UNIVERSITY OF LEEDS

Article:
Bennell, JA orcid.org/0000-0002-5338-2247, Cabo, M and Martínez-Sykora, A (2018) A beam search approach to solve the convex irregular bin packing problem with guillotine cuts. European Journal of Operational Research, 270 (1). pp. 89-102. ISSN 0377-2217

https://doi.org/10.1016/j.ejor.2018.03.029

A beam search approach to solve the convex irregular bin packing problem with guillotine cuts

J.A. Bennell* J.A.Bennell@soton.ac.uk

M. Cabo †‡ marta.cabo@itam.mx

A. Martínez-Sykora * A.Martinez-Sykora@soton.ac.uk

Abstract

This paper presents a two dimensional convex irregular bin packing problem with guillotine cuts. The problem combines the challenges of tackling the complexity of packing irregular pieces, guaranteeing guillotine cuts that are not always orthogonal to the edges of the bin, and allocating pieces to bins that are not necessarily of the same size. This problem is known as a two-dimensional multi bin size bin packing problem with convexirregular pieces and guillotine cuts. Since pieces are separated by means of guillotine cuts, our study is restricted to convex pieces. A beam search algorithm is described, which is successfully applied to both the multi and single bin size instances. The algorithm is competitive with the results reported in the literature for the single bin size problem and provides the first results for the multi bin size problem.

Keywords: cutting; packing; heuristics; beam search; guillotine cuts.

1 Introduction

This paper tackles the two-dimensional (2D) multi bin size bin packing problem with irregular pieces and guillotine cuts (MBSBPPGC) and can also solve the single bin size version of the problem (SBSBPPGC). Guillotine cuts arise due to the cutting process of certain materials, where cuts are restricted to extend from one edge of the stock sheet to another. Research on guillotine cuts can be divided into two main groups: one where all the pieces are rectangular, thus the obvious cuts are orthogonal to the stock sheet; and the other where pieces are irregular convex polygons. For this case, allowing both orthogonal and non-orthogonal

*Business School, University of Southampton, Southampton, SO17 1BJ, UK

†Department of Mathematics, ITAM, Rio Hondo 1, Col Progreso Tizapán, 01080, Mexico City, Mexico

‡Corresponding author

cuts are the best option for minimising the waste. Our research focuses on the second, less studied, version of the problem. The algorithm we present in this research successfully solves two versions of the problem; the multi bin size and the single bin size problem. This extends the literature in this problem, which only tackles the single bin size version.

The 2D-cutting and packing problem with guillotine cuts and rectangular pieces was first introduced by Gilmore and Gomory (1965) and has been widely studied ever since. Lodi et al. (1999b) surveys 2D rectangle bin packing, including algorithms that handle guillotine constraints. Their approach consists of packing pieces onto shelves along the width of the bin. Lodi et al. (1999a, 2015, 2017) consider the case where no rotation is allowed. Charalambous and Fleszar (2011) use an alternative approach and generate patterns across the width of the bin, and then within the free rectangle areas. Pieces may be rotated in order to maximise the area of the biggest free rectangle. Fleszar (2013) propose a constructive heuristic where the insertion decision is made by first-fit, best-fit or critical-fit criteria. Note that in most cases, when dealing with rectangle cutting the number of transitions between horizontal and vertical cuts, called stages, is restricted. At each stage several guillotine cuts can be performed. The problem is known as the n-stage two dimensional bin packing problem, see Puchinger and Raidl (2007), Alvelos et al. (2009) or Malaguti et al. (2014).

When cutting more complex shapes, cutting problems face new challenges due to the geometry of the pieces. The problem is then known as an irregular packing problems. For many years, the literatures focused on the open dimension version of the problem, called irregular strip packing problems. A useful review of methods can be found in Bennell and Oliveira (2009). Moreover, the solution approaches were largely heuristic. More recently researchers have been developing exact methodologies for these problems. One of the recent exact methods for irregular strip packing is the one from Cherri et al. (2016) where they apply a mixed integer linear programming model. Their model builds on the linear programming compaction approaches (for example Bennell and Downsland (2001)) and adds binary variables to activate and deactivate constraints. Larger problems are solved by Rodrigues and Toledo (2017) where their integer programming model approximates the stock sheet by discrete points. A compromise between the two approaches is proposed by Leao et al. (2016) who discretise the stock sheet in the y-axis but allow continuous translations in the x-direction.

A further advancement in the literature is the consideration of multiple fixed dimension stock sheets; irregular bin packing. This is the problem solved by Martinez-Sykora et al. (2017) using exact and heuristic methods. This problem is also tackled by Abeysooriya et al. (2018) who apply a pure heuristic. Both these papers allow any angle of rotation of the pieces. The irregular packing literature provide important findings to consider in our algorithm design, but we point out that none of these papers consider guillotine constraints.

Han et al. (2013) introduce the irregular bin packing problem with guillotine cuts into the literature, and propose a one-stage approach that matches pieces into clusters, enclosed in their convex hull to create a block. The clusters are built using a forest search structure where the forest is populated by matching pieces

with pieces, pieces with clusters, and clusters with clusters. The forest is complete once there are no further matches for the blocks. The bin packing sequentially selects the block with the greatest summed area of pieces, while removing blocks that contain common pieces to those already selected.

Martínez-Sykora et al. (2015) is the only other paper, to the best of our knowledge, that tackles the problem considered in this research. They use a metaheuristic where the relative position of pieces in the bin and guillotine constraints are calculated via mixed integer programming (MIP). The MIP consists of containment constraints, non-overlapping constraints and guillotine cut constraints. It not only determines whether a piece can feasibly be inserted into the bin, but also where to place it. Every time a new piece is inserted, the MIP may change the position of the pieces already in the bin, respecting the guillotine cut structure defined in previous steps. The formulation assumes that pieces are orientated at a certain angle. Hence, the model is solved several times for each piece to be inserted, trying different rotation angles and the reflection of the piece. Pieces are sorted by a certain criterion, and if the next piece in the list does not fit in the current bin, the algorithm tries to insert any of the remaining pieces before opening a new bin. Clearly, since they solve a MIP for each piece inserted in a bin, and the number of constraints grows with the number of pieces in the bin, this heuristic slows down when dealing with instances where there are many pieces per bin.

The authors present new lower bounds, and some of the results obtained are proven to be optimal, improving those on Han et al. (2013). The lower bounds are determined by a simple MIP model that minimises the number of bins needed to allocate all the pieces where only the area of the pieces and the area of the bin are considered as the packing constraint.

The contributions of this paper are as follows. It is the first paper that solves a multi bin size (MBS) bin packing problem with convex irregular pieces and guillotine constraints. It also presents an effective beam search algorithm which obtains fast and competitive solutions without the aid of specialist software, so it can provide small businesses a tool for their cutting operations. Our method produces better results for the single bin size (SBS) problem than those shown on Han et al. (2013), and the results are competitive when compared with those obtained in Martínez-Sykora et al. (2015) while having shorter execution times.

The remainder of the paper is structured as follows: Section 2 describes the problem in detail, and introduces the relevant notation that is used in this paper. Section 3 describes the beam search heuristic. First it gives a general overview of the method and, in Section 3.2, it focuses on how the method is applied to solve this particular problem. Sections 4 and 5 detail the main steps of the beam search heuristics, including how a node is generated, and how a global solution is constructed. Computational results are shown in Section 6. The paper ends with summary and conclusions in Section 7.

2 Problem Description

The problem objective is to cut a set of pieces from the minimum number of stock sheets, hence it is an input minimization problem. There are sufficient rectangular stock sheets available to meet the demand, and these may be of different sizes. Let $\mathcal{P}$ be the demand set of pieces, and each piece is considered to be unique. According to the typology proposed by Wäscher et al. (2007) this is a multiple bin size (MBS) bin packing problem. Further refinements are that all pieces are convex, usually irregular, can be freely rotated and reflected and only guillotine cuts are allowed.

A guillotine cut is defined as a straight line cut that begins at an edge of the stock sheet and ends at another edge. A cut divides the stock sheet creating two component stock sheet with boundaries that can define the start and end of the next guillotine cut. In our case, the cuts are not constrained to be parallel to the edges of the stock sheet. Usually when considering guillotine constraints, pieces must be cut free from the stock sheet with a maximum number of cuts; in our problem there are no limits to the number of cuts. The order of the cuts is important to track in order to successfully execute the cutting plan.

Let $\mathcal{B}$ be the set of bins, and $B \in \mathcal{B}$ denote a particular bin with width W_B and length L_B. We consider T different bin sizes. Let $\mathcal{P}$ be the set of pieces, where $p \in \mathcal{P}$ denotes a piece, which is characterized by an ordered list of vertices $(v_1, \ldots, v_{n_p})$, and let the edges be expressed by $e_j = (v_j, v_{j+1})$, where $j = 1, \ldots, n_p - 1$ and the last edge is $e_{n_p} = (v_{n_p}, v_1)$. Each piece can be freely rotated and reflected.

The objective is to maximise the total bin utilization (U), which is equivalent to minimising total waste. Utilization can be calculated as:

$$U = \frac{\sum_{p \in \mathcal{P}} A_p}{\sum_{B \in \mathcal{B}} A_B} \tag{1}$$

where A_p denotes the area of piece p and A_B is the area of bin B. In practice residual material of a partially packed bin can be reused in subsequent operations. The residual appears when practitioners perform a horizontal or vertical cut to a bin, to separate the packed and unpacked areas. We assume that only the last bin may have a residual. Let L_R be the length of the packed area, and W_R its corresponding width, then the area of the last bin is either $L_R \cdot W_B$ or $L_B \cdot W_R$, whichever is smallest.

3 Beam Search

In this section we explain in detail the beam search heuristic to solve the 2D MBSBPPGC. This heuristic uses a tree search structure of nodes and branches analogous to branch and bound, but only a subset of nodes is evaluated in the search tree. At any level, only the nodes considered to be promising are kept for further branching and the remaining nodes are pruned permanently. Its structure

lends itself to modeling problems where the solution of the problem may be constructed sequentially.

Beam search was first applied for scheduling problems in Sabuncuoglu and Bayiz (1999) and Ghirardi and Potts (2005). More recently, it has been applied to cutting and packing problems as in Akeb et al. (2009) and Akeb et al. (2011) to solve the circular packing problem, or in Bennell and Song (2010) where an irregular shape packing problem is solved by means of beam search. In both cases, they only deal with the packing of one bin, and the beam search will find the best sequencing of pieces based on a placement procedure. To the best of our knowledge, within the field of cutting and packing problems, only Song and Bennell (2014) use beam search when there are multiple bins. They tackle a 2D cutting stock problem. Beam search is used as part of a column generation procedure to pack the pieces into a single bin but not as the main procedure that deals with multiple bins.

In the next sections we present a general overview of this heuristic followed by a description of how it has been applied to our particular problem.

3.1 General Framework

The algorithm alternates between searching the tree breadth first and then depth first, with aggressive pruning of the branches at each stage according to a local and global evaluation function respectively. There is no backtracking and the number of branches at each level of the tree is user defined. Hence, the user somewhat controls the running time of the algorithm and it is polynomial in the size of the problem (Sabuncuoglu and Bayiz, 1999). Below we describe in detail the main ideas of the beam search, and give a general idea of how this procedure has been implemented to solve our particular problem.

Beam search searches the tree from each parent node, creating multiple children. A filtering method discards some children from each parent to reduce the computational burden while ensuring that only promising children are kept for evaluation. The number of children kept at this stage is called the *filter width* (α), and nodes are selected by a *local evaluation function* that only takes into account the performance of the partial solution represented by that node, without considering its impact on the final solution. For each of the filtered nodes, a *global evaluation* is performed by constructing a final solution of the problem, with initial conditions set by the filtered nodes. In our problem, it corresponds to a complete packing scheme, where a subset of pieces have already been used to pack some bins. All of the complete solutions are compared, and only β nodes with the best value for the global evaluation function are kept for further investigation. These nodes are called *beam nodes*, and the value β is known as the *beam width*. Once the global evaluation function is performed, and the β nodes are selected, the search returns to the partial solutions at the local evaluation level and the β nodes become the new parents. Note that when performing the local evaluation, child nodes will only compete with other children branching from the same parent node. However, for global evaluation all children are compared with each other.

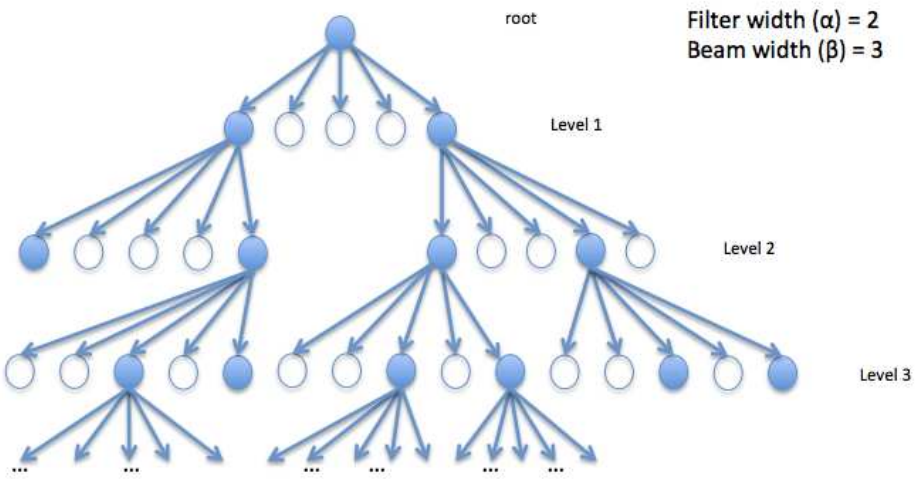

Figure 1: Illustration of a beam search tree

In Figure 1 we illustrate a beam search tree for the SBS problem. The MBS problem beam search tree, only differs in that it will contain T times more child nodes, where T is the number of bin sizes. In this explanatory tree, there are 5 possible children (for the MBS problem, there would be $5 \cdot T$ children). From each parent node, the best two are selected, $\alpha = 2$ (shaded nodes), and the remaining three are pruned. The filtered nodes are globally evaluated and the best $\beta = 3$ across all branches are kept. Usually $\alpha < \beta$ to ensure that at least one node from a different parent is selected, aiming for diversity in the subsequent nodes. Note that the global evaluation is not needed for pruning branches until there are at least a total of β filter nodes on a level. Clearly, the larger the value of α and β the greater the computational time required and also the greater the potential of finding better solutions.

3.2 Overview of Beam Search for the proposed problem

In our implementation of beam search, each node represents a complete packed bin. A child node is created by first selecting a bin size. A constructive algorithm packs the bin, which is deterministic with respect to the selection of the first piece and its orientation. We can create different patterns in the same size bin by changing the orientation of the first piece, or selecting a different piece. Hence for each bin size, multiple children are created from the same parent that can be divided in two groups: those with a different initial piece, and among them, children with a different orientation of the first piece.

For the local evaluation the obvious function would be maximizing the bin utilization, which works well when solving SBS instances. However, when dealing with MBS instances, this approach may be too greedy. Instead, the filter width

takes into account the number of different bin sizes, T, that are available and, for each bin size, it selects α nodes as child nodes. This way nodes are only competing among their own bin type.

As reported in Bennell and Oliveira (2009), most construction algorithms share a strategy of placing difficult to place pieces early in the process, such as large pieces so that small ones can fill the gaps. In order to encourage this behaviour the local evaluation function (F) computes the ratio between the squared area of the pieces on the bin, and the area of the bin as in (2), where $P(B)$ denotes the set of pieces placed in bin B:

$$F = \frac{\sum_{p \in P(B)} A_p^2}{A_B} \tag{2}$$

It then selects the set of nodes with maximum F among all children with the same initial piece; and among these children it finally selects the best α children, using the same criteria. In case of a tie, the algorithm choses the node with more pieces in it. Since this is done for each bin size, each parent keeps $\alpha \cdot T$ children.

Once the filter nodes are selected, the beam search creates a global solution for each of them. For a given node the global solution packs the remaining unpacked pieces bin by bin by generating one bin of each size using the same construction algorithm. The algorithm selects the largest unpacked piece that fits for each bin size as the first piece. The bin with maximum value of F is kept and the algorithm repeats this procedure until all pieces are placed. Hence, the global solution will reflect the advantages of having different bin sizes.

Global evaluation identifies the greedy decisions made at the filter stage that leads to suboptimal complete solutions. It keeps the β branches that lead to the best global solution previously created, discarding all remaining branches. Our implementation of the beam search algorithm uses U, equation (1), as the global evaluation function.

One characteristic of beam search is that, even though we construct complete solutions at each level of the tree, these solutions are discarded and only the information of which nodes led to the best values of the objective function is kept. However, we keep track of the best solution found so far, which allows us to validate the beam search by confirming that the best solution improves as the search progress.

Thus the beam search for the 2-dimensional multiple bin size bin packing problem with guillotine cuts can be summarize as follows:

Step 1: *Generation of nodes.* Order pieces in descending order of their area. Select the first M pieces from set $P \subset \mathcal{P}$, set of unplaced pieces. For each $p_j \in M$ apply r orientations (rotations and reflections). Let p_j^k be the k-th orientation of p_j. For each of the T bin sizes, generate $M \cdot r$ child nodes, with p_j^k as the initial piece. For each new node, update P.

Step 2: *Filtering.* For each of the T bin sizes, select the best α children according to F. Prune the remaining nodes for that bin size. Go to Step 1 until there are at least β nodes at the current level.

Step 3: *Global solution generation.* From each filtered child, construct a global solution.

Step 4: *Global evaluation.* Apply the global evaluation function, U, and keep the best β branches. Prune the remaining ones.

Step 5: Repeat steps 1 - 4 until all branches have placed all pieces. At the final level of each branch, calculate the residual to obtain the utilization of that branch. Select the branch with best overall utilization.

4 Generation of Nodes

In this section we describe in detail how the nodes are generated. As stated before, each node of the tree represents a packed bin. During the beam search procedure, from each node, multiple children are created. To ensure diversity of the proposed solutions, we must ensure that they are different and distributed across the solution space.

We follow the findings of Martínez-Sykora et al. (2015), who show that sorting pieces in non-increasing area provides better solutions for this problem. The packing procedure is deterministic in the sense that we obtain the same solution if the first piece and its orientation are the same. Thus, to create different children from each parent node, a subset of pieces is selected as the initial piece for each child and, for each of the initial pieces selected, children are created with different rotations and reflections of this initial piece.

The heuristic to create child nodes will be evaluated in terms of bin utilization, thus it tries to fill the bin as much as possible. However, guillotine constraints limit available positions for pieces. In this paper we define two types of guillotine cuts. The first one follows the ideas of the *best match* found in Han et al. (2013), where a guillotine cut is defined by the concurrent edges when matching two edges of two convex shapes. It is guaranteed that the matching edge is a valid guillotine cut, however, the limits of this cut are not defined by the edges of the bin, but by the convex hull of the two pieces. The second cut we define, we call a *separation cut* and it follows the ideas found in Martínez-Sykora et al. (2015) where a guillotine cut is performed across the bin to separate the bin into two sections. Combining both methods ensures a better utilization of the bin.

Figure 2 shows the advantages of combining these two cuts. Pieces p_1 and p_2 are placed using the best match method. The edges of the convex hull defined by p_1 and p_2 set the limits of the guillotine cut as shown in Figure 2(a). In the case where p_3 is the next piece, this cannot be placed in the bin by matching the edges of p_3 with the convex hull because it will break the containment constraint of the bin. Piece p_3 can only fit in the bin if it is placed beyond the right-hand boundary of the top edge of the convex hull, which is not permitted by this approach. Alternatively, if only separation cuts were defined, the guillotine cut creates two sections, as shown in Figure 2(b), which also prevents piece p_3 being placed. Combining the two types of cut, allows us to separate pieces p_1 and p_2 by a best match cut (g_1), whose limits are the edges of the convex hull, and then

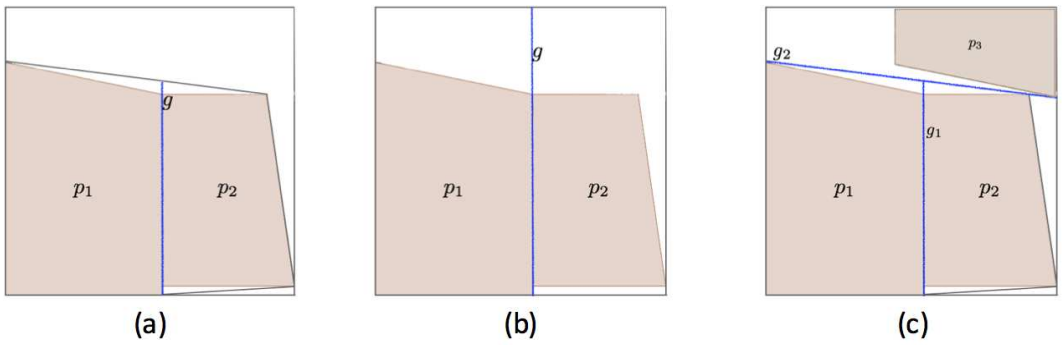

Figure 2: Advantages of combining two methods to place pieces in the bin.

separate p_3 by a separation cut (g_2) as shown in Figure 2(c) allowing us to place piece p_3.

Placement heuristics usually order pieces under a certain criterion and try to place a piece following that order. Our approach uses a more dynamic selection of pieces, which draws the next piece to be placed from a *restricted candidate list*, Q. The candidate list is a dynamic ordered subset of L pieces that are eligible to be placed next. It is more efficient than forcing a certain piece to be placed next, since it takes into account the particular features of the bin as well as the pieces that are already placed. To create the candidate list, we take into account the free area in the bin, and also the quality of the match between the edges of the convex hull of the pieces already placed. The *match degree*, introduced by Han et al. (2013), is a measure of the similarity between the edges of the pieces we are comparing. Let $md(m, n)$ be the match degree between edge e_m of the candidate piece and edge e_n of the partial solution:

$$md(m,n) = 1 - \frac{||e_m| - |e_n||}{\max\{|e_m|, |e_n|\}} \tag{3}$$

where $|e_m|$ and $|e_n|$ represent the length of edges e_m and e_n respectively. The greater the value of $md(m, n)$ the more similar edges e_n and e_m are in length. We only accept a piece as a candidate, if the match degree is greater than a threshold ($md(m, n) \geq \theta_1$), based on the idea that the more similar the edges are, the less waste the convex hull will generate. Algorithm 1 describes how the candidate list is created.

Note that within this algorithm, line 4 refers to the area of S_i^B, a section of a bin which is created after a separation cut is applied. Initially the first section we consider is the entire bin ($S_1^B = B$). $C_{S_i^B}$ represents the convex hull of the pieces already placed in section S_i^B. Next we describe the algorithms to find the best match, or the best separation cut. The final procedure will select one of these two methods to place one of the candidates.

Algorithm 1 Candidate List

$Q = \varnothing$
while $|Q| < L$ **do**
  for all $p \in P$ **do**
    if $Area(C_{S_i^B}) + Area(p) \leq Area(S_i^B)$ **then**
      for all $e_m \in p$ **and** $e_n \in C_{S_i^B}$ **do**
        if $md(m, n) > \theta_1$ **then**
          $Q = Q \cup \{p\}$
        end if
      end for
    end if
  end for
end while

4.1 Finding the Best Match

The function that finds the best match between two polygons is partially derived from the *best match* function found in Han et al. (2013). It is based on the fact that, because of the guillotine cut constraint, only convex shapes can be cut. When matching two edges of two convex shapes, the concurrent edge is guaranteed to be a valid guillotine cut. However, the resulting shape might not be convex, so an immediate waste is generated. The convex hull is used as an approximation of the union of the two polygons, p_1 and p_2. The utilization of the convex hull is defined as:

$$U_{cov}^{(p_1,p_2)} = \frac{Area(p_1) + Area(p_2)}{Area(C(p_1, p_2))} \tag{4}$$

where $C(p_1, p_2)$ represents the convex hull of pieces p_1 and p_2 respectively. Han et al. (2013) packing strategy first clusters pieces and then attempts to fit those clusters into bins, so they look for matches that are more similar to a rectangle shape. Thus they also consider the rectangle enclosure utilization, defined in a similar way as the convex hull utilization, as a criteria for matching. They use a weighted averaged utilization ratio of the two to define the best match. To release pieces from the bin, guillotine cuts must be performed in reverse order to the one created by the algorithm, meaning that the first cut will separate the last piece added to the cluster, and the last cut releases the first and second pieces placed. The main limitation with their procedure is that, because pieces are first clustered outside the bin, when placed into it, an immediate waste is generated between the big block of pieces and the edges of the bin.

Our heuristic also matches pieces together, but since pieces are sequentially placed in the bin, we do not need to consider the rectangle enclosure because the boundary of the bin is taken into account when evaluating the feasibility of a match. When we place a piece in the bin, we do not modify its position, so rather than considering the convex hull utilization of two pieces matched together we consider the convex hull of all the pieces already placed in the bin. Let us denote

by C_B the convex hull of all the pieces already placed in the bin, and $C(C_B, p)$ the convex hull formed after placing piece p in bin B. The convex hull utilization will be defined as:

$$U^p_{C_B} = \frac{Area(C_B) + Area(p)}{Area(C(C_B, p))}. \tag{5}$$

Once a piece p is matched with C_B, we can determine its convex hull utilization. However, the convex hull generated when matching two polygons is strongly dependent on the relative position of the two polygons as shown in Figure 3, where different convex hulls $C(C_B, p)$ are created depending on the rotation of p, even if they are both matched with the same edge of C_B.

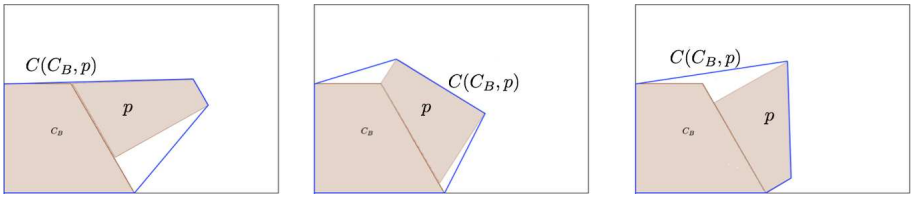

Figure 3: Convex hulls created when matching polygons C_B and different positions of p.

To match two pieces together we use the two heuristics that were first introduced in Han et al. (2013): *Attach* and *Slide*. The *Attach* procedure places edge e_n of p concurrent with edge e_m of C_B. Initially, the starting vertex of e_m is matched with the end vertex of e_n. The *Slide* procedure defines a finite number of points along the edge e_m distance ϵ apart, where the quality of the match is tested.

Since C_B is already fixed in the bin, we can find all edge matches by rotating and translating p. Before sliding along a certain edge, we perform two tests: one is for containment feasibility, which means that we only perform these operations if the resulting polygon lies entirely inside the bin; the second is to check the match degree $md(m, n)$. The match degree was defined in equation (3) and used to restrict the candidate list. Recall that we ensure that candidates have at least one edge with $md(m, n) > \theta_1$; however, the corresponding match may not lie entirely in the bin. Since we still want matching edges with similar length, to minimise the waste of the convex hull, we relax the threshold parameter to $\theta_2 < \theta_1$ and proceed to the attach operation if $md(m, n) \geq \theta_2$. Note that it is possible to find the relative position of two polygons such that the area of their convex hull is minimized, by identifying breakpoints in the shape of the convex hull as explained by Grinde and Cavalier (1995). Since we fix the position of one of the polygons and need to respect the boundaries of the bin, this method will frequently identify infeasible configurations.

This procedure is summarize in Algorithm 2. Note that this algorithm only identifies the candidate piece, edge match and slide distance that gives the maximun convex hull utilization, but does not implement this before checking for separation cuts.

Algorithm 2 Select Best Match

$U_C^* = -1$
for all $p \in Q$ **do**
  for all $e_m \in p$ **and** $e_n \in C_{S_i^B}$ **do**
    if $md(m,n) > \theta_2$ **then**
      for $d = 0$ **to** $d = \max\{0, |e_m - e_n|\}$ **step** ϵ **do**
        $Attach(C_{S_i^B}, p, e_n, e_m)$
        $Slide(C_{S_i^B}, p, e_n, e_m)$
        if $C(C_{S_i^B}, p)$ fits in S_i^B **then**
          $U^p_{C_{S_i^B}} = \frac{Area(C_{S_i^B}) + Area(p)}{Area(C(C_{S_i^B}, p))}$
          if $U^p_{C_{S_i^B}} > U_C^*$ **then**
            $p_{best} = p$, $best(n) = e_n$,
            $best(m) = e_m$, $best(d) = d$
            $U_C^* = U^p_{C_{S_i^B}}$
          end if
        end if
      end for
    end if
  end for
end for

4.2 Finding the Best Separation Cut

Once the best utilization of the convex hull is calculated, we explore the utilization of the best separation cut, to then decide which method to use to place the next piece. In this section, we explain how to find the best separation cut.

A different approach to place pieces in a bin, is to perform a guillotine cut that will separate the pieces already placed from the rest of the bin. This type of guillotine cut separates the bin in two sections: S_i^B which contains the pieces already packed, and S_{i+1}^B which is empty, thus we will call it a *separation cut*. Then, we can start placing pieces in section S_{i+1}^B. Each time a separation cut is performed, we need to update the edges and vertices of the sections involved. At the end of the procedure the bin is sectioned and can be obtained as the union of all the sections. All the sections are convex areas, and only have, at most, one edge in common. When the bin has not been divided $B = S_1^B$. To decide whether to apply this kind of cut, we need to take into account the utilization of the packed area, considering the area already in use, and therefore the convex hull of the pieces already placed in that section: $C_{S_i^B}$. A cut will be concurrent with an edge of the convex hull of pieces already placed. Thus, let us define g_i as the separation cut that separates sections S_i^B from S_{i+1}^B by performing the cut along one edge of $C_{S_i^B}$. The utilization of a section when cut g_i is applied can be calculated as:

$$U_{g_i} = 1 - \frac{Area(S_i^B) - Area(C_{S_i^B})}{Area(S_i^B)}. \tag{6}$$

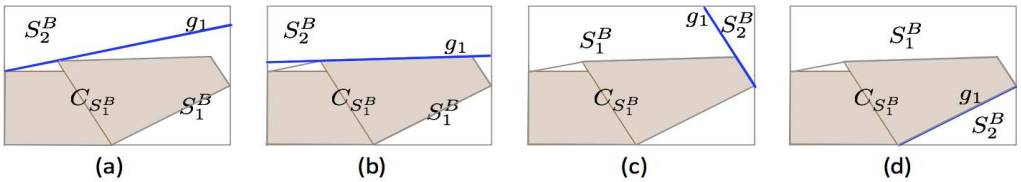

Figure 4: Sections created with different separation cuts from the same convex hull $C_{S_1^B}$.

Figure 4 shows the different sections created when performing each of the possible separation cuts where initially only two pieces are placed. In this figure, we consider the convex hull of the two pieces $C_{S_1^B}$, and for each edge, we perform a cut g_1. Since we do not take into account edges concurrent with the bin, there are four possible cuts. Each cut generates two sections S_1^B and S_2^B. The procedure will keep the separation cut with best utilization of the packed section as described in equation (6). In this particular example this is defined by the cut in Figure 4(b), so the procedure will select this cut. It will compare its utilization with the utilization of the convex hull defined by finding the best match of one of the candidates pieces, as described in the previous section. If the utilization of the separation cut is greater, the procedure will place pieces in S_2^B. Otherwise it places the piece selected in Section 4.1.

Algorithm 3 shows how the procedure to find the best separation cut is implemented. Note that to select the best separation cut, we do not need to consider the piece that we are placing next, but only the utilization of the section we are cutting.

Algorithm 3 Select Best Separation Cut

$U_g^* = -1$
for all $e_m \in C_{S_i^B}$ **do**
    Let $S_{i(m)}^B$ be the section obtained by performing cut g_i along edge e_m.
    $U_{g_i^m} = 1 - \frac{Area(S_{i(m)}^B) - Area(C_{S_i^B})}{Area(S_{i(m)}^B)}$
    if $U_{g_i^m} > U_g^*$ **then**
        $best(g) = g_i^m$
        $U_g^* = U_{g_i^m}$
    end if
end for

4.3 Packing One Bin

We are now in a position to explain in detail the heuristic designed to pack a bin. The packing algorithm takes into account the two functions: best match and best separation cut. It starts by ordering the pieces by non-increasing area. The first piece is placed in the bottom left corner of the bin, with a rotation such that the number of edges concurrent with the bin is maximum. Ties are broken with total edge length concurrent with the bin. There may be degenerate cases where a piece can only fit in a bin in a few orientations that do not include matching edges with the bin. For those cases, we use the original rotation of the piece, assuming it is set with the rotation that will fit.

Using Algorithm 1 we create a candidate list. This is updated every time a new piece is placed in the bin because the free area in the bin is reduced. Another reason for this update, is the match degree as an acceptance criterion. Once the new piece is added, the convex hull of placed pieces changes, and so does its edges. So previous candidates, even if they fit in terms of area, may be discarded for not having a match degree greater than the threshold.

Each candidate is assessed by Algorithm 2, which identifies the best match candidate. We compare the utilization of the best match convex hull and the best separation cut and select the strategy that gives the maximum. If the separation cut is selected, then section S_{i+1}^B will be created and we place the first candidate in the list that fits, with a rotation such that the number of concurrent edges between the piece and the section is maximal. In case of a tie, we consider maximum edge length.

Figure 5 - 8 illustrate this strategy with an example. The first piece, p_1, is placed in the bottom left corner, so that the sum of the lengths of the edges matching the bin is maximum. Then we consider all possible separation cuts g_1 and keep the one with maximum value of U_{g_i} given in (6). In this case, the maximum utilization is given by the cut represented in Figure 5 (b1).

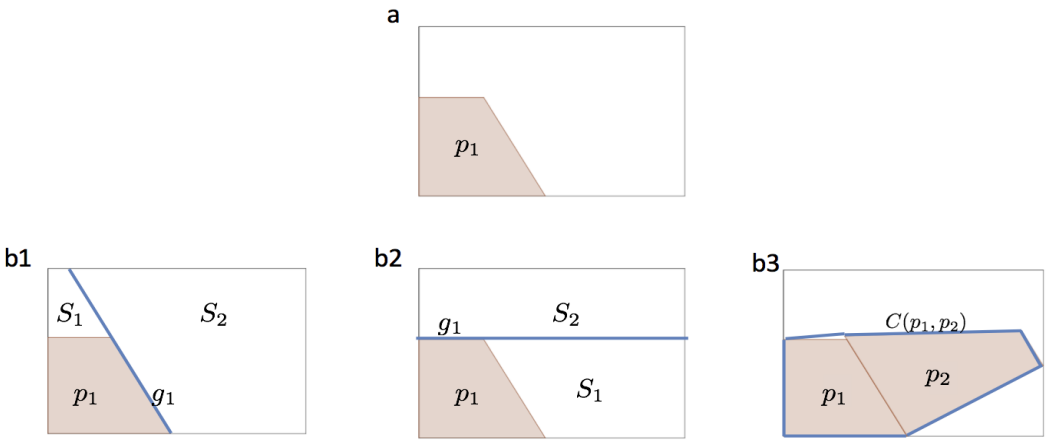

Figure 5: Packing the second piece in a bin.

The algorithm also calculates the maximum convex hull utilization between $C_B = p_1$ and all pieces $p_c \in Q$, $\max U_{C_B}^{p_c}$ using equation (5). It is clear, as shown in (b3), that for this example, there exists a piece such that its convex hull has a greater utilization than performing a guillotine cut.

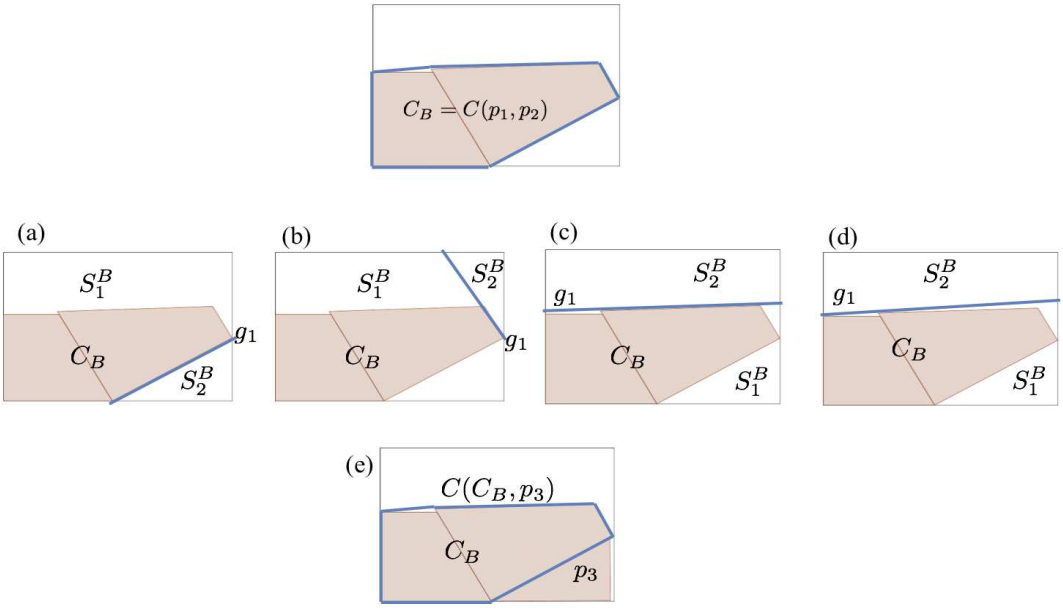

Figure 6: Matching a piece to the convex hull of the pieces already placed.

Now that p_1 and p_2 are matched we consider $C_B = C(p_1, p_2)$, and we can start placing the third piece in the bin. Figure 6 shows how to place the third piece: the second row shows the four possible separation cuts and the sections that arise from them. Only the best separation cut, shown in Figure 6 (c), is kept for comparison against the utilization of the different convex hulls created by the candidates. Figure 6 (e) shows that the best option to place the third piece, is to select $p_3 \in Q$ and match it with the existing convex hull.

Piece p_4 is placed using the same principle and does not result in a separation cut. Note that piece p_4 is a small piece. It is towards the end of the sorted list of pieces, and therefore usually not considered for packing until the very end of the algorithm. Since we are creating the candidate list, a lot of big pieces have been discarded at this point, mainly due to the area constraint, however p_4, apart from being a small piece also has edges of similar size as C_B, as can be seen in Figure 7 (a).

For the next piece, there is only one separation cut as shown in Figure 7 (b). The utilization of this cut is greater than the utilization of the convex hull generated by attaching piece p_5 to C_B. Thus the procedure applies the separation cut, updates the limits for S_1^B and S_2^B and places a candidate piece in S_2^B, as shown in Figure 7 (b). The heuristic continues in this manner until no further candidates will fit in the bin. The final bin is illustrated in Figure 8.

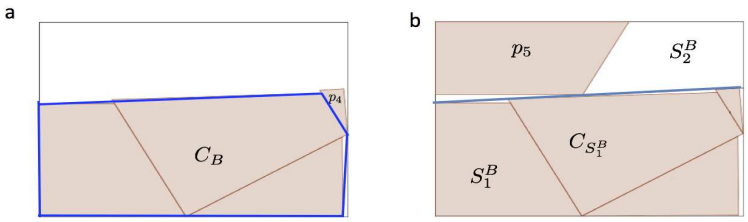

Figure 7: Packing pieces p_4 and p_5.

As a result of using the best match or best separation cut criteria, the order of the guillotine cuts is not straight forward, and depends on which criteria we use to place the piece. In our example, the bin is divided into 5 sections: S_1^B contains pieces p_1, p_2, p_3 and p_4; S_2^B only contains piece p_5; the third section S_3^B contains piece p_6 and sections S_4^B and S_5^B contains pieces p_7 and p_8 respectively. Thus the guillotine cuts must separate first each section, in the order they are created. Then inside each section, guillotine cuts are performed in reverse of the order the pieces are placed, since pieces from the same section are placed using the best match criteria. Figure 8 shows the order in which the pieces are placed with this heuristic; and the order, set by the subscript, in which each guillotine cut must be performed at the end of our example. The guillotine cuts that separate the sections, are numbered with subscript 1 to 4, indicating the order of the cuts. Inside section 1, we need to perform 3 cuts, numbered with superscript 1, indicating the section, and subsubscript 1 to 3 also indicating the order in which the cuts are made.

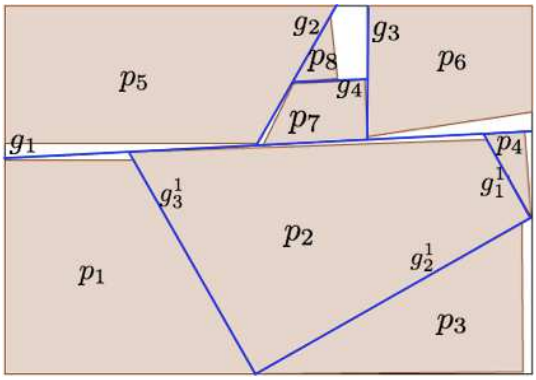

Figure 8: Packing a bin: order in which pieces are placed and the guillotine cuts must be done to separate the pieces.

To summarise the above procedure, let us call P the set of unplaced pieces, $|e_m|$ the length of edge e_m, and Q the set of piece candidates to be placed next. This set is restricted to be of size L. The placing algorithm is as follows:

Placing Procedure:

Step 1: (*Initialization*) Order pieces in descending order of their area, and place the first piece p_1 in the bottom left corner of the bin. Update $P = \mathcal{P} - \{p_1\}$, $i = 1$, $S_i^B = B$ and $C_{S_i^B} = p_1$.

Step 2: (*Create Candidate List*) Apply Algorithm 1 to create a candidate list Q of size L.

Step 3: (*Selection of Best Match*) If $Q \neq \varnothing$ then apply Algorithm 2 to calculate the best utilization of the convex hull, U_C^*. If it is not possible to select any candidate with this criterion, $U_C^* = -1$.

Step 4: (*Selection of the Best Separation Cut*) Apply Algorithm 3 to select the best separation cut with maximum utilization, U_g^*. If it is not possible to perform any separation cut set $U_g^* = -1$.

Step 5: (*Place Candidate*) If $U_C^* \geq U_g^*$ then piece p_{best}, from Algorithm 2, will be attached to the existing convex hull. Else, if it is possible to perform a separation cut and there is a candidate p_c that fits in section S_{i+1}^B, then we update values $P = P - \{p_c\}$ and reset the boundaries of S_i^B and S_{i+1}^B.

Step 6: (*Stopping Criterion*) If there are still pieces to be placed, and sections to be explored go back to Step 2. If all the pieces have been placed or no candidate could be placed with any of the criteria, then the algorithm stops.

The placing heuristic stops for two main reasons: either there are no more pieces to place, or there are no more candidates that fit in the bin. Recall that this procedure is used to generate nodes of the beam search tree. We are interested in nodes with maximum utilization, as this is the objective of the local evaluation. The fact that there are no candidates that fit in the bin, does not mean that no other piece could be placed. So, at the node generation step, once the bin has been filled up to a certain utilization, we apply the procedure with $Q = P$, i.e. all pieces are eligible as candidates.

When selecting the best separation cut, we always move to the next section, discarding empty areas that could still allocate pieces. Thus a final test is done, by dividing each existing section further and trying to place a piece in the new empty areas. With this final test, we ensure that there is no other possibility to place a piece in the bin before declaring it complete and computing its local evaluation function value.

Once child nodes are created and selected through the local evaluation function, beam search constructs the global solution. In the next section we explain how this solution is generated.

5 Global Solution Generation

Once a level of the tree is completed, beam search selects the beam nodes that will determine which branches to keep and explore further, and which ones to discard. To select the beam nodes complete solutions to the problem are constructed, taking the partial solution represented by each node as a starting point. Then, a

global evaluation function will select those child nodes that lead to the best final solution.

The proposed beam search was designed to solve both MBS and SBS problems. The placing procedure described in the previous section can be applied irrespective of the bin size. However, when creating a global solution, we need to differentiate whether we are solving a MBS or a SBS problem, as there are features for each problem that can be exploited. In the following sections we explain how the solution is constructed for each problem.

5.1 Multi Bin Size Bin Packing Problem

Recall that, in Section 3.2 we explain that the global solution for the MBS problem, packs pieces bin by bin. At each stage of the procedure it generates a bin from each bin size, using the construction algorithm described above, and selects the bin with maximum value of F. The algorithm repeats this procedure until all pieces are placed.

Once the global solution is created beam nodes are selected as the ones with maximum value for U, as that is the final objective function. In case of a tie, we select the one with lowest utilization of the last node of the global solution, as the residual cut left on this last bin may ensure a better utilization.

5.2 Single Bin Size Bin Packing Problem

The single bin size problem does not require a bin selection problem to be solved. Hence, the construction algorithm can follow a first fit type strategy where more than one bin can be open at a time. Thus we modify our constructive algorithm to compare this approach with the MBS approach and found that the modified approach performs a little better for the SBS problem.

The procedure to construct a global solution opens a new bin if no piece from Q can be placed in any of the available bins, ensuring that the first piece on each bin is the largest of the remaining pieces. The key difference is that the algorithm does not expand Q to all the unplaced pieces and instead opens a new bin. On each iteration, we update the candidate list for all open bins. This update may accept or discard different pieces based on those already placed on each bin.

We declare a bin *closed* if there are no more possibilities of placing pieces in it. This occurs when no candidate can be placed using the best match procedure. We also explore empty sections of the bin and the possibility of adding new separation cuts, before declaring a bin closed.

Because we are dealing with single bin sizes, it is useful to compute the total number of bins. If R^* denotes the ratio of the packed area after applying the guillotine cut that removes the residual, then $R^* = \min\{L_R/L, W_R/W\}$. The objective is to minimise the fractional number of bins, $N - 1 + R^*$, where N denotes the number of bins used in the solution. In this case the residual is a measure of the utilization of the last bin that helps differentiate between solutions with the same number of bins.

Beam nodes are then selected as the ones with minimum fractional number of bins. Thus, on each global solution created, we calculate the utilization on the last bin by rearranging the pieces to prioritise horizontal and vertical placements, and keeping the one with the best fractional number of bins after the residual cut is made.

6 Computational Experiments

In this section, we present the results obtained with the beam search algorithm for both the MBS and SBS packing problems. Since the MBS version of the problem has not been studied in the literature, we first present results for the SBS and compare our results with the best in the literature in terms of solution quality. Then we present some test to validate the MBS version, and the benefits of using a range of bin sizes.

Beam search, as many other heuristics, has some parameters that need to be fixed, specifically filter width α and beam width β. Experimental tests show that the quality of the solution is not very dependent on these parameters. Choosing $\alpha = 2$ and $\beta = 3$ gave the best results, and increasing these parameters only increased the running time but did not improve the quality of the solutions. For the MBS problem, we want a diversity of bins on each level of the tree so we force the algorithm to choose α nodes for each bin size, increasing the filter width to $\alpha \cdot T$.

To determine how many nodes we are generating from each parent, we select 10 pieces, as the initial piece for each bin. For each of the first pieces and their reflection, we select three rotations. These rotations are chosen so that the number of edges matching the bin is maximum, and ties are broken by selecting those with maximum sum of the lengths of the edges matching the bin. From each parent node, we then create a maximum of 60 children for each bin size, resulting in a total of $60T$ nodes per parent.

The packing heuristic also has a few parameters to set. First, is the size of the candidate list L. We tested different values of L that range from $L = 1$, which means that we are always placing the next piece in the list, to $L = |\mathcal{P}|$ which means that all pieces are available to be candidates. The acceptance criterion for a candidate is also restricted by the match degree. We only accepted candidates with a match degree greater than a threshold $0 < \theta_1 \leq 1$. Values of θ_1 closer to 0, result on either candidates that did not satisfy the acceptance criterion to apply the best match algorithm, or poor utilization of the convex hull after applying this algorithm. Values of θ_1 closer to 1 restrict the candidate list in terms of edge similarity, ensuring a better utilization of the convex hull, but we may not find all L candidates. After several experiments on different instances, we found that $L = 5$ and $\theta_1 = 0.8$ were the best values to restrict the candidate list.

The match degree is also used as a restriction to accept attach operations when placing a selected candidate in the bin, θ_2. Using the same threshold as for the candidate list may results in placements that do not lie entirely in the bin. Thus we relax the threshold to accept a matching edge when placing in the bin,

and select $\theta_2 = 0.5$, ensuring that all candidates will be tested, and possibly with more than one edge matching.

It is important to mention that, although beam search discards all global solutions constructed, we keep the best solution found at any time, as is usual for other local search strategies. However, in most cases, this solution was replicated or improved during the tree search, confirming that our local and global evaluation functions together do not prune nodes that will lead to the best solution.

These parameters were used throughout our computational experiments, for both the SBS and MBS versions of the problem. When applicable, we compare our results against the best in the literature (CA1-2Ph-Imp) presented in Martínez-Sykora et al. (2015). To present a fair comparison, we run their code and our heuristics on the same machine: an Intel Core i7 with 2.8 GHz MacBook Pro with 4 GB of RAM. Parameters were tuned using the original instances for this problem, found in Han et al. (2013), and the heuristic was run over all irregular convex instances found on the ESICUP website (`http://paginas.fe.up.pt/~esicup/datasets`).

6.1 Results for the single bin size problem

In this section we present the results obtained when all bins are identical in size. We only report results from Martínez-Sykora et al. (2015) (CA1-2Ph-Imp) as they outperform those from Han et al. (2013). In Tables 1 and 2 we show the total number of bins used, the percentage utilization, calculated as in (1), and the fractional number of bins, as described in Section 5.2. While the equal number of bins and the percentage utilization provide the same comparative measure, including the utilization indicates the overall quality of the packing. The fractional number of bins, is only meaningful in practice when offcuts are retained and used. However, it help distinguish between solutions that have an equal number of bins, and compare our results with those in the literature. For each instance, we highlight the column with minimum fractional number of bins.

Table 1: Results for single bin instances in Han et al. (2013)

	MBS Algorithm				SBS Algorithm				CA1-2Ph-Imp			
	Total Bins	% Util	Frac No of Bins	Time (secs)	Total Bins	% Util	Frac No of Bins	Time (secs)	Total Bins	% Util	Frac No of Bins	Time (secs)
J40	8	75.99	7.23	56	7	86.84	6.99	39	7	86.84	**6.92**	168
J50	9	84.97	8.99	85	9	84.97	**8.84**	51	9	84.97	8.97	344
J60	11	78.68	10.23	115	10	86.54	**9.97**	79	10	86.54	9.99	204
J70	12	82.24	11.79	144	12	82.24	**11.32**	114	12	82.24	11.54	703
H80	10	81.62	9.45	290	10	81.62	9.32	166	10	81.62	**9.21**	1275
H100	16	83.82	15.67	407	16	83.82	**15.27**	266	16	83.82	**15.27**	1412
H120	16	85.84	15.89	587	16	85.84	15.41	376	16	85.84	**15.37**	2406
H150	22	87.41	21.96	1044	22	87.41	**21.45**	595	22	87.41	21.59	3314

There are two sets of irregular convex instances on the ESICUP website. Table 1 shows the results for the first set of instances, which corresponds to those introduced in Han et al. (2013). It compares the results obtained with

both versions of the beam search, the one designed for MBS and the one for SBS. Appendix A shows the solution obtained for the J40 instance with both MBS and SBS algorithms. It is clear that the modification made to the global solution generation for the SBS version of the problem reduces the number of bins in some instances and the fractional number of bins in all instances. In terms of the number of bins, the SBS algorithm matches the best results to date, and there are small differences in the fractional number of bins. Beam search obtains slightly better results than CA1-2Ph-Imp in half of the instances, and the deviation between the single bin size version of the beam search and CA1-2Ph-Imp is not, in any case, greater than a 22% improvement on the last bin. The most notable difference in performance is in the execution times of the algorithms. Note that the beam search execution is a function of the size of the tree. The fastest instance (J40) in beam search takes less than a minute, while CA1-2Ph-Imp heuristic takes almost three minutes to run. The runtime advantage is more notable for the bigger instance such as H150, where CA1-2Ph-Imp takes just under one hour, and in less than 13 minutes the beam search finds a slightly better result.

The ESICUP website also contains some irregular shaped instances with only convex pieces from Terashima-Marín et al. (2010). The results for these instances are in Table 2. Each class from TA to TR has 30 instances. Martínez-Sykora et al. (2015) do not report results for these instances but we were able to run their code, and the results are given in Table 2. We have stated above that running the MBS version of the beam search does report worse results for instances with a single bin size, thus we only run this latter algorithm for these instances.

Table 2: Results for convex irregular instances in Terashima-Marín et al. (2010)

Class	Instances Solved	CA1-2Ph-Imp				SBS Algorithm			
		Total Bins	% Util	Frac Bins Avge	Time Avge	Total Bins	% Util	Frac Bins Avge	Time Avge
TA	30	120	75.00	**3.57**	31	120	75.00	3.71	8
TB	30	340	88.57	10.94	3	**330**	**90.91**	**10.45**	2
TC	30	220	82.00	6.93	20	**211**	**85.00**	**6.76**	20
TD	30	120	75.00	**3.65**	691	120	75.00	3.91	30
TE	22	88	75.00	**3.69**	355	88	75.00	3.76	24
TF	30	90	66.67	**2.43**	53	90	66.67	2.57	6
TG	30	424	84.78	13.73	8	**408**	**88.81**	**13.05**	5
TH	30	417	86.62	13.47	5	**391**	**92.31**	**12.50**	4
TI	30	120	80.00	**3.41**	99	120	80.00	3.59	4
TJ	30	150	80.00	**4.71**	196	150	80.00	4.96	16
TK	30	210	85.71	**6.72**	38	210	85.71	6.76	22
TL	26	104	75.00	3.61	25	104	75.00	3.61	6
TM	18	108	83.33	5.8	28	108	83.33	**5.74**	14
TN	30	90	66.67	**2.45**	757	90	66.67	2.79	17
TO	20	**159**	**88.26**	7.59	4	160	87.50	**7.42**	3
TP	12	112	85.93	9.03	71	**108**	**88.89**	**8.89**	46
TQ	30	**452**	**99.58**	**15.03**	7	508	88.15	16.54	37
TR	30	313	86.45	10.06	42	**301**	**90.00**	**9.89**	39

Our first observation, is that CA1-2Ph-Imp cannot solve all instances. Specif-

ically, there are five classes where this approach fails to produce a solution for all the instances. This is mainly due to the placement of the first piece, when edges are longer than the bin length or width, and the piece must be placed with a certain angle. This is most notable in class TP where only 12 out of the 30 instances are solved. For comparison purposes, we then only report results on those instances that CA1-2Ph-Imp solves, although beam search is able to solve all instances for all classes. In terms of total number of bins, there are 10 ties out of the 18 instance classes; in 6 of them the beam search uses less bins than CA1-2Ph-Imp heuristic and only on class TQ the beam search performs worse than the CA1-2Ph-Imp heuristic. For class TO, although CA1-2Ph-Imp used one bin less than the beam search, we note that the average fractional number of bins is smaller for the beam search algorithm. This is mainly because, although CA1-2Ph-Imp finds two optimal solutions with 100% utilization, for the remainder of the instances the fractional number of bins used, is always greater than the solution obtained with the SBS algorithm. From those instances where there is a tie in the number of bins, we look at the fractional number of bins. In most of these instances the CA1-2Ph-Imp heuristic works better than the beam search proposed, however, the difference between both solutions is, on average, less than 0.2, which represents a 20% improvement on the utilization of the last bin. Comparing the computational time for both algorithms to run, CA1-2Ph-Imp is worse, since for some classes it took more than 10 minutes to solve, on average. The total running time for the 488 instances CA1-2Ph-Imp was able to solve is around 19 hours, whereas the beam search took, on average, no more than 40 seconds for any instance, and solved all 540 instances in 2.3 hours. This is a significant improvement on time, with very little, or sometimes no, difference in solution quality. On the other hand, we can see the advantages of using a MIP as part of the heuristic in class TQ, which was created by cutting the pieces out of a square by means of guillotine cuts. In this class CA1-2Ph-Imp found 28 out of the 30 optimal solutions very quickly, whereas the beam search performed less well for this particular class, although it was capable of finding 1 optimal solution. Also note, that for classes TB, TH and TR the utilization with the SBS algorithm is greater than 90%, while CA1-2Ph-Imp only achieves greater than 90% utilization in class TQ.

These examples show the strength of our beam search heuristic, and its competitive performance when compared with the best solutions found in the literature. It demonstrates we have met the aim of our research, which is to develop an efficient tool to solve the proposed problem without the aid of specialist commercial software and obtain competitive results.

6.2 Results for the multi bin size problem

The MBS version of the two-dimensional bin packing problem with guillotine cuts has not been studied before, and all instances for convex irregular pieces available to the community only deal with SBS. Thus we need to create some instances to test our algorithm. To compare the efficiency of our algorithm, we need to create different bin sizes and use them with the existing convex pieces in the literature.

We worked with three bin sizes: small (S), medium (M) and large (L), where the medium size bin is the same as the single bin size in the literature. To create a small bin, each side of the original bin is multiplied by $\sqrt{2/3}$, and to create the large bin the multiplying factor is $\sqrt{3/2}$.

Table 3: Beam search results for MBS instances vs. SBS instances

Class	Multi Bin Size		Single Bin Size (Bin M)		Single Bin Size (Bin L)	
	%Util	Time	%Util	Time	%Util	Time
J40	88.5	202	85.3	39	83.9	59
J50	88.5	250	86.6	57	86.3	101
J60	87.6	355	85.3	80	83.7	140
J70	88.2	550	85.9	105	84.4	170
H80	90.1	773	87.8	185	85.6	235
H100	89.8	1538	87.1	290	86.3	436
H120	90.1	2119	88.2	409	86.5	575
H150	90.5	4162	88.9	719	87.4	1010

To present the results, we run the MBS algorithm to pack the same sets of pieces under two scenarios: multiple size bins are available, and just one size bin is available. Using the MBS algorithm for both scenarios allows us to assess the benefit of having multiple bin sizes. Appendix A also shows the solution obtained on these scenarios. For the SBS instances, we are presenting results with a medium and a large bin size. Here we only report the utilization since number of bins does not provide a meaningful comparative metric. For example, it will take many more bins to pack the same set of pieces if small and medium bins are used, as opposed to using large bins, but may give a higher utilization. We do not present results for a small bin size, as not all the pieces fit in this bin size making some instances infeasible. The rationale for this comparison is to show the competitiveness of the algorithm and the benefits of having different bin sizes in terms of bin utilization. Table 3 shows the results for the first set of instances.

For this case, we can see a small improvement on using different bin sizes, with respect to using all bins of size M. However, when it comes to bins of size L, the improvements are greater. This may be explained by our observation that using large bins generates clusters of big pieces in a bin along with unused spaces where small or medium pieces will not fit. Then, extra bins are needed just for these pieces. When using bins of size M, big pieces are spread across bins and small and medium pieces can fit in the gaps. When compared with using multiple bins to only using bins of size M, the improvements range between a 1.6 and 3.2%, whereas if we compare with a large bin the range widens between a 2.2 and a 5.5%. Execution times reflect the difficulty of having more bins, however solving the multi bin instance, is never less than 6 times the time it takes to solve the fastest SBS problem. Recall that all times reported in this paper are in seconds.

Similar results can be seen in Table 4 where we present the same comparison for the second set of instances. For this set of instances the improvement observed when using multiple bins to using only bins of size M are much smaller than on the previous set. This is because, for these instances, pieces were created by

Table 4: Beam search results for MBS instances vs. SBS instances

Class	Multi Bin Size		Single Bin Size (Bin M)		Single Bin Size (Bin L)	
	%Util	Time	%Util	Time	%Util	Time
TA	87.6	62	86.6	11	75.8	3
TB	95.1	41	93.6	6	78.3	9
TC	91.7	144	91.3	28	80.1	5
TD	84.0	207	83.1	39	70.9	3
TE	87.1	129	86.1	26	78.1	3
TF	86.9	45	85.1	8	70.6	2
TG	94.1	61	90.5	9	78.3	10
TH	95.1	63	92.7	8	77.6	10
TI	92.1	26	90.5	5	80.7	2
TJ	86.8	132	84.0	21	76.9	3
TK	92.4	158	91.8	32	82.1	5
TL	90.2	35	88.3	7	80.4	2
TM	90.5	77	89.2	15	83.5	4
TN	84.7	122	83.1	22	67.7	2
TO	95.7	33	96.0	3	81.8	6
TP	91.7	251	90.4	45	85.9	6
TQ	92.6	209	92.0	31	76.8	13
TR	92.6	314	91.3	57	82.8	7

taking a medium bin as a starting point and extracting pieces from it like a jigsaw puzzle. Thus it is expected that this bin will provide the best fit for pieces in these instances. We are not obtaining 100% utilization in any case, since not all of the pieces could be separated with guillotine cuts, when the instances were created. When compared with a large bin, improvements are considerably larger, ranging from a 6% to a 17%. Although we do not show the mix of bin sizes, there is a majority of medium size bins used and all solutions take advantage of having different sizes available and report at least one extra bin size (S or L), and many of them use the three available sizes.

These results show the efficiency of the algorithm proposed in this paper, to solve both the MBS and SBS problems. The beam search developed to solve the MBS problem is competitive when dealing with SBS instances, but we have proved that a slight modification in the global solution construction leads to better results that are competitive with the best ones found in the literature, and outperforming them for some cases. There are no references for solving the two-dimensional bin packing problem with guillotine cuts and MBS, we have created different bin sizes, and compare our results to instances when only one bin size could be used. Using multiple bin sizes reports better solutions in terms of utilization, being more notable when the selected bin size is not the initial bin reported in the original instances.

7 Conclusions

We have presented a new heuristic to solve the two-dimensional multi bin size bin packing problem with irregular pieces and guillotine cuts. This heuristic is based on a beam search implementation, where each node represents a partial

solution, in the form of a packed bin. The constructive heuristic designed to pack the bins, take ideas of best match and best guillotine cut presented previously in the literature, and combines them to build a cutting pattern inside a bin. The constructive heuristic is fast, so packing a bin is not time consuming. Taking advantage of this fact, we decided to build many bins, and select a subset that may lead to good final solutions.

Beam search has been used before in cutting and packing problems, but did not play an important role in bin packing until now. To the best of our knowledge this is the first paper that uses this heuristic entirely to solve a bin packing problem, with highly competitive results. We have tested the implementation in the benchmark instances that are available for this problem, obtaining very similar results than the best heuristic up to date, in shorter time and without the aid of additional specialised software. All the parameters are adjustable, so the user can set them to their needs. We have also created new instances to test our implementation for the multi bin size problem. Results showed that using different bin sizes reports benefits in terms of bin usage, specially when compared to bins that are different to the original ones reported in the given instances.

A Selected Layouts

Here we present some layouts for the J40 instance run under the different scenarios presented in this paper. They correspond to the solutions presented in Tables 1 and 3.

First, we present the solutions obtained for the J40 instance run on a single bin size M with the MBS algorithm (Figure 9) and with the SBS algorithm (Figure 10). These examples correspond to the solutions presented in Table 1. Since we are working with single bin sizes, we compute the residual in the last bin and the utilization reported also considers this fact.

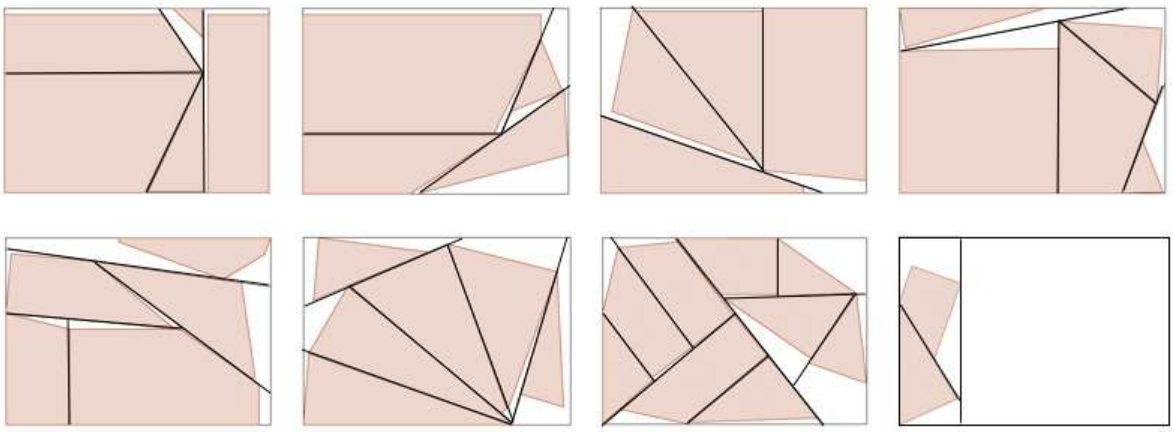

Figure 9: J40 instance, bin size M, MBS algorithm. Utilization: 84.1%

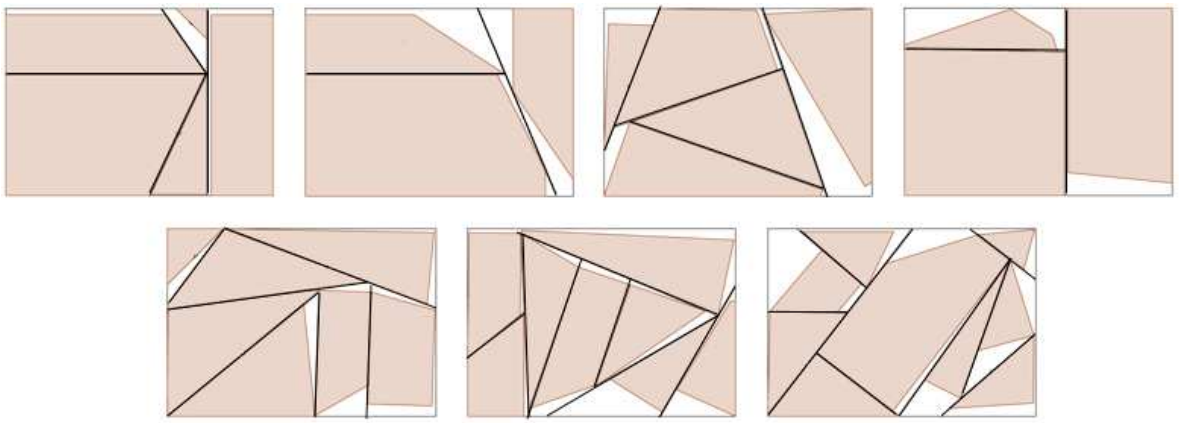

Figure 10: J40 instance, bin size M, SBS algorithm. Utilization: 86.9%

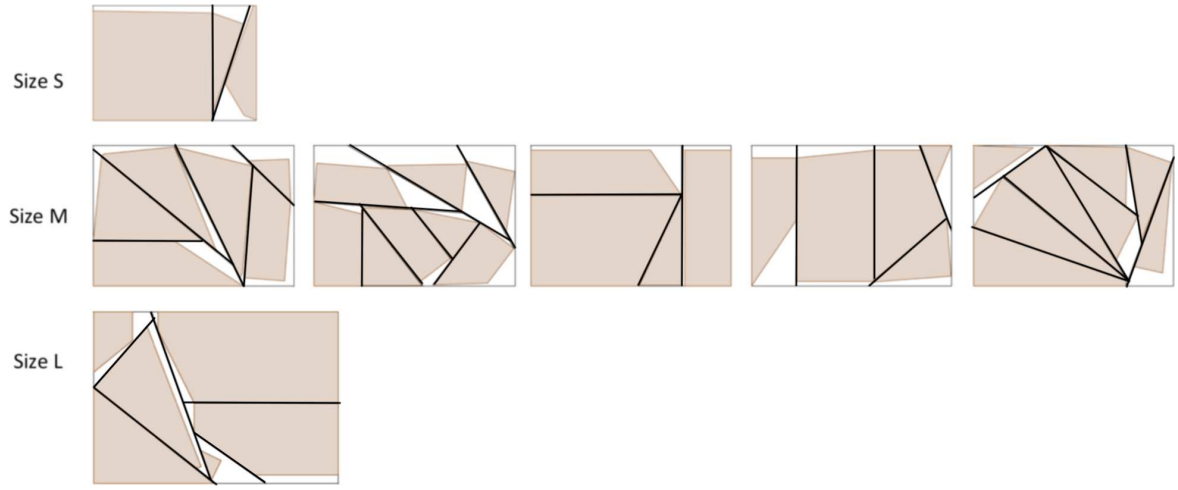

Figure 11: J40 instance, different bin sizes. Utilization: 88.5%

We then show the results presented in Table 3 where we compare the effect of having more than one bin size. These instances were run with the MBS algorithm. In this case, we do not compute the residual of the bin, thus the difference between the solutions in Figures 9 and 12. We show in Figure 11 the layout obtained when having three bin sizes: small (S), medium (M) and large (L). We can observe that, although there is a majority of medium bins, the algorithm takes advantage of the other bin sizes to improve the utilization. In Figure 12 we use only medium bins, and in Figure 13 the pieces are placed in large bins.

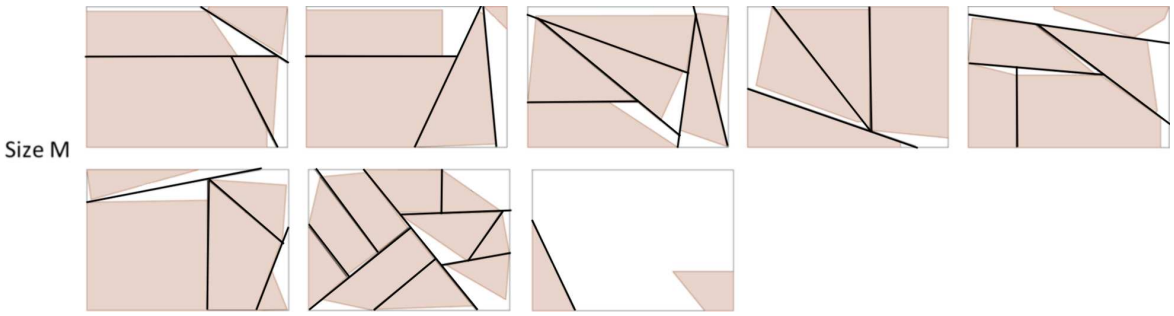

Figure 12: J40 instance, bins of size M. Utilization: 85.3%

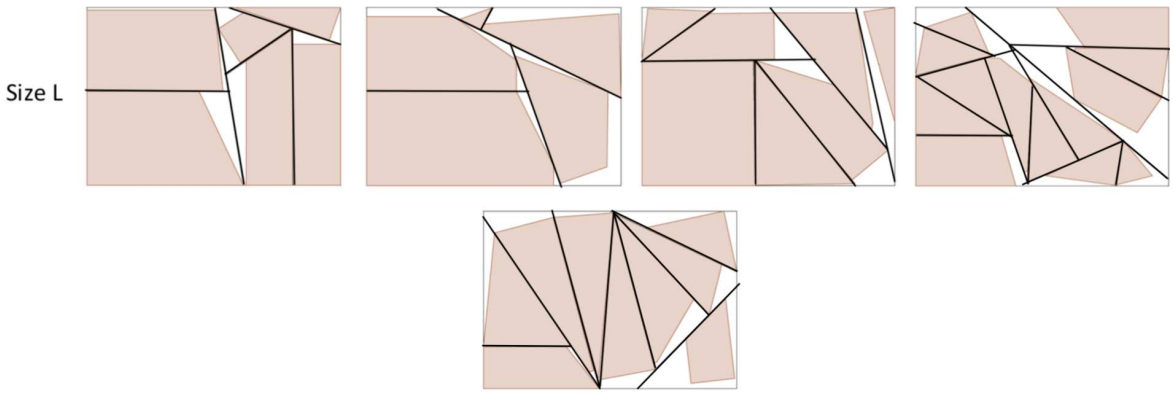

Figure 13: J40 instance, bins of size L. Utilization: 83.9%

Acknowledgements

This work has been funded by UK's Royal Society - Newton Advance Fellowship under grant NA150409.

References

R. P. Abeysooriya, J. A. Bennell, and A. Martinez-Sykora. Jostle heuristics for the 2d-irregular shapes bin packing problems with free rotation. *International Journal of Production Economics*, 195:12 – 26, 2018.

H. Akeb, M. Hifi, and R. M'Hallah. A beam search algorithm for the circular packing problem. *Computers & Operations Research*, 36(5):1513 – 1528, 2009.

H. Akeb, M. Hifi, and S. Negre. An augmented beam search-based algorithm for the circular open dimension problem. *Computers & Industrial Engineering*, 61 (2):373 – 381, 2011.

F. Alvelos, T. Chan, P. Vilaca, E. Silva, and J. M. V. de Carvalho. Sequence based heuristics for two-dimensional bin packing problems. *Engineering Optimization*, 41(8):773 – 791, 2009.

J. A. Bennell and K. A. Downsland. Hybridising tabu search with optimisation techniques for irregular stock-cutting. *Management Science*, 47(8):1160 – 1172, 2001.

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*, 16:167 – 188, 2010.

C. Charalambous and K. Fleszar. A constructive bin-oriented heuristic for the two-dimensional bin packing problem with guillotine cuts. *Computers & Operations Research*, 38:1443 – 1451, 2011.

L. H. Cherri, L. R. Mundim, M. Andretta, F. M. Toledo, J. F. Oliveira, and M. A. Carravilla. Robust mixed-integer linear programming models for the irregular strip packing problem. *European Journal of Operational Research*, 253(3):570 – 583, 2016.

K. Fleszar. Three insertion heuristics and a justification improvement heuristic for two-dimensional bin packing with guillotine cuts. *Computers & Operations Research*, 40:463 – 474, 2013.

M. Ghirardi and C. N. Potts. Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. *European Journal of Operational Research*, 165:457 – 467, 2005.

P. C. Gilmore and R. E. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13(1):94–120, 1965.

R. B. Grinde and T. M. Cavalier. A new algorithm for the minimal-area convex enclosure problem. *European Journal of Operational Research*, 84:522 – 538, 1995.

W. H. Han, J. A. Bennell, X. Zhao, and X. Song. Construction heuristics for two-dimensional irregular shape bin packing with guillotine constraints. *European Journal of Operational Research*, 230:495 – 504, 2013.

A. A. Leao, F. M. Toledo, J. F. Oliveira, and M. A. Carravilla. A semi-continuous mip model for the irregular strip packing problem. *International Journal of Production Research*, 54(3):712–721, 2016.

A. Lodi, S. Martello, and D. Vigo. Approximation algorithms for the oriented two-dimensional bin packing problem. *European Journal of Operational Research*, 112:158 – 166, 1999a.

A. Lodi, S. Martello, and D. Vigo. Heuristics and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, 11(4):345 – 357, 1999b.

A. Lodi, M. Monaci, and E. Pietrobuoni. Partial enumeration algorithms for two-dimensional bin packing problems with guillotine constraints. *Discrete Applied Mathematics*, 2015.

A. Lodi, M. Monaci, and E. Pietrobuoni. Partial enumeration algorithms for two-dimensional bin packing problem with guillotine constraints. *Discrete Applied Mathematics*, 217:40 – 47, 2017.

E. Malaguti, R. M. Durán, and P. Toth. Approaches to real world two-dimensional cutting problems. *Omega*, 47:99 – 115, 2014.

A. Martínez-Sykora, R. Álvarez-Valdés, J. A. 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.

A. Martinez-Sykora, R. Alvarez-Valdes, J. Bennell, R. Ruiz, and J. Tamarit. Matheuristics for the irregular bin packing problem with free rotations. *European Journal of Operational Research*, 258(2):440 – 455, 2017.

J. Puchinger and G. R. Raidl. Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operational Research*, 183:1304 – 1327, 2007.

M. O. Rodrigues and F. M. Toledo. A clique covering mip model for the irregular strip packing problem. *Computers & Operations Research*, 87:221 – 234, 2017.

I. Sabuncuoglu and M. Bayiz. Job shop scheduling with beam search. *European Journal of Operational Research*, 118:390 – 412, 1999.

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*, 65:1037 – 1052, 2014.

H. Terashima-Marín, P. Ross, J. C. Farías-Zárate, E. López-Camacho, and M. Valenzuela-Rendón. 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*, 183:1109 – 1130, 2007.