# Matheuristics for the Irregular Bin Packing Problem with free rotations

**A. Martinez-Sykora[a], R. Alvarez-Valdes[b], J. Bennell[a], R. Ruiz[c], J.M. Tamarit[b]**

*a* University of Southampton, Southampton, UK
J.A.Bennell@soton.ac.uk; A.Martinez-Sykora@soton.ac.uk

*b* Dept. of Statistics and Operations Research, University of Valencia, Doctor Moliner 50, 46100 Burjassot, Spain
ramon.alvarez@uv.es; jose.tamarit@uv.es

*c* Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edificio 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain.
rruiz@eio.upv.es

## Abstract

We present a number of variants of a constructive algorithm able to solve a wide variety of variants of the Two-Dimensional Irregular Bin Packing Problem (2DIBPP). The aim of the 2DIBPP is to pack a set of irregular pieces, which may have concavities, into stock sheets (bins) with fixed dimensions in such a way that the utilization is maximized. This problem is inspired by a real application from a ceramic company in Spain. In addition, this problem arises in other industries such as the garment industry or ship building. The constructive procedure presented in this paper allows both free orientation for the pieces, as in the case of the ceramic industry, or a finite set of orientations as in the case of the garment industry. We explicitly model the assignment of pieces to bins and compare with the more common strategy of packing bins sequentially. There are very few papers in the literature that address the bin packing problem with irregular pieces and to our knowledge this is the first to additionally consider free rotation of pieces with bin packing. We propose several Integer Programming models to determine the association between pieces and bins and then we use a Mixed Integer Programming model for placing the pieces into the bins. The computational results show that the algorithm obtains high quality results in sets of instances with different properties. We have used both industry data and the available data in the literature of 2D irregular strip packing and bin packing problems.

***Keywords***: Cutting and packing; Two-Dimensional irregular bin packing; Integer Programming.

# 1 Introduction

The paper addresses a real and highly relevant problem in industry that up to now has received minimal attention by the cutting and packing research community. This is the two-dimensional bin packing problem with irregular pieces (2DIBPP). The particular characteristics of the problem are: the shapes to be cut are irregular with concavities, the material being cut is homogenous therefore continuous rotation of the pieces are permitted, and a typical demand set of pieces require more than one stock sheet to be satisfied. While two-dimensional (2D) irregular packing problems have been extensively researched, these are almost exclusively strip packing problems. Further, only a few papers consider continuous rotation of pieces and to our knowledge there are no publications that examine problems that combine strongly heterogeneous irregular pieces, multiple stock sheets and continuous rotation. Each of these features adds complexity to the problem. While simple adaptations of known algorithms for the related simpler problem can be used, it is highly likely that these will yield sub-optimal results. Hence in this paper we seek to design an algorithm that considers and optimises the complete problem. The particular contributions of the paper are four fold. This is the first paper to tackle the problem of bin packing irregular shapes with unrestricted rotations, and one of very few that considers pieces with concavities within the bin packing problem. We investigate some methodological design principles, in particular, the value of separating the assignment decision from the packing model versus assignment as a consequence of the packing decision. We also investigate the relative impact of assignment and packing depending on the data instance. We develop new data instances for the irregular bin packing problem based on the benchmark instances of the strip packing problem. Finally, we investigate the bias introduced in data set design with respect to the piece orientation and show how using unrestricted rotation can address this bias.

A Spanish company in the ceramic tile sector, Butech Building Technology, by PORCELANOSA Group, has developed a new solid surface which, due to its properties of strength, durability, and color stability, can be used

for ventilated facades, among many other applications. The material, whose commercial name is Krion®, is produced in plates of 366.6 by 75 centimeters. A particular property of this product is the possibility to cut it into pieces of any polygonal shape, convex or not, according to the architect's design. This is a significant departure from the classical tiles used in ventilated facades, which are always of rectangular shape. The problem is, then, to determine how the demanded pieces are cut from the plates so as to minimize the number of required plates. As this new material is expensive and the number of pieces in a project can be very large, reducing the number of plates to a minimum can produce substantial savings. At the time of undertaking this research, the company manually designed the cutting patterns, where each order could take up to four person weeks to design the cutting pattern.

Although the present study has been motivated by this particular company, there are many other companies in which materials coming in sheets of given sizes have to be cut into smaller pieces with irregular shapes. Therefore, the ideas developed here could be applied to other situations, such as the garment industry, ship building, or the glass industry (Han et al., 2012; Martinez-Sykora et al., 2015), maybe adapting them to the special features of each case.

The above cutting problem is a two-dimensional bin packing problem with irregular (non-convex) pieces (2DIBPP). According to the typology proposed by Wäscher et al. (2007), the 2DIBPP problem can be classified as a two-dimensional Single Bin Size Bin Packing Problem (SBSBPP) with the refinement that the pieces have irregular shape. In the case of the ceramic cutting, there are no restrictions on the angle of rotation, making the problem more difficult to solve efficiently. Further, the pieces cannot be reflected because each side of the plate has a different texture and only one side can face outwards. The objective is to minimize the total number of bins needed to cut all the pieces.

The most studied packing problem involving irregular shapes that can be found in the literature is the two-dimensional strip irregular packing problem, also known as the Nesting Problem (see Bennell and Oliveira (2009) for a survey). There are few publications considering the bin packing problem with irregular pieces and, to our knowledge, there is no publication that additionally allows items to be continuously rotated. Chernov et al (2010) is one example of a number of publications by these authors that permit continuous rotation of irregular pieces. Here the objective is to minimize the area of the single containing object. An interesting application of bin packing problems with irregular pieces arises in the glass industry. In this case pieces can be placed in any orientation, however, since the cutting process forces the use of guillotine cuts, the pieces have to be convex. Han et al. (2012) propose two constructive heuristics and report results on some real data. More recently, Martinez-Sykora et al. (2015) present several MIP-based constructive procedures in which they improve the best known solutions in all the instances and, furthermore, they obtain competitive results on rectangular bin packing problems.

Terashima-Marin et al. (2010) propose a hyper-heuristic algorithm for the 2DIBPP, which combines several placement heuristics. They combine the Bottom-Left algorithm developed by Jakobs (1996) and an improved version proposed by Liu and Teng (1999). For both procedures Terashima-Marin et al. (2010) add the possibility of rotating the pieces by a finite set of angles. Furthermore, they include different modifications of the constructive approach presented by Hifi and M'Hallah (2002). The hyper-heuristic is embedded into a Genetic Algorithm and it is tested on jigsaw puzzle instances where all the pieces are convex. Lopez-Camacho et al. (2013) adapt the Djang and Finch heuristic (DJD), originally proposed for the one-dimension bin packing problems by Ross et al. (2002), to the two-dimensional irregular bin packing problem. Despite the fact the algorithm proposed by Lopez-Camacho et al. (2013) consider non-convex pieces, in the computational experiments they only report results involving pieces with convex shapes. A later paper, Lopez-Camacho et al. (2014), claim to have solved the non-convex instances, but only report the percentage of problems solved with more, the same, or less bins when compared with the problems convex counterpart. In all these approaches, the assignment of the piece to a bin is incidental to the decision of where to place the piece in the bin.

Another study that considers multiple bins and irregular pieces is that of Song and Bennell (2013), who propose a column generation procedure for solving the irregular shape cutting stock problem. The authors use the beam search algorithm proposed in Bennell and Song (2010) for generating the patterns and study three solution approaches: column generation, adapted column generation and a sequential heuristic procedure. In the computational experiments the authors build eleven instances obtained from well-known 2D irregular strip packing instances by fixing the bin length and multiplying the demand of each shape by 100. While the instances include non-convex pieces, the problem is a cutting stock problem where patterns are repeated multiple times, which is somewhat different to the two-dimensional bin packing problem considered in our study.

This paper describes a constructive algorithm that explicitly considers the two aspects of the optimisation problem; the assignment of pieces to bins and the arrangement of assigned pieces in the bin. We implement several strategies for assigning pieces to the bins, including solving IP models for the one-dimensional Bin Packing

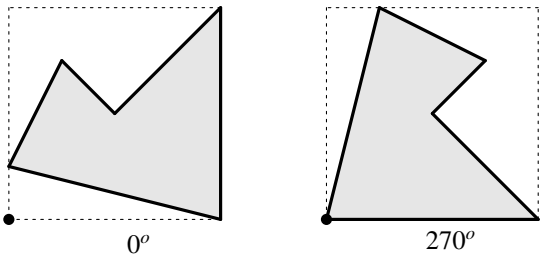



**Figure 1:** *Reference point of a piece in two different orientations*

Problem and knapsack problem and heuristic methods. In a second phase, the subset of pieces assigned to a given bin are sequentially placed in the new bin using a packing procedure. This phase consists of two steps. First, we determine a set of promising rotations for the piece and then, for each rotation, we determine if the piece fits into the bin by solving a Mixed Integer Programming (MIP) model. The packing procedure permits all pieces to move within the bin to accommodate the next piece. Since the assignment is based on piece area, often the packing fails. Hence, we propose some alternative strategies for reassigning pieces. All these approaches are compared to the more common approach of deciding the bin assignment as part of the piece placement strategy.

The paper is organised as follows: in Section 2 we define the problem and some notation. In Section 3 we describe the construction algorithms that determine the assignment of pieces to bins and in Section 4 we provide the formulation for the placement of items into each bin. Section 5 contains two local search procedures designed to improve the solutions obtained. In Section 6 we describe the implementation of the constructive algorithm, including any parameters that need to be set and a full description of the data instances. In Section 7 we present the computational results and in the final section we summarise our conclusions.

# 2 Problem description

Let $P = \{1, \ldots, n\}$ be the set of $n$ pieces, represented as simple polygons, to be placed into identical rectangular bins. We denote the area of piece $k \in P$ by $a_k$. The number of available bins can be considered large enough to pack all the pieces and the objective is to minimize the total number of bins needed to cut all the pieces in $P$. The width and length of the bins are denoted by $W$ and $L$ respectively. The pieces can be rotated continuously and must be placed entirely within the bin and may not overlap with other pieces.

A solution to the described problem is given by a set of bins $B = \{b_1, \ldots, b_N\}$. Each bin $b_i(P_i, O_i, X_i, Y_i)$, $\forall i = 1, \ldots, N$, is composed of a set of pieces $P_i \subseteq P$, a vector $O_i = \{o_1, \ldots, o_{n_i}\}$ defines the orientation of the pieces, where $n_i = |P_i|$ and for a given piece $k$, $k = 1, \ldots, n_i$, $o_k \in [0, 2\pi]$, and the two vectors, $X_i$ and $Y_i$, are the $X$-coordinate and $Y$-coordinate of the reference point of each piece. We define the reference point of the piece $k \in P$ as the bottom left corner of the enclosing rectangle of the piece when in orientation $o_k$. Hence, the position of the reference point, relative to the vertices of the piece, changes according to the orientation, as illustrated in Figure 1. In addition, we consider that the bottom-left corner of each bin is placed at (0,0), so that the coordinates of the reference points are always positive. Note that, since all the pieces have to be placed, then $P = \cup_{i=1}^{N} P_i$.

The stated objective is to minimize the total number of bins. Applying this objective directly will result in many tied solutions making the solution space difficult to navigate, further in many applications, the residual material can be reused provided it is sufficiently large. Lopez-Camacho et al. (2013) propose an alternative measure of performance that maximises the percentage usage of each bin as follows:

$$F = \frac{\sum_{i=1}^{N} U_i^2}{N} \tag{1}$$

where $U_i$ is the utilization ratio of each bin $i \in 1 \ldots, N$, defined as:

$$U_i = \frac{\sum_{k=1}^{n_i} a_k}{LW}, \tag{2}$$

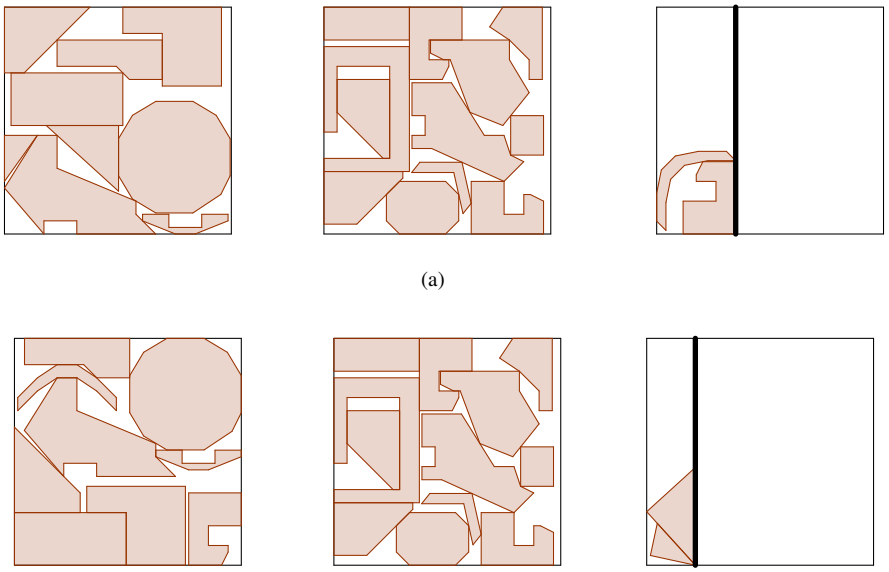
This measure avoids ties among different solutions with the same number of bins, and favours solutions with a high utilization in all but one or two bins over those that evenly distribute the pieces.

An alternative approach is used by Han et al. (2012). In order to account for the reuse of residual material, they apply a vertical or a horizontal cut to separate the non-utilized part of the bin for future use. The cut is applied to the least utilized bin only. We define the fractional number of bins as follows:

$$K = N - 1 + R^* \tag{3}$$

where $R^*$ is the proportion of the bin used to pack pieces after applying the vertical or horizontal cut. Figure 2 shows two different solution of the same instance (*Han* taken from ESICUP benchmark instance). Although both solutions use three bins, solution (b) is better since it has a smaller value of $K$.

We employ both these performance measures, $F$ and $K$, for comparing with the experimental results in addition to the number of bins. Also $F$ is used to guide the local search strategies.



(a)

(b)

**Figure 2:** *Two solutions of instance* han *(Nest-MB) (a)* $K = 2.35$*; (b)* $K = 2.20$.

# 3 Solution algorithm

The bin packing problem has two sets of distinct decision types. Specifically these are: the assignment of pieces to a bin and how to pack the assigned pieces in the bin. Many authors design solution approaches where the assignment decision is incidental to the packing decision, for example packing a bin until it is full, closing the bin, and then opening a new bin to continue packing (Parreño et al. , 2010). In this paper we investigate an alternative approach where the assignment decision is solved separately and prior to determining the layout of pieces in the bin. Inevitably the assignment will often be unfeasible and one or more pieces can not be placed in their allotted bin, hence we investigate alternative strategies for mending the assignment. In this section we describe alternative strategies for assigning to bins and mending the assignment when pieces can not be feasibly packed. The packing procedure is described in Section 4, which packs the assigned pieces to a given bin sequentially in order of non-increasing area using a MIP model. Finally we investigate the benefit of employing local search after the construction of the solution, this is described in Section 5.

The general framework for the procedure is organized as follows:

Step 1, solve a one-dimensional bin packing problem, where the pieces in $P$ are represented by their area, in order to assign all pieces to the minimum number of bins. Sort the bins in non-increasing order of utilization area. If $P$ is empty, stop.

Step 2, for the first bin on the list, use the MIP model described in Section 4 to arrange the assigned 2D irregular pieces into the bin. If not all the pieces fit into the bin, go to Step 3. If all the pieces can be placed in the bin with no overlap, then a feasible solution for the bin is found. Remove the bin from the list. Remove the pieces

packed in the bin from $P$. Store the packed bin to the final solution. If all pieces are packed, stop. Otherwise, repeat Step 2 for the next bin on the list.

Step 3, if all the pieces can not be placed into the bin, mend the assignment of pieces to the bin until a feasible arrangement is found. Remove the bin from the list. Remove pieces packed in the bin from $P$. Store the packed bin to the final solution. Since the original assignment has been disrupted, go to Step 1 to solve a 1D-BPP with the remaining pieces.

Section 3.1 to 3.5 describe alternative procedures for solving the assignment of pieces to bins, including strategies for reassignment when the assignment is found to not be feasible. We also describe a simple construction heuristic that does not preassign pieces to bins.

## 3.1 Bin Packing with Greedy Decisions (BPGD)

The BPGD strategy assigns pieces to bins by solving an IP formulation of the one-dimensional bin packing problem (1DBBP) described below. Since area is a weak approximation of a simple polygon, particularly when concavities are present, the packing procedure, which decides how to arranged the 2D irregular pieces, often fails to place one or more of the assigned pieces inside the bin. While it is possible to solve the entire assignment problem again with constraints to prevent the failed bin being reproduced, there is a high chance the new assignment would result in other infeasible bins. Reaching a feasible packing through this approach would require solving many IP models with an increasing number of constraints leading to a very high computational cost. Hence a faster strategy is to retain the bin that has a feasible arrangement of some of the assigned pieces and try to improve the utilization of the bin by packing additional pieces that have been assigned to other bins. This is the approach taken by our BPGD algorithm.

The IDBBP model is defined as follows: Let each piece be approximated by its area, $a_i$. $\overline{N}$ is an upper bound on the total number of bins required to pack all the pieces, obtained by using the First Fit Decreasing (FFD) algorithm (Johnson et al. (1974)). We define the following binary variables: $s_i$, $\forall i = 1, \dots \overline{N}$, which take value 1 if bin $b_i$ is used in the solution and 0 otherwise, and $z_{ij}$, $i = 1, \dots, \overline{N}$, $j = 1, \dots, n$, which take value 1 if piece $j$ is assigned to bin $i$ and 0 otherwise. An integer programming model for assigning pieces to bins can be formulated in a standard way as a 1DBPP as follows:

$$Min.\ \sum_{i=1}^{\overline{N}} s_i, \tag{4}$$

$$\sum_{j=1}^{n} a_j z_{ij} \leq LW, \qquad 1 \leq i \leq \overline{N}, \tag{5}$$

$$\sum_{i=1}^{\overline{N}} z_{ij} = 1, \qquad 1 \leq j \leq n, \tag{6}$$

$$z_{ij} \leq s_i, \qquad 1 \leq j \leq n,\ 1 \leq i \leq \overline{N}, \tag{7}$$

$$s_i \in \{0,1\},\ z_{ij} \in \{0,1\} \qquad 1 \leq j \leq n, 1 \leq i \leq \overline{N}. \tag{8}$$

The objective function minimizes the total number of bins used in the solution. Constraints (5) ensure that the total area of the pieces assigned to bin $b_i$ does not exceed the area of the bin. Equalities (6) force each piece to be assigned to one bin. Inequalities (7) guarantee that if any piece is placed into a given bin, then the corresponding bin is used in the solution. Finally, constraints (8) force the variables to be binary. Note that, if this IP model is solved to optimality and we pack all the pieces assigned to each bin successfully, the solution would be optimal.

Given the optimal solution to the IDBBP, the bins are sorted in non-increasing order of utilization. The packing procedure is applied to the first bin on this list, where the pieces assigned to the bin are sorted in non-increasing order of area. When trying to pack the pieces assigned to a bin, there are two possible outcomes:

a) The packing procedure finds a feasible placement for all the pieces. In this case the bin and the pieces placed in the bin are removed from further consideration and saved to the final solution. The algorithm will then go on to consider the new most utilized bin on the list and apply the packing procedure.

b) The packing procedure fails when placing a given piece, $j$. In this case the pieces packed in the bin so far are retained. Excluding piece $j$, all pieces that have not been packed (including those assigned to other bins) are sorted in non-increasing order. Next, the packing procedure tries to place each piece in turn in the partially packed bin, accepting each feasible placement. When all the pieces have been tried, the bin and the pieces placed in the bin are removed from further consideration and saved to the final solution. Since the assignment of pieces has been altered, we solve the assignment model again for the unpacked pieces.

The IP model has to be solved each time case (b) occurs because the set of remaining pieces has been modified. Hence the IP model is run at most $N$ times. Note that each time the IP model is solved at least one bin has been closed and the number of the remaining pieces has been reduced. Therefore $\overline{N}$ is updated and the number of binary variables is reduced.

## 3.2 First Fit Algorithm (FF)

An alternative to solving the 1DBPP to optimality using an IP model as described above, is the use of a known heuristic algorithm. Our FF algorithm follows this strategy and adopts the fast and well known First Fit Decreasing Algorithm (FFD) (Johnson et al. (1974)) to solve the 1DBPP. FFD takes an ordered list of pieces and assigns them sequentially to the bins. In order to assign a piece, FFD examines each bin in the order the bins were opened and places the piece in the first bin that can feasibly accommodate the piece. If the piece does not fit into any existing bin, a new bin is created and the piece is assigned to this new bin. This algorithm is very fast and depends critically on the initial ordering of the pieces. Therefore, in order to obtain a good solution we use the FFD assignment scheme, but starting from a unique random permutation of the pieces and repeat the process 5000 times. We keep the solution with the minimum number of bins, breaking ties by the minimum used area of the least utilized bin, and sort the bins in non-increasing order of utilization.

Similar to BPGD, the packing procedure is applied to the first bin on this list. If it finds a feasible placement for all the pieces, the bin and the pieces placed in the bin are saved to the final solution and the algorithm will go on to consider the new most utilized bin on the list.

If the packing procedure fails when placing a given piece, $j$, it stops and all pieces that have not been packed yet are reassigned to bins using FFD, where the partially packed bin is the first open bin and piece $j$ can not be assigned to it. Given the new assignment, the packing procedure tries to place the newly assigned pieces into the first bin. The packing and re-assigning continues until a feasible bin is obtained.

## 3.3 Partial Bin Packing (PBP)

The PBP strategy is motivated by the observation that we frequently change the assignment across the bins to obtain a feasible solution. It also recognizes that assigning across the bins prevents the algorithm being too greedy with respect to the bin under consideration. It is well documented in cutting and packing literature that a greedy solution packs small pieces together early on in the solution construction, leaving the larger more difficult pieces to the end, leading to overall poor solutions. Therefore, the PBP approach avoids reassignment by focusing on the assignment of the pieces to just one bin, but uses an objective function that favours assigning larger pieces. This strategy has been widely used in one-dimensional BPP (see Caprara and Pferschy (2004) for a review and a worst-case analysis).

The assignment approach is a knapsack formulation, where the objective is to maximise the value of the bin by packing some of the available pieces. The following formulation sets the binary variables, $q_j$, $j = 1 \ldots, n$, to take the value 1 when piece $j$ is assigned to the bin and 0 otherwise and $a_{max}$ is the area of the biggest piece. The IP model can be formulated as follows:

$$Max. \ \sum_{j=1}^{n} \left(\frac{a_j}{a_{max}}\right)^2 q_j, \tag{9}$$

$$\sum_{j=1}^{n} a_j q_j \leq LW, \qquad 1 \leq j \leq n, \tag{10}$$

$$q_j \in \{0, 1\}, \qquad 1 \leq j \leq n. \tag{11}$$

As with the above two strategies, the packing procedure may succeed or fail to feasibly pack the bin. If it finds a feasible placement for all the pieces, the packed bin is saved to the final solution and the algorithm will then solve the knapsack formulation with the remaining pieces to generate a new bin.

On the contrary, if the packing procedure fails when placing a given piece, $j$, the knapsack formulation is solved again for the remaining capacity, with $j$ removed from consideration. Given the new assignment, the packing procedure tries to place the newly assigned pieces into the bin. The packing and re-assigning continues until a feasible bin is obtained. Piece(s) $j$ temporarily removed from consideration are returned to the available pool of pieces and the algorithm will then solve the knapsack formulation to generate a new bin.

### 3.4 Two Phases Strategy (TPS)

The TPS strategy aims to enhance the BPGD strategy by attempting to distribute the small pieces across the bins during the assignment phase, as in PBP strategy. While the solutions obtained by the BPGD procedure provides the optimal number of bins required to pack all the pieces (as a 1D problem), the solution does not consider how the small and large pieces are distributed. Therefore for the TPS strategy, after solving the assignment model proposed in the BPGD strategy to find the minimum number of bins, $N'$, we solve a second IP model to reassign pieces across bins. Let $z_{ij}$, $i = 1, \ldots, N'$, $j = 1, \ldots, n$ be the binary variables which take value 1 if piece $j$ is associated with bin $i$.

$$Min. \ \sum_{i=1}^{N} \sum_{j=1}^{n} (\frac{a_j}{a_{max}})^2 \frac{2(N'-i)}{N'^2+N'} \ z_{ij}, \tag{12}$$

$$\sum_{j=1}^{n} a_j z_{ij} \leq LW, \qquad 1 \leq i \leq N', \tag{13}$$

$$\sum_{i=1}^{N'} z_{ij} = 1, \qquad 1 \leq j \leq n, \tag{14}$$

$$z_{ij} \in \{0, 1\}, \qquad 1 \leq i \leq N', \ 1 \leq j \leq n. \tag{15}$$

The objective function (12) forces the assignment of large pieces into the earlier bins while minimising the total number of bins. Note that $(\frac{a_j}{a_{max}})^2$ takes a greater value when $p_j$ has more area and $\frac{2(N'-i)}{N'^2+N'}$ takes a greater value for the earlier bins. The constraints (13) and (14) ensure, respectively, that the pieces associated to one bin does not exceed the area of the bin and that each piece is assigned to one bin.

Once the pieces have been assigned by sequentially solving both IP models, the procedure to pack and reassign pieces is the same as the BPGD strategy.

### 3.5 Simple construction heuristic (SCH)

The SCH strategy is included to benchmark our approach against the commonly used approach of deciding the bin allocation and placement position together. SCH directly packs pieces into a bin in a given sorted order following a Next-Fit Decreasing strategy. The pieces are packed into the bins using the procedure described in Section 4. The structure is as follows:

1) Sort all the pieces in order of non-decreasing area, open bin, $i = 1$

2) For next piece $i$, solve IP model described in Section 4 to pack piece $i$ in the open bin.

3) If a feasible solution is found, $i = i + 1$, if $i > n$, stop, else got to 2. otherwise close bin and open new bin, go to 2.

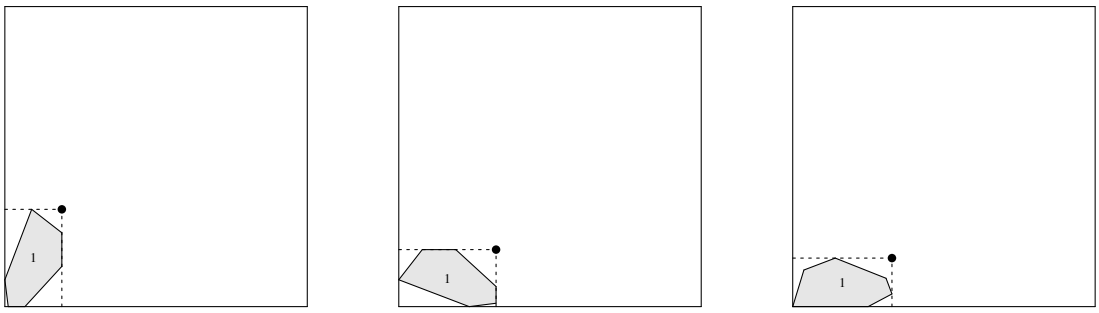
## 4 Packing model

In this section we describe the model that determines the exact location and orientation of the pieces in a bin. We assume that a set of pieces $P_b \subseteq P$, ordered by non-increasing area, has been assigned to the current bin, $b$. The MIP model places one piece in the bin in order to minimise the weighted rectangle of the partial solution. Although, the pieces can be rotated continuously, but the MIP we use to pack the pieces requires that each piece has a fixed orientation. As a result, the MIP model is run several times for each piece solving a model for each orientation we want to consider. This idea was first proposed in Martinez-Sykora et al. (2015).

In order to obtain promising rotations for placing the next piece, we consider the edges of the pieces already placed in the bin and the edges of the bin. We calculate the angles of rotation in such a way that one edge of the piece being placed matches with either one edge of a piece already placed or one edge of the bin. If we obtain more than three different rotations, we sort the angles by the following criteria (GR):

- Non-increasing number of matches between the edges of the polygon obtained by applying the given rotation to the piece, and the edges of all the pieces already placed in the bin and the edges of the bin.
- Ties are broken by the total length of the edges of all the matchings.

If the number of rotations is not specified, we consider the first three rotations for the insertion of each piece. If in a given instance, only some fixed orientations are allowed, we use these orientations.



(d) Rotation 1 (e) Rotation 2 (f) Rotation 3

**Figure 3:** *Three rotations for the first piece (instance poly3a-x2). Rotation* 3 *is selected and fixed*

In what follows we give a description of the MIP model. For the sake of clarity, we first describe the model for the insertion of the first piece, then the second piece and finally the general case. The placement model for the first piece has a linear programming formulation, in which the variables $x_1$ and $y_1$ correspond to the coordinates of the reference point of piece 1:

$$Min.\ \omega L_u + (1-\omega)W_u, \tag{16}$$

$$L_u \leq L, \tag{17}$$

$$W_u \leq W, \tag{18}$$

$$x_1 \leq L_u, \tag{19}$$

$$y_1 \leq W_u. \tag{20}$$

$L_u$ and $W_u$ denote the used length and width of the bin when placing the first piece and $\omega$ is a coefficient which depends on the dimensions of the bin. In order to balance the used length and width when adding pieces we fix $\omega = \frac{1}{(W/L)+1}$. Therefore, with this objective function the pieces will be placed in such a way that the dimensions of the enclosing rectangle are proportional to the dimensions of the bin. Constraints (17) and (18) ensure that the used length and width do not exceed the dimensions of the bin and constraints (19) and (20) force piece 1 to be inside the enclosing rectangle $(L_u, W_u)$.

We solve this linear programming formulation once for each different orientation we consider for the piece. In this step we determine and fix the orientation of the piece for which the objective function is minimized. In Figure 3, the first two rotations are considered because two edges of the piece are parallel to the edges of the bin. The values of $x_1$ and $y_1$ given by the LP solution are not fixed in the following models but the orientation is. The third rotation, which happens to be the best, is considered because the largest edge of the piece is parallel to one edge of the bin.

When inserting the second and later pieces, the main difficulty is to ensure that they do not overlap. We use the *MIP* model, proposed by Alvarez-Valdes et al. (2013), in which non-overlapping constraints are obtained using *No-Fit Polygons (NFP)*. The *NFP* is a polygon derived from every pair of pieces in such a way that its interior represents all the overlapping positions between the pieces, and its boundary represents all the touching positions. Figure 4 gives an example of the NFP, corresponding to pieces 1 and 2 ($NFP_{12}$), where the orientation of piece 1 is fixed in the previous step and piece 2 was the same shape as piece 1 and is rotated $90^0$. Deriving the NFP is not the topic of this paper, see Bennell and Oliveira (2008) for a review of procedures. Given the correct placement of piece 1, the reference point of piece 2 must be placed outside $NFP_{12}$ to avoid overlapping. Alvarez-Valdes et al. (2013), following the idea proposed by Fischetti and Luzzi (2008), divide the outer region into horizontal slices and define a binary variable for each slice so that it takes value 1 if the reference point is in this slice and 0 otherwise.

In the example in Figure 4 we need 13 binary variables. The notation we use to write the formulation is a three-index notation in which the two first indexes represent the pieces, and the third index enumerates the slices (for the sake of clarity in Figure 4 we show only the third index). In general, if pieces have complex shapes, the set of points not belonging to the NFP may have a structure more complex than that of the Figure 4 and it might be needed to divide concave shapes into convex shapes. In these cases, before defining the slices and their associated

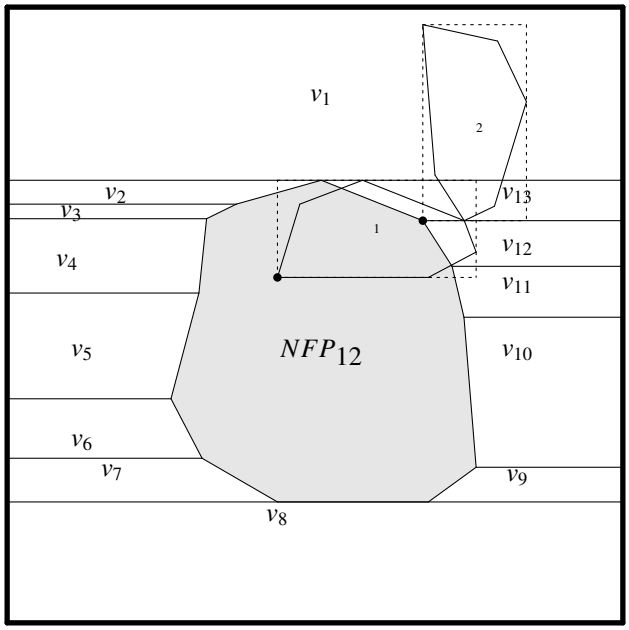



**Figure 4:** $NFP_{12}$

variables, a preprocess is required and additional binary variables are associated with the subsets of the region not included in the slices (see Alvarez-Valdes et al. (2013) for a complete discussion).

Let $m_{12}$ be the total number of binary variables, i.e, the number of slices we derived from $NFP_{12}$ ($m_{12} = 13$ in Figure 4). We denote by $t_{12i}$ the total number of inequalities required to describe slice $S^i$ associated with $v_{12i}$, $i = 1, \ldots, m_{12}$. Despite the fact that in Figure 4 $t_{12i} = 4$, $\forall i = 1, \ldots, m_{12}$, it might be possible that $t_{12i}$ takes different values if the $NFP_{12}$ has a more complex shape. In what follows we describe first the non-overlapping constraints and then the containment constraints.

If, for some slice $k$ of the $NFP_{12}$, $v_{12k} = 1$, the constraints defined by the sides of the slice:

$$\alpha_{12}^{kf}(x_2 - x_1) + \beta_{12}^{kf}(y_2 - y_1) \leq \delta_{12}^{kf} \qquad f = 1, \ldots, t_{12k} \tag{21}$$

where $\alpha_{12}^{kf}$ and $\beta_{12}^{kf}$ are the coefficients defining the slope of the inequality and $\delta_{12}^{kf}$ the intercept, must be satisfied. However, if $v_{12k} = 0$, these constraints must be relaxed. This can be done by using big-$M$ constants:

$$\alpha_{12}^{kf}(x_2 - x_1) + \beta_{12}^{kf}(y_2 - y_1) \leq \delta_{12}^{kf} + (1 - v_{12k})M \qquad f = 1, \ldots, t_{12k} \tag{22}$$

Fischetti and Luzzi (2008) describe a procedure for building these constraints avoiding the big-$M$ constants. Since exactly one slice is used in any feasible solution, this equality holds:

$$\sum_{i=1}^{m_{12}} v_{12i} = 1 \tag{23}$$

Then a tighter constant could be written for each slice, eliminating the big-$M$ and including on the right hand side of the inequality all the binary variables, defined by $NFP_{12}$, multiplied by a given constant. These constants are obtained by solving the following problem:

$$\delta_{12}^{kfh} = max_{p_2 \in S^h} \ \alpha_{12}^{kf}(x_2 - x_1) + \beta_{12}^{kf}(y_2 - y_1) \tag{24}$$

which corresponds to the maximum value of the left hand side when piece 2 lies on slice $h$, $h \in \{1, \ldots, m_{12}\}$.

Then, the non-overlapping inequalities can be written as follows:

$$\alpha_{12}^{kf}(x_2 - x_1) + \beta_{12}^{kf}(y_2 - y_1) \leq \sum_{h=1}^{m_{12}} \delta_{12}^{kfh} v_{12h} \qquad k = 1, \ldots, m_{12}, f = 1, \ldots, t_{12k} \tag{25}$$

The containment constraints ensure that each piece does not exceed the limits of the bin (as with inequalities (19) and (20) in the formulation for one piece). Alvarez-Valdes et al. (2013) describe a lifting procedure for these inequalities, adding some of the binary variables of the corresponding $NFP$, associated with slices on which one piece protrudes either horizontally or vertically from the other. Let us denote by $\overline{y}_{12k}$ and $\underline{y}_{12k}$ ($\overline{x}_{12k}$ and $\underline{x}_{12k}$) the maximum and minimum value of $y_2 - y_1$ (respectively, $x_2 - x_1$), when slice $k$ is used. Let $\underline{X}^{12}$ and $\underline{Y}^{12}$ be the minimum $X$-coordinate and $Y$-coordinate value of $NFP_{12}$ respectively. We define the following classification of the binary variables:

- $U_{12} := \{v_{12k} \mid \underline{y}_{12k} \geq 0\}$, the set of binary variables whose associated slices do not allow piece 2 to protrude from below piece 1. In Figure 4, $U_{12} = \{v_1, v_2, v_3, v_{12}\}$.
- $R_{12} := \{v_{12k} \mid \underline{x}_{12k} \geq 0\}$, the set of binary variables whose associated slices do not allow piece 2 to protrude from the left of piece 1. In Figure 4, $R_{12} = \{v_9, v_{10}, v_{11}, v_{12}\}$.
- $LS_{12} := \{v_{12k} \mid \lambda_{12k} > 0\}$ where $\lambda_{12k} = l_1 - (\overline{x}_{12k} - \underline{X}^{12})$ and $l_1$ is the length of piece 1 in the current orientation. Then, $LS_{12}$ is the set of binary variables whose associated slices force piece 1 to protrude from the right of piece 2. In Figure 4, $LS_{12} = \{v_2, v_3, v_4, v_5, v_6, v_7\}$.
- $DS_{12} := \{v_{12k} \mid \mu_{12k} > 0\}$ where $\mu_{12k} = w_1 - (\overline{y}_{12k} - \underline{Y}^{12})$ and $w_1$ denotes the width of piece 1. Then, $DS_{12}$ is the set of binary variables whose associated slices force piece 1 to protrude from above piece 2. In Figure 4, $DS_{12} = \{v_7, v_8, v_9\}$, variable $v_6$ is not in $DS_{12}$ because $\mu_{126} = 0$.

The containment constraints for piece 1 are:

$$\sum_{k \in R_{12}} \underline{x}_{12k} v_{12k} \leq x_1 \leq L_u - l_1 - \sum_{k \in LS_{12}} \lambda_{12k} v_{12k} \tag{26}$$

$$\sum_{k \in U_{12}} \underline{y}_{12k} v_{12k} \leq y_1 \leq W_u - w_1 - \sum_{k \in DS_{12}} \mu_{12k} v_{12k} \tag{27}$$

Similarly we can build the containment constraints for piece 2. In Figure 4, $U_{21} = \{v_5, v_6, v_7, v_8, v_9, v_{10}\}$, $R_{21} = \{v_2, v_3, v_4, v_5, v_6, v_7\}$, $LS_{21} = \{v_9, v_{10}, v_{11}, v_{12}\}$, and $DS_{21} = \{v_1, v_2, v_3, v_4, v_{11}, v_{12}\}$. The containment constraints for piece 2 can be written as follows:

$$\sum_{k \in R_{21}} \lambda_{21k} v_{12k} \leq x_2 \leq L_u - l_2 - \sum_{k \in LS_{21}} \underline{x}_{12k} v_{12k} \tag{28}$$

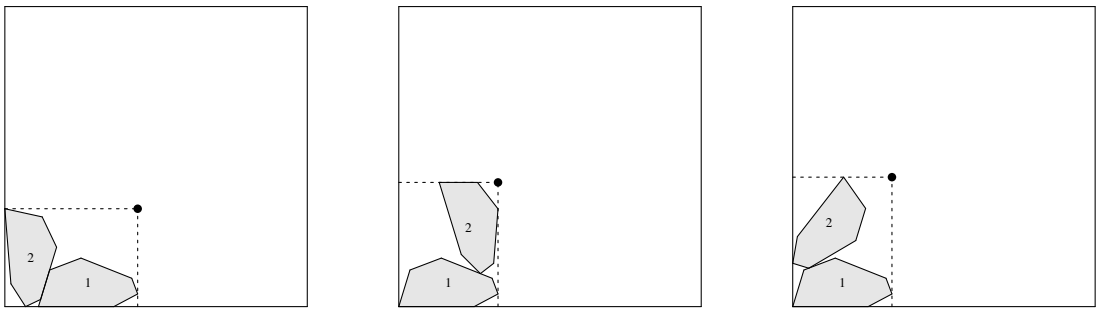
$$\sum_{k \in U_{21}} \overline{y}_{12k} v_{12k} \leq y_1 \leq W_u - w_2 - \sum_{k \in DS_{21}} \mu_{21k} v_{12k} \tag{29}$$

where $\lambda_{21k} = l_2 - (\overline{X}^{12} - \underline{x}_{12k})$ and $\mu_{21k} = w_2 - (\overline{Y}^{12} - \underline{y}_{12k})$.
The MIP formulation we consider to place pieces 1 and 2 has as objective function expression (16), subject to:

- Constraints (17) and (18) to ensure that $L_u$ and $W_u$ do not exceed the limits of the bin.
- Lifted containment constraints (26) and (27) for piece 1.
- Lifted containment constraints (28) and (29) for piece 2.
- Non-overlapping constraints (23) and (25).

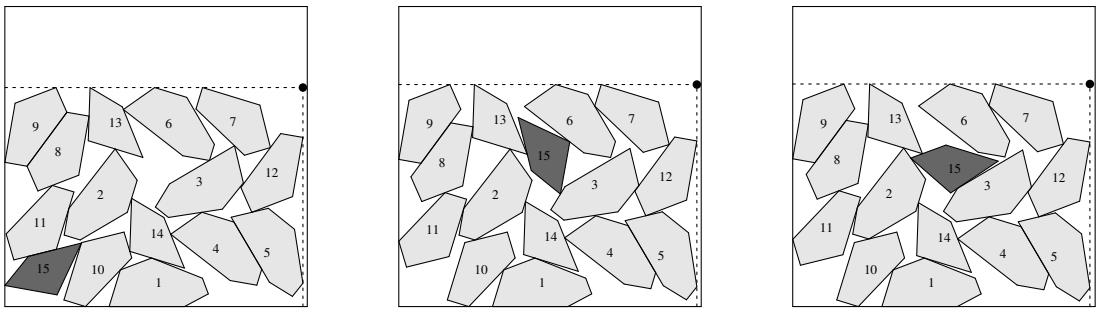
This MIP model is solved three times since we consider three orientations as illustrated in Figure 5. Each time we solve the model for a given rotation and we obtain an improved feasible solution, the value of its objective function is used as an upper bound for the following rotations. Therefore, only models producing improved solutions are solved to optimality. If the orientation being tested cannot produce a better solution, the model is quickly identified as unfeasible by the solver, speeding up the process. In the three solutions depicted in Figure 5 the orientation of piece 1 is fixed, as a result of the previous step, but its position is determined by the solution of the current model.

There are 13 binary variables in each one of the models solved for the insertion of the second piece. Since this number increases exponentially with the number of pieces already placed into the bin, it maybe necessary to fix the relative position between the pieces already placed to reduce this number. If the computational time needed to solve the MIP model to optimality exceeds a given threshold, $\theta$, then for the following insertion we fix the binary variables to the values obtained in the previous model. Otherwise, we allow more flexibility by keeping the binary variables free. Further each MIP model is given a maximum computation time.




(a) Rotation 1 (b) Rotation 2 (c) Rotation 3

**Figure 5:** *Three rotations for the second piece. Rotation* 3 *is selected and fixed*




(a) Rotation 1 (b) Rotation 2 (c) Rotation 3

**Figure 6:** *Three rotations for the 15$^{th}$ piece. Rotation* 1 *is selected and fixed*

For the general case, where the next piece to be inserted is piece $j$, $j-1$ pieces have been inserted and now have fixed orientation. The model for the insertion of $j$ can be written as follows:

$$Min.\ \omega L_u + (1-\omega)W_u, \tag{30}$$

$$L_u \leq L, \tag{31}$$

$$W_u \leq W, \tag{32}$$

$$\sum_{k\in R} \underline{x}_{j_1 j_2 k} v_{j_1 j_2 k} \leq x_{j_1} \leq L_u - l_{j_1} - \sum_{k\in LS_{j_1 j_2}} \lambda_{j_1 j_2 k} v_{12k}, \qquad j_1, j_2 \in \{1,\ldots,j\} \tag{33}$$

$$\sum_{k\in U_{j_1 j_2}} \underline{y}_{j_1 j_2 k} v_{j_1 j_2 k} \leq y_{j_1} \leq W_u - w_{j_1} - \sum_{k\in DS_{j_1 j_2}} \mu_{j_1 j_2 k} v_{j_1 j_2 k}, \qquad j_1, j_2 \in \{1,\ldots,j\} \tag{34}$$

$$\alpha^{kf}_{j_1 j_2}(x_{j_2} - x_{j_1}) + \beta^{kf}_{j_1 j_2}(y_{j_2} - y_{j_1}) \leq \sum_{h=1}^{m_{j_1 j_2}} \delta^{kfh}_{j_1 j_2} v_{j_1 j_2 h}, \qquad 1 \leq j_1 < j_2 \leq j, k = 1,\ldots,m_{j_1 j_2}, f = 1,\ldots,t_{j_1 j_2 k} \tag{35}$$

$$\sum_{i=1}^{m_{j_1 j_2}} v_{12i} = 1, \qquad 1 \leq j_1 < j_2 \leq i \tag{36}$$

$$v_{j_1 j_2 k} \in \{0,1\}, \qquad j_1, j_2 \in \{1,\ldots,j\} \tag{37}$$

$$x_l,\ y_l \geq 0, \qquad 1 \leq l \leq j \tag{38}$$

Constraints (31) and (32) ensure that $L_u$ and $W_u$ do not exceed the dimensions of the bin. The containment constraints (33) and (34) are required for all the pairs of pieces taking into account the order ($j(j-1)$ pairs) and the non-overlapping constraints (35) and (36) are required for all the NFP ($\frac{j(j-1)}{2}$ pairs).

In Figure 6 we show the insertion of piece 15. In this example the previous MIP model was difficult to solve to optimality (the computation time was greater that the threshold $\theta$). Therefore, the relative position of the pieces already placed is fixed while piece 15 is being inserted. We can see that we obtain different solutions with the same objective function value. In this case, the algorithm would solve the first MIP model to optimality (Rotation 1) obtaining a solution. This objective function value will be used in the next two MIPs as upper bound. As a result, the second and third model will be quickly identified as infeasible. Their solutions appear in Figure 6 only

for illustrative purposes.

# 5 Local search

In this section we present two local search strategies referred to as *LS1* and *LS2* respectively. Throughout the different proposed methods we consider that a solution has been improved if its coefficient $F$ (equation 1) increases. Both strategies are based on hill climbing first found improvement and do not include diversification schemes. Both terminate when a local optimum solution is found. The rationale behind such simplistic procedures is that coefficient $F$ has several properties: First, if a move leads to a reduction on the number of bins, $F$ increases and the move is accepted. Second, if a move results in the same number of bins but with one bin with a higher utilisation, $F$ will still increase, accepting the move. These properties prove useful when differentiating between solutions with the same number of bins, removing large plateaux from the solution space.

## 5.1 LS1

Given a solution containing a set of bins $B = \{b_1, \ldots, b_N\}$ sorted by non-decreasing utilization, i.e, $U_1 \leq U_2 \leq \cdots \leq U_N$. The proposed local search procedure (*LS1*) considers all pairs of bins, $b_i$ and $b_j$ with $i < j$, and attempts to move pieces one by one from bin $i$ into bin $j$. If all pieces from both bins fit into only one bin, the new solutions is accepted. Otherwise, the move is accepted if $U_j$ is increased and consequently $U_i$ is decreased. The new solution would have the same number of bins but a higher $F$. The idea is to reduce the usage of the least used bins so that eventually they might be emptied and consequently removed.

A neighbourhood move involves the removal of one piece from the fullest bin, $b_j$, starting with the smallest piece in the bin, and the insertions of a piece from emptiest bin, $b_i$, starting from the largest piece in the bin. Each piece from $b_i$ is tried before the piece from $b_j$ is returned and the next smallest piece is removed. The first improving move found is accepted.

The neighbourhood search procedure *LS1* is as follows: We first sort the pieces from bin $b_j$ by non-decreasing area and the pieces from bin $b_i$ by non-increasing area. Considering the pieces from each bin in their sorted order, remove a piece $p_j \in b_j$. This generates an empty space in which all pieces $p_i \in b_i$ are considered. Each piece from $b_i$ is packed into $b_j$, one at a time. If it fits, the piece is kept in $b_j$ and the next piece in $b_i$ is considered. If not, the piece is placed onto an $Unplaced$ list of pieces. When all pieces of $b_i$ have been considered, we then attempt to pack the removed piece $p_j$ into $b_j$. Again, if it does not fit, it is put onto the $Unplaced$ list.

At this stage, we examine the new utilisation of bin $b_j$. If it has not improved, then the move is discarded. Otherwise, if $Unplaced$ is empty, then bin $b_i$ is also empty and the new solution is accepted. If $Unplaced$ is not empty, all of its pieces are packed into a new bin, again one at a time in non-increasing order of their area. If this packing is unsuccessful, then the new solution has a larger number of bins and is discarded. If all pieces in $Unplaced$ fit, the solution is accepted if the recalculated $F$ is higher. The complete procedure is further detailed in Algorithm 1.

*LS1* considers each pair of bins starting with those with the lowest utilization. Each time an improvement is found we start again from the new solution studying all pairs of bins in which exactly one bin is different from the previous solution and all previously unvisited pairs of bins.

**Data:** $b_i$ and $b_j$ such that $u_i \leq u_j$
**Result:** $b_{i'}$ and $b_{j'}$ such that $u_{i'} \leq u_i$ and $u_{j'} \geq u_j$
$b'_i = b_i$;
$b'_j = b_j$;
**for** *each piece* $p_j$ *in* $b_j$ **do**
  Set $area_{out} = area_{p_j}$, $area_{in} = 0$, $Unplaced = \emptyset$;
  Remove $p_j$ from $b'_j$;
  *List*: Sort pieces in $b_i$ by non-increasing area;
  Add $p_j$ to the end of *List*;
  **for** *each piece p in List* **do**
    Obtain rotations piece $p$ described in Section 4;
    Solve the corresponding MIP models (one per rotation);
    **if** *piece p fits* **then**
      Update $b'_j$;
      $area_{in} += area_p$;
    **else**
      $Unplaced = Unplaced \cup \{p\}$;
    **end**
  **end**
  **if** $area_{in} > area_{out}$ *(bin* $b'_j$ *has a higher utilisation)* **then**
    **if** $Unplaced \neq \emptyset$ **then**
      Sort *Unplaced* by non-increasing area;
      Build $b'_i$ trying to place all the pieces into the bin (see Section 4);
      **if** *All pieces fit* **then**
        return $b'_i$ and $b'_j$;
      **else**
        Set $b'_i = b_i$ and $b'_j = b_j$;
      **end**
    **else**
      return $b'_j$ and $b'_i$ as an empty set (all the pieces fits only in one bin);
    **end**
  **else**
    Set $b'_i = b_i$ and $b'_j = b_j$;
  **end**
**end**

**Algorithm 1:** Procedure to explore the neighbourhood in *LS1*.

## 5.2 LS2

This local search aims to extend the scope of the search in *LS1* by considering a set of bins, $B'$, instead of a single bin $b_j$. As a result, we increase the chances of packing all pieces of bin $b_i$ into a set of bins. As before, the aim is to empty bin $b_i$, as much as possible, from a solution with $N$ bins by moving pieces into any of the bins with greater utilisation than $b_i$. The set of bins, $B'$ is all the bins with greater utilisation that $b_i$ and lower than 0.99, say $B' = \{b_{j_1}, \ldots, b_{j_r}\}, r < N, B' \subseteq B \setminus \{b_i\}$, where $r$ is the number of bins in $B'$.

The main difference is that in *LS1*, once we have built a new bin $b'_j$, we try to rebuild $b_i$. Whereas in *LS2*, we consider $r \geq 2$ bins in such a way that bin $b_i$ is not rebuilt until all the $r$ bins have been considered. If we fail to pack all the pieces from the list of unpacked pieces in the new bin $b'_i$, the whole movement fails and we return to the previous solution. The specific details of the procedure are given as pseudocode in Algorithm 2.

**Data:** $b_i, b_{j_1}, \ldots, b_{j_r}$ such that $u_i \leq u_{j_1} \leq \cdots \leq u_{j_r}$
**Result:** $r$ or $r+1$ new bins packing the same set of pieces with better coefficient $F$
Copy pieces of $b_i$ in *List* ;
$b_i = \emptyset$;
Copy bins $b'_{j_1} = b_{j_1}, \ldots, b'_{j_r} = b_{j_r}$;
improved = true;
**while** *improved* **do**
  improved=false;
  **for** $k = 1$ *to* $r$ **do**
    **for** *each piece* $p_{j_k}$ *in* $b'_{j_k}$ **do**
      Set $area_{out} = area_{p_{j_k}}$, $area_{in} = 0$;
      Remove $p_{j_k}$ from $b'_{j_k}$;
      Copy *List*2 = *List*;
      Add $p_{j_k}$ to the end of *List*;
      **for** *each piece p in List*2 **do**
        Obtain rotations for piece $p$ as described in Section 4;
        Solve the corresponding MIP models (one per rotation);
        **if** *piece p fits* **then**
          Update $b'_{j_k}$;
          $area_{in} += area_p$;
          Remove $p$ from *List*2;
        **else**
          Study next piece;
        **end**
      **end**
      **if** $area_{in} > area_{out}$ **then**
        **if** *List*2 $\neq \emptyset$ **then**
          *List* = *List*2 and sort the list by non-increasing area;
          $b_{j_k} = b'_{j_k}$;
          improved=true;
        **else**
          return $b'_{j_1}, \ldots, b'_{j_r}$ (a solution with one bin less have been found);
        **end**
      **else**
        $b'_{j_k} = b_{j_k}$;
      **end**
    **end**
  **end**
  **if** *List has been modified* **then**
    Try to place all the pieces from *List* into $b_i$, building a new bin $b'_i$;
    **if** *All pieces fit* **then**
      Improvement is found: return $b_i$ and bins $b'_{j_1}, \ldots, b'_{j_r}$;
    **else**
      return (no improvement is found);
    **end**
  **else**
    return (no improvement is found);
  **end**
**end**

**Algorithm 2:** Procedure to explore the neighbourhood in *LS2*.

Local search *LS2* is based on applying the procedure described in Algorithm 2, which is carried out for all the bins $b_i$, $i = 1, \ldots, N-1$. Note that $i$ goes all the way up to the second fullest bin without taking into account bins with an utilisation greater than 0.99.

# 6 Implementation

In Section 3 we present five different construction algorithms for solving the 2DIBBP: BPGD, FFD, PBP, TPS and SCH. The first four approaches initially solve a variant of the one dimensional bin packing problem prior to packing the 2D pieces in the bins. The fifth approach performs the assignment and the packing together. All approaches use the MIP model described in Section 4 to pack the pieces.

In Section 5 we described two local search procedures. Although they can be applied to the solutions obtained by any constructive algorithm, we have tested them only in combination with the constructive algorithm *PBP*. This is because PBP produces more efficient layouts at lower computational cost on average than the other variants (see

Section 7).

There are few parameters to set. The construction algorithms terminate when they find a feasible solution and the local search strategies terminate when they hit a local optima. As described in Section 4, the MIP models can be time consuming, hence for all the approaches we consider $\theta = 20$, so if the time required to solve the MIP exceeds 20 seconds, the binary variables are fixed in the next insertion. We also fix a time limit of 50 seconds per MIP solved. Our experience in developing the algorithms was that this gave a reasonable trade off between computation time and solution efficiency. Clearly if the MIP model is permitted to run for longer, better solutions can be found. The only other parameter is related to the number of rotations attempted in the packing model. For instances in which the pieces can rotate freely, their orientation will be obtained by the criteria described in Section 4. We denote by GR1 the criterion in which only the best rotation is considered and by GR3 the case in which the best three rotations are studied. For instances in the literature in which the orientations of the pieces are limited, usually to two angles of rotation ($0^o$, $180^o$), or four angles of rotation ($0^o$, $90^o$, $180^o$, $270^o$), we consider only these allowed orientations.

The five construction algorithms and the two local search strategies are coded in C++ and implemented using Visual Studio 2008. We use *CPLEX*, version 12.6.0.0, for solving both the Integer Programming models in the assignment strategy and the Mixed Integer Programming models in the packing strategy, and the computational tests are run on a PC with core i7 2600 processor and 4 GB memory.

## 6.1 data

The different variants of the algorithm were tested on four different sets of instances with irregular shapes available in the literature. The first two sets of instances were proposed by Lopez-Camacho et al. (2013). Both sets are jigsaw puzzle instances where the optimal solution is known and equal to 100% utilisation of every bin. The first set, JP1, has a collection of 540 instances where all pieces are convex. These instances are divided in 18 classes with 30 problems in each class with varying numbers of pieces per bin. The second set, JP2, has 480 instances (16 classes with 30 problems each class) in which the pieces have concavities. These two sets of instances are available online at ESICUP web site (*http://paginas.fe.up.pt/ esicup/tiki-index.php*).

Figure 7 shows the solution to two JP1 instances of different types. The size of the bins is the same (but scaled differently in the picture). However, the shape of the pieces and the number of pieces per bin are very different. Both solutions are obtained by algorithm PBP. The top solution belongs to an instance of class L and the solution depicted at the bottom is an instance of class O (in which the algorithm was able to find the optimal solution).

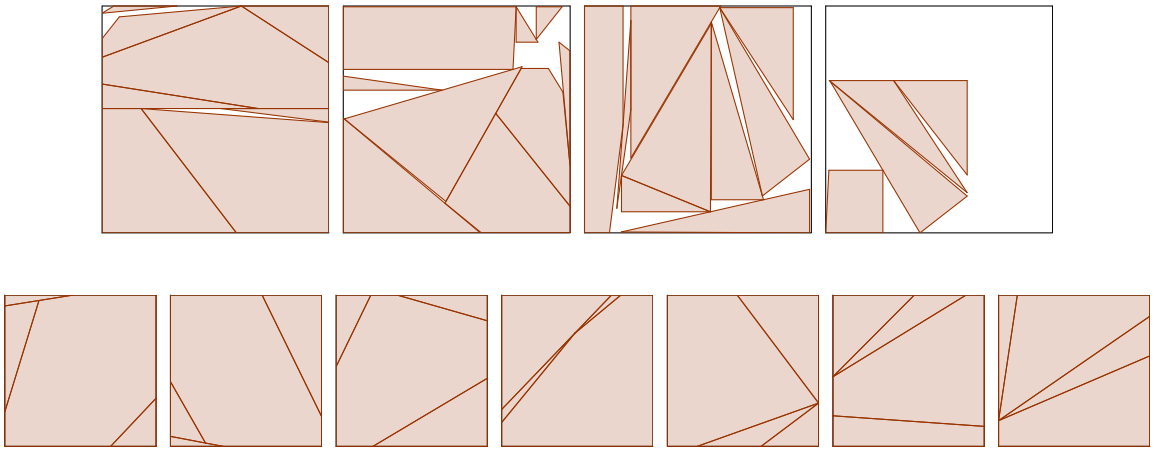


**Figure 7:** *Two solutions of different type instances.*

The third set of instances are created from the well-known irregular strip packing benchmark instances, available on the ESICUP web site. These have a high variety of shapes (convex and non-convex) and the pieces do not fit together exactly, as in the previous two sets. In strip packing problems only the width of the stock sheet is constrained, and the objective is to minimize the total length required to pack all the pieces. In order to modify these instances to our bin packing problem, we only need to define a fixed stock sheet size (width and length). We have built three sets of instances according to the bin size as follows. Let $m_d$ be the maximum length or width

across all the pieces in their initial orientation for a given instance. Then the bin are a fixed dimensional square with the following sizes (given in the same units of the pieces):

- *Nest-SB* (small bins). The bin dimensions are $W = L = 1.1m_d$
- *Nest-MB* (medium bins). The bin dimensions are $W = L = 1.5m_d$
- *Nest-LB* (large bins). The bin dimensions are $W = L = 2m_d$

There are twenty-three irregular strip packing instances, which gives sixty-nine bin packing instances arising from the three bin sizes. Table 1 shows the instances with the corresponding number of pieces, bin size, and rotations permitted. Columns SB, MB and LB shows the size of the bins in each one of the sets.

**Table 1:** *Irregular strip packing instances*

| | N | SB | MB | LB | Rot |
|---|---|---|---|---|---|
| albano | 24 | 3337.40 | 4551.00 | 6068.00 | 0-180 |
| shapes2 | 28 | 5.50 | 7.50 | 10.00 | 0-180 |
| trousers | 64 | 64.90 | 88.50 | 118.00 | 0-180 |
| shapes0 | 43 | 15.40 | 21.00 | 28.00 | 0-90-180-270 |
| shapes1 | 43 | 15.40 | 21.00 | 28.00 | 0-180 |
| shirts | 99 | 26.00 | 19.50 | 14.30 | 0-180 |
| dighe2 | 10 | 77.00 | 105.00 | 140.00 | 0 |
| dighe1 | 16 | 72.60 | 99.00 | 132.00 | 0 |
| fu | 12 | 15.40 | 21.00 | 28.00 | 0-90-180-270 |
| han | 23 | 25.30 | 34.50 | 46.00 | 0-90-180-270 |
| jakobs1 | 25 | 8.80 | 12.00 | 16.00 | 0-90-180-270 |
| jakobs2 | 25 | 17.60 | 24.00 | 32.00 | 0-90-180-270 |
| mao | 20 | 1206.70 | 1645.50 | 2194.00 | 0-180 |
| poly1a | 15 | 14.30 | 19.50 | 26.00 | 0-90-180-270 |
| poly2a | 30 | 14.30 | 19.50 | 26.00 | 0-90-180-270 |
| poly3a | 45 | 14.30 | 19.50 | 26.00 | 0-90-180-270 |
| poly4a | 60 | 14.30 | 19.50 | 26.00 | 0-90-180-270 |
| poly5a | 75 | 14.30 | 19.50 | 26.00 | 0-90-180-270 |
| poly2b | 30 | 14.30 | 19.50 | 26.00 | 0-90-180-270 |
| poly3b | 45 | 14.30 | 19.50 | 26.00 | 0-90-180-270 |
| poly4b | 60 | 14.30 | 19.50 | 26.00 | 0-90-180-270 |
| poly5b | 75 | 14.30 | 19.50 | 26.00 | 0-90-180-270 |
| swimm | 48 | 2133.61 | 2909.47 | 3879.29 | 0-180 |

The rationale for considering three different bin sizes is to investigate the relative importance of the assignment and the packing approaches. Intuitively, for instances *Nest-SB* the assignment problem should be more important than for instances *Nest-MB* and *Nest-LB* because fewer pieces can fit into one bin and the packing problem will be easier. Conversely, in the instances of *Nest-LB*, the packing phase will be more important, where solving the packing problem more efficiently may lead to better solutions even if the assignment is not ideal.

The fourth set of instances have been obtained from Han et al. (2012) for the 2-Dimensional Bin Packing Problem with Irregular convex Pieces and Guillotine Cuts (2DBPPIPGC). The authors test their algorithms on 8 instances. 4 of these instances, called J40, J50, J60 and J70, are real data obtained by a company in the glass cutting industry. The remaining 4 instances, called H80, H100, H120 and H149, were generated randomly taking into account the main properties of the real data. This data set permits free rotation of the pieces. We denote these 8 instances by Jotika.

In addition, we report the solutions we obtain for the three real instances for the ceramic industry provided by a Spanish company located in Castellon, BUTECH-PORCELANOSA. The instances have 209, 340 and 484 pieces. Two of the instances have pieces with concavities and one instance has only convex pieces. In this real application there is no restriction on the orientation of the pieces, and the objective is to minimize the total number of bins ($N$).

# 7 Computational results

In the first computational test we compare the five variants of the constructive algorithm presented in Section 3. In Table 2 we present the average results obtained by the algorithms in all instances in each data set. For each one

of the variants we report the average number of bins ($N$), the average value of $F$ described in (1) and the average fractional number of bins $K$ defined in (3). Table 3 shows the average computational time ($T$) in seconds.

**Table 2:** *Comparison between different variants of the constructive algorithm*

| | SCH | | | BPGD | | | FF | | | PBP | | | TPS | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | N | F | K | N | F | K | N | F | K | N | F | K | N | F | K |
| JP1 | 7.600 | 0.682 | 7.189 | 7.670 | 0.671 | 7.289 | 7.922 | 0.632 | 7.521 | 7.569 | 0.685 | 7.159 | 7.656 | 0.672 | 7.275 |
| JP2 | 7.204 | 0.693 | 6.834 | 7.340 | 0.676 | 7.007 | 7.633 | 0.629 | 7.282 | 7.159 | 0.699 | 6.805 | 7.308 | 0.680 | 6.981 |
| Nest-SB | 9.174 | 0.409 | 8.755 | 9.261 | 0.395 | 8.884 | 9.304 | 0.392 | 8.913 | 9.174 | 0.409 | 8.737 | 9.348 | 0.396 | 8.938 |
| Nest-MB | 4.870 | 0.425 | 4.476 | 4.870 | 0.425 | 4.458 | 4.870 | 0.427 | 4.464 | 4.870 | 0.425 | 4.477 | 4.870 | 0.425 | 4.485 |
| Nest-LB | 2.783 | 0.398 | 2.386 | 2.783 | 0.398 | 2.395 | 2.783 | 0.398 | 2.396 | 2.783 | 0.398 | 2.381 | 2.826 | 0.391 | 2.413 |
| Jotika | 13.125 | 0.695 | 12.542 | 13.375 | 0.678 | 12.790 | 13.500 | 0.664 | 12.892 | 13.125 | 0.698 | 12.514 | 13.375 | 0.671 | 12.884 |
| Av. | 7.459 | 0.550 | 7.030 | 7.550 | 0.540 | 7.137 | 7.669 | 0.524 | 7.245 | 7.447 | 0.553 | 7.012 | 7.564 | 0.539 | 7.163 |

**Table 3:** *Computational time (in seconds) used by all the constructive algorithms*

| | SCH | BPGD | FF | PBP | TPS |
|---|---|---|---|---|---|
| JP1 | 52 | 81 | 54 | 53 | 85 |
| JP2 | 72 | 58 | 40 | 42 | 67 |
| Nest-SB | 78 | 67 | 61 | 72 | 98 |
| Nest-MB | 129 | 119 | 121 | 116 | 124 |
| Nest-LB | 291 | 313 | 284 | 265 | 307 |
| Jotika | 129 | 163 | 154 | 148 | 233 |
| Av. | 125 | 134 | 120 | 116 | 152 |

Although the differences in $N$, the number of bins required by the solutions, are very small, as is usual in bin packing problems, we can observe that in all the data sets PBP performed the best on average in all the measures of performance ($N$, $F$ and $K$), with lower computational effort. Variant SCH is the next best preforming algorithm, although the computational effort is slightly higher than PBP and FF. The approaches that made a full assignment to the bins performed least well (BPGD, FF and TPS). PBP makes a partial assignment by solving the 1D knapsack problem for one bin at a time and reducing the greediness of the approach with a modified objective function. SCH does not pre-assign pieces to bins but simply places the piece in the first bin it will fit. This leads to many more attempts to pack a piece, increasing computation time. Hence, we can conclude that attempting to assign pieces prior to packing is valuable but only for the immediate decision to pack the next bin. Tables 7, 8, 9, 10,11 and 12 in the appendix shows the average results for each one of the classes (or individual instances) in JP1, JP2, Nest and Jotika. Since *PBP* is the best performing algorithm, we use this variant for the remainder of the investigation.

In order to compare the improvement of each one of the local search procedures described in Section 5 we present the average improvement of each local search procedure over the solution obtained by the PBP algorithm. In Table 4 we present the average improvement of $F$ obtained by both local search procedures LS1 and LS2 when applied to the solution obtained by the constructive algorithm PBP for each set of instances. In general there is a significant improvement with both local search strategies. We can observe that LS2 performs better than LS1, obtaining better average results in all sets of instances but one, with a similar computational effort. Table 13 and 14 in the appendix show the the results obtained in each one of the classes on sets JP1 and JP2. The average results obtained on set JP1 by LS1 and LS2, 0.703 and 0.723, respectively, improve on the average solution quality from the best known algorithms results reported by Terashima-Marin et al. (2010), 0.690.

Note that the average improvement of LS2 on instances Nest-LB is much lower than the other sets of instances. This fact suggests that the impact of the local search depends on the ratio between the size of the pieces and the size of the bins. The average number of pieces per bin on instances sets Nest-SB, Nest-MB and Nest-LB is, respectively, 4.35, 8.19 and 14.34. Therefore, if the average number of pieces per bin is big enough then the local search procedure, which explores only the assignment of the pieces into the bin, is less effective. However, the local search produces relevant improvements on instances in which the average number of pieces per bin is lower. This confirms our conjecture that the assignment is more important when the number of pieces per bin is small.

The next set of computational experiments evaluate the impact of allowing free rotation of the pieces. In addition, we investigate the influence of the initial given rotation and how that impacts the solution quality. Our conjecture is that data sets are created with a good initial orientation with respect to packing and therefore keeping this orientation fixed would produce good solutions. For some applications i.e. garment manufacturing the pattern and bias of the material is important, and rotation angles are finite and specific. However, with many other materials there is no practical reason for any given initial rotation. Table 5 shows the results obtained for the Nest-MB instances by the PBP constructive algorithm with three configurations. In columns "Initial Given" the

**Table 4:** *LS1 vs LS2*

| | PBP | | LS1 | | | LS2 | | |
|---|---|---|---|---|---|---|---|---|
| | F | T | F | %Imp | T | F | %Imp | T |
| JP1 | 0.685 | 52 | 0.703 | 1.740 | 129 | 0.723 | **3.806** | 161 |
| JP2 | 0.699 | 42 | 0.716 | 1.753 | 136 | 0.729 | **3.068** | 123 |
| Nest-SB | 0.409 | 72 | 0.434 | **5.802** | 1721 | 0.432 | 5.389 | 1297 |
| Nest-MB | 0.425 | 116 | 0.445 | 4.493 | 2816 | 0.452 | **6.029** | 3172 |
| Nest-LB | 0.398 | 265 | 0.403 | 1.222 | 951 | 0.403 | **1.294** | 767 |
| Jotika | 0.698 | 148 | 0.725 | 3.740 | 1931 | 0.730 | **4.370** | 1426 |

orientation of the pieces in the original data is kept and for each instance we consider the finite rotation angles given in Table 1. In columns "Initial Random" the orientation of the pieces in the original data is rotated a random angle and the permitted orientation of the pieces are considered starting from the new angle of rotation. The columns headed "Free Rot" present the results obtained when the pieces are allowed to rotate freely, regardless of their permitted orientations.

If we compare the results obtained by "Initial Given" and "Initial Random", we observe that "Initial Given" gets better solutions in seventeen of the twenty-three cases, and of those cases where "Initial Given" is worse, all but one arise in the *poly* data sets that are artificial instances with arbitrary shapes for which the initial orientation is not important. This suggests that the initial orientation given to the pieces is advantageous in the majority of nesting instances coming from real problems, allowing good matchings between the pieces and the edges of the bin. In Figure 8 we can observe that the initial angle of rotations on solution (b) is clearly better for packing, allowing the larger pieces to be packed together more effectively. In solution (a), where we randomized the initial angle of rotations, pieces have fewer matching edges leading to a worse solution. A more significant impact can be found in instances *dighe1* and *dighe2*. These are jigsaw puzzle instances where only the initial orientation of the pieces allows the perfect fit.

Comparing these results with those obtained by "Free Rot", this variant found better result than both the finite rotation variants in fourteen cases. Note that our "get rotations" algorithm is not dependent on the initial angle of rotation given to the pieces. Hence, when there is no practical reason for setting specific angles of rotation, our free rotation approach removes any bias from how the data sets are created. Also note that on average the number of bins is better for "Free Rot", but its F value is inferior to that of "Initial Given". Although in most of the instance in which "Free Rot" does not obtain the best result the distance to best is very small, there are very large discrepancies on the two jigsaw instances, especially on *dighe2*, in which the algorithm with free rotation needs an extra bin, and as a result the F value is significantly lower.

In summary, when the instances come from real problems or are jigsaw instances they are given with an initial orientation which will produce the best or near-best solution. Our algorithm with free rotation, in which every time a piece is packed its orientation is chosen according to the partial configuration of the bin, is able to improve on the results obtained using this initial orientation for most of the instances and to get similar solutions for the remaining ones, except for jigsaw instances in which just one bad selection of the orientation of one piece can lead to much worse solutions. Apart from these very specific cases, the proposed algorithm, which is independent of the initial orientation of the pieces, tends to produce better solutions.

In Table 6 we present the results obtained in the three real instances proposed by BUTECH-PORCELANOSA. For these instances the objective is to reduce the total number of bins, $N$. For illustrative purposes the table also shows the average percentage utilization $U = \sum_{i=1}^{N} U_i/N$, as well as the computing times $T$. Concerning the algorithms, since the upper bounds on the number of bins and the number of pieces are very large, solving the corresponding IP models many times, as required by TPS, is computationally prohibitive. This variant of the algorithm is therefore not included. However, algorithm BPGD, which solves a similar IP formulation but far fewer times, produces a solution in reasonable computing time and is included. Nevertheless, the best algorithm in terms of computational time and quality of the solutions is, again, PBP, that always provides average utilizations greater than 90% and up to 97.49%. These solutions considerably improved the efficiency of previous manual solutions.

# 8 Conclusions

In this paper we have addressed a new bin packing problem with irregular pieces that can be freely rotated for which there is no specialized algorithm despite the fact that the problem arises in industry. It was motivated

**Table 5:** *Analysis of different rotation criteria on Nest-MB instances*

| | Initial Random | | Initial Given | | Free Rot | |
|---|---|---|---|---|---|---|
| | N | F | N | F | N | F |
| albano | 4 | 0.332 | 3 | 0.480 | 3 | **0.510** |
| shapes2 | 12 | 0.248 | 10 | 0.349 | 10 | **0.350** |
| trousers | 4 | 0.362 | 3 | 0.555 | 3 | **0.580** |
| shirts | 9 | 0.450 | 8 | 0.518 | 8 | **0.570** |
| shapes0 | 10 | 0.179 | 7 | 0.271 | 6 | **0.390** |
| shapes1 | 9 | 0.150 | 7 | 0.282 | 6 | **0.390** |
| dighe2 | 2 | 0.236 | 1 | **0.823** | 2 | 0.280 |
| dighe1 | 2 | 0.283 | 2 | **0.368** | 2 | 0.290 |
| fu | 5 | 0.247 | 4 | **0.443** | 4 | 0.440 |
| han | 3 | 0.309 | 3 | 0.387 | 3 | **0.390** |
| jakobs1 | 6 | 0.281 | 4 | **0.570** | 4 | 0.530 |
| jakobs2 | 4 | 0.354 | 4 | **0.401** | 4 | 0.380 |
| mao | 3 | 0.272 | 3 | 0.271 | 3 | **0.300** |
| poly1a | 2 | 0.307 | 2 | 0.308 | 2 | **0.320** |
| poly2a | 4 | **0.342** | 4 | 0.339 | 4 | 0.340 |
| poly3a | 5 | 0.420 | 5 | 0.419 | 5 | **0.420** |
| poly4a | 7 | 0.396 | 7 | 0.392 | 7 | **0.410** |
| poly5a | 8 | 0.448 | 8 | 0.449 | 8 | **0.450** |
| poly2b | 4 | 0.363 | 4 | 0.389 | 4 | **0.400** |
| poly3b | 5 | 0.423 | 5 | 0.432 | 5 | **0.440** |
| poly4b | 6 | **0.474** | 6 | 0.465 | 6 | 0.470 |
| poly5b | 7 | **0.486** | 7 | 0.478 | 7 | 0.480 |
| swimm | 5 | 0.385 | 5 | **0.397** | 5 | 0.390 |
| Av. | 5.478 | 0.337 | 4.870 | 0.425 | 4.826 | 0.414 |
| #best | | 3 | | 6 | | 14 |

**Table 6:** *Computational results for the real data instances*

| | SCH | | | BPGD | | | PBP | | | FFD | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | N | U | T | N | U | T | N | U | T | N | U | T |
| BP209 | **48** | 0.9157 | 320 | 52 | 0.8454 | 1280 | **48** | 0.9157 | 293 | 51 | 0.8577 | 589 |
| BP340 | 204 | 0.9354 | 663 | 214 | 0.8918 | 4049 | **202** | 0.9455 | 234 | 210 | 0.9089 | 568 |
| BP484 | **316** | 0.9732 | 22 | 388 | 0.7900 | 1827 | **316** | 0.9749 | 83 | 325 | 0.9471 | 527 |

by the ceramic tile sector, from which we have tested the algorithm in several instances provided by BUTECH (PORCELANOSA group) in Spain. Due to the high cost of the material used, the main objective is to obtain high utilization solutions.

Besides the instances provided by the company we have used several sets of test instances, some of them previously published and other adapted from related problems, in order to have instance with very different characteristics. That has allowed us to study the behavior of the different strategies developed in this paper on a wide range of cases.

We have solved the problem by decomposing it into two phases, the assignment of pieces to bins and the packing of the assigned pieces in the corresponding bin. As one of the objectives of the study was to investigate the value of this separation strategy versus the assignment as a consequence of the packing decision, we have developed and tested several assignment procedures, including the simple sequential assignment method. The results show that a partial assignment is the most successful strategy. This strategy is known to work well in one dimensional and two-dimensional bin packing problems with rectangular pieces and it is also the best in this case of irregular pieces.

We have also investigated the relative impact of assignment and packing depending on the data instance and found it highly dependent on the average number of pieces per bin. The assignment is more important when the number of pieces per bin is small. In particular, the local searches developed to improve the solutions, based on modifying the assignment of pieces to bins, work better when only a few pieces fit into each bin.

Finally, we have studied the impact of the orientation of pieces given in the original data and how free rotation helps to improve the solutions by modifying the orientation of the pieces. Our results show that in some cases, such as jigsaw puzzle instances in which only using the initial orientation of the pieces is possible to obtain the perfect fit, allowing free rotation cannot improve on the solutions and more likely will produce worse solutions if some wrong decisions about the orientation are made in the constructive process. However, in many other cases, our algorithm, which is not dependent of the initial orientation, produces better results, finding better orientations

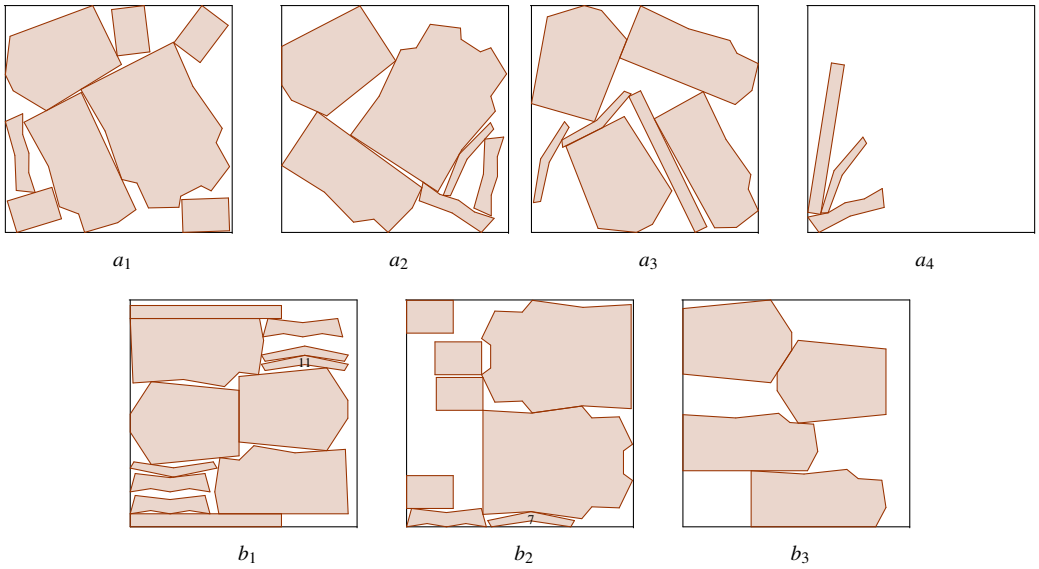



**Figure 8:** *Solution a with "Initial Random", needed 4 bins; solution b with "Initial Given", needed* 3 *bins*

for some pieces.

As future work, we plan to extend the ideas developed here to other bin packing problems with irregular pieces, such as the problems arising from cutting leather, in which bins are neither identical nor rectangular, and can even have some defects, defining non-usable zones.

## Acknowledgments

This study has been partially supported by the Spanish Ministry of Economy and Competitiveness, DPI2011-24977, and by the Generalitat Valenciana, PROMETEO/2013/049. The authors would like to acknowledge the contribution of the professionals at PORCELANOSA Group, in particular to Javier M. Chiva Bartoll, who have contributed to the practical application in this work in a significant way.

## References

R. Alvarez-Valdes, A. Martinez-Sykora, and J.M. Tamarit. A branch & bound algorithm for cutting and packing irregularly shaped pieces. *International Journal of Production Economics*, 145:466–477, 2013.

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

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

J.A. Bennell and X. Song. A beam search implementation for the irregular shape packing problem. *Journal of Heuristics*, 16(2):167–188, 2010.

A. Caprara and U. Pferschy. Worst-case analysis of the subset sum algorithm for bin packing.. *Operations Research Letters*, 32:159–166, 2004.

N. Chernov, Y. Stoyan, and T. Romanova. Mathematical model and efficient algorithms for objects packing problem. *Computational Geometry: Theory and Applications* 43:535–553, 2010.

M. Fischetti and I. Luzzi. Mixed-integer programming models for nesting problems. *Journal of Heuristics*, 15: 201–226, 2008.

W. Han, J.A. Bennell, X. Zhao, and X. Song. Construction heuristics for two dimensional irregular shape bin packing with guillotine constraints. *European Journal of Operations Research*, 230:495–504, 2012.

M. Hifi and R. M'Hallah. A best-local position procedure-based heuristic for two-dimensional layout problems. *Studia Informatica Universalis, International Journal on Informatics*, 2(1):33–56, 2002.

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

D.S. Johnson, A. Demers, J.D. Ullman, M.R. Garvey, and R.L. Graham. Worst-case performace bounds for simple one-dimensional packing algorithms. *SIAM Journal on Comuting*, 3:299–325, 1974.

D. Liu and H.Teng. An improved bl-algorithm for genetic algorithm of the othogonal packing of rectangle. *European Journal of Operational Research*, 112:413–419, 1999.

E. Lopez-Camacho, G. Ochoa, H. Terashima-Marin, and E.K. Burke. An effective heuristic for the two-dimensional irregular bin packing problem. *Annals of Operations Research*, 206:241–264, 2013.

E. Lopez-Camacho, H. Terashima-Marin, Peter Ross and G. Ochoa,. A unified hyper-heuristic framework for solving packing problems. *Expert Systems with Applications*, 41:6876–6889, 2014.

A. Martinez-Sykora, R. Alvarez-Valdes, J. Bennell, and J.M. Tamarit. Constructive procedures to solve 2-dimensional bin packing problems with irregular pieces and guillotine cuts. *Omega*, 52:15–32, 2015.

F. Parreño, R. Alvarez-Valdes, J.F. Oliveira, and J.M. Tamarit A hybrid GRASP/VND algorithm for two- and three-dimensional bin packing *Annals of Operations Research*, 179:203–222, 2010.

P. Ross, S. Schulenburg, and J.G. Marin-Blazquez. Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. *Lecture notes in computer science. Conference on genetic and evolutionary computation*, 942–948, 2002.

X. Song and J.A. Bennell. Column generation and sequential heuristic procedure for solving an irregular shape cutting stock problem. *Journal of the Operational Research Society*, 1–16, 2013.

H. Terashima-Marin, P. Ross, C.J. Farias-Zarate, E. Lopez-Camacho, and M. Valenzuela-Rendon. Generalized hyper-heuristics for solving 2d regular and irregular packing problems. *Annals of Operations Research*, 179: 369–392, 2010.

G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183, 109-1130, 2007.

# 9 Appendix

In this Section we present with more detail the computational results presented and discussed in Section 7.

**Table 7:** *Results of the different constructive algorithms applied on set of instances JP1*

| | SCH | | | | BPDG | | | | FF | | | | PBP | | | | TPS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | N | F | K | T | N | F | K | T | N | F | K | T | N | F | K | T | N | F | K | T |
| A | 4.000 | 0.613 | 3.498 | 51.733 | 4.000 | 0.605 | 3.520 | 61.400 | 4.000 | 0.605 | 3.519 | 47.600 | 4.000 | **0.614** | 3.490 | 51.700 | 4 | 0.605 | 3.5356 | 61.1 |
| B | 11.833 | 0.761 | 11.406 | 0.933 | 11.333 | **0.826** | 11.100 | 27.833 | 11.967 | 0.741 | 11.586 | 1.733 | 11.700 | 0.781 | 11.281 | 1.300 | 11.5 | 0.8059 | 11.243 | 24.3 |
| C | 7.233 | **0.731** | 6.838 | 18.200 | 7.600 | 0.671 | 7.155 | 46.267 | 7.933 | 0.615 | 7.436 | 22.933 | 7.233 | 0.729 | 6.858 | 18.833 | 7.6 | 0.671 | 7.1642 | 46.567 |
| D | 4.000 | **0.591** | 3.638 | 149.500 | 4.000 | 0.586 | 3.668 | 160.340 | 4.000 | 0.581 | 3.693 | 163.700 | 4.000 | 0.590 | 3.636 | 164.533 | 4 | 0.5856 | 3.667 | 168.73 |
| E | 4.400 | **0.523** | 4.098 | 100.500 | 4.400 | 0.519 | 4.102 | 127.367 | 4.467 | 0.510 | 4.141 | 98.300 | 4.400 | 0.522 | 4.098 | 101.167 | 4.5667 | 0.4961 | 4.1762 | 109.4 |
| F | 3.000 | **0.516** | 2.412 | 39.733 | 3.000 | 0.508 | 2.457 | 31.800 | 3.000 | 0.507 | 2.435 | 34.900 | 3.000 | 0.515 | 2.414 | 41.967 | 3 | 0.5041 | 2.4467 | 34.4 |
| G | 14.533 | 0.716 | 14.158 | 1.833 | 14.867 | 0.684 | 14.494 | 78.100 | 14.967 | 0.681 | 14.552 | 3.233 | 14.467 | **0.724** | 14.050 | 3.300 | 14.633 | 0.7082 | 14.272 | 68.967 |
| H | 14.267 | 0.751 | 13.859 | 1.100 | 14.067 | 0.774 | 13.719 | 68.133 | 14.400 | 0.736 | 14.037 | 2.033 | 14.133 | 0.766 | 13.693 | 1.833 | 14 | **0.786** | 13.693 | 66.267 |
| I | 4.000 | **0.629** | 3.383 | 87.067 | 4.000 | 0.622 | 3.410 | 97.300 | 4.000 | 0.617 | 3.414 | 93.233 | 4.000 | 0.628 | 3.382 | 83.367 | 4 | 0.6211 | 3.4057 | 97.733 |
| J | 5.000 | 0.671 | 4.578 | 88.233 | 5.000 | 0.665 | 4.619 | 97.033 | 5.000 | 0.662 | 4.628 | 82.100 | 5.000 | **0.672** | 4.570 | 90.533 | 5 | 0.6656 | 4.6287 | 94.767 |
| K | 7.033 | 0.760 | 6.730 | 47.800 | 7.233 | 0.728 | 6.896 | 72.733 | 7.567 | 0.677 | 7.184 | 37.733 | 7.033 | **0.761** | 6.721 | 51.533 | 7.2333 | 0.7274 | 6.9074 | 73.2 |
| L | 4.067 | **0.589** | 3.751 | 28.467 | 4.067 | 0.580 | 3.806 | 49.367 | 4.100 | 0.569 | 3.856 | 34.867 | 4.067 | 0.588 | 3.754 | 28.600 | 4.0667 | 0.5782 | 3.8204 | 44.5 |
| M | 6.400 | **0.658** | 6.102 | 24.200 | 6.500 | 0.636 | 6.198 | 48.400 | 7.000 | 0.564 | 6.536 | 30.867 | 6.400 | 0.655 | 6.111 | 23.600 | 6.5333 | 0.6326 | 6.2192 | 48.567 |
| N | 3.000 | 0.518 | 2.388 | 199.700 | 3.000 | 0.513 | 2.395 | 218.900 | 3.000 | 0.512 | 2.416 | 210.867 | 3.000 | 0.517 | 2.392 | 186.300 | 3 | 0.5077 | 2.4124 | 230.57 |
| O | 7.967 | 0.826 | 7.641 | 0.833 | 7.433 | 0.920 | 7.333 | 11.733 | 8.800 | 0.675 | 8.490 | 3.567 | 7.933 | 0.831 | 7.596 | 1.533 | 7.3667 | **0.929** | 7.2584 | 10.567 |
| P | 9.867 | 0.713 | 9.420 | 45.533 | 9.967 | 0.688 | 9.577 | 71.233 | 10.133 | 0.655 | 9.835 | 51.533 | 9.800 | **0.720** | 9.383 | 53.967 | 9.9667 | 0.6884 | 9.5771 | 83.233 |
| Q | 15.633 | 0.941 | 15.338 | 14.200 | 16.600 | 0.843 | 16.283 | 137.933 | 17.133 | 0.792 | 16.807 | 10.200 | 15.500 | **0.956** | 15.263 | 7.800 | 16.333 | 0.8692 | 16.061 | 173.23 |
| R | 10.567 | 0.766 | 10.165 | 33.033 | 11.000 | 0.710 | 10.477 | 58.633 | 11.133 | 0.681 | 10.812 | 40.400 | 10.567 | **0.767** | 10.165 | 39.967 | 11 | 0.7103 | 10.462 | 100.67 |
| Av. | 7.600 | 0.682 | 7.189 | 51.811 | 7.670 | 0.671 | 7.289 | 81.361 | 7.922 | 0.632 | 7.521 | 53.878 | 7.569 | 0.685 | 7.159 | 52.880 | 7.656 | 0.672 | 7.275 | 85.376 |

**Table 8:** *Results of the different constructive algorithms applied on set of instances JP2*

| | SCH | | | | BPGD | | | | FF | | | | PBP | | | | TPS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | N | F | K | T | N | F | K | T | N | F | K | T | N | F | K | T | N | F | K | T |
| A | 4.000 | **0.626** | 3.435 | 90.000 | 4.000 | 0.594 | 3.653 | 97.367 | 4.067 | 0.584 | 3.688 | 84.100 | 4.000 | 0.605 | 3.579 | 95.345 | 4 | 0.5926 | 3.6594 | 144.59 |
| B | 12.000 | 0.728 | 11.644 | 74.767 | 12.300 | 0.698 | 11.901 | 51.033 | 12.833 | 0.636 | 12.477 | 22.500 | 11.967 | **0.731** | 11.596 | 18.133 | 12.233 | 0.7014 | 11.848 | 48.467 |
| C | 7.700 | 0.665 | 7.217 | 134.633 | 7.767 | 0.653 | 7.260 | 105.933 | 7.933 | 0.612 | 7.452 | 79.467 | 7.517 | **0.690** | 7.094 | 79.828 | 7.7667 | 0.6522 | 7.2594 | 150.47 |
| F | 3.000 | 0.510 | 2.488 | 225.667 | 3.000 | 0.498 | 2.503 | 83.400 | 3.000 | 0.501 | 2.499 | 80.500 | 3.000 | **0.511** | 2.468 | 79.700 | 3 | 0.4921 | 2.5384 | 123 |
| H | 14.433 | **0.730** | 13.985 | 49.467 | 14.667 | 0.704 | 14.281 | 67.300 | 15.100 | 0.661 | 14.736 | 19.200 | 14.433 | **0.730** | 13.995 | 14.567 | 14.633 | 0.7091 | 14.219 | 88.167 |
| L | 4.200 | 0.563 | 3.883 | 93.600 | 4.133 | 0.564 | 3.886 | 67.967 | 4.167 | 0.556 | 3.914 | 59.667 | 4.167 | **0.571** | 3.823 | 71.933 | 4.1667 | 0.561 | 3.8764 | 70.2 |
| M | 6.633 | 0.627 | 6.264 | 94.767 | 6.833 | 0.594 | 6.354 | 78.200 | 6.967 | 0.564 | 6.524 | 63.667 | 6.533 | **0.641** | 6.194 | 68.067 | 6.8 | 0.6 | 6.328 | 85.467 |
| O | 8.300 | 0.769 | 7.857 | 21.700 | 8.733 | 0.691 | 8.429 | 34.600 | 9.200 | 0.618 | 8.888 | 11.567 | 8.233 | **0.774** | 7.843 | 5.533 | 8.6333 | 0.7038 | 8.3437 | 33.5 |
| S | 3.000 | 0.517 | 2.633 | 27.900 | 2.867 | 0.556 | 2.603 | 16.900 | 2.967 | 0.526 | 2.644 | 14.100 | 2.967 | 0.521 | 2.605 | 13.333 | 2.8667 | **0.561** | 2.5877 | 14.433 |
| T | 10.833 | 0.887 | 10.556 | 9.567 | 11.167 | 0.846 | 10.969 | 24.467 | 12.100 | 0.736 | 11.792 | 3.500 | 10.833 | **0.890** | 10.542 | 2.033 | 11.067 | 0.8599 | 10.883 | 24.333 |
| U | 6.133 | 0.732 | 5.830 | 19.167 | 6.200 | 0.736 | 5.901 | 18.833 | 6.933 | 0.571 | 6.550 | 8.500 | 6.100 | 0.737 | 5.826 | 7.600 | 6.1667 | **0.744** | 5.8871 | 17.133 |
| V | 5.233 | 0.942 | 5.115 | 2.833 | 5.033 | 0.990 | 5.027 | 0.633 | 5.267 | 0.936 | 5.194 | 0.467 | 5.200 | 0.951 | 5.091 | 0.600 | 5.0667 | **0.982** | 5.0395 | 0.4667 |
| W | 5.133 | 0.646 | 4.790 | 49.667 | 5.100 | 0.649 | 4.768 | 45.600 | 5.167 | 0.630 | 4.861 | 26.733 | 5.067 | 0.656 | 4.712 | 30.300 | 5.0333 | **0.671** | 4.7104 | 41.567 |
| X | 4.033 | 0.586 | 3.727 | 134.633 | 4.000 | 0.588 | 3.711 | 80.467 | 4.000 | 0.581 | 3.742 | 69.233 | 4.000 | **0.593** | 3.691 | 69.867 | 4.0345 | 0.5822 | 3.7171 | 78.103 |
| Y | 7.300 | **0.720** | 6.992 | 95.200 | 7.533 | 0.687 | 7.125 | 87.533 | 7.767 | 0.647 | 7.293 | 65.567 | 7.300 | **0.720** | 6.958 | 70.767 | 7.3667 | 0.7044 | 7.0639 | 91.1 |
| Z | 13.333 | 0.847 | 12.924 | 33.700 | 14.100 | 0.763 | 13.734 | 63.933 | 14.667 | 0.708 | 14.251 | 37.967 | 13.233 | **0.856** | 12.866 | 39.100 | 14.1 | 0.7633 | 13.735 | 64.433 |
| Av. | 7.204 | 0.693 | 6.834 | 72.329 | 7.340 | 0.676 | 7.007 | 57.760 | 7.633 | 0.629 | 7.282 | 40.421 | 7.159 | 0.699 | 6.805 | 41.669 | 7.308 | 0.680 | 6.981 | 67.214 |

**Table 9:** *Results of the different constructive algorithms applied on each instance in Nest-SB*

| | SCH | | | | BPGD | | | | FF | | | | PBP | | | | TPS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | N | F | K | T | N | F | K | T | N | F | K | T | N | F | K | T | N | F | K | T |
| albano | 6 | 0.474 | 5.216 | 12 | 6 | 0.419 | 5.811 | 18 | 6 | 0.677 | 5.521 | 10 | 6 | 0.474 | 5.216 | 10 | 6 | 0.419 | 5.811 | 18 |
| shapes2 | 20 | 0.300 | 19.727 | 3 | 20 | 0.300 | 19.727 | 4 | 20 | 0.542 | 19.727 | 1 | 20 | 0.300 | 19.727 | 3 | 20 | 0.300 | 19.727 | 5 |
| trousers | 5 | 0.671 | 4.909 | 202 | 6 | 0.519 | 5.247 | 232 | 5 | 0.831 | 4.909 | 195 | 5 | 0.671 | 4.909 | 181 | 5 | 0.677 | 4.886 | 185 |
| shapes0 | 14 | 0.245 | 13.390 | 8 | 14 | 0.245 | 13.390 | 11 | 14 | 0.500 | 13.390 | 14 | 14 | 0.245 | 13.390 | 11 | 14 | 0.245 | 13.391 | 10 |
| shapes1 | 13 | 0.272 | 12.714 | 22 | 13 | 0.272 | 12.714 | 22 | 13 | 0.528 | 12.714 | 19 | 13 | 0.272 | 12.714 | 21 | 14 | 0.241 | 13.712 | 19 |
| shirts | 14 | 0.571 | 13.839 | 135 | 16 | 0.462 | 15.490 | 115 | 16 | 0.676 | 15.629 | 97 | 14 | 0.571 | 13.839 | 131 | 16 | 0.466 | 15.490 | 169 |
| dighe2 | 3 | 0.359 | 2.654 | 0 | 3 | 0.350 | 2.662 | 1 | 3 | 0.611 | 2.654 | 0 | 3 | 0.359 | 2.654 | 0 | 3 | 0.350 | 2.662 | 1 |
| dighe1 | 3 | 0.406 | 2.771 | 3 | 3 | 0.406 | 2.771 | 3 | 3 | 0.677 | 2.817 | 4 | 3 | 0.406 | 2.771 | 3 | 3 | 0.406 | 2.772 | 2 |
| fu | 8 | 0.362 | 7.455 | 1 | 8 | 0.357 | 7.455 | 1 | 8 | 0.602 | 7.455 | 1 | 8 | 0.362 | 7.455 | 1 | 8 | 0.348 | 7.584 | 1 |
| han | 5 | 0.435 | 4.237 | 33 | 5 | 0.430 | 4.277 | 38 | 5 | 0.619 | 4.277 | 11 | 5 | 0.435 | 4.237 | 27 | 5 | 0.430 | 4.277 | 26 |
| jakobs1 | 9 | 0.371 | 8.341 | 1 | 10 | 0.305 | 9.341 | 10 | 9 | 0.576 | 8.682 | 4 | 9 | 0.371 | 8.341 | 2 | 10 | 0.305 | 9.341 | 10 |
| jakobs2 | 7 | 0.402 | 6.682 | 10 | 7 | 0.401 | 6.707 | 9 | 7 | 0.640 | 6.818 | 11 | 7 | 0.402 | 6.682 | 9 | 7 | 0.401 | 6.707 | 11 |
| mao | 4 | 0.432 | 3.706 | 22 | 4 | 0.430 | 3.831 | 27 | 4 | 0.702 | 3.706 | 16 | 4 | 0.432 | 3.706 | 19 | 4 | 0.428 | 3.706 | 20 |
| poly1a | 3 | 0.438 | 2.892 | 24 | 3 | 0.441 | 2.896 | 14 | 4 | 0.513 | 3.420 | 13 | 3 | 0.438 | 2.892 | 20 | 3 | 0.438 | 2.856 | 24 |
| poly2a | 7 | 0.363 | 6.420 | 52 | 6 | 0.433 | 5.977 | 47 | 7 | 0.575 | 6.420 | 50 | 7 | 0.363 | 6.420 | 49 | 7 | 0.360 | 6.280 | 53 |
| poly3a | 10 | 0.385 | 9.420 | 106 | 9 | 0.434 | 8.824 | 86 | 9 | 0.660 | 8.949 | 82 | 10 | 0.386 | 9.420 | 92 | 10 | 0.384 | 9.420 | 135 |
| poly4a | 13 | 0.381 | 12.515 | 206 | 12 | 0.436 | 11.860 | 147 | 12 | 0.657 | 11.997 | 116 | 13 | 0.386 | 12.439 | 160 | 13 | 0.385 | 12.439 | 242 |
| poly5a | 16 | 0.400 | 15.420 | 297 | 16 | 0.398 | 15.420 | 181 | 16 | 0.630 | 15.517 | 176 | 16 | 0.400 | 15.420 | 274 | 16 | 0.403 | 15.420 | 480 |
| poly2b | 7 | 0.409 | 6.632 | 31 | 7 | 0.395 | 6.843 | 27 | 7 | 0.652 | 6.629 | 39 | 7 | 0.410 | 6.506 | 33 | 7 | 0.409 | 6.615 | 70 |
| poly3b | 9 | 0.444 | 8.916 | 89 | 9 | 0.441 | 8.918 | 98 | 10 | 0.607 | 9.420 | 87 | 9 | 0.444 | 8.916 | 86 | 9 | 0.449 | 8.768 | 90 |
| poly4b | 12 | 0.430 | 11.210 | 151 | 12 | 0.414 | 11.420 | 91 | 12 | 0.645 | 11.420 | 136 | 12 | 0.430 | 11.210 | 147 | 12 | 0.419 | 11.420 | 235 |
| poly5b | 14 | 0.428 | 13.426 | 227 | 14 | 0.432 | 13.420 | 228 | 14 | 0.658 | 13.448 | 208 | 14 | 0.431 | 13.284 | 202 | 14 | 0.431 | 13.420 | 270 |
| swimm | 9 | 0.416 | 8.877 | 165 | 10 | 0.365 | 9.335 | 127 | 10 | 0.588 | 9.486 | 121 | 9 | 0.418 | 8.806 | 165 | 9 | 0.414 | 8.865 | 170 |
| Av. | 9.174 | 0.409 | 8.755 | 78.261 | 9.261 | 0.395 | 8.884 | 66.826 | 9.304 | 0.624 | 8.913 | 61.348 | 9.174 | 0.409 | 8.737 | 71.565 | 9.348 | 0.396 | 8.938 | 97.652 |

**Table 10:** *Results of the different constructive algorithms applied on each instance in Nest-MB*

| | SCH | | | | BPGD | | | | FF | | | | PBP | | | | TPS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | N | F | K | T | N | F | K | T | N | F | K | T | N | F | K | T | N | F | K | T |
| albano | 3 | 0.480 | 2.873 | 26 | 3 | 0.480 | 2.873 | 25 | 3 | 0.491 | 2.667 | 24 | 3 | 0.480 | 2.873 | 22 | 3 | 0.480 | 2.895 | 36 |
| shapes2 | 10 | 0.345 | 9.933 | 3 | 10 | 0.346 | 9.933 | 7 | 10 | 0.347 | 9.933 | 4 | 10 | 0.349 | 9.933 | 5 | 10 | 0.344 | 9.933 | 10 |
| trousers | 3 | 0.555 | 2.667 | 156 | 3 | 0.556 | 2.667 | 194 | 3 | 0.555 | 2.667 | 197 | 3 | 0.555 | 2.667 | 147 | 3 | 0.556 | 2.667 | 241 |
| shapes0 | 7 | 0.271 | 6.762 | 71 | 7 | 0.271 | 6.762 | 58 | 7 | 0.287 | 6.524 | 36 | 7 | 0.271 | 6.762 | 60 | 7 | 0.271 | 6.762 | 66 |
| shapes1 | 7 | 0.282 | 6.476 | 181 | 7 | 0.282 | 6.476 | 166 | 6 | 0.367 | 5.905 | 92 | 7 | 0.282 | 6.476 | 171 | 7 | 0.281 | 6.570 | 172 |
| shirts | 8 | 0.518 | 7.844 | 160 | 8 | 0.514 | 7.849 | 174 | 8 | 0.517 | 7.866 | 195 | 8 | 0.518 | 7.844 | 136 | 8 | 0.514 | 7.891 | 167 |
| dighe2 | 1 | 0.823 | 0.952 | 8 | 1 | 0.823 | 0.952 | 11 | 1 | 0.823 | 0.952 | 14 | 1 | 0.823 | 0.952 | 9 | 1 | 0.823 | 0.951 | 9 |
| dighe1 | 2 | 0.368 | 1.333 | 9 | 2 | 0.368 | 1.333 | 7 | 2 | 0.368 | 1.333 | 8 | 2 | 0.368 | 1.333 | 8 | 2 | 0.368 | 1.333 | 10 |
| fu | 4 | 0.443 | 3.571 | 1 | 4 | 0.443 | 3.571 | 1 | 4 | 0.454 | 3.429 | 1 | 4 | 0.443 | 3.571 | 1 | 4 | 0.443 | 3.571 | 2 |
| han | 3 | 0.387 | 2.203 | 32 | 3 | 0.387 | 2.203 | 27 | 3 | 0.376 | 2.290 | 34 | 3 | 0.387 | 2.203 | 27 | 3 | 0.391 | 2.290 | 27 |
| jakobs1 | 4 | 0.570 | 3.333 | 54 | 4 | 0.567 | 3.250 | 52 | 4 | 0.552 | 3.417 | 53 | 4 | 0.570 | 3.333 | 45 | 4 | 0.567 | 3.250 | 52 |
| jakobs2 | 4 | 0.401 | 3.250 | 36 | 4 | 0.401 | 3.250 | 35 | 4 | 0.412 | 3.333 | 40 | 4 | 0.401 | 3.250 | 36 | 4 | 0.401 | 3.251 | 36 |
| mao | 3 | 0.271 | 2.328 | 20 | 3 | 0.271 | 2.328 | 19 | 3 | 0.295 | 2.171 | 23 | 3 | 0.271 | 2.328 | 18 | 3 | 0.243 | 2.518 | 19 |
| poly1a | 2 | 0.308 | 1.536 | 36 | 2 | 0.300 | 1.598 | 25 | 2 | 0.308 | 1.536 | 39 | 2 | 0.308 | 1.536 | 35 | 2 | 0.300 | 1.597 | 25 |
| poly2a | 4 | 0.339 | 3.308 | 79 | 4 | 0.339 | 3.308 | 63 | 4 | 0.367 | 3.308 | 60 | 4 | 0.339 | 3.308 | 64 | 4 | 0.343 | 3.308 | 59 |
| poly3a | 5 | 0.419 | 4.705 | 184 | 5 | 0.426 | 4.632 | 137 | 5 | 0.414 | 4.790 | 146 | 5 | 0.419 | 4.665 | 165 | 5 | 0.424 | 4.682 | 137 |
| poly4a | 7 | 0.392 | 6.386 | 269 | 7 | 0.413 | 6.308 | 244 | 7 | 0.400 | 6.328 | 216 | 7 | 0.392 | 6.386 | 216 | 7 | 0.398 | 6.308 | 218 |
| poly5a | 8 | 0.443 | 7.850 | 441 | 8 | 0.447 | 7.667 | 362 | 8 | 0.445 | 7.958 | 387 | 8 | 0.449 | 7.905 | 370 | 8 | 0.446 | 7.950 | 339 |
| poly2b | 4 | 0.389 | 3.436 | 78 | 4 | 0.379 | 3.462 | 64 | 4 | 0.368 | 3.526 | 79 | 4 | 0.389 | 3.436 | 72 | 4 | 0.383 | 3.442 | 88 |
| poly3b | 5 | 0.434 | 4.582 | 148 | 5 | 0.430 | 4.564 | 115 | 5 | 0.433 | 4.581 | 133 | 5 | 0.432 | 4.582 | 170 | 5 | 0.441 | 4.507 | 158 |
| poly4b | 6 | 0.465 | 5.801 | 278 | 6 | 0.463 | 5.848 | 245 | 7 | 0.383 | 6.308 | 207 | 6 | 0.465 | 5.801 | 277 | 6 | 0.470 | 5.776 | 265 |
| poly5b | 7 | 0.478 | 6.936 | 386 | 7 | 0.476 | 6.826 | 488 | 7 | 0.477 | 6.933 | 439 | 7 | 0.478 | 6.936 | 368 | 7 | 0.476 | 6.826 | 488 |
| swimm | 5 | 0.397 | 4.891 | 311 | 5 | 0.401 | 4.870 | 227 | 5 | 0.392 | 4.907 | 365 | 5 | 0.397 | 4.891 | 238 | 5 | 0.401 | 4.870 | 232 |
| Av. | 4.870 | 0.425 | 4.476 | 129.000 | 4.870 | 0.425 | 4.458 | 119.391 | 4.870 | 0.427 | 4.464 | 121.391 | 4.870 | 0.425 | 4.477 | 115.652 | 4.870 | 0.425 | 4.485 | 124.174 |

**Table 11:** *Results of the different constructive algorithms applied on each instance in Nest-LB*

| | SCH | | | | BPGD | | | | FF | | | | PBP | | | | TPS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | N | F | K | T | N | F | K | T | N | F | K | T | N | F | K | T | N | F | K | T |
| albano | 2 | 0.387 | 1.472 | 51 | 2 | 0.388 | 1.533 | 53 | 2 | 0.388 | 1.533 | 55 | 2 | 0.387 | 1.472 | 45 | 2 | 0.383 | 1.500 | 70 |
| shapes2 | 5 | 0.439 | 4.800 | 25 | 5 | 0.441 | 4.800 | 27 | 5 | 0.449 | 4.700 | 23 | 5 | 0.452 | 4.700 | 41 | 5 | 0.439 | 4.799 | 22 |
| trousers | 2 | 0.445 | 1.500 | 504 | 2 | 0.437 | 1.500 | 635 | 2 | 0.435 | 1.500 | 578 | 2 | 0.445 | 1.500 | 433 | 2 | 0.450 | 1.500 | 480 |
| shapes0 | 4 | 0.290 | 3.393 | 61 | 4 | 0.290 | 3.393 | 54 | 4 | 0.314 | 3.393 | 35 | 4 | 0.290 | 3.393 | 49 | 4 | 0.291 | 3.393 | 53 |
| shapes1 | 4 | 0.302 | 3.286 | 134 | 4 | 0.302 | 3.286 | 108 | 4 | 0.317 | 3.214 | 86 | 4 | 0.302 | 3.286 | 103 | 4 | 0.285 | 3.428 | 86 |
| shirts | 4 | 0.644 | 3.926 | 554 | 4 | 0.640 | 3.935 | 708 | 4 | 0.642 | 3.928 | 501 | 4 | 0.644 | 3.926 | 456 | 5 | 0.498 | 4.269 | 715 |
| dighe2 | 1 | 0.260 | 0.590 | 14 | 1 | 0.260 | 0.587 | 26 | 1 | 0.260 | 0.587 | 21 | 1 | 0.260 | 0.587 | 21 | 1 | 0.260 | 0.587 | 42 |
| dighe1 | 1 | 0.329 | 0.835 | 43 | 1 | 0.329 | 0.835 | 37 | 1 | 0.329 | 0.835 | 35 | 1 | 0.329 | 0.835 | 35 | 1 | 0.329 | 0.833 | 19 |
| fu | 2 | 0.505 | 1.681 | 23 | 2 | 0.505 | 1.681 | 25 | 2 | 0.505 | 1.681 | 22 | 2 | 0.505 | 1.681 | 18 | 2 | 0.505 | 1.680 | 25 |
| han | 2 | 0.300 | 1.217 | 65 | 2 | 0.300 | 1.217 | 57 | 2 | 0.300 | 1.217 | 52 | 2 | 0.300 | 1.217 | 53 | 2 | 0.311 | 1.217 | 51 |
| jakobs1 | 2 | 0.608 | 1.763 | 72 | 2 | 0.609 | 1.875 | 64 | 2 | 0.608 | 1.812 | 49 | 2 | 0.608 | 1.763 | 67 | 2 | 0.609 | 1.875 | 64 |
| jakobs2 | 2 | 0.450 | 1.875 | 71 | 2 | 0.450 | 1.875 | 70 | 2 | 0.436 | 1.933 | 69 | 2 | 0.450 | 1.875 | 71 | 2 | 0.456 | 1.812 | 79 |
| mao | 2 | 0.231 | 1.346 | 29 | 2 | 0.231 | 1.346 | 32 | 2 | 0.231 | 1.346 | 29 | 2 | 0.231 | 1.346 | 30 | 2 | 0.213 | 1.319 | 35 |
| poly1a | 1 | 0.355 | 0.907 | 80 | 1 | 0.355 | 0.907 | 96 | 1 | 0.355 | 0.907 | 79 | 1 | 0.355 | 0.907 | 96 | 1 | 0.355 | 0.856 | 128 |
| poly2a | 2 | 0.369 | 1.723 | 194 | 2 | 0.371 | 1.744 | 233 | 2 | 0.366 | 1.711 | 151 | 2 | 0.369 | 1.723 | 198 | 2 | 0.366 | 1.770 | 299 |
| poly3a | 3 | 0.380 | 2.634 | 330 | 3 | 0.372 | 2.701 | 378 | 3 | 0.373 | 2.687 | 335 | 3 | 0.380 | 2.634 | 269 | 3 | 0.383 | 2.544 | 343 |
| poly4a | 4 | 0.382 | 3.521 | 682 | 4 | 0.385 | 3.481 | 581 | 4 | 0.384 | 3.500 | 511 | 4 | 0.382 | 3.521 | 538 | 4 | 0.377 | 3.588 | 666 |
| poly5a | 5 | 0.405 | 4.299 | 911 | 5 | 0.398 | 4.348 | 812 | 5 | 0.400 | 4.402 | 807 | 5 | 0.405 | 4.299 | 888 | 5 | 0.404 | 4.273 | 869 |
| poly2b | 2 | 0.442 | 1.898 | 217 | 2 | 0.445 | 1.917 | 239 | 2 | 0.439 | 1.925 | 236 | 2 | 0.442 | 1.898 | 223 | 2 | 0.448 | 1.873 | 180 |
| poly3b | 3 | 0.396 | 2.590 | 255 | 3 | 0.408 | 2.472 | 319 | 3 | 0.395 | 2.565 | 293 | 3 | 0.396 | 2.590 | 260 | 3 | 0.403 | 2.651 | 285 |
| poly4b | 4 | 0.389 | 3.265 | 467 | 4 | 0.398 | 3.269 | 682 | 4 | 0.400 | 3.231 | 593 | 4 | 0.389 | 3.255 | 474 | 4 | 0.393 | 3.270 | 689 |
| poly5b | 4 | 0.472 | 3.754 | 1010 | 4 | 0.464 | 3.828 | 1157 | 4 | 0.467 | 3.853 | 938 | 4 | 0.472 | 3.754 | 949 | 4 | 0.468 | 3.860 | 1063 |
| swimm | 3 | 0.364 | 2.593 | 912 | 3 | 0.375 | 2.557 | 797 | 3 | 0.356 | 2.657 | 1045 | 3 | 0.364 | 2.593 | 772 | 3 | 0.364 | 2.613 | 797 |
| Av. | 2.783 | 0.398 | 2.386 | 291.478 | 2.783 | 0.398 | 2.395 | 312.609 | 2.783 | 0.398 | 2.396 | 284.478 | 2.783 | 0.398 | 2.381 | 264.739 | 2.826 | 0.391 | 2.413 | 306.957 |

**Table 12:** *Results of the different constructive algorithms applied on each instance in Jotika*

| | SCH | | | | BPGD | | | | FF | | | | PBP | | | | TPS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | N | F | K | T | N | F | K | T | N | F | K | T | N | F | K | T | N | F | K | T |
| jotika40 | 8 | 0.612 | 7.378 | 23 | 8 | 0.616 | 7.385 | 19 | 8 | 0.628 | 7.299 | 38 | 8 | 0.612 | 7.378 | 25 | 8 | 0.616 | 7.385 | 47 |
| jotika50 | 10 | 0.620 | 9.327 | 41 | 10 | 0.636 | 9.191 | 28 | 10 | 0.602 | 9.526 | 55 | 10 | 0.641 | 9.191 | 48 | 10 | 0.630 | 9.209 | 63 |
| jotika60 | 11 | 0.659 | 10.299 | 43 | 11 | 0.648 | 10.432 | 57 | 11 | 0.625 | 10.598 | 88 | 11 | 0.668 | 10.194 | 53 | 11 | 0.642 | 10.466 | 76 |
| jotika70 | 12 | 0.709 | 11.358 | 78 | 12 | 0.689 | 11.653 | 132 | 12 | 0.689 | 11.748 | 71 | 12 | 0.695 | 11.536 | 81 | 12 | 0.687 | 11.635 | 127 |
| han80 | 10 | 0.711 | 9.286 | 139 | 10 | 0.697 | 9.422 | 104 | 10 | 0.687 | 9.523 | 172 | 10 | 0.711 | 9.286 | 136 | 10 | 0.694 | 9.485 | 247 |
| han100 | 16 | 0.722 | 15.496 | 161 | 16 | 0.719 | 15.462 | 131 | 17 | 0.659 | 16.158 | 132 | 16 | 0.732 | 15.325 | 195 | 16 | 0.710 | 15.648 | 210 |
| han120 | 16 | 0.749 | 15.670 | 174 | 16 | 0.751 | 15.555 | 283 | 17 | 0.692 | 16.127 | 314 | 16 | 0.749 | 15.679 | 259 | 17 | 0.682 | 16.243 | 313 |
| han150 | 22 | 0.780 | 21.520 | 372 | 24 | 0.666 | 23.222 | 552 | 23 | 0.726 | 22.162 | 366 | 22 | 0.778 | 21.526 | 385 | 23 | 0.708 | 23.000 | 778 |
| Av. | 13.125 | 0.695 | 12.542 | 128.88 | 13.375 | 0.678 | 12.79 | 163.25 | 13.500 | 0.664 | 12.892 | 154.5 | 13.125 | 0.698 | 12.514 | 147.75 | 13.375 | 0.671 | 12.884 | 232.625 |

**Table 13:** *Local search on JP1 instances*

| | PBP | PBP-LS1 | PBP-LS2 | BKR |
|---|---|---|---|---|
| A | 0.614 | 0.614 | 0.614 | 0.605 |
| B | 0.781 | 0.858 | 0.878 | 0.929 |
| C | 0.729 | 0.736 | 0.736 | 0.763 |
| D | 0.590 | 0.591 | 0.590 | 0.579 |
| E | 0.522 | 0.525 | 0.529 | 0.412 |
| F | 0.515 | 0.520 | 0.564 | 0.496 |
| G | 0.724 | 0.797 | 0.809 | 0.814 |
| H | 0.766 | 0.868 | 0.810 | 0.928 |
| I | 0.628 | 0.637 | 0.652 | 0.627 |
| J | 0.672 | 0.675 | 0.701 | 0.665 |
| K | 0.761 | 0.763 | 0.763 | 0.718 |
| L | 0.588 | 0.590 | 0.619 | 0.512 |
| M | 0.655 | 0.655 | 0.655 | 0.589 |
| N | 0.517 | 0.520 | 0.668 | 0.503 |
| O | 0.831 | 0.848 | 0.848 | 0.823 |
| P | 0.720 | 0.720 | 0.852 | 0.678 |
| Q | 0.956 | 0.965 | 0.965 | 1.000 |
| R | 0.767 | 0.767 | 0.767 | 0.771 |
| Av. | 0.685 | 0.703 | 0.723 | 0.690 |
| Av.Time | 52 | 129 | 161 | 50 |
| Impv. | | 1.74% | 3.80% | |

**Table 14:** *Local search on JP2 instances .*

| | PBP | PBP-LS1 | PBP-LS2 |
|---|---|---|---|
| A | 0.605 | 0.642 | 0.680 |
| B | 0.731 | 0.752 | 0.760 |
| C | 0.690 | 0.691 | 0.691 |
| F | 0.511 | 0.530 | 0.551 |
| H | 0.730 | 0.766 | 0.828 |
| L | 0.571 | 0.575 | 0.576 |
| M | 0.641 | 0.664 | 0.681 |
| O | 0.774 | 0.786 | 0.788 |
| S | 0.521 | 0.537 | 0.522 |
| T | 0.890 | 0.921 | 0.921 |
| U | 0.737 | 0.764 | 0.761 |
| V | 0.951 | 0.976 | 0.989 |
| W | 0.656 | 0.664 | 0.669 |
| X | 0.593 | 0.607 | 0.632 |
| Y | 0.720 | 0.725 | 0.726 |
| Z | 0.856 | 0.858 | 0.893 |
| Av. | 0.699 | 0.716 | 0.729 |
| Av. Time | 41.669 | 136.163 | 123.344 |
| Impv. | | 1.753 | 3.068 |