O P E R A T I O N S   R E S E A R C H   A N D   D E C I S I O N S

No. 1 2019
DOI: 10.5277/ord190103

Igor KIERKOSZ[1]
Maciej ŁUCZAK[1]

A ONE-PASS HEURISTIC FOR NESTING PROBLEMS

A two-dimensional cutting (packing) problem with items of irregular shape and rectangular sheets is studied. Three types of problems are considered: single-sheet problems without restrictions on the number of elements, single-sheet problems with restrictions on the number of elements, and cutting stock problems (restricted number of items and unrestricted number of sheets). The aim of the optimization is to maximize the total area of the elements cut from a single plate or to minimize the number of sheets used in cutting. A one-pass algorithm is proposed which uses the popular concept of a no-fit polygon (NFP). The decision on whether an item is cut from a sheet in a given step depends on the value of a fitting function. The fitting function depends on the change in the NFP of individual items. We test eight different criteria for the evaluation of partial solutions. On the basis of numerical experiments, the algorithm that generates the best solution for each of the considered problem types is selected. The calculation results for these algorithms are compared with results obtained by other authors.

Keywords: *cutting, packing, irregular shapes, nesting problem, one-pass algorithm*

1. Introduction

We shall be considering the optimization problem of two-dimensional cutting (packing) for irregularly shaped items from a rectangular sheet. We assume that items can be represented as a set of oriented vertices (polygons). The aim of the optimization is to minimize the waste in the cutting process, equivalent to maximizing material utilization. This problem is known in the literature as the irregular cutting (packing) problem or nesting problem [9].

[1]Faculty of Civil Engineering, Environmental and Geodetic Sciences, Koszalin University of Technology, Śniadeckich 2, 75-453 Koszalin, Poland, e-mail addresses: igor.kierkosz@tu.koszalin.pl, mluczak@wilsig.tu.koszalin.pl

According to the typology proposed by Wäscher et al. [47], we consider here three types of two-dimensional cutting and packing problems: the single knapsack problem (SKP), single large object placement problem (SLOPP) and single stock size cutting stock problem (SSSCSP), where the large object is a rectangular sheet with fixed dimensions and the small items are polygons.

Irregular cutting problems and equivalent problems occur in many practical applications, for example, in the clothing, footwear, furniture and metal industries. The cutting material may be fabric, leather, paper, metal, polystyrene, etc. In different applications, depending on the specific task, we have to deal with different constraints and different criteria for evaluating solutions. It should be noted that in practical applications we often have the more general problem of determining how many sheets should be used to obtain all of the given pieces.

In the literature, the problem of cutting rectangular items from a rectangular sheet is more often described. Although cutting and packing problems are NP-complete, exact algorithms are very important here (e.g., [6, 14, 16, 20, 23, 27]). They can be particularly useful in the case of problems of small and medium sizes, as well as local optimization mechanisms. For more complex instances, various approximate algorithms are used. Many of the approximate methods are algorithms based on metaheuristics [3, 26, 28, 30, 32].

The irregular cutting problem is much less frequently considered. This is mainly due to the geometric complexity of cutting patterns. For this reason, there exist rather few exact algorithms. Exact methods are described, among others, in [15, 22, 31, 46]. The authors of these papers investigate the use of mixed integer programming (MIP) models for solving the nesting problem. However, as the authors of the second paper state, MIP techniques, though still not appropriate for solving the more complex nesting instances, can be useful to improve heuristic methods. Jones [29] uses non-linear programming models (quadratic programming) and a circle representation of pieces to find an optimal solution for instances with two and three nonconvex polygons. For instances with four elements, he achieved the best known solution, but without proof of its optimality. Alvarez-Valdes et al. [4] proposed new integer linear formulations and a branch and bound algorithm that is able to solve instances of up to 16 pieces to optimality.

Because of the complexity of the nesting problem, both literature and practice are dominated by approximate methods. Various heuristic approaches are applied, ranging from simple heuristics [17, 24, 37, 39] to metaheuristic and hybrid algorithms [1, 7, 8, 12, 19, 25, 35, 36, 48].

It should be noted that single stock size cutting stock problems or single bin size bin packing problems (in the typology proposed by Wäscher et al. [47]) with items of irregular shape are very rarely discussed in the literature. We may refer, for example, to the following works: [18, 43] with weakly heterogeneous pieces or [33, 34, 45] with strongly heterogeneous pieces.

Most of the currently used methods are based on the concept of a no-fit polygon, introduced by Art [5] and developed, among others, by Adamowicz and Albano [2].

This is a method of determining layouts of elements so that they do not overlap each other. Although the NFP technique seems the most promising, mention should also be made of other approaches: pixel/raster methods [38, 41], trigonometric methods [21, 40], Φ functions [11, 44] and circle representations [29, 42]. A detailed survey covering the core geometric methodologies is given in [9].

2. Description of the problem

Let $Z = \{P_1, P_2, ..., P_n\}$ be a set of two-dimensional elements that we have to produce during a cutting process. We assume that each item P_i, $(i = 1, 2, ..., n)$ is a polygon whose shape is defined by a list of vertices. In addition, each element can be rotated by certain angles. The set of allowable rotations (in degrees) of each polygon is denoted by $O = \{O_1, O_2, ..., O_n\}$, where $O_i = \{o_1, o_2, ..., o_k\}$ describes rotations of the element P_i. For example: $O_2 = \{0, 90, 180, 270\}$ states that element P_2 can be rotated by any of the listed directed angles. In the solution there may be at most b_i elements P_i. Since some elements from set Z can be the same (congruent), the number of different types of items is denoted by m. The elements are cut from two-dimensional sheets $S = (H, W)$ of rectangular shape with height H and width W. Since the cutting and packing problems are equivalent, we shall use both concepts: cutting an element from a sheet, or placing an element on a plate. We call each possible way of cutting a sheet a cutting pattern, or layout.

As in [18] we consider, according to the typology proposed by Wäscher et al. [47], the following three types of problems:

- Single knapsack problem (KP) – cutting of a single rectangular sheet. Each element P_i may occur in the solution at most once ($b_i = 1$ for each $i = 1, 2, ..., n$). The objective is to maximize the ratio of the total area of elements arranged on the plate to the area of the plate (called here the filling rate).
- Single large object placement problem (PP) – the only difference from the previous type of problem is that there are no constraints on the number of elements of the same type in the layout. Therefore, we can assume that it is a problem where the assortment of small items is weakly heterogeneous.
- Single stock size cutting stock problem (CSP) – we have an unlimited number of sheets and a limited number of elements. We cut a certain number of elements of type P_i so as to use as few sheets as possible. The assortment of small items is weakly heterogeneous. A problem of this type can be associated with the implementation of a specific manufacturing contract.

3. One-pass heuristic

3.1. No-fit polygon

As was mentioned above, the irregular cutting problem (nesting problem) is a much more complex task than cutting rectangular items from a rectangular sheet. In this case it is harder to ensure feasible cutting patterns, where elements do not overlap. In addition, we need to pack elements as close as possible to each other. A recent approach to meeting the above conditions is the no-fit polygon (NFP) technique. Details of the NFP algorithm can be found in [10, 13].

Let *A* and *B* be polygons, where the item *A* is fixed. Let polygon *B* move around the polygon *A*, in such a way that the two polygons are always in contact but never overlap. We choose a point (called the reference point) on the polygon *B*. Moving (orbiting) item *B* around *A*, the point marks out a closed path (polygon), which we will denote by NFP(*A*, *B*) (Fig. 1a). The polygon NFP(*A*, *B*) has the following properties:

• if the reference point of *B* is inside NFP(*A*, *B*), then polygons *A* and *B* overlap,

• if the reference point of *B* is on the boundary of NFP(*A*, *B*), then polygons *A* and *B* are touching;

• if the reference point of B is outside NFP(*A*, *B*), then polygons *A* and *B* are separated.

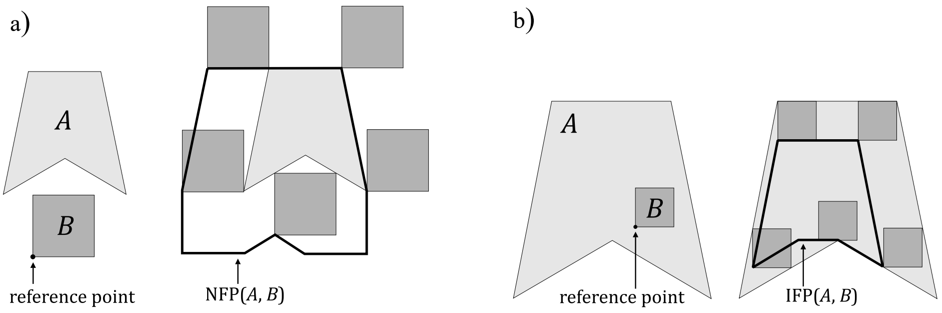

Fig. 1. An example of: a) no-fit polygon, b) inner fit polygon of a fixed polygon *A* and an orbiting polygon *B*

Therefore it is easy to check whether the polygons *A* and *B* overlap. In addition, because we want as dense packing as possible, it is appropriate to arrange items on the sheet so that they touch, i.e., the reference point of *B* should be positioned on the boundary of NFP(*A*, *B*).

Similarly, we can determine the inner fit polygon of two polygons *A* and *B*, denoted by IFP(*A*, *B*). This is a path (polygon) marked out by the reference point of a polygon *B* moving (orbiting) inside the (fixed) polygon *A* (Fig. 1b). In this case we also assume that both polygons are touching. With IFP(*A*, *B*), we can easily check whether the polygon *A* contains the polygon *B* – the reference point of *B* should be positioned on the boundary of or inside IFP(*A*, *B*).

The NFP of a partial packing *State* on the plate *S* and a polygon *P* will be calculated as the set difference of IFP(*S*, *P*) and the sum of NFPs of the elements A_i placed on the plate and the polygon *P* (Fig. 2):

$$\mathrm{NFP}(State, P) = \mathrm{IFP}(S, P) \setminus \bigcup_i \mathrm{NFP}(A_i, P)$$

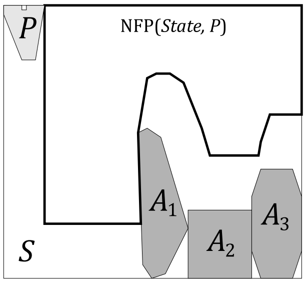

Fig. 2. NFP of a partial packing (*State*) and a polygon *P*

3.2. Description of the algorithm

One-pass heuristics, which use NFP to represent elements arranged on a plate, can be found in the literature [24, 39]. We propose a new algorithm in which NFPs are additionally used in the process of evaluating partial solutions by estimating the free space left after placing a given element on the plate.

In this paper, we propose a one-pass algorithm. Starting with an empty plate, the algorithm places (or cuts) an element (polygon) onto (or from) the current state of the plate. There are no returns or changes in a partial solution, excluding placement of the next item. Packing ends when the items are exhausted, or when it is no longer possible to place any item on the plate.

We will use the following notation: *State* – a plate with some elements packed (partial solution), NFP (*State*, *Polygon*) – the NFP of a partial solution *State* and an element *Polygon*. The NFP can be a polygon (possibly empty) or a sum of disjoint polygons (see Fig. 2), $State_{\mathrm{current}}$ – the current state of the plate; $State_{\mathrm{next}} = State_{\mathrm{current}} \cup Polygon$ – the next state of the plate. An element *Polygon* has been packed at its optimal placing point (a vertex of its NFP), Area – surface area, $\mathrm{Area}_{\mathrm{convex}}$ – surface area of the convex hull, Elems(*State*) – the set of items that can be packed in a *State*, $\mathrm{Verts}(P)$ – the vertices of a polygon *P*, Fit – the fitting function.

At each step of the algorithm, an element is placed depending on the value of the fitting (evaluation) function. The fitting function evaluates both an item and its placement at the so-called placing point. The placing point of an element *P* can be any vertex of the polygon $\mathrm{NFP}(\mathit{State}, P)$. At each step of the algorithm, the element with the highest value of the fitting function for the pair (element *P*, placing point *V*) is cut. The considered fitting functions are based on the current and next state of the plate.

$$(P,V) \leftarrow \max_{P \in \mathrm{Elems}(\mathit{State})} \ \max_{V \in \mathrm{Verts}(\mathrm{NFP}(\mathit{State},P))} \mathrm{Fit}(P, V, \mathit{State})$$

The one-pass nature of the algorithm significantly shortens the packing time. This time depends linearly on the product of the number of elements placed on the plate and the average computation time of the fitting function. On the other hand, one-pass means that any “wrong” placement of an element will be reflected in the final result without the possibility of any adjustment. However, with suitable selection of the fitting function, the algorithm can give very good results not only in terms of execution time, but also in terms of filling rate.

The algorithm described above can be used directly in problems of type KP and PP. In the CSP, we use the algorithm for any sheet of the cutting stock. The items are packed on the first plate, then on the second one, and so on until all elements are packed. In order to optimize the packing time, if on the next plate the same pattern of items can be packed (if the constraints allow it), then this pattern of elements is directly copied onto the next plate (without running the packing algorithm).

3.3. Description of fitting functions

Slightly similar fitting functions are considered in [39]. In that work, the fitting functions depend on the different properties of the elements and their combinations being added, such as: waste, overlap and distance combined with min/max area, length and overlap of rectangular enclosures. However, in our work, the evaluation functions depend only on the surface areas of the corresponding NFPs and estimate the average area of the empty space left to be filled on the plate, thus they are simpler, faster, and easily interpretable.

For the described algorithm, we construct several fitting functions. These functions evaluate both the item (polygon) to be packed and its placing point (vertex of NFP).

Let us introduce the following notation:

$$\mathrm{sumAreaNFP}(\mathit{State}) = \sum_{P \in \mathit{Elems}} \mathrm{Area}\left(\mathrm{NFP}(\mathit{State}, P)\right)$$

$$\mathrm{maxAreaNFP}(\mathit{State}) = \max_{P \in \mathit{Elems}} \mathrm{Area}\left(\mathrm{NFP}(\mathit{State}, P)\right)$$

We considered eight fitting functions:

Opt1. This fitting function depends on the area of the item *Polygon* to be packed and on the change in the sum of the NFP areas due to packing *Polygon* in a particular position (and hence moving to a new state)

$$\mathrm{Fit} = \frac{\mathrm{Area}(Polygon)}{\mathrm{sumAreaNFP}(State_{\mathrm{current}}) - \mathrm{sumAreaNFP}(State_{\mathrm{next}})}$$

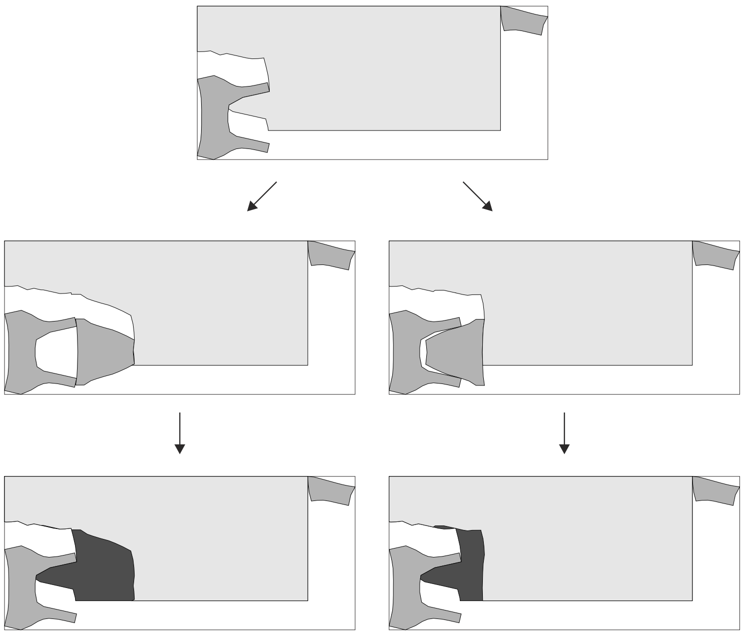

Fig. 3. Reduction in the NFP area depending on how an item fits into the current packing state of the plate. The dark area indicates the decrease in the NFP area for two different packed items

Figure 3 shows the change in NFP area for one element. If the item being packed is ill-fitted to the elements currently placed on the plate (Fig. 3, left), then the NFP area of any item decreases significantly (Fig. 3, left, dark area). If the element being packed is well-fitted to the current state of the plate (Fig. 3, right), then the decrease in the NFP area is smaller (Fig. 3, right, dark area).

Opt2. This fitting function is similar to Opt1, but in the numerator we have the squared area of the element being packed. This fitting function prefers larger items. For two elements with the same NFP change, the element with greater area is packed.

$$\text{Fit} = \frac{\left[\text{Area}(\mathit{Polygon})\right]^2}{\text{sumAreaNFP}(\mathit{State}_{\text{current}}) - \text{sumAreaNFP}(\mathit{State}_{\text{next}})}$$

Opt1.5. This fitting function is similar to Opt1, but in the numerator we have the area of the convex hull of the element being packed.

$$\text{Fit} = \frac{\text{Area}_{\text{convex}}(\mathit{Polygon})}{\text{sumAreaNFP}(\mathit{State}_{\text{current}}) - \text{sumAreaNFP}(\mathit{State}_{\text{next}})}$$

Two items can be very similar in terms of the area available for subsequent items to be packed (similar NFPs relative to each other) while having very different areas. Using the area of the convex hull can reduce this difference.

Opt2.5. This fitting function is similar to Opt1.5, but in the numerator we have the squared area of the convex hull of the element being packed. This fitting function prefers items with convex hulls of larger area. For two polygons with the same change in the NFP, the item with the greater convex hull is packed.

$$\text{Fit} = \frac{\left[\text{Area}_{\text{convex}}(\mathit{Polygon})\right]^2}{\text{sumAreaNFP}(\mathit{State}_{\text{current}}) - \text{sumAreaNFP}(\mathit{State}_{\text{next}})}$$

Opt3. This fitting function depends on the area of the item *Polygon* to be packed and on the change in the maximum NFP area due to packing *Polygon* (and hence moving to a new state)

$$\text{Fit} = \frac{\text{Area}(\mathit{Polygon})}{\text{maxAreaNFP}(\mathit{State}_{\text{current}}) - \text{maxAreaNFP}(\mathit{State}_{\text{next}})}$$

Opt4. Similarly as in Opt3, but in the numerator, we have the square of the area. This fitting function prefers larger elements.

$$\text{Fit} = \frac{\left[\text{Area}(\mathit{Polygon})\right]^2}{\text{maxAreaNFP}(\mathit{State}_{\text{current}}) - \text{maxAreaNFP}(\mathit{State}_{\text{next}})}$$

Opt3.5. Similarly as in Opt3, but in the numerator, we have the area of the convex hull.

$$\text{Fit} = \frac{\text{Area}_{\text{convex}}(\mathit{Polygon})}{\text{maxAreaNFP}(\mathit{State}_{\text{current}}) - \text{maxAreaNFP}(\mathit{State}_{\text{next}})}$$

Opt4.5. Similarly as in Opt3.5, but in the numerator we have the squared area of the convex hull. This fitting function prefers items with larger convex hulls.

$$\text{Fit} = \frac{\left[\text{Area}_{\text{convex}}(Polygon)\right]^2}{\text{maxAreaNFP}(State_{\text{current}}) - \text{maxAreaNFP}(State_{\text{next}})}$$

The algorithm (for all eight fitting functions) places items in a characteristic way. Since, for a rectangular sheet, the best fitting is realized in the corners or edges of the sheet, the elements are packed in a circular fashion. After all items are packed, an empty space is left in the centre of the plate. Figure 4 presents a partial solution for the instance DAGLI (Table 1) with the NFP of the next element to be packed.

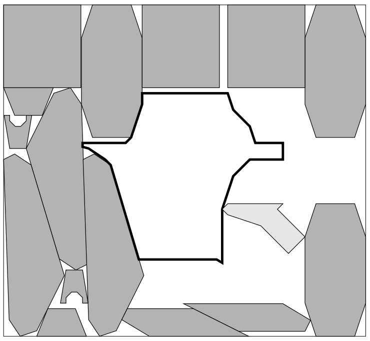
Fig. 4. An example partial solution for the instance DAGLI

4. Computational experiments

4.1. Experimental setup

The described algorithm was implemented in the C#.net 4.0 programming language and tested on an Intel Pentium Dual CPU 2.20 GHz, 2 GB RAM personal computer with the Microsoft Windows Vista operating system. For polygon operations, the Clipper library by Angus Johnson (http://www.angusj.com/delphi/clipper.php) was used with the following operations: set addition and multiplication, surface area function, and Minkowski sum functions. The NFP computations were performed using Minkowski sum procedures from that library. All the procedures of the algorithm and fitting functions were implemented and executed in single-threaded fashion.

There are few test instances, either in the literature or in online databases, which fulfil all of the conditions listed in Section 2. The most commonly available instances represent a slightly different type of problem, where only one dimension (e.g., the height) of the sheet is fixed. In this type of problem we minimize the width of the sheet.

In Wäscher et al. typology [47], a problem of this type is called a two-dimensional irregular open dimension problem (ODP).

As was mentioned in the Introduction, irregular two-dimensional cutting stock problems are rarely presented in the literature. In the present work, computational experiments were performed on well-known datasets with modifications as proposed by Del Valle et al. [18] and Song and Bennell [43]. In some papers [33–45], the results of computations are given for a slightly different class of problems: irregular single bin size bin packing problems. However, some of the features of the test instances proposed in those works (a strongly heterogeneous assortment of small items, small number of optimal cutting patterns for single plates, mostly with 100% occupied area) exclude our algorithm from competition with other methods – a one-pass heuristic cannot generate better results than multi-pass methods on such test data. Test datasets of the type proposed in [36] (for a single knapsack problem) have similar characteristics and will not be considered in this paper.

Table 1. Characteristics of benchmark data instances

Name	Number of different pieces (m)	Total number of pieces (n)	Height	Width	Rotations (directed angle)
FU	12	12	38	34	{0, 90, 180, 270}
JACKOBS1	25	25	40	13	{0, 90, 180, 270}
JACKOBS2	25	25	70	28.2	{0, 90, 180, 270}
SHAPES0	4	43	40	63	{0}
SHAPES1	4	43	40	59	{0, 180}
SHAPES2	7	28	15	27.3	{0, 180}
DIGHE1	16	16	100	138.14	{0}
DIGHE2	10	10	100	134.05	{0}
ALBANO	8	24	4900	10122.63	{0, 180}
DAGLI	10	30	60	65.6	{0, 180}
MAO	9	20	2550	2058.6	{0, 90, 180, 270}
MARQUES	8	24	104	83.6	{0, 90, 180, 270}
SHIRTS	8	99	40	63.13	{0, 180}
SWIM	10	48	5752	6568	{0, 180}
TROUSERS	17	64	79	245.75	{0, 180}

Finally, we decided to carry out computational experiments on well-known benchmark problems (originally used in strip packing problems) from the European Working Group in Cutting and Packing (ESICUP) website (http://www.fe.up.pt/esicup). We performed computational experiments on the following datasets. Three types of problems are considered. Modifications of the original instances are also presented:

1. Single knapsack problem: 15 instances with modifications as proposed by del Valle et al. [18]. Table 1 contains basic information on the test instances.

2. Single large object placement problem: the same instances as in 1 but with no constraints on the number of items of each type used in the cutting patterns.

3. Single stock size cutting stock problem:

A. The same instances as in 1 (Table 1) with the constraints on the number of items as in [18] (generated randomly from [1, 100]). The data for the number of items in each dataset are available on the website: http://katmat.wilsig.tu.koszalin.pl/projects/p1/.

B. The same instances as in 1 with modifications as proposed by Song and Bennell [43]. The width of the sheet is equal to its height (a square with the length of a side equal to the Height value from Table 1). The number of items of each type P_i is equal to 100. The total number of pieces for each instance is thus $100n$.

Note that the rotation notation (last column in Table 1) differs slightly from that used in other papers [18, 25]. We prefer to denote possible rotations of items as a set of directed angles.

4.2. Results

A number of preliminary experiments were carried out, the aim of which was to select the best fitting function for a given type of problem. Tables 2–5 show the results obtained by our algorithm with the eight fitting functions for all the test instances from the benchmark set and for all types of problems. For problems of type KP (Table 2) and PP (Table 3), the results of the computations are presented as the ratio of the total area of elements packed on the plate to the area of the plate (column Area) and the computation time in seconds spent solving an instance (column Time), excluding the time required for NFP computations. The best result among the fitting functions used in the algorithm is printed in bold.

The data in Table 2 show that the algorithm Opt2.5 gives the best results for problems of type KP. Algorithm Opt2.5 finds layouts with the best filling rate for 12 out of the 15 test instances. Algorithm Opt4 is only slightly worse, with 11 best results. The worst among the fitting functions is Opt3 – it finds the best result for only 5 instances. Comparing the computation times of the algorithms, we see that they are similar. Only algorithm Opt4.5 seems to be slightly faster than the others.

For problems of type PP (Table 3), the algorithm Opt3 gives the best results for total area (for 9 out of 15 instances). Computation times are now more varied. Algorithms Opt2, Opt2.5, Opt4 and Opt4.5 are notably faster than the other four. This is due to the fact that the fitting functions from the first group prefer larger elements. With no restriction on the number of elements, shorter computation times are produced.

Table 4 presents results for the CSP, where we pack a fixed number of items on as small as possible a number of plates. The columns of the table show the number of plates used (# Plates) and computation time in seconds (Time). The best algorithm is Opt4 (for 10 instances). It is also the fastest algorithm (in terms of both the number of instances with the shortest computation time, and the average execution time).

Table 2. Comparison of algorithms Opt1–Opt4.5 for problems of type KP

Name	Opt1		Opt2		Opt1.5		Opt2.5		Opt3		Opt4		Opt3.5		Opt4.5	
	Area	Time [s]	Area	Time [s]	Area	Time [s]	Area	Time [s]	Area	Time [s]	Area	Time [s]	Area	Time [s]	Area	Time [s]
FU	**0.8382**	1.00	**0.8382**	1.01	**0.8382**	0.93	**0.8382**	1.00	0.6865	1.22	**0.8382**	1.02	0.6865	1.22	**0.8382**	1.01
JACKOBS1	**0.7538**	18.84	**0.7538**	19.60	**0.7538**	22.83	**0.7538**	21.52	0.7154	15.87	**0.7538**	20.52	0.7058	37.29	**0.7538**	19.71
JACKOBS2	**0.6844**	30.52	**0.6844**	24.17	**0.6844**	31.91	**0.6844**	27.50	**0.6844**	34.00	**0.6844**	27.26	**0.6844**	34.79	**0.6844**	29.43
SHAPES0	0.5857	0.68	0.5984	0.6	0.5762	0.80	**0.6095**	0.70	0.6016	0.73	0.5587	0.44	0.5476	0.78	0.6016	0.55
SHAPES1	0.6593	2.83	0.6441	1.73	0.6458	4.73	**0.6763**	2.71	0.6763	3.30	0.6288	1.83	**0.6763**	4.26	**0.6763**	2.82
SHAPES2	0.7375	2.07	0.7643	1.81	0.7179	2.04	0.7668	1.73	0.7375	1.75	**0.7717**	1.31	0.7521	1.36	0.7375	1.09
DIGHE1	**0.7240**	0.43	**0.7240**	0.42	**0.7240**	0.45	**0.7240**	0.43	0.6561	0.40	**0.7240**	0.39	**0.7240**	0.41	0.6667	0.46
DIGHE2	**0.7460**	0.14	0.7042	0.12	**0.7460**	0.12	0.7042	0.10	**0.7460**	0.14	**0.7460**	0.12	**0.7460**	0.13	0.6285	0.08
ALBANO	0.7832	3.87	0.8297	2.52	0.6764	4.99	**0.8606**	2.76	0.7447	3.61	0.8357	2.15	0.7378	4.95	0.8287	2.74
DAGLI	**0.7731**	3.83	**0.7731**	3.58	**0.7731**	4.49	**0.7731**	3.50	0.6537	4.95	**0.7731**	3.61	**0.7731**	3.47	**0.7731**	3.80
MAO	**0.7160**	19.86	**0.7160**	16.26	**0.7160**	16.24	**0.7160**	15.21	**0.7160**	17.52	**0.7160**	18.21	**0.7160**	19.23	**0.7160**	14.27
MARQUES	0.7711	11.51	**0.8274**	5.23	0.7646	14.21	**0.8274**	7.31	0.7249	9.09	**0.8274**	9.75	**0.8274**	9.40	**0.8274**	8.31
SHIRTS	0.7958	44.42	0.8542	27.78	0.8025	39.75	0.8482	23.27	0.8025	20.70	**0.8554**	23.01	0.8025	23.94	**0.8554**	22.91
SWIM	**0.6734**	178.77	**0.6734**	159.94	**0.6734**	174.57	**0.6734**	178.85	**0.6734**	131.99	**0.6734**	131.03	0.6538	175.37	**0.6734**	111.28
TROUSERS	0.8122	105.81	**0.8863**	69.89	0.8507	71.97	**0.8863**	74.54	0.7367	62.96	0.8774	85.54	0.7866	89.01	0.8774	84.07

Table 3. Comparison of algorithms Opt1–Opt4.5 for problems of type PP

Name	Opt1		Opt2		Opt1.5		Opt2.5		Opt3		Opt4		Opt3.5		Opt4.5	
	Area	Time [s]	Area	Time [s]	Area	Time [s]	Area	Time [s]	Area	Time [s]	Area	Time [s]	Area	Time [s]	Area	Time [s]
FU	0.8390	2.58	**0.9412**	0.81	0.8390	2.57	**0.9412**	0.80	0.8189	3.86	0.9063	0.78	0.8189	3.90	0.9063	0.79
JACKOBS1	0.9058	150.36	0.8952	7.44	0.8712	168.27	0.8875	16.74	**0.9192**	193.07	0.8904	9.42	0.8808	75.95	0.8808	19.47
JACKOBS2	0.8409	87.02	0.8308	8.98	**0.8693**	192.53	0.8186	9.04	0.8354	77.29	0.8308	8.59	0.8141	111.31	0.8186	8.86
SHAPES0	0.6921	1.19	0.7397	0.32	0.6286	1.31	0.5571	0.66	**0.7810**	0.74	0.7317	0.33	0.6571	1.42	0.5540	0.61
SHAPES1	0.6932	3.59	0.7475	0.78	0.6559	6.90	0.6797	2.94	0.7373	3.37	**0.7559**	0.78	0.6593	6.16	0.6424	2.68
SHAPES2	0.8791	6.18	0.7900	1.76	0.6996	3.46	0.6093	2.08	**0.8864**	5.25	0.8132	1.59	0.6557	2.13	0.5873	1.78
DIGHE1	0.7669	1.87	0.7249	0.35	0.7669	1.85	0.7710	0.41	**0.7991**	1.52	0.7490	0.50	0.7565	1.54	0.7333	0.43
DIGHE2	0.7840	0.24	**0.8188**	0.12	0.7840	0.25	**0.8188**	0.10	0.7699	0.26	0.7292	0.08	0.7699	0.27	0.7292	0.09
ALBANO	0.8635	12.66	0.8360	2.52	0.8304	16.18	0.8474	2.65	**0.8822**	6.15	0.8331	3.14	0.7858	44.74	0.8331	3.19
DAGLI	0.8465	13.79	0.8633	2.92	0.7872	31.19	0.8569	3.50	**0.8947**	4.75	0.8615	2.29	0.8637	11.96	0.8456	2.30
MAO	**0.9197**	830.77	0.8737	19.43	0.9070	700.98	0.8647	19.10	0.9189	74.79	0.8687	10.93	0.9084	64.52	0.8746	11.13
MARQUES	**0.8821**	39.40	0.8341	4.12	0.8538	43.56	0.8357	4.72	0.8702	22.83	0.8432	7.16	0.7803	151.52	0.8432	7.61
SHIRTS	0.9512	323.30	0.8479	10.87	0.9437	319.72	0.8463	10.50	**0.9706**	37.93	0.8677	11.45	0.9069	63.86	0.8661	11.29
SWIM	0.7828	497.17	0.7643	211.32	0.7398	386.52	0.6869	130.39	**0.8081**	530.50	0.8025	181.06	0.7209	199.10	0.6563	63.64
TROUSERS	0.9302	1020.01	0.9089	23.75	0.9340	795.21	0.9089	23.41	**0.9367**	408.35	0.9122	63.63	0.9300	275.48	0.9122	64.21

Table 4. Comparison of algorithms Opt1–Opt4.5 for problems of type CSP (dataset 3A)

Name	Opt1		Opt2		Opt1.5		Opt2.5		Opt3		Opt4		Opt3.5		Opt4.5	
	# Plates	Time [s]	# Plates	Time [s]	# Plates	Time [s]	# Plates	Time [s]	# Plates	Time [s]	# Plates	Time [s]	# Plates	Time [s]	# Plates	Time [s]
FU	70	15.55	**67**	19.35	70	15.66	**67**	19.32	73	17.09	**67**	14.48	73	17.11	**67**	14.57
JACKOBS1	50	832.77	47	541.23	48	826.70	**46**	373.54	49	675.66	**46**	414.79	47	661.44	47	327.93
JACKOBS2	39	846.00	39	794.72	39	1205.53	**38**	833.24	**38**	808.54	**38**	624.44	39	1010.50	**38**	859.40
SHAPES0	51	3.41	51	3.41	51	3.64	**49**	5.09	50	6.27	51	4.24	52	3.43	50	4.02
SHAPES1	48	11.34	**47**	17.45	49	19.84	**47**	14.22	**47**	18.05	48	13.79	48	18.84	49	11.90
SHAPES2	67	21.56	65	14.93	68	14.06	66	14.04	67	24.18	**64**	11.61	67	18.13	65	11.05
DIGHE1	58	17.79	48	9.50	51	9.97	46	12.55	**37**	11.70	47	8.75	49	10.70	47	9.33
DIGHE2	**40**	2.01	**40**	1.70	**40**	2.25	**40**	1.60	**40**	1.46	43	1.41	**40**	1.37	42	1.11
ALBANO	86	36.91	**78**	49.27	86	82.49	**78**	20.97	84	33.85	**78**	38.14	82	79.39	**78**	21.77
DAGLI	54	86.50	51	48.49	56	134.83	51	50.97	53	57.32	**50**	42.34	55	91.22	**50**	50.98
MAO	43	1254.35	41	140.52	43	1103.03	41	125.85	**39**	722.02	41	109.59	42	262.67	41	154.55
MARQUES	52	180.34	50	181.63	54	240.97	57	112.63	53	126.19	**49**	137.97	52	394.46	50	166.90
SHIRTS	47	1210.85	43	256.72	46	1079.16	43	250.47	46	498.70	**42**	198.24	45	382.00	**42**	192.98
SWIM	50	2264.98	48	2023.85	50	2536.11	**47**	1996.71	50	2861.27	**47**	1554.47	49	1927.21	**47**	1729.82
TROUSERS	53	4111.82	47	2018.66	52	4558.91	**47**	2107.14	51	2380.73	**47**	1440.29	50	2847.15	**47**	1459.41

Table 5. Comparison of algorithms Opt1–Opt4.5 for problems of type CSP (dataset 3B

Name	Opt1		Opt2		Opt1.5		Opt2.5		Opt3		Opt4		Opt3.5		Opt4.5	
	# Plates	Time [s]	# Plates	Time [s]	# Plates	Time [s]	# Plates	Time [s]	# Plates	Time [s]	# Plates	Time [s]	# Plates	Time [s]	# Plates	Time [s]
FU	95	23.17	88	24.99	95	23.03	88	25.20	92	25.55	**86**	18.20	92	25.41	**86**	17.78
JACKOBS1	30	7124.88	29	8174.81	30	13541.36	**28**	7379.89	30	8140.65	29	6359.71	29	7622.93	29	7637.56
JACKOBS2	34	11445.07	34	10363.43	34	14863.52	**33**	8321.25	34	5680.36	**33**	5708.01	34	5692.61	**33**	6057.03
SHAPES0	193	2.59	**168**	3.15	191	1.54	169	2.80	188	3.57	171	3.22	182	1.96	169	2.38
SHAPES1	161	5.74	**157**	8.54	166	4.34	158	6.51	158	4.83	159	7.42	168	8.26	159	6.72
SHAPES2	207	5.95	194	6.58	213	5.53	**191**	3.92	205	7.14	193	4.55	210	6.16	196	4.62
DIGHE1	156	5.81	**138**	7.28	145	5.11	140	4.20	144	5.88	144	4.34	148	6.30	142	5.39
DIGHE2	**135**	0.98	139	0.84	**135**	0.98	144	1.05	148	1.33	136	1.12	139	1.26	147	1.12
ALBANO	233	5.88	240	5.74	227	7.07	**220**	4.13	226	4.97	217	3.57	227	6.72	223	2.03
DAGLI	107	73.50	100	54.60	108	96.81	100	51.87	**85**	41.16	100	74.83	108	61.18	101	68.39
MAO	75	2748.90	71	233.10	74	2661.26	72	177.17	76	1222.06	**70**	206.29	74	437.99	**70**	181.86
MARQUES	82	324.87	**76**	315.49	86	659.26	**76**	247.24	81	245.42	77	240.03	82	601.44	77	169.96
SHIRTS	175	392.14	160	100.10	174	320.04	159	113.68	172	224.84	**158**	89.04	171	139.30	**158**	77.00
SWIM	115	1827.70	108	1375.08	111	1280.72	**107**	1357.51	114	1906.17	108	1027.46	111	1137.99	**107**	1095.78
TROUSERS	388	243.67	327	62.72	**227**	145.88	327	63.56	385	374.22	343	56.07	344	234.64	343	56.14
POLY 5B	157	15460.41	157	10632.30	157	15211.28	155	11129.09	**154**	13234.76	156	9914.31	155	12656.35	155	9563.33

Table 5 presents results of computations for algorithms Opt1–Opt4.5 for problems of type CSP with the constraints proposed by Song and Bennell [43] (data instances 3B). For completeness, we performed calculations for all the instances from Table 1 (in the paper), and also for those not considered by Song and Bennell [43]. The columns of the table show the number of plates used (# Plates) and computation time in seconds (Time). The best algorithm is Opt2.5 (for 6 instances).

In Tables 6–9 we compare our algorithms with the algorithms used by Del Valle et al. [18] for all the types of problems. For problems of type KP (Table 7) and PP (Table 8), the results of the computations are presented as the ratio of the total area of elements packed on the plate to the area of the plate (column Area) and the computation time in seconds spent solving an instance (column Time), excluding the time required for NFP computations.

Table 9 presents results for the CSP, where we pack a fixed number of items on as small as possible a number of plates. The columns of the table show the number of plates used (# Plates) and computation time in seconds (Time). The best result among the fitting functions used in the algorithm is printed in bold. We should note the differences in speed between implementations and computers. The algorithms from [18] were implemented in C++ and performed on a PC with Intel Core 2 Quad CPU 2.4 GHz. Our software was written in C# .net 4.0 and computed on a PC with Intel Pentium Dual CPU 2.2 GHz. The processor clock speeds are similar, but it must be noted that the same code implemented in C++ is usually twice as fast as when implemented in C# .net. Therefore we can assume, as an approximation, that our computational platform is two times slower than the test platform from Del Valle et al. [18].

Table 6. Time [s] required for NFP computations

Name	Del Valle et al. [18]	Opt [s]
FU	0.26	0.50
JACKOBS1	59.49	1.94
JACKOBS2	53.80	1.91
SHAPES0	51.32	0.02
SHAPES1	202.56	0.03
SHAPES2	17.90	0.07
DIGHE1	0.12	0.10
DIGHE2	0.15	0.07
ALBANO	14.40	0.19
DAGLI	26.16	0.17
MAO	102.91	1.52
MARQUES	57.50	0.43
SHIRTS	208.49	0.11
SWIM	1088.21	2.86
TROUSERS	48.22	0.38

In Table 6, the computation times of NFPs for all the pairs of item types in the test instance are compared. We see that computations of NFPs in our algorithm are mostly much shorter (by as much as two orders of magnitude) than in the algorithm used by Del Valle et al. [18].

Table 7 presents a comparison of the GRASP algorithm used by Del Valle et al. [18] and the algorithm Opt2.5 which is the best among the tested fitting functions for problems of type KP. Opt2.5 is worse than GRASP for only one instance (out of 15). Computation times are mostly much shorter. We can also see that the algorithm Opt2.5 finds more optimal solutions, i.e., it places all the items to be packed on the plate. In Table 7, optimal solutions are printed in bold in the Quantity of items columns.

Table 7. Comparison of the performance of algorithms for problems of type KP

Name	Del Valle et al. [18] GRASP			Opt2.5		
	Area	Quantity of items	Time [s]	Area	Quantity of items	Time [s]
FU	**0.8382**	**12**	21.79	**0.8382**	**12**	1.00
JACKOBS1	**0.7538**	**25**	8.30	**0.7538**	**25**	21.52
JACKOBS2	**0.6844**	**25**	565.71	**0.6844**	**25**	27.50
SHAPES0	0.6016	41	1552.46	**0.6095**	40	0.70
SHAPES1	0.6424	41	3891.60	**0.6763**	**43**	2.71
SHAPES2	0.7289	26	1048.12	**0.7668**	26	1.73
DIGHE1	**0.7240**	**16**	10.68	**0.7240**	**16**	0.43
DIGHE2	**0.7460**	**10**	0.07	0.7042	9	0.10
ALBANO	0.8038	23	961.59	**0.8606**	**24**	2.76
DAGLI	0.7586	29	1106.40	**0.7731**	**30**	3.50
MAO	**0.7160**	**20**	120.42	**0.7160**	**20**	15.21
MARQUES	**0.8274**	**24**	217.77	**0.8274**	**24**	7.31
SHIRTS	0.7702	96	14317.13	**0.8482**	93	23.27
SWIM	0.6427	46	39781.43	**0.6734**	**48**	178.85
TROUSERS	0.7866	62	5796.59	**0.8863**	**64**	74.54

Figure 5 shows some patterns for cutting found by the method Opt2.5 for the KP. For these instances, the algorithm Opt2.5 found optimal solutions, while the GRASP algorithm did not. We can see that for a few layouts, not only are all the items placed on the sheet, but some free space also remains on the sheet. For all the tested algorithms, it is characteristic that the free space is located in the centre of the plate.

Table 8 presents a comparison of the algorithm Solve2KP used by Del Valle et al. [18] and the algorithm Opt3, which is the best among the tested methods for problems of type PP.

Overall, Opt3 is worse than Solve2KP. This is due to the fact that Solve2KP is a rectangular-oriented algorithm. For instances without restrictions on the number of items, Solve2KP uses rectangular pieces to get the best result. If there are no rectangular items in an instance, Solve2KP does not find similarly good solutions. In this case, the algorithm Opt3 is better, as it is specially tuned to solve problems with irregular shapes. The execution times of Opt3 are mostly much shorter than those of Solve2KP. Examples of solutions for problems of type PP are shown in Fig. 6. For these instances, Opt3 is better than Solve2KP.

Table 8. Comparison of performance of the algorithms Solve2KP and Opt3 for problems of type PP

Name	Del Valle et al. [18] Solve2KP (av.)		Opt3	
	Area	Time [s]	Area	Time [s]
FU	**0.9892**	5.01	0.8189	3.86
JACKOBS1	**0.9870**	49.60	0.9192	193.07
JACKOBS2	**0.9851**	66.08	0.8354	77.29
SHAPES0	0.5873	134.43	**0.7810**	0.74
SHAPES1	0.6893	248.15	**0.7373**	3.37
SHAPES2	**0.9183**	49.37	0.8864	5.25
DIGHE1	0.6909	7.62	**0.7991**	1.52
DIGHE2	0.7657	3.14	**0.7699**	0.26
ALBANO	**0.9653**	37.93	0.8822	6.15
DAGLI	**0.9196**	50.99	0.8947	4.75
MAO	**0.9644**	71.53	0.9189	74.79
MARQUES	**0.9515**	43.76	0.8702	22.83
SHIRTS	**1.0000**	508.92	0.9706	37.93
SWIM	0.7272	2206.42	**0.8081**	530.50
TROUSERS	**0.9986**	193.54	0.9367	408.35

Table 9 presents a comparison of the algorithm Solve2CS used by Del Valle et al. [18] and the algorithm Opt4, which is the best among the tested methods for the CSP. We can see that even for the naïve algorithm which we use to solve a cutting stock problem, the results can be very good. Only for one instance is Opt4 slightly worse than Solve2CS. For the other instances Opt4 finds much better results than Solve2CS. Note the huge difference between the computation times of the compared methods. Execution times of the algorithm Opt4 can be several orders of magnitude shorter than those of Solve2CS. This arises from the fact that Opt4 is a one-pass algorithm. On the other hand, Opt4 packs elements cleverly enough to use a smaller number of sheets than Solve2CS. This is a feature of all of the algorithms tested in this study.

Table 10 presents results of computations for problems of type CSP with the constraints proposed by Song and Bennell [43] (data instances 3B). For completeness of results, we performed calculations for all instances from Table 1, and also for those missed in [43]. The columns of the table show the number of plates used (# Plates) and computation time in seconds (Time). In this table, a comparison of the algorithm Opt2.5 with the results of methods CG1, CG10, CG100 from Song and Bennell [43] is shown. These are the best algorithms presented in that paper.

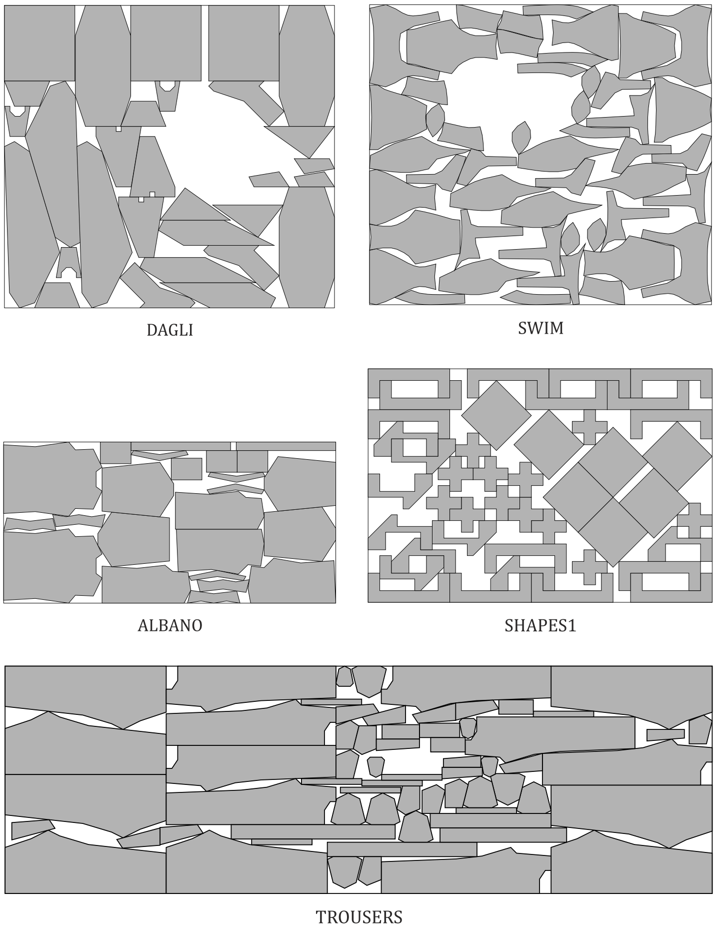

Fig. 5. Optimal solutions found by the algorithm Opt2.5 for problems of type KP

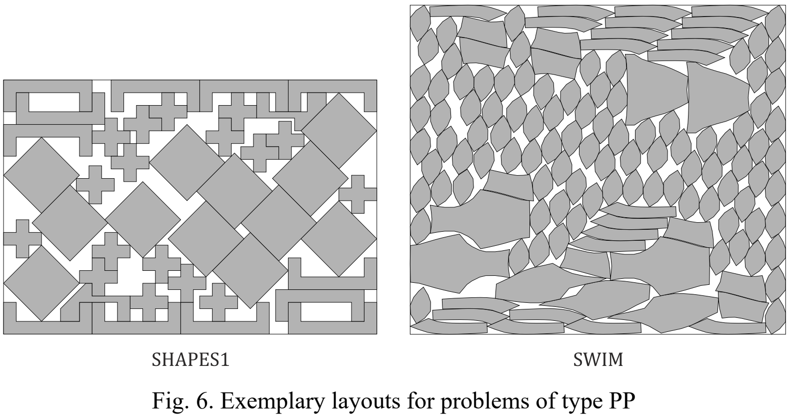

Fig. 6. Exemplary layouts for problems of type PP

We would also draw attention to the large difference in the result for the instance Jakobs1, which may suggest a mistake in reporting of the results for algorithms CG1–CG100. The results (# Plates) obtained for OPT 2.5 are comparable to CG1–CG100. While the computation times for OPT 2.5 are clearly better, especially compared with the best method CG100, where times vary by several orders of magnitude, note must be taken of the differences in speed between implementations and computers.

Table 9. Performance comparison of the algorithms Solve2CS and Opt4 for problems of type CSP

Name	Del Valle et al. [18] Solve2CS		Opt4	
	# Plates	Time [s]	# Plates	Time [s]
FU	74	228.93	**67**	14.48
JACKOBS1	50	6,726.73	**46**	414.79
JACKOBS2	47	6,897.00	**38**	624.44
SHAPES0	55	26,701.14	**51**	4.24
SHAPES1	56	59,601.60	**48**	13.79
SHAPES2	**63**	9,413.40	64	11.61
DIGHE1	61	252.30	**47**	8.75
DIGHE2	46	45.18	**43**	1.41
ALBANO	84	6,750.17	**78**	38.14
DAGLI	58	9,304.73	**50**	42.34
MAO	49	7,073.23	**41**	109.59
MARQUES	53	8,298.07	**49**	137.97
SHIRTS	45	1,195,677.01	**42**	198.24
SWIM	60	466,819.10	**47**	1554.47
TROUSERS	53	273,141.60	**47**	1440.29

Table 10. Performance comparison of algorithms CG1–CG100 and Opt4 for problems of type CSP

Name	Song and Bennell [43]						Opt2.5	
	CG1		CG10		CG100			
	# Plates	Time [s]	# Plates	Time [s]	# Plates	Time [s]	# Plates	Time [s]
ALBANO	213	15	211	74	**207**	2181	220	4.13
SHAPES2	178	5	180	56	**174**	476	191	3.92
DAGLI	101	387	99	5979	**97**	74 449	100	51.87
DIGHE2	124	15	111	252	**108**	2002	144	1.05
DIGHE1	128	79	121	984	**114**	11 627	140	4.2
FU	112	8	110	77	108	580	**88**	25.2
JAKOBS1	329	78	301	1705	296	20 524	**28**	7379.89
MAO	164	137	161	3225	136	51 627	72	177.17
MARQUES	137	49	134	905	132	10 713	76	247.24
SHAPES1	144	6	143	69	**140**	335	158	6.51
SHIRTS	158	136	155	6983			159	113.68
SWIM	114	533	106	6373			107	1357.51
TROUSERS	367	1471	348	17430			**327**	63.56

The algorithms from Song and Bennell [43] were implemented in C++ and performed on a PC with CPU 1.6 GHz. Our software was written in C# .net 4.0 and computed on a PC with Intel Pentium Dual CPU 2.2 GHz. In view of the differences in processor speed and the fact that the same code implemented in C++ is usually twice as fast as when implemented in C#.net, we can assume as an approximation that our computational platform is slightly slower – approximately 70% of the speed of the test platform from Song and Bennell [43].

5. Conclusions

This work has concerned the problem of placing two-dimensional elements with irregular shape on one or several rectangular plates with fixed dimensions. We considered three types of problems: knapsack problems (KP), placement problems (PP) (the unconstrained version of KP), and cutting stock problems (CSP). In problems of type KP and PP the objective is to place elements on a plate so as to obtain the best total area of the items. In the CSP we aim to carry out a manufacturing contract to cut a fixed number of items from the smallest number of sheets.

For these problems we have developed a heuristic that places elements on the plate in one pass. The value of a fitting function determines which element is packed in each step of the algorithm. We propose eight fitting functions to evaluate partial solutions. Methods are based on the well-known concept of no-fit polygon. Fitting functions consider both the area of the element being packed and the reduction in the NFP area of an element.

The new algorithms were tested on instances found in the literature. We identified the best algorithm for each of three types of cutting problem. The results of the methods were compared with those published by Del Valle et al. [18], who considered the same types of problems. For KP and CSP instances the new methods appear to be much better than the methods described in the aforementioned paper. The new methods are superior in terms of both the total area of the elements packed and the computation time. For problems of type PP, our proposed methods are better only for a few instances, especially those with no rectangular items. For instances in which all elements have irregular shape, the new algorithms outperform the methods of Del Valle et al. [18]. For instances of type CSP the new method was also compared with the algorithms from Song and Bennell [43]. The results (# Plates) are comparable, while the computation times for the new method are clearly better, sometimes by several orders of magnitude.

The advantage of the presented one-pass heuristic is its simplicity, easy implementation and relatively short calculation time. It is also easy to implement the algorithms for parallel computation, which can significantly reduce computation time on computer systems with multicore processors.

One of the possibilities of further research on the presented algorithms could be the use of other sequences of elements included in the evaluation of partial solutions and new fitting functions.

In future work we also plan to adapt the algorithms presented here to problems with plates of non-rectangular shape and for cutting elements with holes. We intend also to modify the methods to work with problems where only one dimension of the sheet is fixed, i.e., for those classified as two-dimensional irregular open dimension problems or irregular strip packing problems.

References

[1] ABEYSOORIYA R.P., BENNELL J.A., MARTINEZ-SYKORA A., *Jostle heuristics for the 2D-irregular shapes bin packing problems with free rotation*, Int. J. Prod. Econ., 2018, 195, 12–26.

[2] ADAMOWICZ M., ALBANO A., *Nesting two-dimensional shapes in rectangular modules*, Comp. Aid. Des., 1976, 8 (1), 27–33.

[3] ALVAREZ-VALDES R., PARREÑO F., TAMARIT J.M., *A Tabu Search algorithm for two-dimensional non-guillotine cutting problems*, Eur. J. Oper. Res., 2007, 183, 1167–1182.

[4] ALVAREZ-VALDES R., MARTINEZ A., TAMARIT J.M., *A branch and bound algorithm for cutting and packing irregularly shaped pieces*, Int. J. Prod. Econ., 2013, 145 (2), 463–477.

[5] ART Jr. R.C., *An approach to the two-dimensional irregular cutting stock problem*, Technical Report 36.Y08, 1966, IBM Cambridge Centre.

[6] BEASLEY J.E., *An exact two-dimensional non-guillotine cutting tree search procedure*, Oper. Res., 1985, 33, 49–64.

[7] BENNELL J.A., DOWSLAND K.A., *A tabu thresholding implementation for the irregular stock cutting problem*, Int. J. Prod. Res., 1999, 37 (18), 4259–4275.

[8] BENNELL J.A., DOWSLAND K.A., *Hybridising tabu search with optimisation techniques for irregular stock cutting*, Manage. Sci., 2001, 47 (8), 1160–1172.

[9] BENNELL J.A., OLIVEIRA J.F., *The geometry of nesting problems. A tutorial*, Eur. J. Oper. Res., 2008, 184, 397–415.

[10] BENNELL J.A., SONG X., *A comprehensive and robust procedure for obtaining the nofit polygon using Minkowski sums*, Comput. Oper. Res., 2008, 35, 267–281.

[11] BENNELL J.A., SCHEITHAUER G., STOYAN Y., ROMANOVA T., *Tools of mathematical modeling of arbitrary object packing problems*, Ann. Oper. Res., 2010, 179, 343–368.

[12] BŁAŻEWICZ J., HAWRYLUK P., WALKOWIAK R., *Using a tabu search approach for solving the two-dimensional irregular cutting problem*, Ann. Oper. Res., 1993, 41, 313–325.

[13] BURKE E.K., HELLIER R.S.R., KENDALL G., WHITWELL G., *Complete and robust no-fit polygon generation for the irregular stock cutting problem*, Eur. J. Oper. Res., 2007, 179, 27–49.

[14] CAPRARA A., MONACI M., *On the two-dimensional knapsack problem*, Oper. Res. Lett., 2004, 32, 5–14.

[15] CHERRI L.H., MUNDIM L.R., ANDRETTA M., TOLEDO F.M.B., OLIVEIRA J.F., CARRAVILLA M.A., *Robust mixed-integer linear programming models for the irregular strip packing problem*, Eur. J. Oper. Res., 2016, 253 (3), 570–583.

[16] CLAUTIAUX F., CARLIER J., MOUKRIM A., *A new exact method for the two-dimensional orthogonal packing problem*, Eur. J. Oper. Res., 2007, 183 (3), 1196–1211.

[17] DALALAH D., KHRAIS S., BATAINEH K., *Waste minimization in irregular stock cutting*, J. Manuf. Syst., 2014, 33, 27–40.

[18] DEL VALLE A.M., DE QUEIROZ T.A., MIYAZAWA F.K., XAVIER E.C., *Heuristics for two-dimensional knapsack and cutting stock problems with items of irregular shape*, Expert Syst. Appl., 2012, 39 (16), 12589–12598.

[19] ELKERAN A., *A new approach for sheet nesting problem using guided cuckoo search and pairwise clustering*, Eur. J. Oper. Res., 2013, 231, 757–769.

[20] FEKETE S.P., SCHEPERS J., *A new exact algorithm for general orthogonal d-dimensional knapsack problems*, Lect. Notes Comput. Sci., 1997, 1284, 144–156.

[21] FERREIRA J.C., ALVES J.C., ALBUQUERQUE C., OLIVEIRA J.F., FERREIRA J.S., MATOS J.S., *A flexible custom computing machine for nesting problems*, Proc. 13th DCIS, Madrid, Spain, 1998.

[22] FISCHETTI M., LUZZI I., *Mixed-integer programming models for nesting problems*, J. Heur., 2009, 15, 201–226.

[23] GILMORE P.C., GOMORY R.E., *The theory and computation of knapsack functions*, Oper. Res., 1966, 14 (6), 1045–1074.

[24] GOMES A.M., OLIVEIRA J.F., *A 2-exchange heuristic for nesting problems*, Eur. J. Oper. Res., 2002, 141 (2), 359–370.

[25] GOMES A.M., OLIVEIRA J.F., *Solving irregular strip packing problems by hybridising simulated annealing and linear programming*, Eur. J. Oper. Res., 2006, 171 (3), 811–829.

[26] GONÇALVES J.F., *A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem*, Eur. J. Oper. Res., 2007, 183, 1212–1229.

[27] HADJICONSTANTINOU E., CHRISTOFIDES N., *An exact algorithm for general, orthogonal, two-dimensional knapsack problems*, Eur. J. Oper. Res., 1995, 83, 39–56.

[28] HADJICONSTANTINOU E., IORI M., *A hybrid genetic algorithm for the two-dimensional single large object placement problem*, Eur. J. Oper. Res., 2007, 183, 1150–1166.

[29] JONES D.R., *A fully general, exact algorithm for nesting irregular shapes*, J. Global Optim., 2014, 59, 367–404.

[30] KIERKOSZ I., ŁUCZAK M., *A hybrid evolutionary algorithm for the two-dimensional packing problem*, Cent. Eur. J. Oper. Res., 2014, 22 (4), 729–753.

[31] LEAO A.A.S., TOLEDO F.M.B., OLIVEIRA J.F., CARRAVILLA M.A., *A semi-continuous MIP model for the irregular strip packing problem*, Int. J. Prod. Res., 2016, 54 (3), 712–721.

[32] LEUNG T.W., CHAN C.K., TROUTT M.D., *Application of a mixed simulated annealing-genetic algorithm heuristic for the two-dimensional orthogonal packing problem*, Eur. J. Oper. Res., 2003, 145 (3), 530–542.

[33] LÓPEZ-CAMACHO E., OCHOA G., TERASHIMA-MARIN H., BURKE E.K., *An effective heuristic for the two--dimensional irregular bin packing problem*, Ann. Oper. Res., 2013, 206, 241–264.

[34] LÓPEZ-CAMACHO E., TERASHIMA-MARIN H., ROSS P., OCHOA G., *A unified hyper-heuristic framework for solving packing problems*, Expert Syst. Appl., 2014, 41 (15), 6876–6889.

[35] MARTINEZ-SYKORA A., ALVAREZ-VALDES R., BENNELL J.A., RUIZ R., TAMARIT J.M., *Matheuristics for the irregular bin packing problem with free rotations*, Eur. J. Oper. Res., 2017, 258 (2), 440–455.

[36] MARTINS T.C., TSUZUKI M.S.G., *Simulated annealing applied to the irregular rotational placement of shapes over containers with fixed dimensions*, Exp. Syst. Appl., 2010, 37 (3), 1955–1972.

[37] MUNDIM L.R, ANDRETTA M., CARRAVILLA M.A., OLIVEIRA J.F., *A general heuristic for two-dimensional nesting problems with limited-size containers*, Int. J. Prod. Res., 2018, 56 (1–2), 709–732.

[38] OLIVEIRA J., FERREIRA J., *Algorithms for nesting problems. Applied simulated annealing*, [in:] R. Vidal (Ed.), *Lecture Notes in Economics and Maths Systems*, Vol. 396, Springer-Verlag, 1993, 255–274.

[39] OLIVEIRA J.F., GOMES A.M., FERREIRA J.S., *TOPOS. A new constructive algorithm for nesting problems, OR Spektrum*, Vol. 22, Springer-Verlag, 2000, 263–284.

[40] PREPARATA F.P., SHAMOS M.I., *Computational geometry. An introduction*, Springer-Verlag, 1985.

[41] RAMESH BABU A., RAMESH BABU N., *A generic approach for nesting of 2-d parts in 2-d sheets using genetic and heuristic algorithms*, Comp.-Aided Des., 2001, 33, 879–891.

[42] ROCHA P., RODRIGUES R., GOMES A.M., TOLEDO F.M.B., ANDRETTA M., *Circle covering representation for nesting problems with continuous rotations*, Proc 19th IFAC World Congress, 2014, 5235–5240.

[43] SONG X., BENNELL J.A., *Column generation and sequential heuristic procedure for solving an irregular shape cutting stock problem*, J. Oper. Res. Soc., 2014, 65 (7), 1037–1052.

[44] STOYAN Y., TERNO J., SCHEITHAUER G., GIL N., ROMANOVA T., *Phi-functions for primary 2d-objects*, Studia Inf. Univ., 2001, 2, 1–32.

[45] TERASHIMA-MARÍN H., ROSS P., FARÍAS-ZÁRATE C.J., LÓPEZ-CAMACHO E., VALENZUELA-RENDÓN M., *Generalized hyper-heuristics for solving 2d regular and irregular packing problems*, Ann. Oper. Res., 2010, 179, 369–392.

[46] TOLEDO F.M., CARRAVILLA M.A., RIBEIRO C., OLIVEIRA J.F., GOMES A.M., *The dotted-board model. A new MIP model for nesting irregular shapes*, Int. J. Prod. Econ., 2013, 145 (2), 478–487.

[47] WÄSCHER G., HAUSSNER H., SCHUMANN H., *An improved typology of cutting and packing problems*, Eur. J. Oper. Res., 2007, 183 (3), 1109–1130.

[48] YANG Q., *No fit polygon for nesting problem solving with hybridizing ant algorithms*, J. Softw. Eng. Appl., 2014, 7 (5), 433–439.

Received 14 June 2018
Accepted 5 February 2019