Cherri, Luiz Henrique; Carravilla, Maria Antónia; Ribeiro, Cristina; de Toledo, Franklina Maria Bragion

Article

Optimality in nesting problems: New constraint programming models and a new global constraint for non-overlap

Operations Research Perspectives

Provided in Cooperation with:
Elsevier

Suggested Citation: Cherri, Luiz Henrique; Carravilla, Maria Antónia; Ribeiro, Cristina; de Toledo, Franklina Maria Bragion (2019) : Optimality in nesting problems: New constraint programming models and a new global constraint for non-overlap, Operations Research Perspectives, ISSN 2214-7160, Elsevier, Amsterdam, Vol. 6, pp. 1-19, https://doi.org/10.1016/j.orp.2019.100125

Operations Research Perspectives 6 (2019) 100125

Contents lists available at ScienceDirect

Operations Research Perspectives

journal homepage: www.elsevier.com/locate/orp

Optimality in nesting problems: New constraint programming models and a new global constraint for non-overlap

Luiz Henrique Cherri[*,a,b], Maria Antónia Carravilla[c], Cristina Ribeiro[c], Franklina Maria Bragion Toledo[b]

[a] Optimized Decision Making (ODM), São Carlos - SP, Brasil
[b] Universidade de São Paulo, São Carlos - SP, Brasil
[c] INESC TEC, Faculdade de Engenharia, Universidade do Porto, Portugal

ARTICLE INFO

Keywords:
Irregular cutting and packing
Constraint programming
Global constraint
Exact models

ABSTRACT

In two-dimensional nesting problems (irregular packing problems) small pieces with irregular shapes must be packed in large objects. A small number of exact methods have been proposed to solve nesting problems, typically focusing on a single problem variant, the strip packing problem. There are however several other variants of the nesting problem which were identified in the literature and are very relevant in the industry.

In this paper, constraint programming (CP) is used to model and solve all the variants of irregular cutting and packing problems proposed in the literature. Three approaches, which differ in the representation of the variable domains, in the way they deal with the core constraints and in the objective functions, are the basis for the three models proposed for each variant of the problem. The non-overlap among pieces, which must be enforced for all the problem variants, is guaranteed through the new global constraint NoOverlap in one of the proposed approaches.

Taking the benchmark instances for the strip-packing problem, new instances were generated for each problem variant. Extensive computational experiments were run with these problem instances from the literature to evaluate the performance of each approach applied to each problem variant. The models based on the global constraint NoOverlap performed consistently better for all variants due to the increased propagation and to the low memory usage.

The performance of the CP model for the strip packing problem with the global constraint NoOverlap was then compared with the Dotted Board with Rotations using larger instances from the literature. The experiments show that the CP model with global constraint NoOverlap can quickly find good quality solutions in shorter computational times even for large instances.

1. Introduction

Two-dimensional irregular cutting and packing problems have been largely studied in the literature for more than five decades. The problem consists in placing polygonal shapes (pieces) in a given object (board). The pieces and the board may have any irregular shape. Different objectives can be explored depending on the application, for example to maximize the number of pieces on the board, to minimize the used board length when cutting all the pieces or to minimize the number of boards needed to cut the pieces. Wäscher et al. [33] proposed a classification of the variants of the two-dimensional cutting and packing problems, building on classifications by Dyckhoff et al. [14,15].

In the two-dimensional irregular cutting problems, the most challenging constraints to deal with are the ones that ensure that the pieces do not overlap. Although this problem has been already solved by different approaches, the choice of an adequate representation for the pieces and the right use of the non-overlap methods are crucial to boost the efficiency of the solution methods. When the pieces are all rectangular, simple computations over the piece dimensions can be done in order to check for overlap. However, if one or more pieces are not rectangular, more sophisticated tools are required to assess whether the pieces are overlapping or not.

Some geometric tools have been proposed to detect piece overlap. The raster points approach discretizes the board and the pieces into matrices. Using these matrices, the places where a piece cannot be placed to avoid overlap can be defined. This approach is simple, but the quality of the solution depends on the accuracy of the discretization.

A more rigorous approximation can be reached by using D-functions or nofit polygons. The D-function returns the relative position between a point and a line and can be used in several ways to avoid overlap. The D-

* Corresponding author at: Universidade de São Paulo, São Carlos - SP, Brasil.
E-mail address: lhcherri@icmc.usp.br (L.H. Cherri).

https://doi.org/10.1016/j.orp.2019.100125
Received 25 March 2019; Received in revised form 15 August 2019; Accepted 24 September 2019
Available online 25 September 2019

function is a rigorous tool, however the complexity of the analysis highly depends on the shapes of the pieces and can be huge. The relative positions between two polygons are captured by the nofit polygon, which reduces evaluating piece overlap to checking if a point is inside a polygon, a task that can be done using the D-functions. A complete review of geometric tools for these problems is available in [8].

Several heuristics were proposed to solve the variants of the nesting problem. The irregular open dimension problem or strip packing problem, where only one dimension is open, is the most studied irregular cutting problem variant and is currently solved by heuristics [6]. Recently, [16] proposed a guided cuckoo search heuristic for the same problem. Baldacci et al. [3] presented a heuristic to solve the irregular bin packing problem in case of an irregular board with defects and quality regions where only a subset of the pieces can be placed. To solve the irregular cutting stock problem, [28] proposed a heuristic column generation procedure. Valle et al. [31] created heuristics for irregular binary knapsack problems and irregular unconstrained knapsack problems. Using these heuristics, the authors also proposed a heuristic column generation procedure to solve the irregular cutting stock problem. To solve the irregular bin packing problems with guillotine cuts, [22] proposed a constructive heuristic based on a mixed integer programming model. More recently, for the same problem, [7] proposed a beam search algorithm to solve multi and single bin instances and used the no-fit raster to deal with the geometry of the pieces, [23] developed a biased random-key genetic algorithm to solve the irregular strip packing problem, and [1] presented an iterated jostle heuristic with diversification mechanisms. Sato et al. [27] proposed a heuristic to solve the irregular strip packing problem using the raster penetration method to determine the positions of the pieces.

In contrast with the number of available heuristics, only a few exact methods were proposed to solve irregular cutting and packing problems and all of them considered only the irregular strip packing problem. The first exact method for the irregular strip packing problem was presented by Carravilla et al. [9] and used constraint programming. The optimality of this method is subject to a discretization inherent to the finite domain constraint programming paradigm. Ribeiro and Carravilla [24] proposed the first global constraint for nesting problems. Mixed integer programming models were also used to tackle the problem. Fischetti and Luzzi [17] developed a model where the overlap of the pieces is avoided by using nofit polygons. In this approach, the exterior part of the nofit polygon is divided into convex areas (slices) and the non-overlap constraints are built using these structures. Alvarez Valdes et al. [2] proposed two procedures to generate the slices of the [17] model. The authors also presented a mixed integer programming model based on the linear compaction model by Gomes and Oliveira [18]. Cherri et al. [12] developed two models, both allowing the pieces to be rotated in a finite set of angles. One model uses only direct trigonometry to avoid overlap while the other uses the nofit polygon to avoid the overlaps. Also using direct trigonometry to avoid overlap among pieces, [11] proposed a mixed-integer quadratically-constrained model to solve the problem considering the free rotation of the pieces. The decision variables used in all the models mentioned so far were the (X,Y)-coordinates of the positioning points of the pieces. Toledo et al. [29] proposed the dotted-board model where the board is represented as a mesh of dots. The binary decision variables, associated to each dot and piece type, are set to 1 if a piece of the corresponding type has its reference point positioned on the dot. As in [9], the optimality of this model depends on the discretization used. Leão et al. [20] proposed a mixed integer programming model to solve the problem. In their approach, the piece position is discretized over the y-axis but not on the x-axis, where continuous variables are assumed. For an overview of articles on exact methods, see [21].

This paper proposes constraint programming models to solve irregular cutting and packing problems where the decision variables are the dots in the board as in [29]. For the domains of the decision variables two approaches are explored: binary domains as in [29], and integer domains. At the core of the irregular cutting and packing problems is the constraint that pieces may not overlap. The new global constraint NoOverlap is proposed, with the corresponding algorithms for propagation and constraint reduction. The NoOverlap constraint can be used in all the variants of irregular cutting and packing problems.

Using the classification of irregular cutting and packing problems in the typology of [33], we propose models for all problem variants combining the non-overlap constraints with variant-specific constraints and objective functions. To the best of our knowledge it is the first time in the literature that exact methods are proposed for all the problem variants of irregular cutting and packing problems.

The computational experiments show that our constraint programming approach can deal with all the variants of irregular cutting and packing problems. Moreover, using the NoOverlap global constraint, the computational time required to prove optimality (whenever it is reached within the given time limit) and the memory used are reduced with respect to integer programming approaches. Moreover, and although the computational time to prove optimality is typically high in all the proposed approaches, all the models can quickly find good quality solutions.

The main contributions of this work are: (1) innovative representations of discrete placement points for pieces, using variables with binary and integer domains; (2) constraint programming models for all the variants of irregular cutting and packing problems classified by Wäscher et al. [33] (the previous studies in the literature propose models for only one of these variants); (3) piece rotation at a finite number of angles as part of the proposed models (only one exact method in the literature provides piece rotation and only for one variant of the irregular cutting and packing problem); (4) NoOverlap, a global constraint to avoid the overlap among pieces, and the corresponding algorithms for constraint propagation; and (5) extensions of the benchmark instances to the variants of irregular cutting and packing problems, essential to evaluate our models.

The remainder of the paper is organized as follows: Section 2 has an overview of the Constraint Programming paradigm and of Global Constraints. Section 3 presents the definition of the irregular cutting and packing problems that will be solved and some geometric definitions used in the models. Section 4 presents decision variables with binary and integer domains to represent the problem and proposes non-overlap constraints for both types of variables. Furthermore, a new global constraint to avoid the overlap between pieces is introduced. The NoOverlap global constraint is used with integer domain variables and is tailored to irregular cutting and packing problems. Section 5 systematically defines a set of constraints to represent each variant of the irregular cutting and packing problems as defined in the typology of [33]. In Section 6, the proposed models are run on several benchmark problems. The computational results are analyzed, showing the performance of each model and also the versatility of the proposed constraint programming approach to solve all the cutting and packing problem variants. Section 7 presents computational experiments with larger instances. Focusing on the irregular strip packing problem, this section compares the results obtained with the constraint programming model with the global constraint NoOverlap with the results of [10] (I1ODP – DBM with rotations). Finally, in Section 8 the conclusions and main open issues are presented.

2. Constraint programming and global constraints

Constraint programming is a computational paradigm where constraints are at the core and the methods for manipulating and propagating constraints are tightly integrated with the optimization strategies. A conventional program to determine the feasibility of a problem may be seen as a single-layer where the constraints must be explicitly stated. In constraint programming a Constraint Solver is added as a second layer in the execution environment (see Fig. 1). The gain in efficiency stems from the fact that the constraints in the Constraint

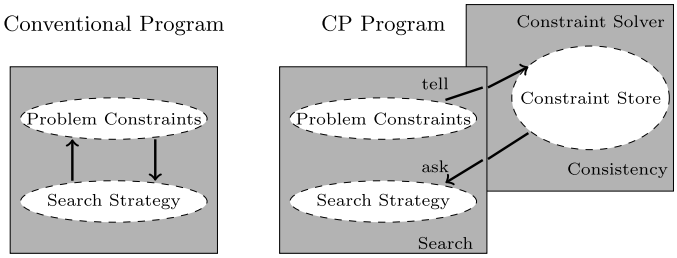


Fig. 1. Conventional, single-layer program versus dual-layer CP model (adapted from Carravilla et al. [9]).

Store are actively used for reducing the domains of the variables and that the Constraint Solver can use a variety of methods depending on the nature of the constraints.

Constraints in a constraint programming language can be as basic as linear constraints, but can also express the structure of the problem at a higher semantic level. Current constraint programming systems offer a broad selection of constraints that can handle problems such as scheduling or shortest routes, and have proven to be very effective in real-life problems, namely when feasibility is an issue. Constraint programming systems are designed to handle a large number of heterogeneous constraints and have been successfully applied in several combinatorial optimization problems. To solve the resource portfolio planning of make-to-stock products, [32] proposed a constraint programming-based genetic algorithm. Clautiaux et al. [13] presented constraint programming to solve the two-dimensional orthogonal packing problem, outperforming previous approaches. To solve the project scheduling problem under resource constraints, [30] proposed a constraint programming approach using the cumulative global constraint. The goal here is to give support to the decision maker by proposing a set of optimal solutions to the problem. Salas et al. [25] developed non-overlap constraints based on Minkowski sums for polygons described by non-linear constraints. However, their approach is sensitive to the polygon shape, i.e., the more complex the shapes are, the harder the problem becomes.

The strength of constraint programming comes from the possibility of modeling problems at a higher level, using the so-called global constraints. A global constraint is a specialized constraint for a given problem, and allows the solver to use features of the problem that are not manageable if the model is expressed at the atomic level of basic constraints, such as linear inequalities. Global constraints are at the core of the solution method for many classes of problems. The Global Constraint Catalog [4] has an extensive list of global constraints, which have been incorporated in various constraint programming solvers. Global constraints have been developed to solve several combinatorial optimization problems. Kovács and Beck [19] proposed a global constraint for the total weighted completion time of activities for a single capacity resource. Saldanha and Morgado [26] created a global constraint to solve the set partitioning problem, which is easy to modify and has an efficient propagator.

For nesting problems, a global constraint to avoid overlap between pieces was proposed by Ribeiro and Carravilla [24]. The `outside` constraint is based on a model where the decision variables are the (X,Y)-coordinates of the positioning points of the pieces. The main limitation of the approach stems from the two-dimensional nature of the problem: having X and Y coordinates represented as different variables limits the effectiveness of constraint propagation. Besides that, the model does not take advantage of piece types, i.e., the existence of pieces with the same shape. Therefore, this approach cannot be easily adapted to solve the variants of the irregular cutting problems where the number of pieces to be cut is not limited. To overcome these limitations and to improve the constraint propagation, we propose in this paper constraint programming models to solve irregular cutting and packing problems where the decision variables are the dots in the board instead of the (X,Y)-coordinates, with two approaches for the domains of the decision variables: binary domains and integer domains.

3. Irregular cutting and packing problems

The aim of the two-dimensional irregular cutting problem is to place convex or non-convex pieces on a board in order to optimize a given objective while ensuring that the pieces are inside the board and do not overlap. Depending on the specific variant of the problem considered, other constraints are added to the models.

3.1. Problem variants and applications

According to the typology of [33], cutting and packing problems can be classified into six basic variants: Identical Item Packing Problem (*IIPP*), Placement Problem (*PP*), Knapsack Problem (*KP*), Cutting Stock Problem (*CSP*), Bin Packing Problem (*BPP*) and Open Dimension Problem (*ODP*). Furthermore [33] classifies *IIPP, PP* and *KP* as output maximization problems and *CSP, BPP* and *ODP* as input minimization problems. Fig. 2 illustrates the irregular two dimensional cutting and packing problem variants.

Considering the output maximization problems, in the Identical Item Packing Problem (*IIPP*), the goal is to place as many pieces as possible of the same piece type on the board. In the Placement Problem (*PP*) there are several piece types and multiple pieces of each type that must be placed on the board. The number of pieces of each type to be cut can be finite (*PPc*) or large enough to be considered infinite (*PP*). If at most one piece of each type has to be placed on the board, we have the Knapsack Problem (*KP*). In all variants of the output maximization problems, the board has finite dimensions and there is typically no space on the board to cut all the demanded pieces. One or more boards can be considered to cut the pieces. The objective is to extract the maximum value from the set of pieces cut (packed) on the board.

Considering the input minimization problems, the resources are sufficient to cut or pack all the pieces and the usage of these resources must be minimized. In the Cutting Stock Problem (*CSP*), the goal is to use the minimum number of boards to place multiple pieces of several piece types. When only one piece of each type has to be placed on the board, the problem is called the Bin Packing Problem (*BPP*). In the Open Dimension Problem (*ODP*), many piece types with several pieces of each are placed on a board with one (1*ODP*) or two (2*ODP*) free dimensions. The objective for the *CSP, BPP*, 1*ODP* and 2*ODP* is typically to minimize the amount of resources used to cut all the demanded pieces. This objective can be reached by minimizing the number of boards used or the length of the board used to perform the cut. When solving the 2*ODP*, several objectives can be considered, such as minimizing the area of the bounding box of the packing, its perimeter, or other functions of the used length and height of the board.

As all the problems considered in this paper involve irregular pieces, the prefix *I* will be used from now on in the names of the problem

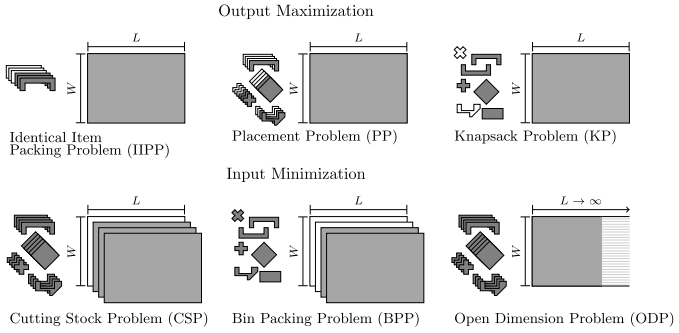


Fig. 2. Irregular cutting and packing problem variants according to the typology of [33].

variants.

Many real-world applications in the clothing, shoe manufacturing, sheet metal cutting and furniture industries require the solution of problems that fall into some of these variants. The problem variant that best captures each application domain depends on the industry and actual problem characteristics.

3.2. Dealing with the geometry

In all the variants of the irregular cutting problems, a finite number of piece types T, with a fixed number of allowed rotations R, are placed on a board. In a feasible solution, the pieces are positioned inside the board and do not overlap. The objective to be considered is either to maximize the value extracted from the board or to minimize the used portion of the board (or the number of boards). In the representations adopted here, each piece is represented by an ordered set of vertices and by a point, chosen as the piece reference point. The board is discretized as a regular mesh of dots D, the potential placement positions for the reference points of the pieces on the board. The mesh is regular, i.e. the vertical and horizontal distances between the dots are a multiple of a parameter Δ which determines the gauge of the mesh.

The first condition in a positioning is that each piece is entirely inside the board. To enforce this condition, we use a geometric construction built for one piece with respect to one board, the so-called inner-fit polygon (*IFP*). The *IFP* of piece type t at rotation r (IFP_{tr}) defines the region of the board where the reference point of this piece type can be placed, such that t is entirely inside the board. The boards considered in this work are rectangular, and therefore the inner-fit polygon can be easily defined based on the vertical and horizontal distances between the reference point chosen for the piece type and the sides of the bounding box.

Fig. 3 a illustrates piece type t at rotation r and the horizontal distances from the reference point to the left- (right-) hand side of the piece bounding box, l_{tr}^{left} (l_{tr}^{right}) and the vertical distances from the reference point to the bottom (top) of the piece bounding box h_{tr}^{bottom} (h_{tr}^{top}).

Fig. 3 b shows IFP_{tr}, the inner-fit polygon for piece type t at rotation r, i.e. the locus on the board for the positioning point of this piece type such that the piece is entirely inside the board. After taking into account the mesh discretisation, the corresponding set of dots, $\mathcal{IFP}_{tr}$, is represented in Fig. 3c.

The second condition in a positioning is that pieces do not overlap. We use the nofit polygon (*NFP*) to enforce this condition. The *NFP* of t at rotation r and t' at rotation r' ($NFP_{tr}^{t'r'}$) summarizes the geometric relation between these pieces. Using the *NFP*, we can reduce the test of overlap between two polygons to a verification of whether a point is inside, on the border or outside a polygon.

Fig. 4 illustrates the nofit polygon of t at rotation r and t' at rotation r'. The pieces are presented in Fig. 4a, the $NFP_{tr}^{t'r'}$ is shown in Fig. 4b (where piece type t is also outlined), and the corresponding set of points, $\mathcal{NFP}_{tr}^{t'r'}$, in Fig. 4c. If the reference point of t' at rotation r' is inside the $NFP_{tr}^{t'r'}$ the pieces overlap; if the reference point of t' at rotation r' is on the boundary (outside) $NFP_{tr}^{t'r'}$, the pieces are touching (apart). Note that the $NFP_{tr}^{t'r'}$ is represented relative to the placement point of piece type t.

In the models presented in this paper, the board is represented by a mesh of dots. To avoid the overlap, it is therefore sufficient to consider only the dots of the mesh inside the nofit polygon. Specifically, given piece t at rotation r placed at dot d and piece t' at rotation r', the set of dots where piece t' at rotation r' cannot be placed, to enforce the non-overlap with piece t at rotation r, is denoted by $\mathcal{NFP}_{tr}^{t'r'}$ (Fig. 4c).

4. Constraint programming models for irregular cutting and packing problems: Decision variables and non-overlap constraints

This section provides two approaches for the representation of cutting and packing problems in Constraint Programming (CP), building on the dotted board model proposed in [29]. There, the concept of dot refers to a two-dimensional point in a discretised plane, and pieces, whatever their shape and dimensions, have a reference point that is assigned to a specific dot in a solution. Our first approach is the one proposed in [29]: a binary variable captures the fact that a piece of a given type is positioned in a dot, with some rotation. In the second approach, integers are used to uniquely identify piece types and their rotations, and each dot is represented by a finite domain variable: when solving a problem, a dot is associated with a set of integer values (the candidate pieces to be placed there); in a solution, each dot is bound to a single integer value, either a non-zero value (for some piece positioned there) or zero if no piece happens to have its reference point on the dot.

We now introduce the notation for both approaches and the non-overlap constraints constructed using these variables.

For the remainder of the paper, dots are represented as $d \in \mathcal{D}$, where $\mathcal{D} = \{1...D\}$, is the set of dots for the corresponding board. A piece of a given type can have multiple occurrences in a problem, so we assume piece type $t \in \mathcal{T}$, where $\mathcal{T} = \{1...T\}$, can take one of a specified set of rotations $r \in \mathcal{R}_t$, where $\mathcal{R}_t = \{1...R_t\}$.

The $\mathcal{IFP}_{tr}$ and $\mathcal{NFP}_{tr}^{t'r'}$ are used as the basis for the geometric constraints. The constraints to handle the non-overlap of pieces are the most challenging ones, both in their expression and in the algorithms for propagating their effects. They convey the essence of all the irregular cutting and packing problem variants.

4.1. CP models based on the dotted board model

In the dotted board model [29], boards are discretized as meshes of dots, and there is a binary variable for each combination of a piece type with one of its possible rotations in each dot of the board.

4.1.1. Binary representation for pieces on board dots

The dotted board model uses a binary representation for piece types on board dots. A binary variable δ_{trd} represents piece type t with rotation r placed on dot d. Specifically, variable δ_{trd} is defined as 1 if the reference point of a piece of type t at rotation r is on dot d; and 0 otherwise. δ_{trd} is therefore zero for all dots $d \notin \mathcal{IFP}_{tr}$.

This representation follows closely the approach in several

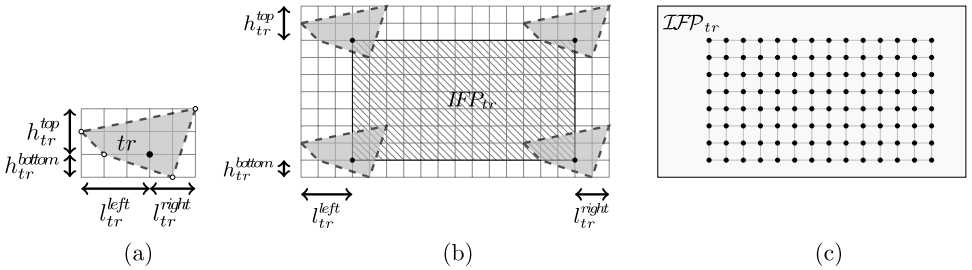


Fig. 3. (a) vertical and horizontal distances from a reference point of a piece (represented by a black dot) to its bounding box; (b) *IFP* of the piece; (c) set of dots $\mathcal{IFP}$.

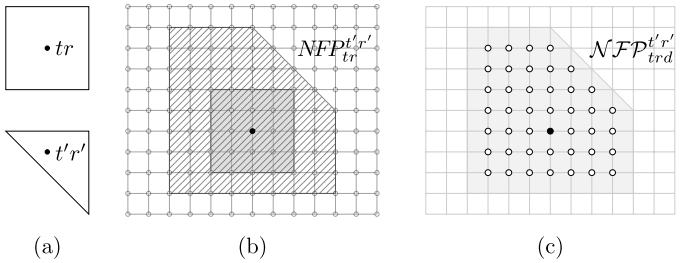


Fig. 4. (a) piece type t at rotation r and piece type t' at rotation r'; (b) the nofit polygon $NFP_{tr}^{t'r'}$; (c) the set of dots $\mathcal{NFP}_{trd}^{t'r'}$.

mathematical programming models for the irregular cutting and packing problem in the literature, where each variable carries the information concerning a specific piece type, rotation and dot. One feature of this representation is that it has to deal with a large number of variables with small (binary) domains. For example, a small problem where ten piece types, with four possible rotations each, are candidates to be placed in a board with one hundred dots (a 10×10 mesh) needs a number of binary variables in the order of four thousand ($10 \times 4 \times 100 = 4000$ minus the variables that are eliminated by the inner-fit polygon) to represent the placement of the pieces in a solution.

Non-overlap constraints based on the binary representation

Consider that a piece of type t at rotation r is placed on dot $d \in \mathcal{D}$. In order to enforce the non-overlap between t and piece type t' at rotation r', all the variables $\delta_{t'r'd'}$, for the dots d' inside the nofit polygon between the two piece types and rotations ($d' \in \mathcal{NFP}_{trd}^{t'r'}$), are set to 0 (*if-then* constraint (1)).

$$\text{If } (\delta_{trd} = 1) \text{ Then } \left(\delta_{t'r'd'} = 0\right), \forall\, t, t' \in \mathcal{T}, r \in \mathcal{R}_t, r' \in \mathcal{R}_{t'}, d \in \mathcal{D}, d' \in \mathcal{NFP}_{trd}^{t'r'}. \tag{1}$$

This constraint propagates when one δ_{trd} is set to 1 in the search, i.e. when one piece is positioned.

To avoid the overlap for all the pieces, $\sum_{d \in D} \sum_{t=1}^{T} \sum_{r=1}^{R_t} \sum_{t'=t}^{T} \sum_{r'=1}^{R_{t'}} |\mathcal{NFP}_{trd}^{t'r'}|$ constraints are needed, where $|.|$ denotes the cardinality of the set.

4.1.2. Integer representation for pieces on board dots

Using binary variables to represent each scenario of a piece type, a rotation and a dot where the piece is positioned, leads to a large number of variables. As an alternative, we consider an integer domain variable to represent the status of a board dot. With this representation, the number of variables is reduced while their domains have more values. For the integer representation we map each piece type $t \in \mathcal{T}$ at a particular rotation $r \in \mathcal{R}_t$ to a single number n_{tr} given by

$$n_{tr} = (t-1) \times R^{\max} + r,$$

where $R^{\max} = \max_{t \in \mathcal{T}} R_t$. Note that this mapping returns a unique integer for a specific piece type t at rotation r and has a simple inverse transformation. Given n_{tr}, we obtain the piece type t and its rotation r as

$$t = \lceil \frac{n_{tr}}{R^{\max}} \rceil$$

$$r = n_{tr} - \lfloor \frac{n_{tr}}{R^{\max}} \rfloor R^{\max}.$$

Let γ_d, $d \in \mathcal{D}$, be the decision variable associated with dot d of the board. The domain of γ_d is the set of possible values for n_{tr} corresponding to the piece types t and rotations r that can be placed on dot d. The value zero represents the situation where no piece reference point can be placed on the dot. For ease of notation, and where no ambiguity may occur, we use the same symbol γ_d for the domain of variable γ_d, initially defined as follows

$$\gamma_d = \{n_{tr} | t \in \mathcal{T}, r \in \mathcal{R}_t\} \cup \{0\}, \qquad \forall\, d \in \mathcal{IFP}_{tr}.$$

Consider an example with three piece types. Piece type 1 has two possible rotations ($R_1 = 2$) and piece types 2 and 3 have one rotation ($R_2 = 1$ and $R_3 = 1$), therefore $R^{\max} = 2$ and n_{tr} will take the following values:

$$n_{11} = (t-1) \times R^{\max} + r = 0 \times 2 + 1 = 1$$
$$n_{12} = (t-1) \times R^{\max} + r = 0 \times 2 + 2 = 2$$
$$n_{21} = (t-1) \times R^{\max} + r = 1 \times 2 + 1 = 3$$
$$n_{31} = (t-1) \times R^{\max} + r = 2 \times 2 + 1 = 5.$$

The mapping results in a unique identification for each piece type and rotation. Note that the mapping may have gaps in the ranges.

By using this approach, the number of decision variables is significantly smaller, specifically, of order $T \times R^{\max}$. There is however still a problem with this representation: in a solution, each variable must be assigned a single value, and therefore this representation prevents the reference point of more than one piece from being placed on the same dot. Fig. 5a illustrates two pieces whose chosen reference points lead to a feasible positioning pattern which would not be possible with our mapping. But this problem is overcome by choosing only reference points in the interior of the pieces such that if two pieces are placed on the same dot, then they overlap (Fig. 5b).

Non-overlap constraints for the integer representation

Consider that a piece of type t at rotation r, mapped as n_{tr}, is placed on dot $d \in \mathcal{D}$. In order to enforce the non-overlap between t and piece type t' at rotation r', all the variables $\delta_{t'r'd'}$, for the dots d' inside the nofit polygon between the two piece types and rotations ($d' \in \mathcal{NFP}_{trd}^{t'r'}$), piece type t' at rotation r' must be removed from the domain (*if-then* constraint (2))

$$\text{If } \gamma_d = \{n_{tr}\} \text{ Then } \gamma_{d'} \neg \supset \left\{n_{t'r'}\right\},$$
$$\forall\, t, t' \in \mathcal{T}, r \in \mathcal{R}_t, r' \in \mathcal{R}_{t'}, d \in \mathcal{D}, d' \in \mathcal{NFP}_{trd}^{t'r'}. \tag{2}$$

Constraints (2) propagate when variable γ_d becomes ground to some positive integer value, i.e., when the reference point of a specific piece t at rotation r is positioned on dot d. When the domain of a variable is reduced, more than one feasible value may still be left, corresponding to the set of piece types and rotations that may still be positioned in the

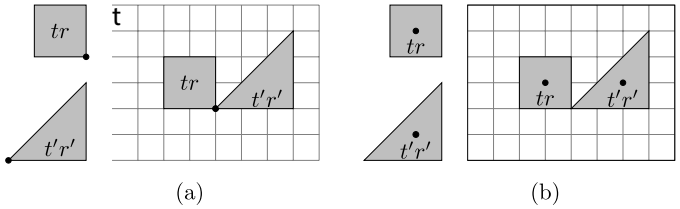


Fig. 5. (a) placement position for a feasible solution that cannot be expressed in the integer representation; (b) example with reference points in the interior of the pieces where the placement position may be expressed using the integer representation.

dot. Although the number of constraints to enforce non-overlap is exactly the same in the integer and binary representations, this approach uses less variables.

4.2. NoOverlap: *a new global constraint*

Despite the amount of information that the integer representation has, the non-overlap constraints derived from built-in generic constraints such as *if-then* are not very effective at propagating the effects of piece positioning. This is a well-known problem in constraint programming, where the very flexible expressions for constraints are not matched by the effectiveness of the built-in constraint propagation and domain filtering methods. This has led to the systematic exploration of the mathematic properties of constraints for many well-known problems, and the development of algorithms to take advantage of problem structure in constraint programming. Beldiceanu et al. [4] present a systematic report of 235 global constraints, ranging from general-purpose methods for handling simple dependencies in domain variables, such as the well-known *alldiff* constraint, to constraints aimed at real-world problems such as scheduling or vehicle routing. The common feature of global constraints is that they include some propagation methods that account for the specifics of the problem at hand.

For cutting and packing, we already mentioned previous work that included the development of global constraints. In these approaches the limitations were due to the difficulty of enforcing the 2-D constraints intrinsic to the problem, in models where $< x, y >$ representations were used. The dot-based representations we explore here do not present such difficulties, and it is therefore easier to take into account the structure of cutting and packing constraints. We argue that developing a new global constraint tailored for the problem reduces the number of non-overlap constraints and makes propagation more effective.

The new global constraint NoOverlap is now proposed to deal with the geometric constraints intrinsic to the nesting problem. Let us consider a scenario where piece type t at rotation r is placed on dot d. We then define Φ_{trd} (constraint 3) to represent the set of dots where no other piece can be positioned

$$\Phi_{trd} = \left\{ d' | d' \in \bigcap_{t' \in \mathcal{T}, r' \in \mathcal{R}_{t'}} \mathcal{NFP}_{trd}^{t'r'} \right\}. \tag{3}$$

Fig. 6 illustrates Φ_{trd} for an example where three pieces (Fig. 6a) must be positioned. Fig. 6b presents the nofit polygons of piece type t at rotation r with the two other pieces and corresponding rotations. When piece t at rotation r is placed on the dot d, $NFP_{trd}^{t'r'}$ and $NFP_{trd}^{t''r''}$ intersect as shown in Fig. 6c. The dots strictly inside the intersection of the shaded regions define the set Φ_{trd}.

Φ_{trd} is the set of all dots where no piece can be positioned if piece type t at rotation r is placed on dot d. Using this set, the NoOverlap constraint is defined. With NoOverlap, we look for opportunities to exclude regions of the plane that are known to be forbidden to any piece type, and can therefore be filtered from the domains of the dot variables. The exclusion is based on the pre-computation of intersections between nofit polygons, that can be performed prior to the resolution of the problem, and used repeatedly in the search process.

$$\begin{aligned} &\forall\, d \in \mathcal{D},\ t, t' \in \mathcal{T},\ r \in \mathcal{R}_t,\ r' \in \mathcal{R}_{t'} \\ &\quad \text{If } \gamma_{d'} = \left\{ n_{t'r'} \right\}, \text{ Then } \quad \forall\, d' \in \Phi_{trd} \qquad \gamma_{d'} = \{0\} \\ &\quad\qquad\qquad \text{and} \quad \forall\, d' \in \mathcal{NFP}_{trd}^{t'r'} \quad \gamma_{d'} \neg \supset \left\{ n_{t'r'} \right\}. \end{aligned} \tag{4}$$

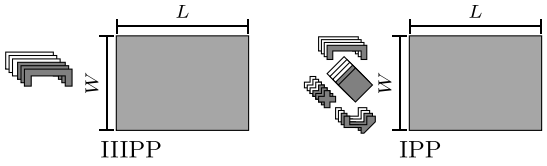
Given sets Φ_{trd} and $\mathcal{NFP}_{trd}^{t'r'}$, the rules for activation of the constraint propagator are easily stated, and so are their effects. This constraint is activated each time variable γ_d is ground, i.e. takes a specific value. The domain of all variables in Φ_{trd} set can then be reduced to $\{0\}$. Moreover, the domain of each $\gamma_{d'}$, $d' \in \mathcal{NFP}_{trd}^{t'r'}$ must be reduced by the value $n_{t'r'}$, for all $t' \in \mathcal{T}$ and $r' \in \mathcal{R}_{t'}$.

The constraint propagator is represented in Algorithm 1. In the second step of the algorithm the filtering of values from the domain of $\gamma_{d'}$ can be made more efficient by avoiding the dots that were processed in the previous step, namely those in set Φ_{trd}. The NoOverlap constraint propagates just in case γ_d is bound to a value other than zero. A single constraint is assigned to each dot and the propagation method is used to reduce the domains of the variables.

5. CP models for all the variants of irregular cutting and packing problems

While the non-overlap constraints are present in all irregular cutting and packing problem variants, for each variant a set of additional constraints has to be considered. In the following, we present the constraint programming models for each problem variant shown in Section 3.1. These models are composed by a set of built-in constraints based on the binary or integer variable representation and the non-overlap constraints presented in Sections 4.1.1, 4.1.2 and 4.2.

5.1. *Irregular Placement Problem* (IPP) *and Irregular Identical Item Placement Problem* (IIIPP)



The models for the IPP and the IIIPP are similar, as the difference between these two problem variants is in the number of piece types and this is a characteristic of the problem instance.

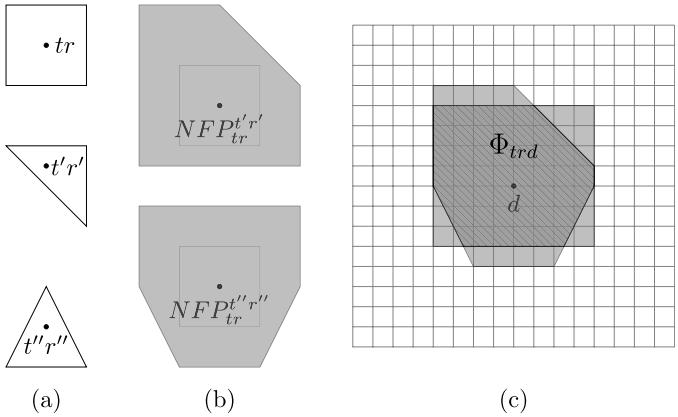


Fig. 6. Example of set Φ_{trd}.

- **Input:** The variables γ_d and sets Φ_{trd} and $\mathcal{NFP}^{t\,r'}_{trd}$, for all $d \in \mathcal{D}, t \in \mathcal{T}, r \in \mathcal{R}_t$;
- **begin**

Calculate $t = \lceil \frac{n_{tr}}{R^{\max}} \rceil$ and $r = n_{tr} - \lfloor \frac{n_{tr}}{R^{\max}} \rfloor R^{\max}$;
For all $d' \in \Phi_{trd}$ do

$\gamma_{d'} = \{0\}$;
For all $\left(t' \in \mathcal{T}, r' \in \mathcal{R}_{t'}, d' \in \mathcal{NFP}^{t\,r'}_{trd}\right)$ do

Remove $n_{t'r'}$ from $\gamma_{d'}$ domain;
Return;

- **end.**

Algorithm 1. NoOverlap propagator.

In these problems, the board dimensions are defined by the instance, and therefore the initial domains can be determined in the pre-processing phase using the *IFP*, as shown in Section 3.2. The models for the IPP (or IIIPP) are completed by adding the non-overlap constraints and an appropriate objective function.

In the binary representation, the non-overlap constraints are represented in (1), capturing the fact that when a piece is positioned, any other piece is excluded from the points in the *NFP* for the two pieces.

The IPP and the IIIPP are both output maximization problems, therefore the number of boards available to be cut is fixed and the objective function must maximize the value of the pieces cut (extracted) from these boards. Specifically, considering that each piece of type $t = 1, ..., T$ has a value v_t, the objective function that maximizes the value can be expressed as:

$$\text{maximize} \sum_{d \in D} \sum_{t=1}^{T} \sum_{r=1}^{R_t} v_t \delta_{trd}. \tag{5}$$

If v_t is defined as the area of the pieces, this objective minimizes the waste. The objective function (5) and non-overlap constraints (1) compose the binary formulation of the IPP and IIIPP.

In the integer representation, the non-overlap constraints may be the ones in (2) or the proposed global constraint NoOverlap (4), capturing the fact that when a piece is positioned, other pieces are excluded from the points in the $\mathcal{NFP}$ between the two pieces.

As the pieces are associated to integers that code their position, the objective function is slightly different. For each piece of type $t = 1, ..., T$ at rotation $r = 1, ..., R_t$, the number of dots (γ variables) that were bound to n_{tr} must be counted and then this number is multiplied by the piece value. The sum of these values is the objective function value. This expression can be formulated using the built-in constraint count that is usually available in constraint programming solvers. Expression (6) is the objective function for output maximization problems formulated with integer-domain variables.

$$\text{maximize} \sum_{t=1}^{T} \underset{d \in D, r=1,...,R_t}{\text{count}} (\gamma_d = n_{tr}) \times v_t. \tag{6}$$

The objective function (6) together with constraints (2) define the integer formulation of IPP and IIIPP.

The integer formulation with the global constraint NoOverlap is obtained if the objective function (6) is combined with the new global constraint NoOverlap (4).

*5.2. Constrained Irregular Placement Problem (*IPPc*) and Irregular Knapsack Problem (*IKP*)*

The models for the IPPc and the IKP can be built on the model for the IPP simply by adding a constraint that limits the number of pieces of each type to cut. In these two variants, as in the IPP, the board dimensions are defined by the instance, thus, the initial domains can be determined in the pre-processing phase using the $\mathcal{IFP}$ (see Section 3.2).

The IPPc and the IKP have similar models, because the difference between these two problem variants is the demand for each piece type, again a characteristic of the problem instance.

In the binary representation, a set of constraints is required to limit the number of pieces to cut. These constraints count the number of times that a piece type is present in the solution and ensure that this number is less than or equal to the limit q_t for piece type t:

$$\underset{d \in D, r=1,...,R_t}{\text{count}} (\delta_{trd} = 1) \leq q_t, \quad t = 1, ..., T. \tag{7}$$

The model for IPPc and IKP with binary decision variables is therefore obtained by adding the objective function (5) and constraints (1) and (7).

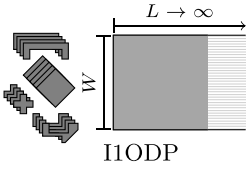
In the integer representation, a set of constraints is required to ensure that the demand for the pieces is not exceeded. For all piece types t, the constraints count the number of times the variables assume the value n_{tr}, for all $r = 1, ..., R_t$, and require that this number is less than or equal to the limit q_t for piece type t:

$$\underset{d\in D, r=1,...,R_t}{\text{count}} (\gamma_d = n_{tr}) \leq q_t, \quad t = 1, ..., T. \tag{8}$$

The model for IPPc and IKP for decision variables with integer domains is obtained by adding the objective function (6) and constraints (2) and (8).

The integer formulation with the global constraint NoOverlap is complete with the objective function (6), constraints (8) and the global constraint NoOverlap (4).

5.3. Irregular one open dimension problem (I1ODP)



The I1ODP is an input minimization problem, therefore the length of the board is not known. However, defining an upper bound to the solution length ($\bar{L}$) is enough for the board to be considered finite and rectangular. Knowing the value of $\bar{L}$, the initial domains of the decision variables can be determined in the pre-processing phase using the *IFP* (see Section 3.2).

In all input minimization problems, the demand q_t of each piece of type $t = 1, ..., T$ is known and needs to be met. The objective is to minimize the board length guaranteeing that the demand is met.

In the binary representation, the set of constraints required to ensure that the demand for the pieces is met count the number of times that piece type t is in the solution and ensure that this number is equal to the demand q_t:

$$\underset{d\in D, r=1,...,R_t}{\text{count}} (\delta_{trd} = 1) = q_t, \quad t = 1, ..., T. \tag{9}$$

To express the objective function, the position of each piece is analyzed to determine the value of the objective function. As the goal is to minimize the used board length, the piece that occupies the rightmost position on the board determines the value for the objective function. Considering the binary formulation this objective can be represented as:

$$\text{minimize} \quad \max_{\substack{d\in D \\ t=1,...,T \\ r=1,...,R_t}} (d_x + l_{tr}^{right})\delta_{trd}. \tag{10}$$

where d_x is the horizontal distance from the left side of the board to dot d.

Together, the objective function (10) and constraints (1) and (9) comprise the model of I1ODP using binary variables.

In the integer representation, a set of constraints is required to account for the demand. For each $t = 1, ..., T$, the number of times that variable γ_d assumes the value n_{tr} is counted, for all $r = 1, ..., R_t$, an required to be equal to demand q_t of piece type t:

$$\underset{d\in D, r=1,...,R_t}{\text{count}} (\gamma_d = n_{tr}) = q_t, \quad t = 1, ..., T. \tag{11}$$

In order to measure the objective of I1ODP, additional constraints are used. Consider the variable $\mathcal{L}$ that measures the solution length. We must ensure that $\mathcal{L}$ will be at least as long as the solution length.

$$\text{If}(\gamma_d = n_{tr})\text{Then}(\mathcal{L} \geq d_x + l_{tr}^{right}), \quad d \in D, t = 1, ..., T, r = 1, ..., R_t. \tag{12}$$

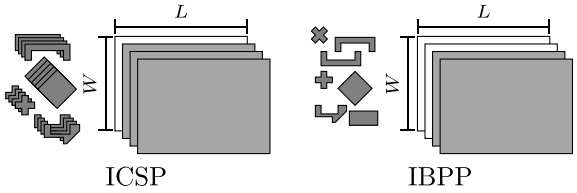
Using constraints (12), the objective function for the irregular 1*ODP* can be expressed as:

$$\text{minimize} \quad \mathcal{L}. \tag{13}$$

The model for the I1ODP with integer-domain variables can be represented by objective function (13) supported by Constraints (2), (11) and (12).

The integer formulation with the global constraint NoOverlap comprises the objective function (13), constraints (11), (12) and the global constraint NoOverlap (4).

5.4. Irregular cutting stock problem (ICSP) and irregular bin packing problem (IBPP)



The difference between ICSP and IBPP is the demand for each piece type, i.e. the formulations may be the same and only the instances differ.

This problem aims to cut all the demanded pieces from N boards, minimizing the number of used boards. Each piece must be completely inside one of the boards and it is considered that all the boards have the same height H and length L_{board}.

To address this problem, the number of boards required to cut all the pieces is estimated. In order to represent the problem using the same definition for the variables, consider an extended board of height H and length $L = N \times L_{board}$. $N - 1$ vertical cuts are made in the extended board dividing it into the N original boards. In order to avoid placing the pieces over the cuts, some additional constraints on the *IFP*s must be considered and will be presented for each kind of variable domain. Fig. 7 illustrates an example of a board and the new *IFP*s for a piece of type t at rotation r (IFP_{tr}).

According to this board definition, the objective function for ICSP and IBPP is the same as the one used to represent I1ODP. This objective ensures that the number of boards used will be minimized and that the used length of the last board will be reduced, so that the material waste of this last board is also minimized.

In the binary representation, to define the *IFP*s, consider that each dot $d \in D$ has coordinates (d_x, d_y). The region of IFP_{tr} can be inferred by fixing the domain of δ_{trd} to zero to avoid piece type t in rotation r to be over the cuts, i.e. $d_x - l_{tr}^{left} < k \times \frac{L}{N}$ or $d_x + l_{tr}^{right} > (k+1) \times \frac{L}{N}$, for $k = 1, ..., N-1$.

Note that as the dots, the dimensions of the pieces and the dimensions of the board are known, the domains of the variables can be reduced in a pre-processing phase.

Considering the definition of the board and the corresponding domain reductions, the objective function (10) together with Constraints (1) and (9) models the ICSP and the IBPP with binary variables.

In the integer representation, similarly to the binary case, the *IFP*s can be used to reduce the domains of γ_d by n_{tr} if $d_x - l_{tr}^{left} < k \times \frac{L}{N}$ or $d_x + l_{tr}^{right} > (k+1) \times \frac{L}{N}$ for $k = 1, ..., N-1$. These domain reductions can be done in the pre-processing phase.

Considering the board definition presented in this section and the corresponding domain reductions, the objective function (13) together with Constraints (2) and (11) models the ICSP and the IBPP with integer variables.

The integer formulation with the global constraint NoOverlap comprises the objective function (13) together with constraints (11) and the global

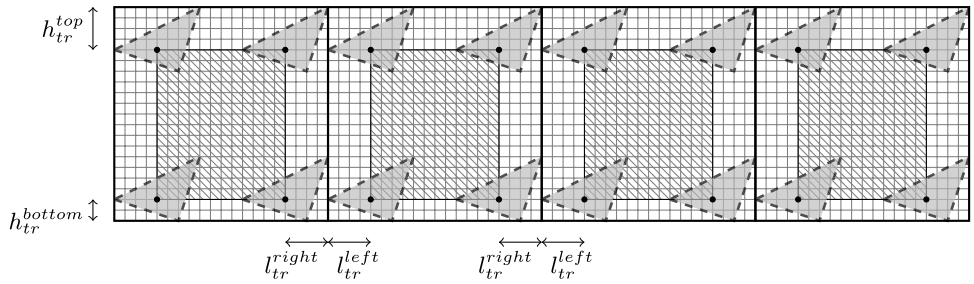
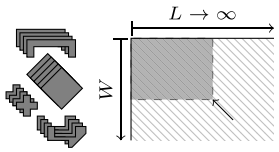


Fig. 7. Board used on irregular cutting stock problems and irregular bin packing problems.

constraint NoOverlap (4).

5.5. Irregular two open dimension problem (I2ODP)



I2ODP

Developing exact methods for the I2ODP demands some more effort compared with the other cutting and packing problem variants. In this problem variant, two board dimensions are estimated. This may lead to a large number of dots on the board and, consequently, to a formulation with a large number of variables. Nevertheless, depending on the objective function, the domains of some variables may be reduced in the pre-processing phase.

If the objective is to minimize the area of a rectangle that contains all the pieces and if we assume that all the pieces fit in a rectangle of area A, then, for each piece type t and rotation r, the (X,Y)-coordinates of the positioning point of the piece have to respect the following inequality:

$$(x + l_{tr}^{right}) \times (y + h_{tr}^{bottom}) \leq A. \tag{14}$$

Note that placing a piece that does not respect this inequality on a dot makes the used board exceed the area A. The objective function takes advantage of this observation. It is also clear that the board will be at least as long (high) as the longest (highest) piece type t at rotation r. Fig. 8 shows a rectangular board where the region defined by inequality (14) is represented in light gray. The x and y coordinates in this board represent $x + l_{tr}^{right}$ and $y + h_{tr}^{top}$, respectively. Any dot in the gray area has to be considered, but the rest of the board is excluded under the assumption of the total area not exceeding A. It is important to highlight that as these irregular boards are defined at a pre-processing phase, the domain of some variables can be inferred, reducing the number of constraints in the model.

When two dimensions are open and the objective is to minimize the used area, the objective is non linear. This poses no problem to constraint programming, as CP models deal with non-linearity in constraints or in the objective function.

It is important to highlight that open dimension problems can have many different objectives. The minimization of the board length and the minimization of the area of the rectangle were chosen because these objectives were already studied in the literature.

In the binary representation, to minimize the objective function, this rectangular area is expressed as:

$$\text{minimize} \quad \max_{\substack{d\in D\\ t=1,\ldots,T\\ r=1,\ldots,R_t}} (d_x + l_{tr}^{right})\delta_{trd} \times \max_{\substack{d\in D\\ t=1,\ldots,T\\ r=1,\ldots,R_t}} (d_y + h_{tr}^{bottom})\delta_{trd}. \tag{15}$$

To represent the I2ODP, constraints (1) and (9) are combined with objective function (15).

In the integer representation, additional constraints are used to define

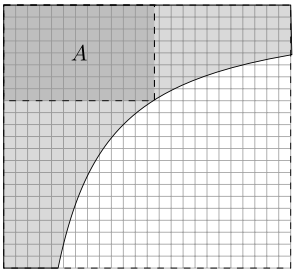


Fig. 8. Board for open dimension problems: the area under the $x \times y = A$ curve.

the objective function. Consider the variable $\mathcal{L}$ to represent the length of the solution and $\mathcal{W}$ to represent the height of the solution. Constraints (12) ensure that $\mathcal{L}$ represents the used board length; for the board width, a similar set of constraints is required:

$$\text{If}(\gamma_d = n_{tr})\text{Then}(\mathcal{W} \geq d_y + h_{tr}^{top}), \quad d \in D,\ t = 1,\ \ldots, T,\ r = 1,\ \ldots, R_t. \tag{16}$$

To minimize the area of the rectangle that contains all the pieces, the objective function is:

$$\text{minimize} \ \ \mathcal{L} \times \mathcal{W}. \tag{17}$$

The formulation of the I2ODP with integer variables is obtained using objective function (17) subject to constraints (2), (11), (12) and (16).

The integer formulation with the global constraint NoOverlap comprises the objective function (17) together with constraints (11), (12), (16) and the global constraint NoOverlap (4).

6. Computational experiments with the constraint programming models

This section presents computational experiments with all the constraint programming models proposed. The experiments were run on a computer with an Intel Xeon Processor E5-2450 with 64 GB of memory using the operating system Scientific Linux 6. The maximum solution time allowed for each problem with any method was one hour. Each problem formulation was implemented and solved by using the constraint programming solver provided by IBM ILOG CPLEX 12.6.

In order to identify the problem variant and the constraint programming model used to solve it, we used an abbreviation of the problem variant's name and a designation of the model. The constraint programming models are abbreviated as Bin (binary variables), Int (integer variables) or IGC (integer variables and the global constraint NoOverlap). As an example, the irregular placement problem (IPP) solved by the constraint programming model with binary variables (Bin) is called IPP − Bin.

6.1. Defining instances

The instances used in the computational experiments are based on well-known instances from the literature on irregular open-dimension

problems with one open dimension (I1ODP). As this is the most studied variant of the problem, many instances were proposed to evaluate solution methods. To evaluate and compare the performance of the models proposed, a subset of instances was chosen from ESICUP[1]. The chosen instances are Three, Threep2, Threep2w9, Threep3, Threep3w9, Blaz1, Blaz2, Shapes0, Shapes1, Fu and Dagli. In Table 1, we present more details for these instances.

As in the *IIPP* only one piece type must be placed on the board, we created eight new instances based on instances Blaz2 and Shapes1 that have convex and non-convex pieces. Each instance has only one piece type of the original instance, and whenever in the original instance a piece has allowed rotations, this characteristic is maintained in the new instance.

The mesh used for instances Blaz1, Blaz2, Shapes0 and Shapes1 and all instances Three has a refinement $\Delta = 1$ and the mesh used to solve Fu and Dagli has a refinement $\Delta = 2$. Choosing different values of Δ for the instances ensures that they can be solved by the three proposed models. Fig. 9 shows how the refinement Δ changes the mesh.

The experiments were run in two phases. In the first phase, this set of instances and the proposed Δ were used. In the second phase, the model that performed best in the first phase was used to solve the larger instances Blaz1, Blaz2, Shapes0, Shapes1, Fu and Dagli and also Jakobs1, Jakobs2 and Shirts with a more refined value for Δ, adapting the instances to each problem variant studied. The mesh refinement Δ used in each instance in the second phase is specified in the results tables.

The instances used for each problem variant were derived from these base instances. Details on the changes made for adapting them to each problem variant are given in Appendix A.

6.2. Performance of the formulations proposed

This section is dedicated to analyzing the performance of the three models presented in the previous sections. For that, we solved a set of instances by using the three models. All the details on the computational experiments can be found in Appendix B.

Table 2 presents a summary of the results. The first and second columns show, respectively, the problem name and the number of instances evaluated.

For each problem variant and model, we report the number of feasible solutions (feas.), the number of optimal solutions (opt.) and the number of best (or equal) solutions (best). The average time to solve the instances (Avg. time) is also provided. To perform a fair comparison of the models, the average time only takes into account the instances in which optimality was proven by the three formulations.

The performance of the Binary and Integer models is similar. Both models obtained the same number of feasible solutions for all the problem variants. Furthermore, the number of instances for which the solution was proven optimal and the number of best solutions obtained were close. The average computational time for the integer formulation is about 24% smaller than the average computational time of the binary formulation.

In general, the reason why the Binary and Integer models do not find feasible solutions for all the instances is that they exceed the computational resource which is limited to 64 GB of RAM in our experiments. As the IGC formulation drastically reduces the number of variables and constraints of the model, this problem is overcome.

The IGC model found feasible solutions for all the instances tested. The number of instances for which optimality was proven is slightly higher compared with the other two formulations and, in most cases, the solutions of the IGC model are better or equal regarding the computational time. In average the IGC model is about 63% faster than the Binary model and 51% faster than the Integer model.

These experiments show that constraint programming can be used to represent and solve all the cutting and packing problems classified in [33], even when they involve non-linearities such as in the I2ODP variant. Moreover, these constraint programming models can easily incorporate particular features for some applications. In addition to the flexibility of the formulations, using the proposed global constraint NoOverlap reduces the number of non-overlap constraints and makes propagation more effective. This proved to be effective in solving larger instances.

6.3. Memory usage

An important information when an approach is chosen to solve a problem is the amount of memory it uses. It is clear that each problem variant uses a different amount of memory since the number of variables and constraints are different. Notwithstanding the contrast in memory consumption of the problem variants, the main aspects of the constraint programming models proposed can be observed in all the problem variants in different scales.

In this section, the memory requirements of the constraint programming models proposed is presented for two irregular cutting and packing problem variants: an output maximization problem, the IPP, and an input minimization problem, the I1ODP. As the memory usage of instances Three was below 0.03 gigabytes these results were not considered in the tables.

6.3.1. Memory usage for an output maximization problem, the irregular placement problem (IPP)

Table 3 shows the memory used to solve the instances presented in Section B.1.2 for the IPP using the three constraint programming models proposed. In the table, the first column shows the instance names. Columns two, three and four show the memory (in gigabytes) used by the binary model, the integer model and the integer model with NoOverlap constraint, respectively.

Comparing the amount of memory used by all the methods, it can be observed that the memory usage of the IPP − IGC model is an order of magnitude lower than in the IPP − Bin and IPP − Int models. The low amount of memory required by the IPP − IGC is a result of the reduced number of non-overlap constraints. The number of variables of IPP − Int is lower than in IPP − Bin, however the memory usage is similar in both approaches. This happens because the number of constraints needed to represent the problem in both approaches is similar and the number of constraints is considerably higher than the number of variables.

Specifically, as stated in Sections 4.1.1 and 4.1.2, the number of constraints needed to enforce the non-overlap among pieces in the Binary and Integer models is $\sum_{d\in D}\sum_{t=1}^{T}\sum_{t'=t}^{T}\sum_{r=1}^{R_t}\sum_{r'=1}^{R_{t'}}|\mathcal{NFP}_{trd}^{t'r'}|$ while the number of constraints needed to avoid this overlap using the NoOverlap constraint is $|\mathcal{D}|$. Furthermore, the number of variables needed to create the non-overlap constraints in the binary model is $|\mathcal{D}| \times \sum_T R_t$ while in the integer models only $|\mathcal{D}|$ variables are needed.

Naturally, the pattern of memory usage for the IPPc is very similar to the one for the IPP since the models are very similar. The problem variant IKP also follows the same pattern but globally the number of constraints and variables is higher, preventing some instances from being solved. The problem variant IIIPP uses other instances, however the pattern is similar to the one presented in Table 3.

6.3.2. Memory usage for an input minimization problem, the irregular one open dimension problem (I1ODP)

Input minimization problems usually use more memory than output maximization ones since the number of decision variables and constraints depends on the initial number of dots and consequently on the initial estimation (upper bound) used for the board length.

Table 4 represents the memory used for the problem variant I1ODP to solve the same instances as in Table 3. The content of the columns is as described for Table 3.

For this problem variant the I1ODP − Bin and the I1ODP − Int used also considerably more memory to solve the instances compared with

[1] EURO Special Interest Group on Cutting and Packing: https://www.euro-online.org/websites/esicup/

Table 1
Characteristics of the instances used in the computational experiments.

Instance	Number of piece types	Total number of pieces	Vertices by piece (avg.)	Feasible rotations	Piece shapes
Three	3	3	3.7	0	Convex
Threep2	3	6	3.7	0	Convex
Threep2w9	3	6	3.7	0	Convex
Threep3	3	9	3.7	0	Convex
Threep3w9	3	9	3.7	0	Convex
Blaz1	7	20	6.3	0, 180	Convex and Non-convex
Blaz2	4	16	7.5	0, 180	Convex and Non-convex
Shapes0	4	43	8.7	0	Convex and Non-convex
Shapes1	4	43	8.7	0, 180	Convex and Non-convex
Fu	12	12	3.6	0, 90, 180, 270	Convex
Dagli	10	30	7.3	0, 180	Convex and Non-convex

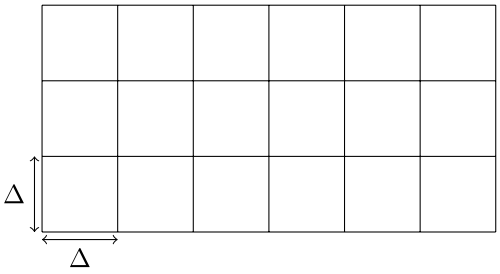


Fig. 9. A mesh with refinement Δ.

pattern in IBPP and ICSP and is higher for IBPP when there is more than one copy of each piece type. Finally the I2ODP problem variant has a larger memory consumption compared with I1ODP or ICSP since the board used in this model has two dimensions to be estimated. Despite the differences between the models, the pattern of memory usage is the same as the one presented for the other variants of input minimization problems.

7. Solving larger instances and comparing with the literature

Section 6.2 showed the flexibility of constraint programming models to solve different irregular cutting and packing problem variants. It can be observed that, for the same problem variants, the constraint programming model with the NoOverlap global constraint - IGC - performs better than the two other constraint programming models proposed. Furthermore, as seen in Section 6.3, the memory usage of IGC is smaller than in the other models by more than one order of magnitude. For these two reasons, the IGC was the constraint programming model chosen to solve the larger instances in Section 7.1 and to make comparisons with the literature in Section 7.2.

Table 2
Comparison of the performance of the three proposed models.

		Bin				Int				IGC			
Prob.	Num.	Number of Avg.				Number of Avg.				Number of Avg.			
Vari.	of inst.	feas.	opt.	best	time	feas.	opt.	best	time	feas.	opt.	best	time
IIIPP	8	8	2	7	959.4	8	2	7	680.1	8	2	7	107.8
IPP	11	9	5	7	14.9	9	5	7	8.6	11	5	10	5.9
IPPc	11	9	5	6	17.6	9	5	6	7.2	11	5	11	2.9
IKP	11	7	4	6	5.8	7	4	6	9.5	11	5	11	385.7
I1ODP	11	7	5	5	59.6	7	5	6	49.7	11	5	11	28.7
ICSP	11	7	4	7	4.4	7	4	6	3.2	11	5	11	3.0
IBPP	11	6	5	6	498.8	6	4	6	430.3	11	5	11	48.6
I2ODP	11	7	5	5	4.4	7	5	6	3.2	11	5	10	3.0
	Average	7.5	4.4	6.1	195.6	7.5	4.2	6.2	149.0	10.6	4.6	10.2	73.2

Table 3
Memory usage (in GB) to solve the Irregular Placement Problem (IPP).

Instance	IPP – Bin	IPP – Int	IPP – IGC
Blaz1	3.50	3.00	0.10
Blaz2	1.40	0.96	0.10
Shapes0	16.60	15.40	0.80
Shapes1	35.80	33.40	1.00
Fu	om	om	0.30
Dagli	om	om	1.60

om: out of memory.

Table 4
Memory usage (in GB) to solve the Irregular one Open Dimension Problem (I1ODP).

Instance	I1ODP – Bin	I1ODP – Int	I1ODP – IGC
Blaz1	9.80	8.60	0.15
Blaz2	2.10	3.00	0.33
Shapes0	om	om	2.00
Shapes1	om	om	1.40
Fu	om	om	0.17
Dagli	om	om	0.84

om: out of memory.

the I1ODP – IGC. As in this case the need for memory is high, only instances Blaz1 and Blaz2 could be solved by all the methods. As expected, the binary and the integer approaches have a similar memory usage, while the approach with the NoOverlap constraint needs a small amount of memory compared to them.

The ICSP problem variant is very similar to the I1ODP, differing only in the definition of the boards, and consequently their memory usage is similar. The difference between IBPP and ICSP is that in IBPP each piece copy is considered as a unique piece type and in ICSP, they are grouped by types. Therefore, the memory usage has the same

In Section 7.1, three problem variants are analyzed, specifically IPP, ICSP and I1ODP. These problem variants were chosen because of their practical relevance. In Section 7.2, the results obtained from the IGC for the problem variant I1ODP are compared with the results from the Dotted Board Model proposed by Toledo et al. [29]. All the experiments in both sections were run using the computer and the solver as described in Section 6.

7.1. Solving larger instances

The size of an irregular cutting and packing instance depends on various factors such as the number of piece types, the total number of pieces to be placed, the number of piece rotations allowed and the board size. In our approach, the number of variables and constraints of the problem is directly related to the discretization of the board, i.e. the number of dots (or admissible positioning points) on the board. Considering this, an instance can be considered small or large depending on the discretization used and, in this section, all the instances are represented by using a mesh with refinement $\Delta = 0.5$.

IPP, ICSP and I1ODP, the variants of the irregular cutting and packing problem selected for these experiments were chosen because of their relevance in the literature. IPP is a classical problem highly studied in the one-dimensional and regular two dimensional cutting and packing problems. This problem emerges as a cutting pattern generator in column generation techniques to solve the cutting stock problem, which can clearly be extended to the irregular case. Column generation techniques lead to the optimal relaxed solution of the cutting stock problem and a feasible integer solution should be obtained by heuristics or branch-and-price techniques. On the other hand, the ICSP model proposed solves the cutting stock problem exactly and each feasible solution found during the search is an integer feasible solution to the problem. Lastly, the I1ODP is the most studied variant among the irregular cutting and packing problems, therefore it is a natural choice for the evaluation with large instances.

The instances used are again the ones proposed for the I1ODP problem: Blaz1, Blaz2, Shapes0, Shapes1, Fu, Dagli, Shirts, Jakobs1, Jakobs2. The instances were taken from the ESICUP website. As in the previous section, the board lengths for the IPP and the ICSP are defined as equal to the board height. Since the lengths of the solutions for instances Jakobs1 and Jakobs2 are shorter than the board height, the board height (and length) considered for these instances was half the one of the original instances to make them more interesting to be solved by the ICSP.

All the instances and problem variants were solved by using the constraint programming model with the global constraint NoOverlap (IGC) and the computational results obtained are presented in Table 5. The first column has the instance name, the value of the best solution found and the computational time needed to find this solution with the IPP (ICSP and I1ODP) are presented in columns two and three (four and five and six and seven respectively).

Even for this smaller discretization, a feasible solution was found for all the problem variants and instances evaluated. Solving the IPP with a smaller discretization produces strictly better solutions compared with the results presented in Table 8 (Section B.1.2) for instances Blaz2, Shapes0, Shapes1, Fu and Dagli. The solution for instance Blaz1 is however worse than the one presented in Table 8. This may be explained by the size of the solution space, which is about four times larger and this may disturb the search for solutions. For the three remaining instances that were not considered in the previous tests, Shirts, Jakobs1 and Jakobs2, feasible solutions were found.

For the ICSP, a solution with equal or better quality was found for instances Blaz1, Blaz2, Shapes0, Shapes1, Fu and Dagli, with a discretization $\Delta = 0.5$, comparing with the solutions presented in Table 11 (Appendix B). Furthermore, feasible solutions were found for instances Shirts, Jakobs1 and Jakobs2. Instances with thousands of pieces, as solved by the heuristic in [5], could not be solved with this constraint programming model, but this approach can be an alternative when the cutting stock problem instances are small. Moreover, using this model, the used length of each board is minimized, leading to a reduction in waste which can be specially advantageous in problems with a relatively small number of pieces.

Table 5
The best solution found for all the problem variants and instances with $\Delta = 0.5$.

	IPP − IGC		ICSP − IGC		I1ODP − IGC	
Instance	Solution	Time to find	Solution	Time to find	Solution	Time to find
Blaz1	203.5	687.4	30.0	1905.3	30.0	193.2
Blaz2	187.5	1717.9	22.0	2400.8	21.5	50.8
Shapes0	1176.0	2995.9	68.0	986.7	65.0	2938.9
Shapes1	1172.0	1124.5	68.0	2605.0	68.0	1867.4
Fu	1430.0	1243.6	34.0	3462.9	34.0	336.3
Dagli	2967.8	2932.6	75.0	524.3	70.0	367.5
Shirts	1815.0	2517.2	65.0	1427.6	65.5	1353.7
Jakobs1	378.5	1506.7	27.0	1729.6	26.5	620.0
Jakobs2	1058.0	2365.3	56.0	144.1	57.0	2983.5

By using a discretization of $\Delta = 0.5$ for the I1ODP, better solutions were obtained for instances Blaz2, Shapes0, Shapes1, Fu and Dagli, compared with the ones presented in Table 13 (Section B.2.3). In the case of instance Blaz1, a worse solution was found. This is possible since the solution space is larger ($\Delta = 0.5$) and better solutions may not be reached within the time limit. For instances Shirts, Jakobs1 and Jakobs2, the model proposed for I1ODP was able to find feasible solutions.

Comparing the solutions in columns ICSP − IGC and I1ODP − IGC in Table 5, it can be observed that for instances Shirts and Jakobs2 the ICSP could find better solutions than the ones found in I1ODP. This behavior is expected since the ICSP is more constrained and thus its solution space is smaller. If, in this smaller search space, there are better solutions for the ICSP and, consequently, for the I1ODP, they may be reached during a time-limited search.

7.2. Comparing with the literature

The results presented in Section 6 demonstrate that an exact approach based on constraint programming models is flexible and can be used to solve many variants of cutting and packing problems. A question that arises is how the constraint programming approach for irregular cutting and packing problems compares with other exact methods in the literature. In fact, to the best of our knowledge, the only irregular cutting and packing problem variant addressed with exact methods in the literature is the I1ODP, therefore, the comparison can only be made with this problem variant. Among the exact approaches proposed in the literature for the I1ODP, we had to choose the one with the same solution space, i.e., one where the reference point of the pieces can only be placed over dots of a discretized board. The Dotted Board Model (DBM) proposed in [29] has these characteristics, thus, the solutions obtained by both methods are comparable. Cherri et al. [10] studied the influence of the discretization used in the model of [29] and proposed a reformulation allowing the pieces to be rotated.

The instances presented in the beginning of this section were used to compare the results of DBM and IGC in the resolution of the I1ODP. Moreover, four sets of smaller instances were created based on the instances Blaz1, Blaz2, Shapes0 and Shapes1. Specifically, the number of pieces in the instance was reduced and the other instance features were

Table 6
The best solution found for all the instances with $\Delta = 0.5$.

	I1ODP − DBMR		I1ODP − IGC	
Instance	Solution	Time	Solution	Time
Blaz1-1	**7.5**	TL	**7.5**	TL
Blaz1-2	20.0	TL	**14.0**	TL
Blaz1-3	28.0	TL	**21.0**	TL
Blaz2-1	**7.0**	**35.9**	7.0	520
Blaz2-2	**11.0**	**5745.9**	11.0	TL
Blaz2-3	**15.0**	TL	**15.0**	TL
Shapes0-1	**14.0**	**211.5**	14.0	TL
Shapes0-2	**14.0**	**223.7**	14.0	TL
Shapes0-3	26.0	TL	**23.0**	TL
Shapes1-1	**14.0**	**435.0**	14.0	TL
Shapes1-2	**14.0**	**1942.1**	14.0	TL
Shapes1-3	25.0	TL	**20.0**	TL
Blaz1	om	om	**30.0**	TL
Blaz2	**21.0**	TL	21.5	TL
Shapes0	om	om	**65.0**	TL
Shapes1	om	om	**68.0**	TL
Fu	om	om	**34.0**	TL
Dagli	om	om	**70.0**	TL
Shirts	om	om	**65.5**	TL
Jakobs1	om	om	**26.5**	TL
Jakobs2	om	om	**57.0**	TL

om: out of memory; TL: Time limit (5 h).

kept. For example, the instance Shapes0-1 includes one copy of each piece type while Shapes-2 and Shapes0-3 respectively should cut two and three copies of each piece type. All the new instances were created using this method.

In Table 6, we compare the computational results obtained by solving the I1ODP using the constraint programming model with the global constraint (I1ODP − IGC) with the ones presented in [10] I1ODP − DBMR – Dotted Board Model with Rotations. The name of the instance is in the first column of the table, columns two and three (four and five) show the solution length and computational time obtained.

The DBM proved optimality for 6 out of 21 instances while the IGC proved optimality for only one instance. On the other hand DBM did not find a solution for eight instances due to lack of memory, while IGC returned solutions for all the instances. Specifically considering the standard instances, none of the models was able to prove optimality within the time limit. Except for the Blaz2 instance, IGC always found better or equal solutions when compared to DBM. It demonstrates how the proposed constraint programming model with the global constraint NoOverlap is powerful to handle large scale instances and indicates a promising area in the study of specialized lower bounds for I1ODP.

8. Conclusions

This work introduces constraint programming models to solve all the variants of irregular cutting and packing problems and presents a new global constraint NoOverlap to enforce non-overlap between pieces.

For each problem variant, three constraint programming models are presented. The models proposed are the first in the literature to solve some instances to optimality.

The constraint programming models use two kinds of variables: those with binary and those with integer domains. The models with binary variables are based on the dotted board model with rotations by Cherri et al. [10], in which a binary variable is defined for each dot, piece type and piece rotation. In the models with integer domain variables, there is only one variable defined for each dot and the values in the variable domain represent the candidate piece types and their rotations. The new global constraint NoOverlap is proposed to take advantage of all the information available for the domains of these integer variables. This global constraint ensures that the pieces do not overlap. The constraint is tailored for the problem, and therefore promotes faster propagation. The computational results show the effectiveness of this new global constraint in solving all the problem variants.

Constraint programming is very flexible for modeling combinatorial optimization problems and allows the use of linear, non-linear or logical constraints to represent the solution space of the problems. Therefore, the proposed constraint programming models lend themselves very easily to the adaptation to real-world problems where custom requirements are frequent and additional constraints need to be included.

The constraints in the proposed models are general enough to be used in other problem variants. As there are some differences in these problem variants, an interesting future research line is to develop and tailor specific global constraints for each problem variant. In addition, combining the power of constraint programming to find good quality solutions with other optimization techniques, such as mixed integer programming, is a promising research line.

Acknowledgments

This research was supported by FAPESP (2018/07240 − 0, 2015/24987 − 4, 2014/10740 − 4, 2013/07375 − 0, 2012/18653 − 8), CNPq (308761/2018 − 9) and CAPES (*A*026 − 2013) from Brazil and by the ERDF European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia within project POCI-01-0145-FEDER-029609 (Delta C&P).

Appendix A. Adapting the instances to all the problem variants

This appendix is dedicated to explain the modifications performed in the instances in order to adapt them to the variants of the cutting and packing problem presented in [33].

For the irregular *ODP* with one open dimension, an upper bound on the length of the board $\overline{L}$ must be defined. The value of $\overline{L}$ must be carefully set, because on one hand, it needs to be large enough to contain feasible solutions but, on the other hand, the number of problem variables increases with this parameter. $\overline{L}$ is defined here as the length of the first solution found by the model with the global constraint run with a board size big enough to find a feasible solution in less than one minute. This model was chosen for this processing step because, as shown in Section 6.3, it uses less memory than the other two approaches. Therefore, for instances Blaz1, Blaz2, Shapes0, Shapes1, Fu, Dagli, Shirts, Jakobs1 and Jakobs2, the board length $\overline{L}$ was defined as 32, 24, 80, 77, 38, 85, 89, 16 and 34, respectively.

For the problem variants for which the board has fixed dimensions (*KP, PP, IIPP, BPP* and *CSP*), the board length is defined as equal to the height, following the approach of [28], and the board height is known since the instances are originally from the strip packing problem.

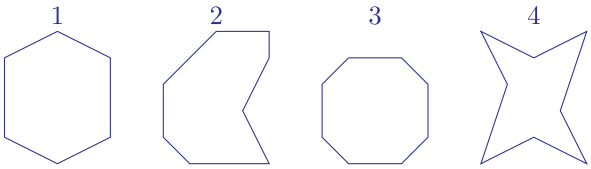


Fig. 10. Assigning numbers to pieces in instance Blaz2.

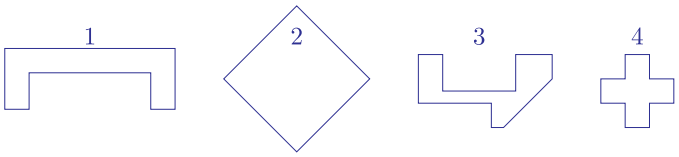


Fig. 11. Assigning numbers to pieces in the Shapes1 instance.

For all the input minimization problems, where demand is strict, the demand of the original instance is maintained. In the problem variants were the pieces must have a value, the value of each piece is set as its area.

Each *IIPP* instance contains only one piece type of Blaz2 or Shapes1 instances as depicted in Figs. 10 and 11. The new instances kept the piece rotations from the pieces of the original instance.

The new instances are identified by the name of the instance where the piece was taken off and the suffix $-<i>$, where $<i>$ is the number associated to the piece in the instance.

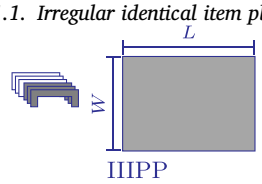
Appendix B. Results for all the problem variants and the three proposed models

In this section we present the results obtained for all the problem variants with the constraint programming models proposed in the paper.

B1. Output maximization problems

Regarding output maximization problems, the board is well defined, and the problem is to decide which pieces will be cut and their positioning points on the board. The results comparing the three models proposed for the variants of this problem class are presented in the following sections.

*B1.1. Irregular identical item placement problem (*IIIPP*)*



The irregular identical item placement problem (IIIPP) consists of cutting multiple copies of a single irregular piece type from a board with fixed dimensions. Table 7 presents the results for the IIIPP obtained by the three models proposed. The instance name is in the first column, columns two and three present the value of the best solution found and the time to obtain this solution for the model with binary domains. Columns four and five (six and seven) have the corresponding content of columns two and three for the model with integer domains variables (for the model with the proposed NoOverlap constraint).

As this problem variant deals with only one piece type, the binary and integer representations are very similar. Considering the solution values, all the approaches reach the same values except for instances Shapes1-3, where the best solution was found by the model with binary decision variables and, Shapes1-4, where the best solution was found by the integer program with the global constraint proposed NoOverlap.

The integer program with the global constraint was able to find solutions that are better or have the same value, faster than the other formulations proposed except for instances Shapes1-3 and Blaz2-3. This is due to the fact that, even for one piece type, the global constraint can significantly reduce the number of constraints necessary to avoid piece overlap.

Table 7
Results for the irregular identical item placement problem (IIIPP).

	IIIPP − Bin		IIIPP − Int		IIIPP − IGC	
Instance	Solution	Time to find	Solution	Time to find	Solution	Time to find
Blaz2-1	144.0*	1040.2	144.0*	779.8	**144.0***	**0.0**
Blaz2-2	155.0*	1.4	155.0*	1.3	**155.0***	**0.4**
Blaz2-3	**168.0***	**0.2**	168.0*	0.3	168.0*	0.3
Blaz2-4	121.0*	878.5	121.0*	580.3	**121.0***	**215.5**
Shapes1-1	880.0*	119.4	880.0*	473.3	**880.0***	**1.8**
Shapes1-2	1080.0*	174.5	1080.0*	30.9	**1080.0***	**0.8**
Shapes1-3	**952.0***	2982.4	924.0*	1128.9	924.0*	3067.4
Shapes1-4	1160.0*	115.0	1180.0*	301.1	**1220.0***	4.4

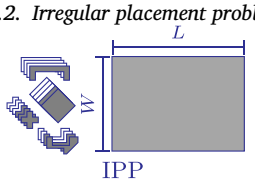
*: optimal solution.

Table 8
Results for the irregular placement problem (IPP) without demand constraints.

Instance	IPP − Bin		IPP − Int		IPP − IGC	
	Solution	Time to find	Solution	Time to find	Solution	Time to find
Three	36.0*	0.3	36.0*	0.3	**36.0***	**0.0**
Threep2	36.0*	0.3	36.0*	0.3	**36.0***	**0.0**
Threep2w9	81.0*	36.7	81.0*	21.0	**81.0***	**14.8**
Threep3	36.0*	0.3	36.0*	0.3	**36.0***	**0.0**
Threep3w9	81.0*	36.7	81.0*	21.0	**81.0***	**14.8**
Blaz1	**214.0***	1449.6	209.0*	773.2	210.0*	1596.5
Blaz2	178.0*	333.6	178.0*	293.3	**178.0***	**102.9**
Shapes0	1104.0*	1934.0	1108.0*	2946.1	**1128.0***	3156.4
Shapes1	1096.0*	400.9	1104.0*	1888.6	**1104.0***	**467.9**
Fu	om*	om	om*	om	**1402.0***	3053.0
Dagli	om*	om	om*	om	**2493.5***	1459.0

*: optimal solution. om: out of memory.

*B1.2. Irregular placement problem (*IPP*)*

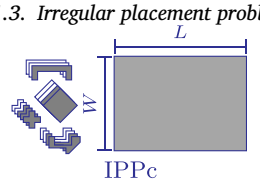


The irregular placement problem (IPP) aims to place a number of pieces on a board in order to maximize the resulting value. Table 8 presents the results for the IPP. Refer to the description of Table 7 for a detailed description of the contents of the columns.

All the models were able to prove optimality for the five instances Three, although the IPP − IGC proved it faster than the other models. Note that for the IPP, instances Three, Threep2 and Threep3 are equal since there are no demand constraints, and the board has the same dimensions. For the same reason, the results of Threep2w9 and Threep3w9 are the same. For instances Blaz1, Balz2, Shapes0, Shapes1, Fu and Dagli none of the three models was able to prove optimality in the given time limit. IPP − IGC obtained solutions with the same or better quality than the two other models except for the instance Blaz1.

Regarding computational times, for the instances where all the models reached solutions with the same value, IPP − IGC was the fastest model. For the instance Shapes1, IPP − Int and IPP − IGC found solutions with the same value but IPP − IGC was more than three times faster than IPP − Int.

*B1.3. Irregular placement problem with demand constraints (*IPPc*)*



The irregular placement problem with demand constraints (IPPc) has a smaller solution space compared with the irregular *PP* without demand constraints. This happens because, with a specified number of pieces to allocate, situations where the domains of the variables can be reduced are more likely to occur. Table 9 presents the results of the computational experiments for IPPc and for each proposed model. Refer to the description of Table 7 for a detailed description of the contents of the columns.

Table 9
Results for the irregular placement problem with demand constraints (IPPc).

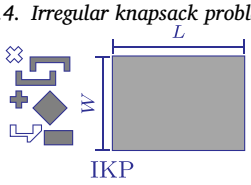
Instance	IPPc − Bin		IPPc − Int		IPPc − IGC	
	Solution	Time to find	Solution	Time to find	Solution	Time to find
Three	23.0*	0.1	23.0*	0.1	**23.0***	**0.0**
Threep2	**34.0***	**0.2**	**34.0***	**0.2**	**34.0***	**0.2**
Threep2w9	46.0*	0.3	46.0*	0.3	**46.0***	**0.0**
Threep3	**35.0***	**0.3**	**35.0***	**0.3**	**35.0***	**0.3**
Threep3w9	57.0*	87.3	57.0*	35.3	**57.0***	**13.9**
Blaz1	191.5*	601.4	192.0*	2820.8	**192.5***	963.0
Blaz2	168.0*	28.5	168.0*	47.6	**168.0***	**6.5**
Shapes0	1024.0*	2501.7	1064.0*	1458.5	**1072.0***	1782.7
Shapes1	1044.0*	2265.7	1048.0*	2478.0	**1052.0***	601.2
Fu	om*	om	om*	om	**1083.0***	1.4
Dagli	om*	om	om*	om	**2688.1***	2102.4

*: optimal solution. om: out of memory.

The IPPc − IGC was the only approach able to find a feasible solution for instances Fu and Dagli. It was also possible to prove optimality for instance Fu in eight seconds. This happens because, as the demand for the items is constrained, all the pieces can be placed inside the board, thus, there is no better solution to cut all the pieces from the board. For instance Blaz2, the IPPc − Bin, IPPc − Int and IPPc − IGC found solutions with the same value, although the IPPc − IGC reached this solution more than four times faster than the IPPc − Bin and more than seven times faster than the IPPc − Int. For instances Three, all models proved the solution optimality, however the IPPc − IGC proved this optimality in smaller or equal computational time.

Considering the differences between the problems, it may seem that, for the same instance, the solutions for the IPPc will be worse than the solutions for the IPP. However, for instance Dagli, the solution obtained by IPPc − IGC has a better value than the one obtained by the IPP − IGC. This is due to the fact that, with the demand constraints, the IPPc solution space can be drastically reduced allowing the solution method to make a more thorough search.

*B1.4. Irregular knapsack problem (*IKP*)*



The irregular knapsack problem (IKP) can be viewed as a special case of in which the demand of each piece is limited to one unit. An instance of IPPc can therefore be easily converted into an instance of IKP considering that each demanded item is a piece of a different type. Clearly it is better to consider the pieces of the same type together since it reduces the number of constraints used to represent the problem in all models. In order to evaluate the performance of the solution method over this problem variant, the same instances solved by the IPPc were solved by the IKP, considering each piece copy as a different piece type. Table 10 presents the results of the computational experiments for IKP and for each proposed model. Refer to the description of Table 7 for a detailed description of the contents of the columns.

For this problem, the binary and integer programs can only solve instances Three and Blaz1 and Blaz2. For the other instances, the memory used exceeds the limit imposed (64 gigabytes).

Except for Threep3w9, all the models prove the optimal solution for instances Three. Note that, despite IKP − Bin and IKP − Int having the same solution found by IKP − IGC, only IKP − IGC could prove the solution optimal. The solution values obtained for instance Blaz1 with IKP − Bin and with IKP − Int are worse than those from IKP − IGC. For instance Blaz2, the three models reached the same solution value, however the IKP − IGC was able to find this solution faster than the other models.

The IKP − IGC model found feasible solutions for all the instances. This happened because, by using the proposed global constraint to avoid overlap, the number of constraints needed to represent the feasible solution space is reduced, consuming less memory. Furthermore, as this constraint propagates faster, good quality solutions could be found, even considering that each demanded piece is of a different type.

As presented in Section 5.2, the models for the IPPc and IKP are the same. It can be observed that no solution found by IKP is better than the solution found by IPPc. This behavior was expected since considering the same-type pieces in the same constraints reduces the number of constraints in the model and avoids symmetric solutions.

B2. Input minimization problems

Unlike output minimization problems, in input minimization problems either the board size or the number of the boards to be used are unknown. In this case, bounds for the board size or for the number of boards should be estimated in order to define the mesh of dots used to represent the board.

Table 10
Results for the irregular knapsack problem (IKP).

	IKP − Bin		IKP − Int		IKP − IGC	
Instance	Solution	Time to find	Solution	Time to find	Solution	Time to find
Three	23.0*	0.1	23.0*	0.1	**23.0***	**0.0**
Threep2	34.0*	2.7	34.0*	1.7	**34.0***	**0.6**
Threep2w9	46.0*	1.0	46.0*	1.0	**46.0***	**0.0**
Threep3	35.0*	22.6	35.0*	41.8	**35.0***	**5.0**
Threep3w9	57.0*	2.8	57.0*	2.8	**57.0***	**1923.1**
Blaz1	180.5*	996.9	185.5*	2889.4	**190.5***	3166.0
Blaz2	168.0*	171.9	168.0*	202.2	**168.0***	**26.1**
Shapes0	om*	om	om*	om	**948.0***	1291.5
Shapes1	om*	om	om*	om	**1036.0***	1388.0
Fu	om*	om	om*	om	**1083.0***	1.4
Dagli	om*	om	om*	om	**2572.9***	1278.6

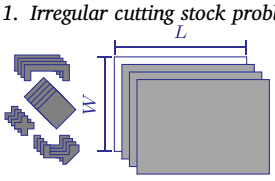
*: optimal solution. om: out of memory.

Table 11
Results for the irregular cutting stock problem (ICSP).

Instance	ICSP − Bin		ICSP − Int		ICSP − IGC	
	Solution	Time to find	Solution	Time to find	Solution	Time to find
Three	6.0*	0.2	6.0*	0.1	**6.0***	**0.0**
Threep2	11.0*	0.3	11.0*	0.3	**11.0***	**0.2**
Threep2w9	8.0*	3.5	8.0*	1.0	**8.0***	**0.3**
Threep3	17.0*	10.0	17.0*	9.4	**17.0***	**8.8**
Threep3w9	12.0*	7.9	12.0*	5.4	**12.0***	**5.2**
Blaz1	30.0*	2746.3	34.0*	72.7	**30.0***	**1385.1**
Blaz2	23.0*	18.4	23.0*	8.3	**23.0***	**5.2**
Shapes0	om*	om	om*	om	**69.0***	740.3
Shapes1	om*	om	om*	om	**70.0***	1250.7
Fu	om*	om	om*	om	**34.0***	63.8
Dagli	om*	om	om*	om	**76.0***	3081.0

*: optimal solution. om: out of memory.

*B2.1. Irregular cutting stock problem (*ICSP*)*

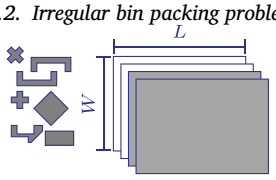


ICSP

The irregular cutting stock problem (ICSP) aims to cut all the pieces from boards using the minimum number of boards. In this paper, the irregular cutting stock problem is formulated similarly to I1ODP, minimizing the number of boards used to perform the cuts and also minimizing the used length of the "last" board. The computational results for ICSP are presented in Table 11. Refer to the description of Table 7 for a detailed description of the contents of the columns.

For all the models proposed of ICSP, feasible solutions were only found for instances Three, Blaz1 and Blaz2. Considering instances Three, the optimality was proven in less than ten seconds by all the models and ICSP − IGC was the fastest approach. For instance Blaz1, ICSP − Bin and ICSP − IGC found a better quality solution than ICSP − Int. In this case ICSP − IGC was twice as fast as ICSP − Bin to find this solution. All the methods reach the same solution value for Blaz2, but ICSP − IGC was slightly faster than the other methods. For the other instances, ICSP − IGC was able to find feasible solutions within the time limit.

*B2.2. Irregular bin packing problem (*IBPP*)*



IBPP

As in the ICSP, the irregular bin packing problem (IBPP) aims to cut all the demanded pieces by using the minimum number of boards, but the demand of each piece is limited to one unit. The model of this problem is equal to the ICSP model, but the instances were adapted to represent the problem as described in Section B.1.4, i.e. considering that each demanded item is a piece of a different type. The results for the IBPP are presented in Table 12. Refer to the description of Table 7 for a detailed description of the contents of the columns.

Table 12
Results for IBPP.

Instance	IBPP − Bin		IBPP − Int		IBPP − IGC	
	Solution	Time to find	Solution	Time to find	Solution	Time to find
Three	6.0*	0.2	**6.0***	**0.1**	**6.0***	**0.1**
Threep2	11.0*	5.2	**11.0***	**1.0**	**11.0***	**1.0**
Threep2w9	8.0*	22.0	8.0*	8.4	**8.0***	**3.5**
Threep3	17.0*	2168.4	17.0*	3.5	**17.0***	853.2
Threep3w9	12.0*	1967.7	12.0*	1711.9	**12.0***	**189.6**
Blaz1	om*	om	om*	om	**34.0***	14.2
Blaz2	23.0*	312.1	23.0*	146.8	**23.0***	5.3
Shapes0	om*	om	om*	om	**73.0***	3424.9
Shapes1	om*	om	om*	om	**75.0***	1649.2
Fu	om*	om	om*	om	**34.0***	63.8
Dagli	om*	om	om*	om	**78.0***	2453.5

*: optimal solution. om: out of memory.

Table 13
Results for the irregular one open dimension problem (I1ODP).

	I1ODP − Bin		I1ODP − Int		I1ODP − IGC	
Instance	Solution	Time to find	Solution	Time to find	Solution	Time to find
Three	6.0*	0.1	6.0*	0.1	**6.0***	**0.0**
Threep2	10.0*	0.8	10.0*	0.8	**10.0***	**0.4**
Threep2w9	8.0*	1.2	8.0*	1.2	**8.0***	**0.5**
Threep3	14.0*	42.6	14.0*	44.9	**14.0***	**37.8**
Threep3w9	12.0*	253.1	12.0*	201.37	**12.0***	**105.1**
Blaz1	30.0*	2571.9	30.0*	893.3	**28.0***	2791.2
Blaz2	23.0*	73.9	22.0*	50.1	**22.0***	**5.3**
Shapes0	om*	om	om*	om	**65.0***	673.81
Shapes1	om*	om	om*	om	**71.0***	1313.3
Fu	om*	om	om*	om	**34.0***	221.1
Dagli	om*	om	om*	om	**76.0***	1577.6

*: optimal solution. om: out of memory.

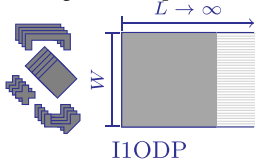
Table 14
Results for the irregular two open dimension problem (I2ODP).

	I2ODP − Bin		I2ODP − Int		I2ODP − IGC	
Instance	Solution	Time to find	Solution	Time to find	Solution	Time to find
Three	40.0*	0.2	40.0*	0.2	**40.0***	**0.1**
Threep2	70.0*	12.3	70.0*	14.6	**70.0***	**5.6**
Threep2w9	70.0*	15.6	70.0*	15.2	**70.0***	**6.9**
Threep3	96.0*	649.7	96.0*	726.0	**96.0***	**442.0**
Threep3w9	96.0*	678.4	96.0*	1250.5	**96.0***	**430.0**
Blaz1	520.0*	2800.8	494.0*	3184.9	**423.0***	866.2
Blaz2	284.0*	2482.6	**280.0***	4.9	288.0*	37.9
Shapes0	om*	om	om*	om	**3024.0***	1080.9
Shapes1	om*	om	om*	om	**2704.0***	1558.4
Fu	om*	om	om*	om	**1225.0***	2349.1
Dagli	om*	om	om*	om	**4290.0***	3557.6

*: optimal solution. om: out of memory.

IBPP − Bin and IBPP − Int found a solution only for instance Blaz2 and for instances Three. On the other hand, IBPP − IGC found solution for all the instances. All the models reached the same solution value for instance Blaz2 but IBPP − IGC was more than 20 times faster than IBPP − Int to find the solution and IBPP − Int found the solution 2 times faster than IBPP − Bin. Comparing the solutions of IBPP − IGC with the solutions obtained for equivalent model ICSP − IGC for the same set of instances, with ICSP − IGC strictly better solutions were found, except for instance Fu, because this instance has already a demand of one for all the pieces. For instance Threep3, only IBPP − Int could not prove the solution optimal. For the other instances Three, all the methods proved the solution optimal and IBPP − IGC was always faster than IBPP − Bin and IBPP − Int.

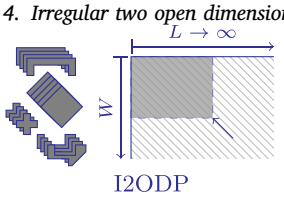
*B2.3. Irregular one open dimension problem (*I1ODP*)*



The irregular one open dimension problem (I1ODP) is the variant of irregular cutting and packing problems most found in the literature. Table 13 presents the results obtained by the I1ODP models proposed for the set of selected instances. Refer to the description of Table 7 for a detailed description of the contents of the columns.

The initial length of the board ($\overline{L}$) had to be estimated for all the instances, following the procedure presented in Appendix A. The estimated values of $\overline{L}$ are larger than the board height, meaning that, for these instances, more dots are needed to represent the board compared to output maximization problems, and therefore, the models demand more resources. Consequently, instances Shapes0, Shapes1, Fu and Dagli could only be solved by the model with the NoOverlap constraint. For the instance Blaz1, the I1ODP − IGC model was able to find a better solution compared with the other two approaches proposed. Moreover, I1ODP − IGC and I1ODP − Int found a better solution for Blaz2 compared with the one obtained with the I1ODP − Bin, but I1ODP − IGC was about ten times faster than I1ODP − Int. For instances Three, all the models were able to prove the solution optimality and the I1ODP − IGC was faster than the other models.

*B2.4. Irregular two open dimension problem (*I2ODP*)*



Considering the irregular two open dimension problem (I2ODP) the two dimensions of the board have to be estimated. Therefore, even if some dots are not considered, as proposed in Section 5.5, the number of dots in the I2ODP models is higher than the number of dots in the other input minimization problems presented. Table 14 presents the computational results for the I2ODP models. Refer to the description of Table 7 for a detailed description of the contents of the columns.

The only model that could find solutions for all the instances was the I2ODP − IGC. Optimal solutions were proved by all the models for instances Three, however the I2ODP − IGC was faster. The two other models were only able to solve instances Three, Blaz1 and Blaz2. For instance Blaz1, the best solution was found by the I2ODP − IGC model and the worst solution was found by the I2ODP − Bin model. But for the instance Blaz2 the solution obtained by I2ODP − IGC was the worst solution compared to the other models. This can occur as the problem is small enough to be well explored by all models and some solutions can be found faster by some models. The I2ODP − IGC was the only model able to find solutions for the remaining instances.

Conflict of interest

The authors have no conflict of interest.

References

[1] Abeysooriya RP, Bennell JA, Martinez-Sykora A. Jostle heuristics for the 2D-irregular shapes bin packing problems with free rotation. Int J Product Econ 2018;195:12–26.

[2] Alvarez-Valdes R, Martinez A, Tamarit J. A branch & bound algorithm for cutting and packing irregularly shaped pieces. Int J Product Econ 2013;145(2):463–77.

[3] Baldacci R, Boschetti MA, Ganovelli M, Maniezzo V. Algorithms for nesting with defects. Discrete Appl Math 2014;163, Part 1(0):17–33.

[4] Beldiceanu N, Carlsson M, Demassey S, Petit T. Global constraint catalogue: past, present and future. Constraints 2007;12(1):21–62.

[5] Bennell J, Scheithauer G, Stoyan Y, Romanova T, Pankratov A. Optimal clustering of a pair of irregular objects. J Global Optim 2015;61(3):497–524. https://doi.org/10.1007/s10898-014-0192-0.

[6] Bennell JA, Oliveira JF. A tutorial in irregular shape packing problems. J Operat Res Soc 2009;60:S93–105.

[7] Bennell J, Cabo M, MartÃnez-Sykora A. A beam search approach to solve the convex irregular bin packing problem with guillotine cuts. Eur J Operat Res 2018;270(1):89–102.

[8] Bennell JA, Oliveira JF. The geometry of nesting problems: a tutorial. Eur J Operat Res 2008;184(2):397–415.

[9] Carravilla MA, Ribeiro C, Oliveira JF. Solving nesting problems with non-convex polygons by constraint logic programming. Int Trans Operat Res 2003;10:651–63.

[10] Cherri LH, Cherri AC, Carravilla MA, Oliveira JF, Toledo FMB, Vianna ACG. An innovative data structure to handle the geometry of nesting problems. Int J Product Res 2018;56(23):7085–102.

[11] Cherri LH, Cherri AC, Soler EM. Mixed integer quadratically-constrained programming model to solve the irregular strip packing problem with continuous rotations. J Global Optim 2018;72(1):89–107.

[12] Cherri LH, Mundim LR, Andretta M, Toledo FM, Oliveira JF, Carravilla MA. Robust mixed-integer linear programming models for the irregular strip packing problem. Eur J Operat Res 2016;253(3):570–83. https://doi.org/10.1016/j.ejor.2016.03.009.

[13] Clautiaux F, Jouglet A, Carlier J, Moukrim A. A new constraint programming approach for the orthogonal packing problem. Comput Operat Res 2008;35(3):944–59.

[14] Dyckhoff H, Finke U. Cutting and packing in production and distribution: typology and bibliography. Heidelberg: Springler-Verlag; 1992.

[15] Dyckhoff H, Kruse H-J, Abel D, Gal T. Trim loss and related problems. Omega 1985;13(1):59–72. https://doi.org/10.1016/0305-0483(85)90083-0.

[16] Elkeran A. A new approach for sheet nesting problem using guided cuckoo search and pairwise clustering. Eur J Operat Res 2013;231(3):757–69.

[17] Fischetti M, Luzzi I. Mixed-integer programming models for nesting problems. J Heur 2009;15(3):201–26.

[18] Gomes AM, Oliveira JF. Solving irregular strip packing problems by hybridising simulated annealing and linear programming. Eur J Operat Res 2006;171(3):811–29.

[19] Kovács A, Beck JC. A global constraint for total weighted completion time for cumulative resources. Eng Appl Artif Intell 2008;21(5):691–7.

[20] Leão AAS, Toledo FMB, Oliveira JF, Carravilla MA. A semi-continuous MIP model for the irregular strip packing problem. Int J Product Res 2016;54(3):712–21. https://doi.org/10.1080/00207543.2015.1041571.

[21] Leão AAS, Toledo FMB, Oliveira JF, Carravilla MA, Alvarez-Valdés R. Irregular packing problems: a review of mathematical models. Eur J Operat Res 2019. https://doi.org/10.1016/j.ejor.2019.04.045.

[22] Martinez-Sykora A, Alvarez-Valdes R, Bennell J, Tamarit JM. Constructive procedures to solve 2-dimensional bin packing problems with irregular pieces and guillotine cuts. Omega 2015;52:15–32. https://doi.org/10.1016/j.omega.2014.10.007.

[23] Mundim LR, Andretta M, de Queiroz TA. A biased random key genetic algorithm for open dimension nesting problems using no-fit raster. Expert Syst Appl 2017;81:358–71.

[24] Ribeiro C, Carravilla MA. A global constraint for nesting problems. Artif Intell Rev 2008;30(1–4):99–118.

[25] Salas I, Chabert G, Goldsztejn A. The non-overlapping constraint between objects described by non-linear inequalities. Principles and practice of constraint programming. Lecture Notes in Computer Science 8656. Springer International Publishing; 2014. p. 672–87.

[26] Saldanha R, Morgado E. Solving set partitioning problems with global constraint propagation. Progress in artificial intelligence. Lecture Notes in Computer Science 2902. Springer Berlin Heidelberg; 2003. p. 101–15.

[27] Sato AK, Martins TC, Gomes AM, Tsuzuki MSG. Raster penetration map applied to the irregular packing problem. Eur J Operat Res 2019.

[28] Song X, Bennell JA. Column generation and sequential heuristic procedure. J Operat Res Soc 2014;65(7):1037–52.

[29] Toledo FMB, Carravilla MA, Ribeiro C, Oliveira JF, Gomes AM. The dotted-board model: a new MIP model for nesting irregular shapes. Int J Product Econ 2013;145(2):478–87.

[30] Trojet M, H'Mida F, Lopez P. Project scheduling under resource constraints: application of the cumulative global constraint in a decision support framework. Comput Ind Eng 2011;61(2):357–63.

[31] Valle AMD, de Queiroz TA, Miyazawa FK, Xavier EC. Heuristics for two-dimensional knapsack and cutting stock problems with items of irregular shape. Expert Syst Appl 2012;39(16):12589–98.

[32] Wang SM, Chen JC, Wang K-J. Resource portfolio planning of make-to-stock products using a constraint programming-based genetic algorithm. Omega 2007;35(2):237–46. https://doi.org/10.1016/j.omega.2005.06.001.

[33] Wäscher G, Haußner H, Schumann H. An improved typology of cutting and packing problems. Eur J Operat Res 2007;183(3):1109–30.