44	1.51	**1.24**
swim	48	10	21.90	1600	218.97	77.40	**51.18**
trousers	64	17	5.06	4624	12.60	2.96	**1.27**
Total time					**922.20**	**309.10**	**174.61**

Table 1 shows, for each algorithm, the time spent (in seconds) in the generation of NFPs, for the respective instance. Each row of this table shows the following information: the instance name (Name); the total number of polygons (Tot); the number of distinct polygons of the instance (Dstc); the average number of vertices of distinct items (V. avg); total number of NFPs of distinct items, given the four rotations used (NFPs); the spent time to generate all NFPs of distinct items for the orbital method (Gen1); the spent time to generate all NFPs of distinct items for the MS via decomposition (Gen2); the spent time to generate all NFPs of distinct items for the MS via boundary addition (Gen3).

From Table 1, we can see a great difference in the execution times of the techniques of NFP generation. In most cases, Gen3 performs better, becoming, on average, over 5 times faster than Gen1. However, we emphasize that Gen3 does not work with polygons with holes, which does not rule out the possibility of using Gen1 in problems with polygons that type. For Gen2, there are three cases in which the generation time is less, when compared to the other two generators, but not enough to justify its use, view its characteristic of neither working with polygons with hole nor generate the exact slides and exact points.

Although the results obtained by Gen3 generally have the lowest runtime, there is a point that we should highlight. From the results obtained, it is clear that the cost of NFP generation grows with increasing number of vertices of a polygon. However, the convexity of a polygon is also crucial that spent time, as may be seen by comparing dighe2 and dagli instances, in which dagli has a greater amount of nonconvex polygons.

To analyze the impact of non-convexity in the creation of NFPs by algorithms, we apply a change to the items in these same instances: for each item i

of an instance, add m reflexes vertices, so that i carry one simple polygon, but with m additional nonconvex vertices. The addition of a reflex vertex r is made in the median position of the edge formed by the pair of consecutive vertices v and $v+1$ of the item i, to a perpendicular distance p of this edge (the value of p is 15% of the lower edge of i, and is updated at each complete cycle of vertices added to i); if the vertex r result in a non-simple polygon, another reflex vertex was calculated, considering the next edge. After the addition of vertices, the NFPs between all pairs of the new items were generated in a similar way to the previous experiment. We consider the value of m being 1, 2, 5, 10 and 15. The results for this change are presented in Tables 2 (for Gen1), 3 (for Gen2) and 4 (for Gen3).

Comparing the results of Table 1 with Tables 2, 3 and 4, we note that the non-convexity of the items contributes to the execution time of the generators. For a better analysis, Figure 1 shows the average execution time of these ones, for each case. We can observe that with the addition of up to 5 reflexes vertices, the behaviour of generators is similar to running with the original instances. From this point we have a reversal, so that Gen2 executes faster. For the case in which are added 15 reflexes vertices, its final execution time is significantly lower if compared to the two other generators. Nevertheless, we must take into account the inability of Gen2 to generate some important characteristics of the NFP and, therefore, it can greatly reduce the time spent generating when items become more complex. Finally, considering Gen1 and Gen3, we have observed

Table 2: NFPs generator times – modified instances for Gen1.

Name	Amount of reflex vertices added				
	Plus 1	Plus 2	Plus 5	Plus 10	Plus 15
albano	9.58	13.98	30.41	94.81	240.20
blaz2	1.89	2.71	6.67	18.62	52.22
dagli	11.34	15.00	35.41	110.27	264.50
dighe1	10.64	15.21	37.28	116.51	298.24
dighe2	5.03	6.87	16.16	49.20	123.17
fu	4.59	7.04	20.59	93.99	274.87
han	96.75	130.67	306.16	999.87	2098.88
jakobs1	42.41	58.46	174.75	674.71	1696.38
jakobs2	59.62	88.97	251.54	892.80	2163.74
mao	25.75	31.17	62.18	167.65	416.46
marques	7.99	10.90	23.24	75.30	178.97
poly1a	11.75	16.80	44.69	131.90	337.36
poly2b	53.96	77.90	192.90	593.77	1419.97
poly3b	118.09	171.55	412.45	1260.29	3033.13
poly4b	206.04	294.53	697.14	2147.59	5102.98
poly5b	312.34	443.35	1058.85	3222.42	7706.24
shapes1	6.13	9.51	23.04	69.49	166.88
shapes2	4.30	6.16	14.79	50.28	137.53
shirts	7.76	10.70	23.81	65.08	173.24
swim	257.45	305.16	497.17	921.18	1766.81
trousers	16.97	23.15	53.51	192.71	539.87
Total	**1270.40**	**1739.78**	**3982.75**	**11948.42**	**28191.63**

Table 3: NFPs generator times – modified instances for Gen2.

Name	Amount of reflex vertices added				
	Plus 1	Plus 2	Plus 5	Plus 10	Plus 15
albano	5.04	8.23	18.18	42.61	83.79
blaz2	1.17	1.44	4.19	7.33	14.91
dagli	6.09	7.88	20.02	49.62	95.55
dighe1	7.14	9.15	39.02	122.82	244.27
dighe2	3.79	4.79	14.57	43.56	82.21
fu	2.84	3.60	16.06	47.01	93.27
han	34.96	45.77	103.16	238.18	385.32
jakobs1	18.79	28.58	83.09	219.47	387.54
jakobs2	25.45	33.96	86.81	226.25	448.09
mao	10.89	12.71	25.87	61.13	102.50
marques	5.20	5.77	14.64	35.09	62.02
poly1a	9.56	13.50	39.49	91.81	193.18
poly2b	37.08	53.95	153.27	359.64	765.29
poly3b	77.80	113.84	335.46	817.52	1666.06
poly4b	143.43	201.75	594.43	1408.87	2896.70
poly5b	217.86	308.90	911.27	2199.91	4547.52
shapes1	1.50	2.20	4.78	9.35	17.11
shapes2	2.83	3.46	10.30	21.44	43.60
shirts	3.93	5.78	13.33	31.64	56.30
swim	87.76	95.88	134.30	211.78	275.85
trousers	10.94	13.67	41.90	112.18	220.26
Total	**714.04**	**974.80**	**2664.14**	**6357.19**	**12681.32**

Table 4: NFPs generator times – modified instances for Gen3.

Name	Amount of reflex vertices added				
	Plus 1	Plus 2	Plus 5	Plus 10	Plus 15
albano	3.02	4.6	13.58	48.2	178.33
blaz2	0.59	1.01	3.99	12.56	28.83
dagli	3.64	5.84	21.43	81.16	197.84
dighe1	1.49	3.66	19.84	92.29	388.62
dighe2	0.92	1.8	8.75	28.37	92.14
fu	1.22	3.68	12.78	42.13	177.9
han	40.2	65.99	195.1	532.16	1067.92
jakobs1	23.83	44.3	121.14	303.72	830.92
jakobs2	21.75	39.88	131.32	399.69	1151.9
mao	7.8	11.32	29.23	100.48	297.51
marques	4.15	5.76	12.58	33.29	84.8
poly1a	2.41	5.34	23.26	79.02	272.08
poly2b	11	24.25	96.7	342.21	1059.01
poly3b	22.71	49.18	208.7	774.19	2330.11
poly4b	39.81	84.96	356.53	1271.78	3816.9
poly5b	60.15	128.58	538.71	1927.34	6070.92
shapes1	4.82	6.76	15.28	35.24	101.53
shapes2	1.25	2.34	9.43	29.41	82.58
shirts	2.38	4.46	14.66	33.62	87.06
swim	71.38	97.87	234.77	627.17	1385.04
trousers	2.7	8.26	34.16	72.81	242.98
Total	**327.22**	**599.84**	**2101.95**	**6866.82**	**19944.93**

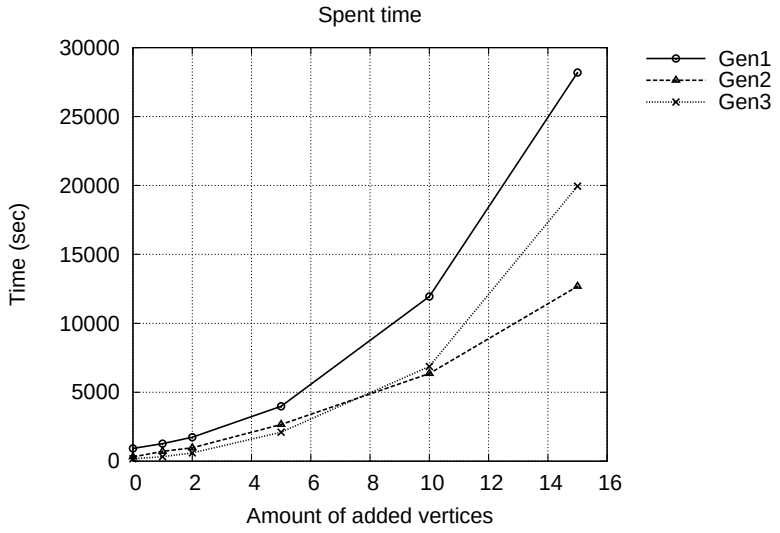

Figure 1: Average execution time of the NFP generators.

that Gen3 has the greatest variation of time after the addition of the new reflexes vertices, despite its shorter execution time.

3. Packing heuristics

In this section we present the proposed heuristics for the packing problems considered in this paper, which are based on a constructive approach: build solutions by adding one item at a time and maintaining the viability of the solution, i.e., without overlap between items and all items within the container.

3.1. Elementary heuristics

The proposed heuristics have in common some elementary heuristics to generate solutions. Among them, we have the heuristic of grouping of items, called "Grouping Heuristic", and the heuristic of ordering of the packing items vector, based on meta-heuristic "Genetic Algorithm", and they are shown below.

3.1.1. Grouping heuristic

Aiming to create a grouping of items that is compact, using geometric properties of items during construction of a solution is shown as a good alternative, being considered for jobs with good practical results (Oliveira et al. (2000); Bennell and Song (2010); Sato et al. (2012)). Following this strand, the Grouping Heuristic (GH) makes use of extended search proposed by Adamowicz and Albano (1976).

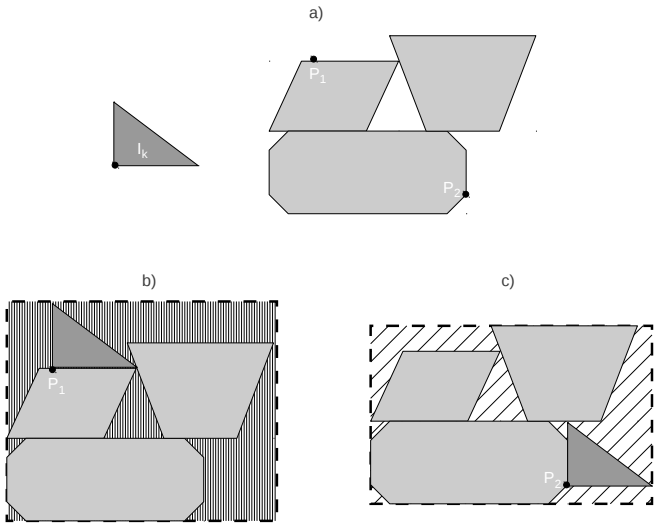

Figure 2: Representation of extended search considering only two packing points (P_1 and P_2) of the item I_k in the partial solution for the analysis of rectangular hull.

In the extended search, given two simple polygons P and Q, and NFP_{PQ}, the idea is to find a point on the edge of NFP_{PQ} such that when Q is packed at this point, the rectangular hull covering P and Q has minimum area. Figure 2 presents the steps taken by the extended search to analyze and choose a grouping of items, considering the respective rectangular hull of the partial grouping. Given the partial packing and one item in 2 a), the extended search finds the position in which these items are grouped together and generate the rectangular hull of minimum area (2 b) and 2 c), respectively). Taking the two generated grouping and evaluating the area of the respective rectangular hull of each one, we note that 2 c) generates a smaller rectangular area, representing the selected group. However, there are cases where the grouping produced by the extended search is not the rectangular hull of the smaller area.

GH considers that the items to be packed are in a list in some initial order. The goal is to remove an item from the list and package it in the container, considering a partial packing D of previously packed items. At each iteration, GH selects a new item Q and tests, using the extended search, the packing of Q in D, in each of their rotations. The grouping of the extended with rectangular hull of the smaller area is then used and passed to the evaluation of a new item. Note that the order of the list of items to be packed influences in the solution of the GH. This is the key of the Genetic Algorithm that we implemented, since a solution is always characterized by the order of the items in the input list.

By itself, the use of GH only based on greedy choices of the extended search can lead to the generation of a final grouping not compact. Figure 3 depicts a case where this happens, in which the instance shown in 3 a) would not be grouped in order to obtain the best grouping, whatever the order considered by

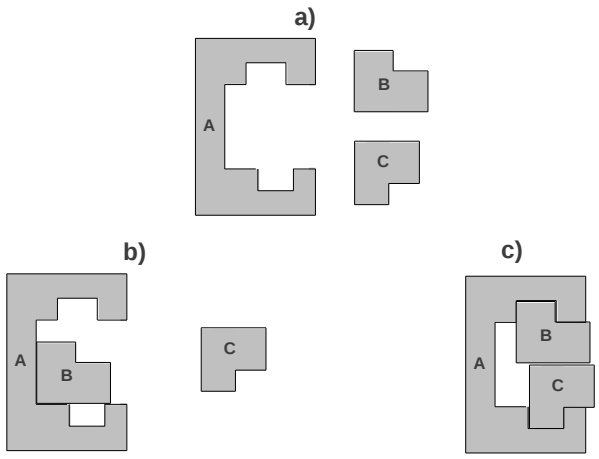

Figure 3: Instance in which the optimal solution can not be generated by GH. a) instance formed by the items A, B and C, and rotations are not allowed; b) initial packing likely made by the Grouping Heuristic; c) optimum packing (better utilization).

GH.

Therefore, we apply two variations in the extended search. Instead of using the positions on the edges of the NFP that the extended search analyzes to find the grouping with rectangular hull of minimum area, we created a new set of test positions of two different ways: the first, called Discretized Search, choose the position of the next item based on a set of test points distributed at equidistant spaces on the edge of their NFP; the second, called Random Search, choose the position of the next item based on a random set of test points on the edge of their NFP. As these forms of grouping items are extensions of the extended search, the evaluation of the position of the item to be packed to consider each point of the test set can be done using both the area of the rectangular hull (similar to the evaluation of the extended search) as the area of the convex hull obtained with the partial solution, providing new possibilities for the packing of an item. Figure 4 shows an example of a set of test points, for each implemented variation.

During the packing of an item Q in a partial solution D, choosing groups which generate shorter container width may be more interesting to more compact grouping, but with greater width, since, for example, the best alternative for Strip Packing problem is to build a solution that uses the smallest width of the container. Therefore, to decide at what point package Q in D, we use a selection criterion based on a loss parameter α as follows: let the area utilization of any grouping be the ratio of the area of the items in D plus Q over the area of the hull (rectangular or convex) of the grouping of D with Q. Based on this utilization, consider the positions analyzed by the extended search (or any of its variations); let A_R be the utilization when packing Q at the point that minimizes the area of the hull with D; and let A_C be the utilization when

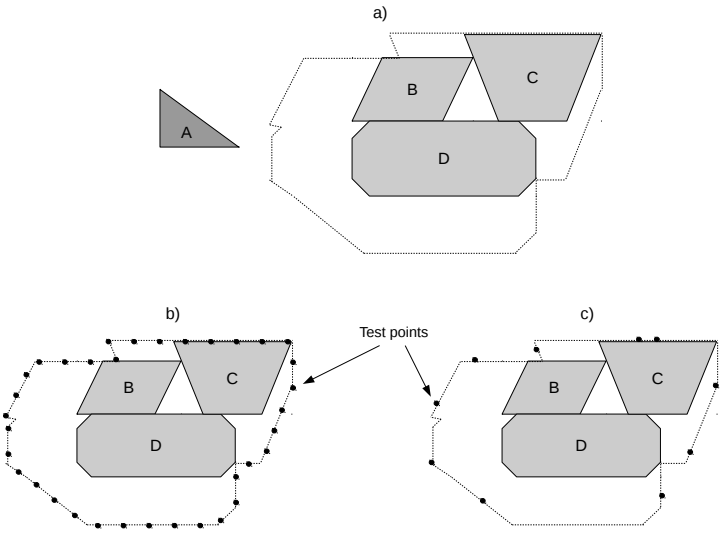

Figure 4: Variation in the extended search, where: a) A is the next item to be grouped together to the partial solution (items B, C and D), and NFP between them; b) set of test points for Discretized Search; c) set of test points for Random Search.

packing Q at the point that minimizes the width of the resulting packing with D, considering the same type of hull. Thus, if

$$A_R - A_C < \alpha, \tag{2}$$

will be given preference to the second packing, even if their utilization is less. An example is shown in Figure 5, in which the values of utilization of the groups of Figure 5 a) and 5 b), showed in 5 c), will be used by Equation 2.

Finally, the Algorithm 1 summarizes the steps taken by the GH to perform the grouping of the items. A point to note this heuristic is with respect to the Packing method, which uses the structure of NFPs to verify feasible points of grouping, considering the partial solution. In this method, the NFP used in such searches is not recalculated by any of the algorithms described in section 2, every time that a new item is packed, but was used the union of NFPs of individual items of the partial solution, for reasons of efficiency. Therefore, the generation of all edges of the resulting NFP (boundary edges, exact sliding and exact points, viewed in section 2) will also be dependent on the method of joining the individual polygons.

3.1.2. Genetic Algorithm

The Genetic Algorithm (GA), proposed by Holland (1992), is a meta-heuristic of stochastic search based on evolutionary computing that mimics the process of natural evolution and is widely used as an optimization method for difficult problems.

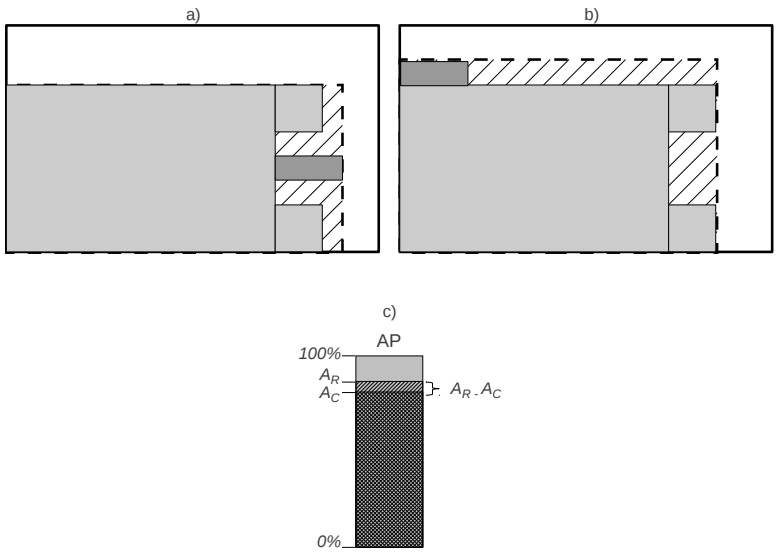

Figure 5: Test of grouping according to the utilization and width of the solution; a) packing with better utilization AP; b) packing with shorter width in the container; c) representation of the loss factor of the previous utilizations.

For packing problems considered, the idea to use the GA is apply it in order to get different orders of the list of items to be packed. The modeling of the structures of GA is described below. packaged

Individuals

An individual is represented by a list of items to be packed in a certain order. Given this list, only one solution is obtained by using the GH (seção 3.1.1).

The creation of the individuals of the initial population is made in a deterministic way in which the choice of the sequence of items is done by keeping items of larger area in the initial positions of the list.

The fitness of an individual depends upon the problem to be solved, which may be the utilization of the container (Knapsack) or the width of the packing (Strip Packing) found by GH.

Reproduction

The generation of new individuals is entirely based on mutations. The mutation consists of the application of a single operation on the item list (chromosome) to change it, among which the following operators can be used on the items (alleles):

Algorithm 1: GROUPHEURISTIC(i, sol, $search\_t$, $hull\_t$, α, R)

Input : i: item to be packed;
sol: partial solution of the packing problem;
$search\_t$: type of search for item packing;
$hull\_t$: type of hull for calculating utilization ;
α: loss parameter for choice the partial solution;
R: container for the partial solution.

Output: The best solution according to the loss parameter α.

```
Begin
    testpoints ← GeneratePoints(sol, i, search_t);
    For all p ∈ testpoints do
        aux_solution ← Packing(p, i, sol, R);
        If search_t = EXTENDED_SEARCH then
            EvaluateSolution(aux_solution, RetangularHull);
        else
            EvaluateSolution(aux_solution, hull_t);
        solutions_set ← solutions_set ∪ aux_solution;
    If is using parameter α then
        return ChoiceSolution(solutions_set, α);
    else
        return BestAPSolution(solutions_set);
```

- *swap*: exchange alleles between two random positions on a chromosome;
- *insertion*: removes and reinserts an allele randomly chosen positions;
- *invertion*: reverses the order of alleles within a random interval of the chromosome.

It is important to note that despite the crossover to be considered very important to the GA, we do not apply this operation to the individuals because we do not know a way to group parties considered a good solution, since the GH works on the partial solution of the packing.

Selection

Initially, the selection of the next generation is made keeping the best individual in the population. Then, the remaining individuals are chosen by the method of the tournament: two individuals are randomly chosen, and passes to the next generation the best individual between them. This process is repeated until the number of individuals of the next generation is not complete.

Other structures and operations

In addition to the basic structures of the GA, some data structures were used to improve the process of building solutions. Among them, we have:

Solution cache. Set of operations on a chromosome, in order to avoid recalculate all packing after the mutation of the respective individual. Works as follows: given a chromosome, find the first position of the list of items with the allele that is in a different position when compared with the list of items before the mutation. Considering this position as the index i, the packing with the $i-1$ initial items will be kept (since $i > 0$), since the grouping of these in the solution is the same.

Calculation of solution of the sequence of mutations. Verification that avoid recalculate the solution of a chromosome, if the change of a mutation results in the same packing. In this case, will not be applied the GH, since there is already the fitness value for this individual.

Restart the population. Reset of the GA with a new population. The heuristic checks whether the solution stabilized in local minimum, and if there are no improvements of the fitness of the best individual in a given period of time, the entire population is reseted, shuffling the order of the items of the list for each individual, so that new solutions can be produced. It is noteworthy that the best individuals generated so far is preserved and reinserted into the newly formed population.

Pseudo-code

Algorithm 2 presents the steps of the Genetic Algorithm heuristic-based for the packing problems.

3.2. Proposed heuristics

Based on the basic heuristics of the previous section, we propose the packing heuristics for Knapsack problem.

3.2.1. Basic heuristic

The first full heuristic called basic heuristic is based on the heuristics described in section 3.1. Therefore, the ordering of the vector of items to be packed is made by GA, while the packing of the items in the container is made by GH. By owning only a single container of fixed width during the packing, the packing heuristic for the Knapsack problem will discard the next item of the list when it can not be packed. For Strip Packing, all items are always packed.

Algorithm 2: GABasedHeuristic($configs$)

input : $configs$: configuration parameters of the GA heuristic-based.
output: The individual with the best fitness of the population, i.e., the individual with the best packing generated by GH.

```
Begin
    ConfiguraEstruturas(configs);
    population ← generate_population();
    CalculateFitness(population);
    While ¬StopCondition(population, CurrentTime()) do
        Mutation(population);
        CalculateFitness(population);
        Selection(population);
        If stabilized_solution() then
            Restart(population);
            last_improve = CurrentTime();
    best_individual ← GetBest(population);
    Return best_individual;
```

3.2.2. Adapted heuristic

During a packing, items sometimes are discarded due to the bad nesting of the items within the container. However, if the items of this partial solution were repacked, but in a different way, such a solution could assume a more compact layout and more items could be added to the partial solution. On this basis, we propose an adaptation of the basic heuristic for the Knapsack problem incorporating in this the basic heuristic to the Strip Packing problem as follows: taking a specific container, the items contained therein are removed and then repacked with the use of basic heuristic for the Strip Packing problem. Thus, the basic heuristic for the Strip Packing problem will act as a compactor of the partial solution of the Knapsack problem in the container.

The complete algorithm is as follows: given the list of items I to be packaged and the algorithm for the Strip Packing problem, the basic heuristic for the Knapsack problem is performed up to fill the container $r(j)$, so that the item i does not fit him. In this moment, the items of $r(j)$ are repacked by the algorithm of the Strip Packing problem. Completed this last optimization, a new packing attempt is made with the item i in $r(j)$: if possible, i is positioned in its place; otherwise, the item i is discarded. The adapted heuristic to the Knapsack problem is described by Algorithm 3.

A point to note on the integration of the heuristics for packing problems is that the time taken for compactation performed by the algorithm of the Strip Packing problem can adversely affect the final solution if such compactations do

Algorithm 3: AdaptedHeuristic(I, *compactor*)

Input : I: items set to be packed;
compactor: algorithm for the irregular Strip Packing problem.
Output: Container R with the packing solution.

```
Begin
    initialize(R) ;
    compactedbin ← false ;
    While I ≠ ∅ do
        packed ← TryPackingAtBin(R, i) ;
        If packed then
            compactedbin ← false ;
        else if ¬compactedbin then
            Compact(R, compactor) ;
            packed ← TryPackingAtBin(R, i);
            If packed then
                compactedbin ← false ;
            else
                compactedbin ← true ;
        I ← I \ i ;
    Return R
```

not contribute to improve the grouping in the container. Thus, in such cases, the spent time in trying to compress a container causes a smaller amount of new individuals are generated by GA, resulting in a smaller exploration of the space of solutions. Thus, the use of the compactation in the adapted heuristic was conditioned to the behavior of the best individual of the GA population, as follows: for every individual, is generated the packing by the basic heuristic and stores the fitness of the solution. For the best individual of the population, storing the values of fitness when used basic and adapted heuristics. But for the new individuals, the adapted heuristic is executed only when any of these generate, using the basic heuristic, a fitness that is better than the fitness of the best individual of the population, considering its value calculated by the basic heuristic too.

4. Results of the packing heuristics

This section presents the experimental settings and the results of packing heuristics proposed in this paper.

Table 5: ESICUP instances parameter settings.

Instance	Height(h)	Width(w)	Allowed rotations(°)	Time(sec)
albano	4900	10122.63	0; 180	1200
blaz2	15	25.20	0; 180	1200
dagli	60	65.60	0; 180	1200
dighe1	100	138.13	0	600
dighe2	100	134.05	0	600
fu	38	34.00	0; 90; 180; 270	600
han	58	43.57	0; 90; 180; 270	1200
jakobs1	40	13.00	0; 90; 180; 270	600
jakobs2	70	28.20	0; 90; 180; 270	600
mao	2550	2058.60	0; 90; 180; 270	1200
marques	104	83.60	0; 90; 180; 270	1200
poly1a	40	13.90	0; 90; 180; 270	1200
poly2b	40	29.99	0; 90; 180; 270	1200
poly3b	40	40.72	0; 90; 180; 270	1200
poly4b	40	51.70	0; 90; 180; 270	1200
poly5b	40	57.71	0; 90; 180; 270	1200
shapes0	40	63.00	0	1200
shapes1	40	59.00	0; 180	1200
shapes2[a]	15	27.30	0; 180	1200
shirts	40	63.13	0; 180	1200
swim	5752	6568.00	0; 180	1200
trousers	79	245.75	0; 180	1200

[a]Instance also known in the literature as blaz1.

4.1. Settings

The execution environment and the characteristics of the implementation for testing the proposed heuristics is the same as described in section 2.3.

We used the same instances of irregular items of the section 2.3, with some additional settings specified in Table 5. It is noteworthy that the rotations allowed for each instance were obtained based on the most recent works for the Strip Packing problem.

Were performed 10 runs for each instance, for the respective algorithm. Importantly, for some instances, the execution time was slightly higher than that described in Table 5, because we stop the algorithm execution only at the end of the construction of a solution, if this is still being generated.

The proposed heuristics were adjusted with the following common parameters: verification of overlap of the items based on the technique of NFPs, with the generation of these structures by the method "Boundary Addition" for simple polygons – described in section 2 –; loss parameter α equivalent to 5%, with such a test based on the utilization of the grouping of the items considering the convex hull; and grouping heuristic, when using the Discretized or Random Search, using the convex hull of the items to evaluate the utilization of the

grouping. On these types of searches, we have the following settings: for the Discretized search, was used a range for the points generation with size referent to 20% of the perimeter of the respective NFP. This value is halved when not find, at least, one packing position that is within of the container; for the Random search, the points set was formed by 20 random points on the outer boundary of the NFP or 20 random points on each of its edges, which is a choice of equal probability. Note that, for any search, if there is not a viable position for a item in the packing, this one is excluded of the solution.

The GA parameters were adjusted using a brute-force search, with a fixed parameters set. Each of this configuration was run for a time of 1200 seconds. The most promising parameters of such parametric configuration are described in Table 6.

Table 6: GA parameter settings.

Parameter	Configuration
Population	4
Offspring per individual	1
Ext. Search	90%
Disc. Search	5%
Rand. Search	5%
Swap	33%
Insertion	34%
Invertion	33%
Restart the population	300 seconds
Selection	Tournament

Finally, for the adapted heuristic – described in section 3.2.2– , was set a time of 3 seconds of optimization for the algorithm of the Strip Packing problem.

4.2. Results and discussion

The results of the heuristics for the Knapsack problem are described in Table 7.

In order to verify the performance of both heuristics (basic and adapted), we also present the best results found in the literature. The information presented in this table, for each row, are: instance name (Problem instance); maximum occupancy of the instance with the packing of all items (Max. oc.); utilization of the best solution (Best Util), average of utilization (Average Util), average number of packed items (Items) and average time (Time) of the basic heuristic; and these same informations for the adapted and literature heuristics with the best results found (except the utilization of the best solution of the literature, because these results were not found). To facilitate the analysis, we highlight the values of the better utilization (bold) and the higher average utilization (underlined), for each instance.

As regards the results of the literature for instances of the ESICUP base, we found the work of Del Valle et al. (2012), which considers most instances resolved. For the instances not present in the work them, was used a width of

Table 7: Knapsack results.

Problem instance	Max. oc.(%)	Basic Heuristic				Adapted Heuristic				Literature		
		Best	Average			Best	Average			Average		
		Util(%)	Util(%)	Items	Time(sec)	Util(%)	Util(%)	Items	Time(sec)	Util(%)	Items	Time(sec)
albano	86.06	**84.81**	83.84	21.8	1200.5	84.27	83.48	21.4	1203.2	80.38	23	975.99
blaz2	74.74	**71.83**	71.17	18.8	1200.5	71.83	70.95	18.7	1201.0	*	*	*
dagli	77.31	**77.31**	77.31	30.0	26.2	**77.31**	77.31	30.0	109.4	75.86	29	1132.56
dighe1	72.40	**72.40**	72.40	16.0	31.6	**72.40**	72.40	16.0	18.0	72.40	16	10.80
dighe2	74.60	**74.60**	74.60	10.0	1.7	**74.60**	74.60	10.0	7.8	74.60	10	0.22
fu	83.82	**83.82**	83.82	12.0	13.2	**83.82**	83.82	12.0	69.7	83.82	12	22.05
han	77.95	**77.95**	76.71	22.0	1162.7	77.16	76.37	21.7	1200.6	*	*	*
jakobs1	75.38	**75.38**	75.38	25.0	13.6	**75.38**	75.38	25.0	10.4	75.38	25	67.79
jakobs2	68.44	**68.44**	68.44	25.0	38.8	**68.44**	68.44	25.0	32.9	68.44	25	619.51
mao	71.60	**71.60**	71.60	20.0	10.0	**71.60**	71.60	20.0	31.1	71.60	20	223.33
marques	82.74	**82.74**	82.74	24.0	58.2	**82.74**	82.74	24.0	63.2	82.74	24	275.27
poly1a	73.73	71.75	70.15	13.8	1200.0	**73.74**	70.33	13.9	1110.0	*	*	*
poly2b	75.40	72.65	71.49	27.0	1201.7	**73.88**	72.01	27.8	1201.6	*	*	*
poly3b	74.90	**73.12**	72.55	42.3	1202.7	72.69	72.22	42.0	1202.8	*	*	*
poly4b	74.80	**74.34**	73.02	57.1	1206.6	73.21	73.01	56.8	1207.2	*	*	*
poly5b	79.53	**74.51**	74.09	65.9	1214.3	74.40	73.77	65.5	1214.8	*	*	*
shapes0	63.33	**61.75**	60.33	40.5	1200.4	61.11	59.71	40.0	1201.6	60.16	41	1603.78
shapes1	67.63	**65.59**	63.90	39.5	1202.0	**65.59**	63.59	39.6	1203.1	64.24	41	4094.16
shapes2	79.12	76.43	75.87	26.7	1200.2	**77.66**	76.12	26.8	1201.0	72.89	26	1066.02
shirts	85.54	**84.71**	83.60	92.1	1207.5	83.89	83.23	89.1	1220.1	77.02	96	14525.62
swim	67.35	**67.35**	66.90	47.3	793.1	**67.35**	67.08	47.5	820.9	64.27	46	40869.64
trousers	88.63	87.45	86.89	59.1	1202.0	**88.30**	87.14	59.3	1206.4	78.66	62	5844.81

* Not available.

container that represents the best solution of the literature for the Strip Packing problem, considering only those with similar parametric settings to the described in the section 4.1. For the instances poly1a, poly2b, poly3b, poly4b and poly5b was used a width of the container based on the work of Burke et al. (2006); for the instances blaz2 and han, their values refer to the obtained from Silveira (2013). It is noteworthy that we do not use the values obtained in the work of Mundim and de Queiroz (2012) because, for some instances, the width value is different of the container width used by Del Valle et al. (2012). Regarding the execution environment of these experiments, Del Valle et al. (2012) have been performed on a PC with a Core2 Quad 2.4GHz CPU and 4GB RAM.

Comparing the results obtained by the proposed heuristics, it is seen that the basic version obtained better results in various aspects. First, it had a higher average number of results with higher utilization. Another important point was the time spent, which was lower in most cases, especially for the instances in which we obtained the optimal solution. This shows that the time spent in compacting a container by the adapted heuristic is not interesting in some instances, given a fixed time of the optimization process.

Despite these characteristics, there are cases where to apply the compactation is interesting. For example, in some experiments in which the instances solved are more complex and have a greater number of distinct items (such as a swim, trousers etc.), the compactation performed in the solution that being constructed by adapted heuristic tends to produce better solutions.

Comparing the results obtained by each of the proposed heuristics with the results of the literature, we emphasize that we obtained only two inferior results when compared the average utilization found by heuristic adapted, and only an inferior result when compared the average utilization found by basic heuristic. However, note that in these cases, the average time used by both heuristics was lower due to standards adopted. Analyzing the occupation of the container, this was higher for some instances, even considering the time limit, that was significantly lower. For the execution time of the instances in which were found optimal solutions, some results of the work of Del Valle et al. (2012) show a little time spent in comparison with other heuristics, but it is not very significant.

Regarding the results in Table 7, we believe that the initial ordering of items (not ascending order of area) was important to obtaining the results, since the average number of items is smaller in some problems, when compared with the results obtained by Del Valle et al. (2012). Another important point is relative to the generation time of NFPs, which showed an insignificant value when compared to the total optimization time.

The best packing for each instance of the Knapsack problem is shown in Figure 6.

5. Conclusions and future works

This paper presented a new heuristic for packing problems with irregular items, using a constructive approach to generate a feasible solution. The basic

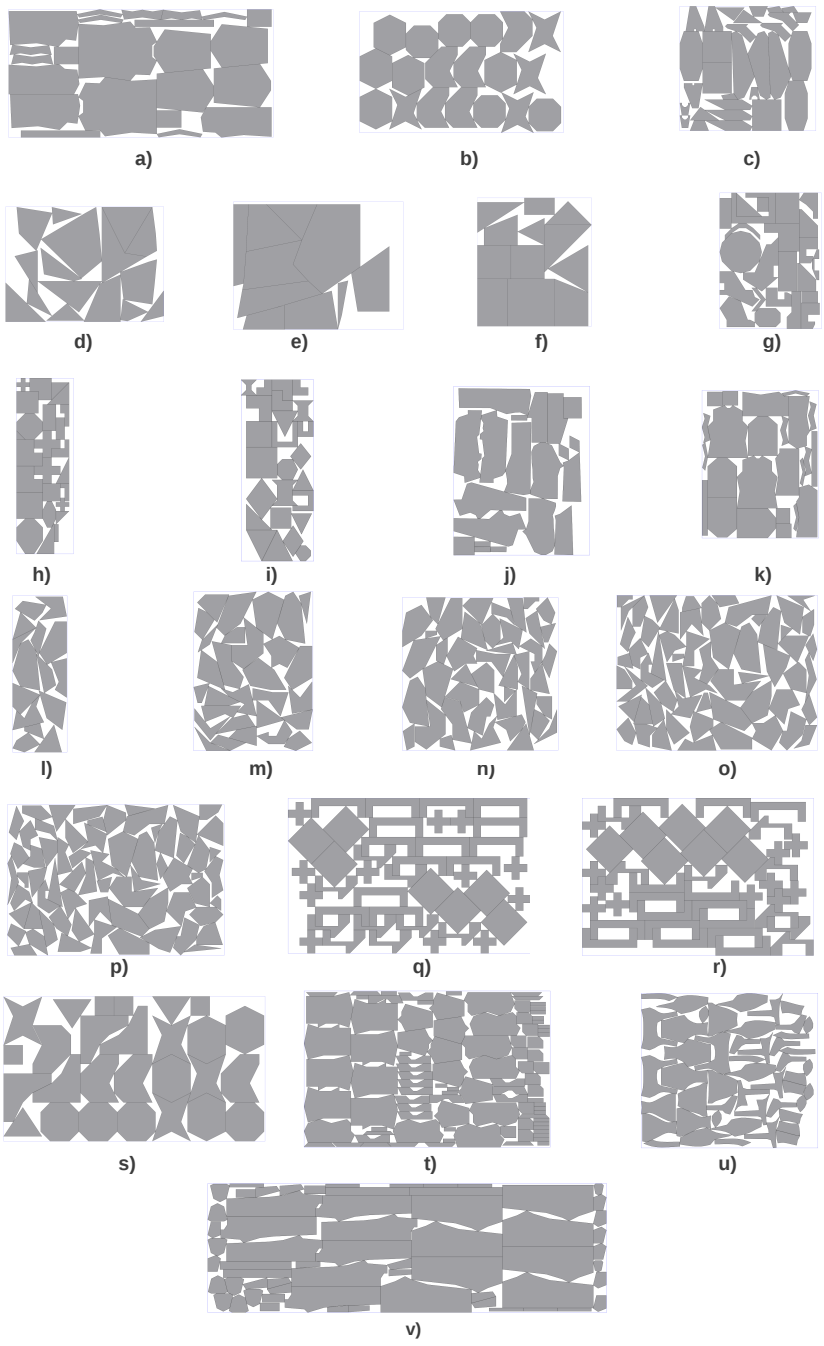

Figure 6: Results for Irregular Knapsack problem for the instances: a) albano; b) blaz2; c) dagli; d)dighe1; e) dighe2; f) fu; g) han; h) jakobs1; i) jakobs2; j) mao; k) marques; l) poly1a; m) poly2b; n) poly3b; o) poly4b; p) poly5b; q) shapes0; r) shapes1; s) shapes2; t) shirts; u) swim; v) trousers.

idea was to use the meta-heuristic genetic algorithm to guide the process of constructing new solutions to solve the Knapsack problem. We carried out a comprehensive computational experiment to analyse the algorithm proposed. The results showed that this meta-heuristic is able to guide efficiently the heuristics in the search space, however the use of suitable geometric structures is fundamental in the optimisation process to generate good results when considering irregular items.

It was also proposed a heuristic that makes the integration between the algorithms of packing problems. We implemented an algorithm for the Strip Packing problem based on the packing heuristic proposed to act as a compactor algorithm in order to generate more optimised solutions to the Knapsack problem. In this case, the proposed heuristic is indicated for cases in which the instances to be solved are complex. Nevertheless, the flexibility provided by the new heuristic is interesting in the sense that any heuristic developed to the Strip Packing problem can be used as a new algorithm to compact solutions.

Regarding to NFP generators, the respective analysis will help other researchers when choosing methods to verify overlap among irregular items for packing problems.

As future work, we will implement new algorithms to verify overlap, such as "Phi-Functions" (Chernov et al. (2010)) and other Minkowski sum approaches, in order to check their performance when using irregular items. Additionally, we will extend the algorithms of this work to deal with simple polygons with holes, generalising the proposal to cover all types of simple polygon. When it comes to heuristics, we propose to compare packing approaches, with and without using NFPs, with the goal of analyzing the impact of the meta-heuristics with different packing heuristics.

References

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

J. Bennell, X. Song, A beam search implementation for the irregular shape packing problem, Journal of Heuristics 16 (2010) 167–188.

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

J.A. Bennell, J.F. Oliveira, A tutorial in irregular shape packing problems, Journal of the Operational Research Society 60 (2009) S93–S105(13).

J.A. Bennell, X. Song, A comprehensive and robust procedure for obtaining the nofit polygon using minkowski sums, Comput. Oper. Res. 35 (2008) 267–281.

J. Berkey, P. Wang, Two-dimensional finite bin-packing algorithms, Journal of the operational research society (1987) 423–429.

E. Burke, R. Hellier, G. Kendall, G. Whitwell, A New Bottom-Left-Fill Heuristic Algorithm for the Two-Dimensional Irregular Packing Problem, OPERATIONS RESEARCH 54 (2006) 587–601.

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

E.K. Burke, R.S.R. Hellier, G. Kendall, G. Whitwell, Irregular packing using the line and arc no-fit polygon, Oper. Res. 58 (2010) 948–970.

CGAL, Cgal, Computational Geometry Algorithms Library, version 4.0, 4.0-2012. Http://www.cgal.org.

B. Chazelle, D.P. Dobkin, Optimal convex decompositions, Computational Geometry (1985) 63–133.

N. Chernov, Y. Stoyan, T. Romanova, Mathematical model and efficient algorithms for object packing problem, Computational Geometry 43 (2010) 535–553.

R. Cunninghame-Green, Geometry, shoemaking and the milk tray problem, New Scientist 12 (1989) 50–50.

A.M. Del Valle, T.A. de Queiroz, F.K. Miyazawa, E.C. Xavier, Heuristics for two-dimensional knapsack and cutting stock problems with items of irregular shape, Expert Systems with Applications (2012).

K.A. Dowsland, W.B. Dowsland, Solution approaches to irregular nesting problems, European Journal of Operational Research 84 (1995) 506 – 521.

H. Dyckhoff, A typology of cutting and packing problems, European J. Operational Research 44 (1990) 145–159.

M.R. Garey, D.S. Johnson, Computers and intractability: a guide to the theory of np-hardness, 1979.

P.K. Ghosh, An algebra of polygons through the notion of negative shapes, CVGIP: Image Underst. 54 (1991) 119–144.

P.K. Ghosh, A unified computational framework for minkowski operations, Computers and Graphics 17 (1993) 357 – 378.

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

J.H. Holland, Adaptation in natural and artificial systems, MIT Press, Cambridge, MA, USA, 1992.

http://paginas.fe.up.pt/∼esicup/, Euro special interest group on cutting and packing, 2013.

S.C. Leung, Y. Lin, D. Zhang, Extended local search algorithm based on nonlinear programming for two-dimensional irregular strip packing problem, Computers & Operations Research 39 (2012) 678 – 686.

A. Mahadevan, Optimization in computer-aided pattern packing., Dissertation Abstracts International Part B: Science and Engineering[DISS. ABST. INT. PT. B- SCI. & ENG.], 46 (1985).

S. Martello, D. Vigo, Exact solution of the two-dimensional finite bin packing problem, Management science 44 (1998) 388–399.

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

L.R. Mundim, T.A. de Queiroz, A hybrid heuristic for the 0–1 knapsack problem with items of irregular shape, in: Informatica (CLEI), 2012 XXXVIII Conferencia Latinoamericana En, IEEE, pp. 1–6.

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

A.K. Sato, T. de Castro Martins, M. de Sales Guerra Tsuzuki, An algorithm for the strip packing problem using collision free region and exact fitting placement., Computer-Aided Design 44 (2012) 766–777.

T. Silveira, Problemas de empacotamento com itens irregulares: Heurísticas e avaliação de construtores de nfp, 2013.

H. Terashima-Marín, P. Ross, C. Farías-Zárate, E. López-Camacho, M. Valenzuela-Rendón, Generalized hyper-heuristics for solving 2d regular and irregular packing problems, Annals of Operations Research 179 (2010) 369–392.

G. Wäscher, H. Haußner, H. Schumann, An improved typology of cutting and packing problems, European Journal of Operational Research 183 (2007) 1109 – 1130.