CIRRELT-2021-35

A Branch-and-Cut and a Parallel Algorithm for Packing Two-Dimensional Rectangles in a Circular Container

Allyson Silva
Leandro C. Coelho
Maryam Darvish
Jacques Renaud

August 2021

Document de travail également publié par la Faculté des sciences de l'administration de l'Université Laval, sous le numéro FSA-2021-010.

A Branch-and-Cut and a Parallel Algorithm for Packing Two-Dimensional Rectangles in a Circular Container

Allyson Silva[1,*], Leandro C. Coelho[1,2], Maryam Darvish[1], Jacques Renaud[1]

1. Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT) and Department of Operations and Decision Systems, Université Laval, Québec, Canada
2. Canada Research Chair in Integrated Logistics, Université Laval, Québec, Canada

Abstract. We study a two-dimensional packing problem where rectangular items are placed into a circular container to maximize either the number or the total area of items packed. We adapt a mixed integer linear programming model from the case with a rectangular container and design a branch-and-cut algorithm to solve this problem by adding linear cuts to forbid items from being placed outside the circle. We show that this linear model allows us to prove optimality for instances larger than those solved so far using the state-of-the-art non-linear model for the same problem. We also propose a simple parallel algorithm that efficiently enumerates all non-dominated subsets of items and verifies whether pertinent subsets fit into the container using an adapted version of our linear model. Computational experiments using large benchmark instances attest that this enumerative algorithm generally provides better solutions than the best heuristics from the literature when maximizing the number of items packed. Instances with up to 30 items are now solved to optimality, against the eight-item instance previously solved.

Keywords: Packing, combinatorial optimization, circular container, branch-and-cut, parallel computing

Acknowledgements. The authors thank Compute Canada for providing high-performance parallel computing facilities. This work was partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under grants 2019-00094, 2020-00401, and 2018-03712.This support is gratefully acknowledged.

---

* Corresponding author: allyson.fernandes-da-costa-silva.1@ulaval.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2021

1. Introduction

In packing problems, a set of items has to be packed into containers considering their physical characteristics [26, 38]. In this paper, we consider a packing problem in which items are represented by rectangles of different sizes and values to be packed into a circular container. We call this problem the *two-dimensional circular packing problem* (2D-CPP). Although we refer to the 2D-CPP as a packing problem, it can be applied in the cutting context without loss of generality. The objective is to choose a subset of rectangles that fits into a circular container to maximize the value of packed items, which can be represented by the number of items or their total area. As usually considered in the packing and cutting literature, items are required to be packed orthogonally to each other [26].

Two-dimensional packing problems with rectangular items have been widely studied since at least 1963 [22]. However, the packing problem with a circular container of fixed radius was formulated only recently in Hinostroza et al. [25] as a case study on lumber sawing in the forestry sector, being later studied again in López and Beasley [32]. Both studies present non-linear formulations to the problem that can only solve instances of very small size. Therefore, both studies proposed the use of heuristics to solve larger instances, being a simple constructive heuristic and a simulated annealing (SA) in Hinostroza et al. [25] and a search space algorithm in López and Beasley [32]. A new efficient heuristic procedure that is integrated with a variable neighborhood search (VNS) and another SA is presented in Bouzid and Salhi [5] to quickly find good solutions for instances with up to 200 items.

A relevant subproblem of packing problems is to determine whether a feasible packing exists given a set of items. For the two-dimensional case, this is known as the *two-dimensional orthogonal packing problem* (2D-OPP) [11, 26]. The 2D-OPP is a decision problem since its objective is to answer the above statement without optimizing any metric. Although simple to state, the 2D-OPP is a hard problem to solve [14, 34], belonging to the class of NP-complete problems [26]. It appears as a subproblem of many 2D packing problems [13, 36], which are NP-hard since they contain the one-dimensional bin packing problem [21], and it is not any different for the 2D-CPP [32].

In this paper, we present two new methods to solve the 2D-CPP using mixed integer linear programming (MILP) models. In the first one, the problem is formulated as a *two-dimensional rectangular packing problem* (2D-RPP) and solved using branch-and-cut (B&C). Precisely, the container is initially designed as a square where a circular container is inscribed. Then, whenever items are packed outside the circular container, linear cuts are added dynamically to reduce the size of the square container and to enforce that only items placed inside the circular container are accepted. The cuts are found using simple trigonometric functions. In our second method, we suitably enumerate all subsets of items to verify whether each set fits into the circular container. We prove the feasibility of a packing by adapting our B&C to solve the 2D-OPP. Once a set is proven to be feasible, the remaining sets dominated by it are discarded. We speed up the process using parallelism by verifying the feasibility of multiple sets simultaneously in different threads. Computational experiments are performed on two available benchmark datasets from the literature and on a new one generated based on them.

The main contributions of this paper are to model the 2D-CPP and the 2D-OPP linearly, and to solve them exactly. In order to achieve this, we:

- adapt a 2D-RPP model and design a B&C algorithm to solve the 2D-CPP and the 2D-OPP with a circular container, resulting in the first linear model in the literature to solve both problems;
- introduce numerous valid inequalities to speed up the solution process of B&C, including cuts to remove parts of a rectangular container that are outside of the circular container inscribed in it;
- propose an efficient parallel enumeration algorithm (PEA) to solve the problem using an adapted version of our B&C for the 2D-OPP with a circular container;
- attest the performance of both methods (B&C and PEA) by proving optimality for instances more than three times the size of the largest one solved in the literature;
- analyze the performance of these methods considering the characteristics of the problem (objective function, container size, number of items to pack, possibility to rotate the items);
- show that a truncated version of our parallel algorithm can find many new best known solutions (BKS) for large size instances in a comparable run time as state-of-the-art metaheuristics.

This paper is structured as follows. Section 2 presents a brief literature review on exact methods to solve the 2D-RPP. Section 3 defines the 2D-CPP and presents the B&C used to solve it. In Section 4, we describe the parallel enumeration procedure to solve the 2D-CPP. The computational experiments are presented in Section 5. Finally, the conclusions follow in Section 6.

2. Literature review

A two-dimensional packing problem with rectangular items was first described in Gilmore and Gomory [22] as the cutting stock problem, where a required number of items from a given set are to be packed in an unlimited number of identical containers. This problem belongs to the class of bin packing problems, which along with the knapsack, the strip packing, and the orthogonal packing constitute the major classes of cutting and packing problems [26]. In this section, we briefly review the literature on two-dimensional packing problems. We refer the reader to the reviews of Iori et al. [26] and Wäscher et al. [38].

Unlike the bin packing problem described in Gilmore and Gomory [22], knapsack problems consider that a single container is available and a maximum value should be obtained from packing a number of items from a given set. The problem of packing rectangular items into a rectangular container is the most commonly found in the literature [4, 6, 8], and it includes many variants, such as packing with orthogonal rotation [10] and arranging items to consider guillotine cuts [20, 31, 33]. We can also find the packing of circular items into a rectangular container [24, 35], rectangular items into an arbitrary convex region [7], and even irregular shaped items into rectangular containers [3].

The problem of packing rectangles into a circular container is still underinvestigated, despite its importance in practice. Examples of circular containers found are: wooden logs, which can be approximated to rounded shapes when choosing the wooden boards of predetermined sizes to be cut [25]; bearing plates inside satellite modules, where a set of weighted objects are to be packed to guarantee the satellite stability, control, and performance [40]; and silicon wafers, a thin round slice of crystalline silicon used as a substrate for microelectronic devices, such as integrated circuits and solar cells, and where dies with square or rectangular shapes should be cut in a process called silicon wafer dicing [16].

2.1. Exact methods for the 2D-RPP

Several exact methods for the 2D-RPP are found and used as an inspiration to the methods developed to solve the 2D-CPP. When all items have integer length and height dimensions, Beasley [2] and Hadjiconstantinou and Christofides [23] propose integer linear programming (ILP) models that consider positions inside the container as discrete points. The position of an item within the container is tracked by the coordinates of its bottom-left corner. A pseudo-polynomial number of constraints are used to check whether the items overlap. These models are solved using branch-and-bound algorithms with improved upper bounding techniques and reductions derived from Lagrangean relaxations to improve their performance. Instances with up to 15 items are solved. Other methods to enumerate possible positions of the items also exist [12, 27].

A more effective approach to solve the same problem with integer dimensions is proposed by Fekete and Schepers [18]. It is based on solving the 2D-RPP using a two-level approach. At the first level, an enumeration scheme is used to select subsets of items to be packed, then, at the second level, a feasible packing is searched. This procedure was later improved by Caprara and Monaci [6] and Baldacci and Boschetti [1] in which the first level is solved using an ILP for the one-dimensional knapsack problem, while the second level solves a 2D-OPP.

Other recent studies propose exact methods such as dynamic programming [10] and MILP models [20, 33] for 2D-RPP variants, e.g., with guillotine cuts. Adapting models from different classes of packing problems is also possible [8]. Several heuristic approaches for the 2D-RPP can also be found [15, 30, 39]. A generator of instances for the 2D-RPP is proposed by Silva et al. [37]. Three-dimensional variants, known as the container loading problem, are also well studied [17, 29], and studies even for n-dimensional cases exist [19]. An efficient model for the three-dimensional case proposed by Chen et al. [9] is adapted here to solve the 2D-CPP and is shown in Section 3.

2.2. Related studies for the 2D-CPP

When considering circular containers, many variants exist as shown in Bouzid and Salhi [5]. The 2D-CPP similar to the one considered here was first investigated in Hinostroza et al. [25], where a container with a fixed size is given, the items to be packed have continuous sizes, no item rotation is allowed, and the problem is formulated as a mixed integer non-linear programming (MINLP) model. Due to its non-linearity, its practical use is limited. When using a commercial solver, only an instance with up to eight items could have its optimality proven within one hour. Therefore, the authors proposed an ordering heuristic, based on packing items iteratively from the lowest to the highest width, and an SA is used to improve the initial solution generated by the ordering heuristic to deal with larger instances. The study of Hinostroza et al. [25] remains, up to this date and to the best of our knowledge, the only one that presented results for the 2D-CPP using an exact method.

A second related study is that of López and Beasley [32], which considered the same 2D-CPP as in Hinostroza et al. [25] but allowing items to be rotated orthogonally. The 2D-CPP without rotation is also modeled as an MINLP. They showed that the variant with rotation can be solved using the same model by duplicating each item to represent its rotated version, and adding new constraints to impose that either the regular or the rotated item can be packed. The authors reported to have used a solver that is capable of solving their non-linear formulation. However, none of the instances used, with 10 items or more, were solved within the time limit imposed. The authors then proposed an implementation of a formulation search space (FSS) algorithm to solve the problem, which relies heavily on the use of the solver. Instances with up to 30 rectangles were generated and solved. Good results are provided using the FSS though requiring a relatively long time.

The third related study is presented in Bouzid and Salhi [5]. They introduced a heuristic procedure that iteratively packs items following a given order. Items are placed contiguous to others already packed, forming a polygon that fits in the circular container boundaries. This procedure was embedded into SA and VNS frameworks used to modify the order of the items hoping that the packing procedure results in better solutions. They solved the instances created by López and Beasley [32] very fast, easily outperforming their FSS algorithm. These metaheuristics were tested on newly generated larger instances with up to 200 items, which were also quickly solved.

In our paper, we prove optimality for many of the instances from López and Beasley [32] for the first time, some with up to 30 items. We also show that a truncated version of our parallel algorithm finds many new BKS for the larger instances of Bouzid and Salhi [5] for similar run times when maximizing the number of items packed.

3. Mathematical model and branch-and-cut algorithm for the 2D-CPP

Let $\mathcal{I}$ be a set of rectangular items to be packed with dimensions (l_i, h_i), $\forall i \in \mathcal{I}$, where l is the item length (horizontal edge) and h is the item height (vertical edge), both representing continuous values. Each item has an associated value α_i. The objective of the 2D-CPP is to choose a set $\mathcal{J} \subseteq \mathcal{I}$ that fits into a circular container of radius R such that the value of $\mathcal{J}$, i.e., the sum of the value of its items, is maximized.

We present next the mathematical models and the B&C algorithm used to solve the problem. We first present the 2D-RPP version without rotation (Section 3.1), then the version with rotation (Section 3.2), the 2D-OPP still with a rectangular container (Section 3.3), then we show the cuts added to solve these problems for a circular container (Section 3.4), and finally we present valid inequalities used to speed up the solution process (Section 3.5). In Section 3.6, we summarize the configurations of the models described for the many variants handled in this paper.

3.1. Two-dimensional rectangular packing problem without rotation

Consider a circle with its center located at $C = (R, R)$ and with a radius R. The 2D-CPP can be relaxed to a 2D-RPP by considering a square container with sides equal to $2R$. The representation of a solution for this 2D-RPP in the continuous space can be done considering a Cartesian coordinate system, where the bottom-left vertex of the square container is located at the origin and its center is at the point (R, R). Then, for the 2D-CPP, the circular container is concentric to the square.

Let a_i be the assignment variables indicating whether item $i \in \mathcal{I}$ is packed. The placement variables (x_i, y_i) are the coordinates in the Cartesian system of the bottom-left vertex of item i. The interassignment variables z_{ij} represent whether items i and j, $i \neq j$, are both packed. Finally, the non-overlap variables δ_{ijp} represent whether items i and j do not overlap, where $p = \{1, 2, 3, 4\}$ indicates whether item i is to the left, or to the right, or below, or above item j, respectively. In the example of Figure 1, the placement of i_1 to the left of j implies that $\delta_{i_1 j 1} = 1$, packing i_2 to the right of and above j implies that $\delta_{i_2 j 2} = \delta_{i_2 j 4} = 1$, and since i_3 is below j but only partially to the left of it, we only have that $\delta_{i_3 j 3} = 1$.

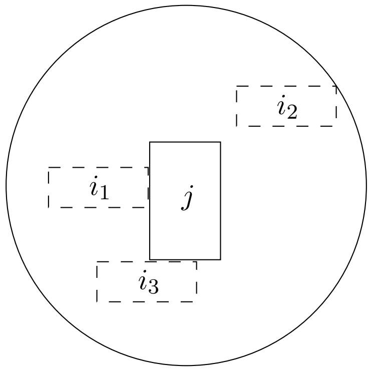


Figure 1: Example of non-overlapping placements of items

Based on the formulation of Chen et al. [9] for the container loading problem, a continuous space formulation for the 2D-RPP without rotation is given as:

$$\max \sum_{i \in \mathcal{I}} \alpha_i a_i, \tag{1}$$

subject to

$$a_i + a_j - 1 \leq z_{ij} \leq a_i, a_j, \quad \forall i < j \in \mathcal{I}, \tag{2}$$

$$z_{ij} \leq \sum_{p=1}^{4} \delta_{ijp} \leq 2z_{ij}, \quad \forall i < j \in \mathcal{I}, \tag{3}$$

$$x_i + l_i \leq x_j + M_{ij1}(1 - \delta_{ij1}), \quad \forall i < j \in \mathcal{I}, \tag{4}$$

$$x_i \geq x_j + l_j - M_{ij2}(1 - \delta_{ij2}), \quad \forall i < j \in \mathcal{I}, \tag{5}$$

$$y_i + h_i \leq y_j + M_{ij3}(1 - \delta_{ij3}), \quad \forall i < j \in \mathcal{I}, \tag{6}$$

$$y_i \geq y_j + h_j - M_{ij4}(1 - \delta_{ij4}), \quad \forall i < j \in \mathcal{I}, \tag{7}$$

$$a_i, z_{ij}, \delta_{ijp} \in \{0, 1\}, \quad \forall i < j \in \mathcal{I}, p = \{1, 2, 3, 4\}, \tag{8}$$

$$0 \leq x_i \leq 2R - l_i, \quad \forall i \in \mathcal{I}, \tag{9}$$

$$0 \leq y_i \leq 2R - h_i, \quad \forall i \in \mathcal{I}. \tag{10}$$

The objective function (1) maximizes the value of the items packed. When $\alpha_i = 1$ for all items, then we maximize the number of items packed. When $\alpha_i = l_i h_i$, we maximize the total area packed, or, equivalently, minimize the container area wasted. Constraints (2) set the value of variables z. When $a_i = a_j = 1$, then $z_{ij} = 1$, and when either $a_i = 0$ or $a_j = 0$, then $z_{ij} = 0$, since z cannot be negative. Constraints (3) impose that when $z_{ij} = 1$, then one or two δ_{ijp} should be activated, i.e., item i must be located to the left or to the right and/or above or below item j. Constraints (4)–(7) impose items i and j not to overlap. Big-M values are used to deactivate these constraints when a respective δ variable is equal to zero. We show in Section 3.5.6 how M_{ij1} to M_{ij4} are set to their lowest possible valid values. Constraints (8) define variables a, z, and δ as binary. Finally, constraints (9)–(10) define the feasible values that variables x and y can assume considering the top and rightmost points where each item can be located.

3.2. Two-dimensional rectangular packing problem with rotation

In the previous formulation for the 2D-RPP, rotation of items is not allowed. So, when an item is packed, its horizontal edge (length) must be parallel to the x-axis and its vertical edge (height) must be parallel to the y-axis. By allowing items being rotated, the problem becomes more complex, although rotation might lead to better solutions [32]. Many papers in the 2D-RPP literature allow rotation, being the 90° turn the most common situation enabled [10, 28, 32]. We explain next how a simple modification in our formulation can deal with 90° rotation of items, while keeping the linearity of the model.

Let r_i be a binary variable indicating whether item i is rotated when packed. The 2D-RPP with rotation is modeled by subjecting the objective function (1) to constraints (2), (3), and (8), and to the following new constraints:

$$x_i + (1 - r_i)l_i + r_i h_i \leq x_j + M_{ij1}(1 - \delta_{ij1}), \quad \forall i < j \in \mathcal{I}, \tag{11}$$

$$x_i \geq x_j + (1 - r_j)l_j + r_j h_j - M_{ij2}(1 - \delta_{ij2}), \quad \forall i < j \in \mathcal{I}, \tag{12}$$

$$y_i + (1 - r_i)h_i + r_i l_i \leq y_j + M_{ij3}(1 - \delta_{ij3}), \quad \forall i < j \in \mathcal{I}, \tag{13}$$

$$y_i \geq y_j + (1 - r_j)h_j + r_j l_j - M_{ij4}(1 - \delta_{ij4}), \quad \forall i < j \in \mathcal{I}, \tag{14}$$

$$0 \leq x_i \leq (1 - r_i)(2R - l_i) + r_i(2R - h_i), \quad \forall i \in \mathcal{I}, \tag{15}$$

$$0 \leq y_i \leq (1 - r_i)(2R - h_i) + r_i(2R - l_i), \quad \forall i \in \mathcal{I}, \tag{16}$$

$$r_i \in \{0, 1\}, \quad \forall i \in \mathcal{I}. \tag{17}$$

Constraints (11)–(14) are the modified versions of the overlapping constraints (4)–(7), and (15)–(16) modify the items placement bounds constraints (9)–(10). They consider that when item i is rotated, i.e., $r_i = 1$, its length l_i becomes its height h_i, and vice versa. Finally, constraints (17) indicate that variables r are binary.

Solving the above formulation is considerably more effective than solving the 2D-RPP model of Section 3.1 by duplicating items to consider rotated versions of them, as suggested by López and Beasley [32], since it only requires the addition of $|\mathcal{I}|$ new variables and constraints instead of doubling the model size.

3.3. Two-dimensional orthogonal packing problem

Given a set $\mathcal{J} \subseteq \mathcal{I}$ of items, we can solve the 2D-OPP to verify whether a feasible packing exists. The 2D-OPP is a subproblem of the 2D-RPP that can be formulated using constraint programming, i.e., the problem is to find whether a feasible solution exists instead of optimizing an objective function. It is optional to implement an objective function when using constraint programming. However, many solvers require that any constant value is passed to solve the model properly.

The 2D-OPP is modeled based on the 2D-RPP models previously described. Since the set of items to be packed is given, variables a and z are no longer required. For the remaining variables, we model the 2D-OPP without rotation as:

$$\max \sum_{j \in \mathcal{J}} \alpha_j, \tag{18}$$

subject to (4)–(10), and to

$$\sum_{p=1}^{4} \delta_{ijp} \geq 1, \quad \forall i < j \in \mathcal{J}. \tag{19}$$

Here, the objective function (18) is a constant, representing the value of the set of items being checked. Constraints (4)–(10) are written for set $\mathcal{J}$ instead of $\mathcal{I}$. The new constraints (19) replace constraints (3) in the original formulation to consider the removal of variables z.

The 2D-OPP with rotation is modeled using constraints (8), (11)–(17), and (19).

3.4. Circular container cuts

Given that the four models described – 2D-RPP and 2D-OPP, with and without rotation – consider a square container with the circular container inscribed in it, some items may be positioned outside the boundaries of the circle.

In this section, we derive valid cuts based on the geometric properties of the circle. The objective is to bring the extremities of the items, which are outside the circle, to its perimeter using linear equations corresponding to tangents to the circle.

For each item i, we identify its four corners as follows: c_1 corresponds to its coordinate (x_i+l_i, y_i+h_i), c_2 to (x_i, y_i+h_i), c_3 to (x_i, y_i), and c_4 to (x_i+l_i, y_i). Recall that the center of the circle is located at $C = (R, R)$ and R is its radius. We also define $d(O, C) = \sqrt{(R-x_o)^2 + (R-y_o)^2}$ as the Euclidean distance between vertex O and C, and $\overline{OC}$ the corresponding line segment.

In order to maximize the impact of each cut, for each item having at least one corner outside the circle, we identify the corner furthest from C. Its distance from the center of the circle should be greater than R (in order to be outside the circle) and greater than the distances of all other corners to the center. Thus, corner c_1 is the furthest one outside the circle if $d(c_1, C) > R$ and $d(c_1, C) > \max\{d(c_2, C), d(c_3, C), d(c_4, C)\}$. This condition is generalized similarly for c_2, c_3 and c_4. Finally, we define $\mathcal{S}_k$, $k \in \{1, 2, 3, 4\}$, as the set of all items for which corner c_k is the furthest with respect to the above conditions.

Figure 2 shows a blue rectangle for which its corner $c_3 = (x_o, y_o)$ is outside the circle. The tangent defined at point A is a valid cut for the placement of corner c_3 of this item as it will move this corner closer to the circle perimeter. To define this cut we need to find the coordinates of point A and the equation of its tangent line.

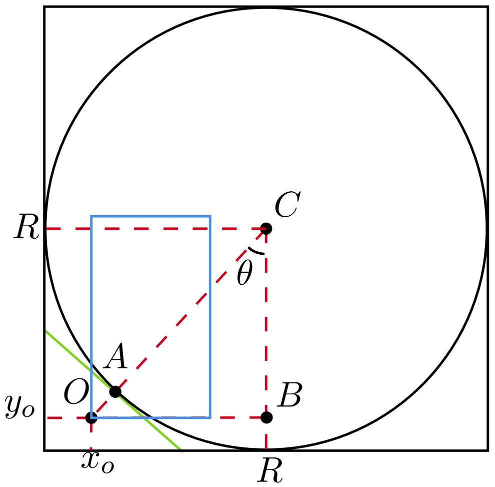


Figure 2: Line tangent to the closest point A between an item's vertex O located outside the container and its boundary

The coordinates of A are determined as follows. From Figure 2, we have that $\overline{OA} = \overline{OC} - \overline{AC}$, thus $d(O, A) = d(O, C) - R$, given that $d(A, C) = R$. For a point $B = (R, y_o)$, we have that $d(O, B) = R - x_o$ and $d(B, C) = R - y_o$. An angle θ is formed between $\overline{OC}$ and $\overline{CB}$. From these, it follows that A is located at the point $A = (x_a, y_a) = (x_o + d(O, A)\sin\theta, y_o + d(O, A)\cos\theta)$, where $\sin\theta = \dfrac{d(O,B)}{d(O,C)} = \dfrac{R - x_o}{\sqrt{(R - x_o)^2 + (R - y_o)^2}}$ and $\cos\theta = \dfrac{d(B,C)}{d(O,C)} = \dfrac{R - y_o}{\sqrt{(R - x_o)^2 + (R - y_o)^2}}$. The line tangent to a point $A = (x_a, y_a)$, where A lies on the circle, is given by $y = y_a + m(x - x_a)$, where $x, y \in \mathbb{R}$, and $m = -\left(\dfrac{x_a - R}{y_a - R}\right)$. Thus, the constraint $y_i \geq y_a + m(x_i - x_a)$ is a cut for an item for which corner $c_3 = (x_i, y_i)$ for all items of $\mathcal{S}_3$. Similarly, for an item in $\mathcal{S}_1$ with corner c_1 located at $(x_i + l_i, y_i + h_i)$ the constraint becomes $y_i + h_i \leq y_a + m(x_i + l_i - x_a)$. For items in $\mathcal{S}_2$ the constraint is $y_i + h_i \leq y_a + m(x_i - x_a)$ and for items in $\mathcal{S}_4$ the constraint is $y_i \geq y_a + m(x_i + l_i - x_a)$.

Formally, the following four types of cuts are added:

$$y_i + h_i \leq y_a + m(x_i + l_i - x_a), \forall i \in \mathcal{S}_1, \tag{20}$$

$$y_i + h_i \leq y_a + m(x_i - x_a), \forall i \in \mathcal{S}_2, \tag{21}$$

$$y_i \geq y_a + m(x_i - x_a), \forall i \in \mathcal{S}_3, \tag{22}$$

$$y_i \geq y_a + m(x_i + l_i - x_a), \forall i \in \mathcal{S}_4. \tag{23}$$

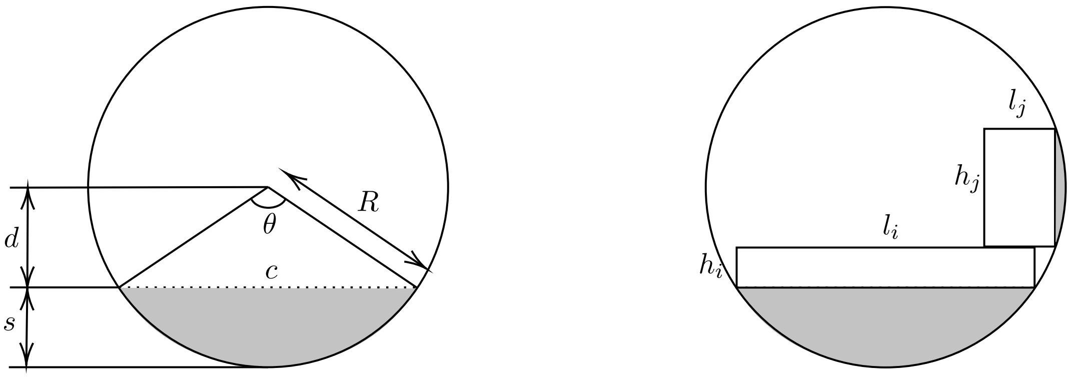
3.5. Valid inequalities for the circular container and big-M values

We now present several valid inequalities for all models previously introduced to accelerate solving the 2D-CPP.

3.5.1. Items placement bounds

The minimum and maximum bounds defined in constraints (9) and (10) for the placement variables can be narrowed when considering a circular container. Consider the circle with radius R of Figure 3a. The chord of a circle is the straight line segment with both endpoints on it. The chord of length c is associated with the sagitta of length s and the apothem of size d. It follows that $R = s + d$ and, from the triangle formed between the center, the point in the circumference that separates d and s, and the midpoint of c, $R^2 = d^2 + \left(\frac{c}{2}\right)^2$. After some algebra, the sagitta is given as:

$$s = R - \sqrt{R^2 - \frac{c^2}{4}}. \tag{24}$$



(a) Trigonometric relations **(b)** Extreme placements

Figure 3: Representation of the sagitta and apothem of a chord of a circle

Given an item i with edges (l_i, h_i), Figure 3b shows that the bottommost place that i can be packed is when l_i defines a chord. We define s_i^l as the sagitta of a chord of length l_i, and s_i^h as the sagitta of a chord of length h_i as:

$$s_i^l = R - \sqrt{R^2 - \frac{l_i^2}{4}}, \tag{25}$$

$$s_i^h = R - \sqrt{R^2 - \frac{h_i^2}{4}}. \tag{26}$$

It follows that the bottommost and uppermost positions for the bottom-left edge of i are bounded by the sagitta s_i^l. Likewise, we can determine the leftmost and rightmost positions for the bottom-left edge of i as bounded by s_i^h. Therefore, constraints (9) and (10) in the 2D-RPP formulation can be redefined for the 2D-CPP as:

$$s_i^h \leq x_i \leq 2R - (l_i + s_i^h), \forall i \in \mathcal{I}, \tag{27}$$

$$s_i^l \leq y_i \leq 2R - (h_i + s_i^l), \forall i \in \mathcal{I}. \tag{28}$$

These inequalities are also valid for the 2D-OPP. For the versions with rotation, we modify constraints (27) and (28) to consider the rotation variables as follows:

$$(1 - r_i)s_i^h + r_i s_i^l \leq x_i \leq 2R - \Big((1 - r_i)(l_i + s_i^h) + r_i(h_i + s_i^l)\Big), \forall i \in \mathcal{I}, \tag{29}$$

$$(1 - r_i)s_i^l + r_i s_i^h \leq y_i \leq 2R - \Big((1 - r_i)(h_i + s_i^l) + r_i(l_i + s_i^h)\Big), \forall i \in \mathcal{I}. \tag{30}$$

3.5.2. Maximum number of items

Given the area of item $i \in \mathcal{I}$ as $A_i = l_i h_i$, where the items in $\mathcal{I}$ are sorted by non-decreasing values of their areas, and the area of the circular container as $A = \pi R^2$, an upper bound for the number of items N that fits into the circular container is:

$$N = \left\{ \max n | \sum_{i=1}^{n} A_i \leq A \right\}.$$

Then, the following inequality is a valid constraint for the 2D-CPP:

$$\sum_{i \in \mathcal{I}} a_i \leq N. \tag{31}$$

3.5.3. Infeasible subsets of items

Given a subset of items $\mathcal{K} \subset \mathcal{I}$, if

$$\sum_{k \in \mathcal{K}} A_k > A,$$

then the following inequality is a valid constraint for the 2D-CPP:

$$\sum_{k \in \mathcal{K}} a_k \leq |\mathcal{K}| - 1. \tag{32}$$

Since a full enumeration of subsets may result in too many constraints, we enumerate up to a maximum number $|\mathcal{K}|$ of items and add the valid inequalities that satisfy the previously mentioned condition when solving the 2D-CPP. From preliminary experiments, we set $|\mathcal{K}| = \left\lceil \sqrt{|\mathcal{I}|} \right\rceil$.

3.5.4. Infeasible pairs of items

Given a pair of items $i, j \in \mathcal{I}$, if $l_i + l_j \geq 2R - (s_i^h + s_j^h)$, then i and j do not fit side-by-side, and we can impose $\delta_{ij1} = 0$ and $\delta_{ij2} = 0$. Also, if $h_i + h_j \geq 2R - (s_i^l + s_j^l)$, then both items do not fit one above the other and we can impose $\delta_{ij3} = 0$ and $\delta_{ij4} = 0$. Furthermore, if both conditions are true, we can add to the 2D-CPP model the following constraint:

$$a_i + a_j \leq 1, \quad \forall i, j \in \mathcal{I} | l_i + l_j \geq 2R - (s_i^h + s_j^h) \wedge h_i + h_j \geq 2R - (s_i^l + s_j^l). \tag{33}$$

For the case with rotation, the only condition to be verified is $\min(l_i, h_i) + \min(l_j, h_j) \geq 2R - (\min(s_i^h, s_i^l) + \min(s_j^h, s_j^l))$. If this statement holds true, then both items cannot be packed together.

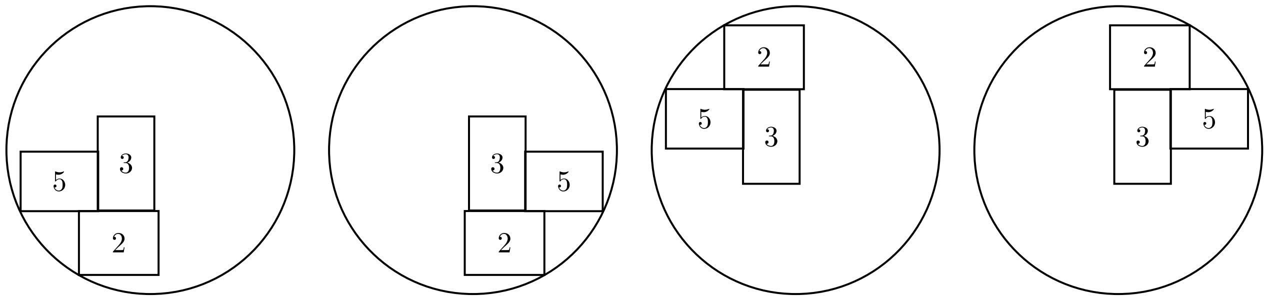
3.5.5. Symmetry breaking

Any feasible packing has three other equivalent solutions (see example in Figure 4) due to the symmetry of the container and items shapes. In order to break symmetries in the 2D-CPP model, constraints are added as follows:

$$x_i + l_i/2, y_i + h_i/2 \leq R \left(1 + \sum_{j=1}^{i-1} a_j\right), \forall i \in \mathcal{I}. \tag{34}$$

These constraints enforce that the center of the lowest indexed item packed is placed in the bottom-left sector of the circle. The left-hand side represents the packing coordinates of the center of an item i. The sum in the right-hand side represents all items packed with an index lower than i. So, if a_i is the lowest indexed item packed, the sum will be zero, and the coordinates of the center of i are bounded above by R. Otherwise, the sum is equal to at least one, and the constraint becomes inactive. In the example in Figure 4, $a_2 = a_3 = a_5 = 1$ and $a_1 = a_4 = 0$. It follows from (34) that for $i = 2$, $x_2 + l_2/2 \leq R$ and $y_2 + h_2/2 \leq R$; for all other items packed, x and y are bounded by at least $2R$, which inactivates the constraint. As the example shows, the symmetric solutions in Figures 4b, 4c, and 4d violate the constraint, so only the packing of Figure 4a can be used.

The symmetry breaking constraints can be easily adapted to the 2D-OPP by adding the constraints to any arbitrary item. For instance, $x_1 + l_1/2 \leq R$ and $y_1 + h_1/2 \leq R$ enforce the center of the first



(a) Original **(b)** Horizontal flip **(c)** Vertical flip **(d)** Both flips

Figure 4: Example of symmetric solutions

item to be packed in the bottom-left sector of the circle.

3.5.6. Setting the big-M

Setting the big-M to the lowest possible value may speed up the optimality proof. Given a pair of items $i, j \in \mathcal{I}$, it follows from constraints (4)–(7) that M_{ij1}, M_{ij2}, M_{ij3}, and M_{ij4} must be large enough to inactivate these constraints when any $\delta = 0$. We prove next that setting M_{ij1} to $2R - (s_i^h + s_j^h)$ is the lowest possible value to achieve this goal for constraints (4).

Corollary 1. *The minimum value for M_{ij1} to turn constraints (4) inactive when $\delta_{ij1} = 0$ is $M_{ij1} = 2R - (s_i^h + s_j^h)$.*

Proof. From constraints (4) it follows that when $\delta_{ij1} = 0$, then $x_i + l_i \leq x_j + M_{ij1}$, $\forall i, j \in \mathcal{I}$. When item i is placed to the rightmost possible position and item j is placed to the leftmost possible position we have:

$$\begin{aligned} M_{ij1} &\geq sup(x_i) + l_i - inf(x_j), \\ M_{ij1} &\geq 2R - l_i - s_i^h + l_i - s_j^h, \\ M_{ij1} &\geq 2R - (s_i^h + s_j^h). \end{aligned} \tag{35}$$

On the other hand, when i is placed to its leftmost possible position and j to its rightmost possible position:

$$\begin{aligned} M_{ij1} &\geq inf(x_i) + l_i - sup(x_j), \\ M_{ij1} &\geq s_i^h + l_i - (2R - l_j - s_j^h), \\ M_{ij1} &\geq -2R + s_i^h + l_i + s_j^h + l_j. \end{aligned} \tag{36}$$

From the inequalities (35) and (36), it follows that:

$$M_{ij1} \geq \max\{2R - (s_i^h + s_j^h), -2R + s_i^h + l_i + s_j^h + l_j\}. \tag{37}$$

Since $l_i + l_j < 2R$ must hold for both items to fit side by side, then it follows from (25) and (26) that $s_i^h + s_j^h < R$ for any combination of l_i and l_j. Therefore, either both items do not fit side by side or the first term is the maximum. □

The same procedure can be done to prove that the same value for M_{ij2} is valid for constraints (5), and that $M_{ij3} = M_{ij4} = 2R - (s_i^l + s_j^l)$ is the minimum value to turn constraints (6) and (7) inactive when $\delta_{ij3} = 0$ and $\delta_{ij4} = 0$, respectively.

3.6. Summary

Due to the large number of models and valid inequalities presented, we summarize here the models used to solve each of the problems presented:

- *2D-CPP* without rotation: we optimize (1) subject to (2)–(8), (27), (28), (31)–(34), and add the circular container cuts dynamically as described in Section 3.4;
- *2D-OPP* without rotation: given a subset of items, we optimize (18) subject to (4)–(8), (19), (27), (28), and add the circular container cuts as above;
- *2D-CPP* with rotation: we optimize (1) subject to (2), (4), (8), (11)–(17), (29)–(34), and add the circular container cuts as above;
- *2D-OPP* with rotation: given a subset of items, we optimize (18) subject to (8), (11)–(17), (19), (29), (30), and add the circular container cuts as above.

4. Parallel enumeration algorithm to solve the 2D-CPP

We can find an optimal solution for the 2D-CPP by solving the previously introduced MILP models. An alternative exact method to solve it is by testing whether each element in the powerset of $\mathcal{I}$, i.e., each element of $\mathbb{P}(\mathcal{I})$, has a feasible packing for the given circular container based on the two-level approach proposed by Fekete and Schepers [18] for the 2D-RPP.

We describe in Algorithm 1 this alternative method, which we call *parallel enumeration algorithm* (PEA). We first define the best solution s^* of value $f(s^*)$ found in the search (line 1) and a list l_{inf} containing all sets proven infeasible (line 2). PEA starts by sorting items in $\mathcal{I}$ (line 3). When the objective function is to maximize the number of items packed, items are sorted by increasing areas. When the objective is to maximize the area packed, the items are sorted in two separate lists by decreasing and increasing areas, and the final list alternates between large and small items until all items are in it. From preliminary experiments, this pattern resulted in better solutions than sorting items by only increasing or decreasing areas. After that, we create a pool of P parallel threads to run tasks (line 4). The elements j of the powerset $\mathbb{P}(\mathcal{I})$ are generated from lower to higher values of cardinality starting with items with the lowest to the highest indices (line 5). Starting by a low cardinality helps speed up the solution process since it is easier for the 2D-OPP model to prove feasibility with fewer items. A parallel architecture is used to solve P different sets at the same time. For each element j of the powerset, the algorithm computes its objective function value (line 6), i.e., $f(s_j) = \sum_{i \in j} \alpha_i$. If s_j is not an improving solution or it contains a subset already proven infeasible, which is in l_{inf}, then we discard j. Otherwise, we create a task *Solve-2D-OPP* to verify whether the items in j can be packed into the circular container and assign it to a free thread (lines 7–9).

The task to evaluate the feasibility of a set j is described in Algorithm 2. It starts by generating the 2D-OPP model to be solved and, then, the B&C process begins. The process can be stopped at any time if s^* is updated by another thread and set j cannot improve it. In this case, j is simply discarded and its thread is freed. If B&C is run until feasibility is proven, s^* is updated (line 4 in Algorithm 2). However, if a packing is proven to be infeasible, then l_{inf} is updated (line 7). PEA finishes returning the best solution found. Given enough time, PEA returns an optimal solution. We can also use a truncated version (see Section 5.4.2) where a global time limit is used to truncate the search.

Algorithm 1 Parallel enumeration algorithm

```
1: Set the best solution: $s^*$;
2: Set a list containing proven infeasible sets: $l_{inf} = \emptyset$;
3: Sort items in $\mathcal{I}$;
4: Create a thread pool containing $P$ parallel threads;
5: for all sets $j \in \mathbb{P}(\mathcal{I})$ do
6:     Calculate the objective function value $f(s_j)$ of subset $j$;
7:     if $f(s_j) > f(s^*)$ and no subset of $j$ is in $l_{inf}$ then
8:         Solve-2D-OPP($s_j$, $s^*$, $l_{inf}$) with a free thread in the pool;
9:     end if
10: end for
11: return best solution $s^*$.
```

Algorithm 2 *Solve-2D-OPP(s_j, s^*, l_{inf})*

```
1: Create 2D-OPP model for set $s_j$;
2: Solve model;
3: if $s_j$ is feasible then
4:     $s^* \leftarrow s_j$;
5: end if
6: if $s_j$ is infeasible then
7:     $l_{inf} = l_{inf} \cup j$;
8: end if
```

5. Computational experiments

This section describes the results obtained from extensive computational experiments performed to test both methods proposed to solve the 2D-CPP and compare their performances with state-of-the-art methods. The experiments were run in computers with an Intel Gold 6148 Skylake CPU with a 2.4 GHz clock, 32 GB of RAM, and 8 cores. PEA was implemented in C++, with the parallelism implemented using the *thread* library from the STL. The exact models were solved using Gurobi 9.10. All instances and results are available in `https://www.leandro-coelho.com/packing-circular-container/`.

5.1. Test instances

Three sets of instances are used to solve the 2D-CPP. The first one (set 1) is a benchmark set from López and Beasley [32] and contains instances with 10, 20 and 30 items with dimensions randomly generated from [1,5]. Each instance size has two versions: one where all items are rectangles and another where they are squares. Each of these six sets of instances is tested for three different container radii, where the area of the circle in relation to the total area of the items, represented by ρ, is equal to $\rho = 33\%$, 50%, and 67%. Each instance is solved for maximizing the number of items and the total area packed, and considering the variants with and without orthogonal rotation of items. This totals 54 tests.

The second set (set 2) is a larger set from Bouzid and Salhi [5], which followed the same instructions to generate instances with 100, 150 and 200 items. This set also totals 54 tests.

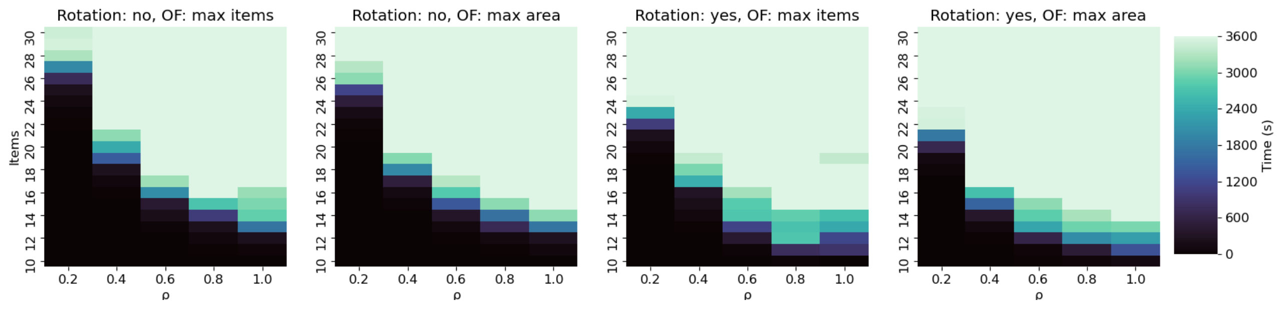
We generated a third set (set 3) with items' dimensions also randomly generated from [1,5]. This new set contains five instances with 30 items each. We set $\rho = 100\%$. Set 3 is used to set up our methods without biasing them towards the benchmark sets.

Unfortunately, the larger instances solved by the heuristics of Hinostroza et al. [25] are not made public despite our best efforts, so a comparison against their method is not possible.

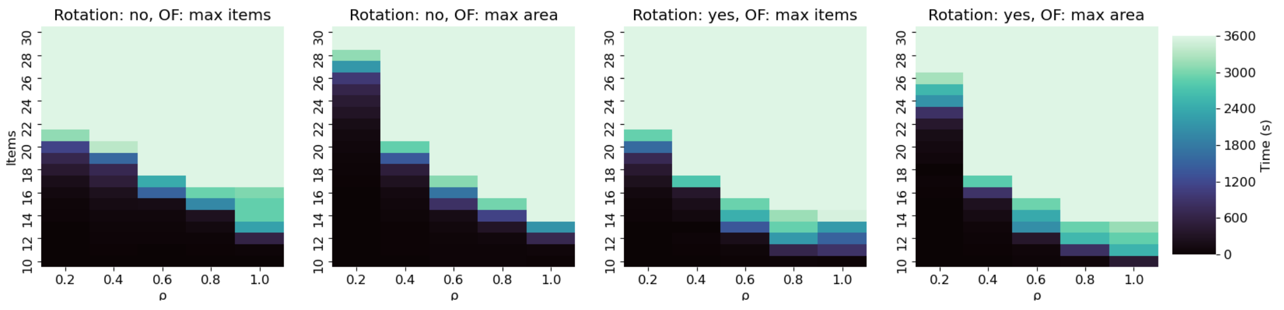
5.2. Maximum instance size each method (B&C and PEA) can prove optimality within one hour

In this round of experiments, we use set 3 to observe the largest instance size that both our methods can solve within one hour. Each instance was solved considering the first 10 generated items up to all 30 items. We also tested different container sizes for $\rho = 20\%$ to 100%. The results are shown in Figure 5. It presents a heat map indicating the average run time when using the original B&C for the

2D-CPP and PEA to solve five instances for each combination of number of items and container sizes. The area of the heat map with the lightest color indicates that no instance was solved within one hour. Meanwhile, the black area shows the combinations tested where all five instances were solved very quickly, mostly in less than one second.



(a) B&C



(b) PEA

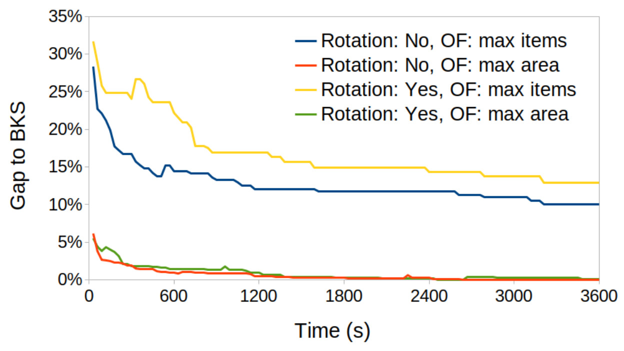
Figure 5: Maximum instance size solved within one hour

Analyzing the results for B&C in Figure 5a, we observe that instances are easier to prove optimality when the container size is small, when rotation is not allowed, and when we maximize the number of items packed. Only one instance with 30 items could be solved within the one-hour limit, and it was the largest instance size solved by any method. For PEA, Figure 5b shows that it is also easier to prove optimality when the container size is small, when rotation is not allowed – although the difference is less than when using B&C – and when maximizing the total area packed. The largest instance solved has 29 items. Hence, both methods perform well for the two objective functions, proving optimality for 29 and 30 items.

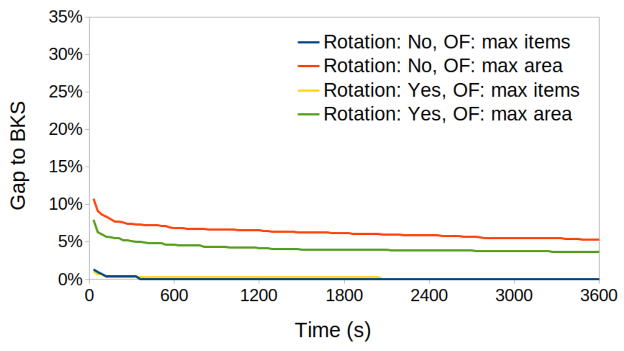
It should be noted that the largest instance solved in Hinostroza et al. [25] when maximizing the total area packed without rotation contains eight items, and the container used has approximately $\rho = 65\%$.

5.3. *Convergence analysis*

Another round of experiments was performed to observe how quickly the best solution improves during the execution of each method. We used again the instances of set 3, now with $\rho = 33\%$, 50%, and 67% to meet the same characteristics of the other two benchmark sets. Only instances with 30 items were tested as they are hard enough to highlight the differences in performance between the proposed methods. Therefore, each problem variant was tested 15 times (five seeds, three container sizes), for a one-hour run each. The results obtained are summarized in Figure 6. Each graph shows, for each problem variant, the average gap to the BKS, i.e., the best feasible solution found after the one-hour run among both methods. Since in some runs it may take some time to find the first feasible solution, mainly for B&C, it is possible to observe an increase in the gap to the BKS as we are showing averages over 15 instances.



(a) B&C



(b) PEA

Figure 6: Convergence of the best solution found

Figure 6 reveals some very interesting findings. It shows that compared to B&C, PEA starts with a much better average solution, and it converges much faster when maximizing the number of items packed. However, B&C converges much faster than PEA when maximizing the total area packed. These results are consistent with the first tests of the previous section. The difference in both cases is so significant that a quick run of a few seconds of one method leads to solutions that the full one-hour run of the other could not yet find. This confirms that each method is better suited to solve the problem for one of the two objective function metrics.

5.4. *Comparison against state-of-the-art heuristics*

In this section, we use B&C and PEA to prove optimality for many of the benchmark instances of set 1 for the first time. Also, the previous findings raise a question of whether a truncated version of PEA can be competitive against heuristic methods due to its quick convergence. To answer this question, we use PEA to solve the instances of set 2 with run times of similar magnitudes as those reported for state-of-the-art metaheuristics. The results of these two rounds of experiments are presented next.

5.4.1. Proving optimality for instances from set 1

We showed in Section 5.2 that our methods can prove optimality for instances about the same size as those in set 1. We now run both exact methods for eight hours to prove the optimality of many of these instances for the first time. The results are presented in Tables 1 and 2, where each row represents an instance from set 1. In the tables, we show the best solutions found by the FSS of López and Beasley [32], and the SA and VNS of Bouzid and Salhi [5] as reported in their papers. We highlight that our goal here is not to compete against the heuristics, since both B&C and PEA are exact methods and, naturally, require a longer time to prove optimality of a solution. But it is still valuable to observe how far the solutions provided by the heuristics are from the optimal ones, therefore assessing their real quality. We provide the best bounds – lower (LB) and upper (UB) – found by B&C and the best solution found by PEA. Solutions reported with a run time of less than 28,800 seconds are those with optimality proven. Overall, SA and VNS find the solutions reported in less than one minute each, while FSS can take several hours, as reported in their papers. Values in boldface indicate a BKS, while underlined values are new BKSs.

Table 1 shows the results when maximizing the number of items packed. All instances from set 1 had either the optimality proven or a new BKS found using PEA. Meanwhile, Table 2 shows the same for B&C when maximizing the total area packed, except for three instances. When maximizing the number of items, PEA proves optimality for 12 instances – B&C proves it for another one – and improves the BKS for the remaining 15 instances. When maximizing the total area packed, B&C proves optimality for 13 instances – PEA proves it for another one – and improves the BKS for a total of 16 instances, five of them optimal.

Although exact and heuristic methods have clear distinct goals, it might be necessary to come up, in practice, with a well-defined scheme to decide when to use one over another. From our experiments, the best option is clearly defined by the number of items n to pack. So, for $n = 10$, and possibly lower, exact methods are as fast as the heuristics and have the advantage of guaranteeing optimality. For $n = 20$, although the best heuristics can quickly provide good solutions, we see that the exact methods could find better solutions given enough time. Finally, for $n = 30$, either the SA and VNS heuristics

Table 1: Best solutions found by state-of-the-art heuristics and our exact methods when maximizing the number of items packed for set 1

Set	*n*	*ρ*	*FSS*	*SA*	*VNS*	*B&C*			*PEA*		*BKS*
						LB	*UB*	*T(s)*	*Sol*	*T(s)*	
Rectangular (no rotation)	10	33	**5**	**5**	**5**	**5**	5	0.2	**5**	0.6	**5***
		50	**6**	**6**	**6**	**6**	6	0.2	**6**	0.7	**6***
		67	**7**	**7**	**7**	**7**	7	0.3	**7**	0.7	**7***
	20	33	7	7	7	**8**	8	153.0	**8**	2253.8	**8***
		50	10	10	10	**11**	14	28800.5	**11**	28800.1	**11**
		67	11	13	12	13	16	28800.2	**14**	28816.4	**14**
	30	33	13	14	14	**15**	21	28801.0	**15**	28818.1	**15**
		50	16	18	17	18	23	28800.3	**19**	28823.1	**19**
		67	19	21	21	21	25	28800.1	**22**	28824.3	**22**
Rectangular (rotation)	10	33	**5**	**5**	**5**	**5**	5	0.1	**5**	0.7	**5***
		50	**6**	**6**	**6**	**6**	6	0.3	**6**	0.8	**6***
		67	**7**	**7**	**7**	**7**	7	1.1	**7**	0.7	**7***
	20	33	**8**	**8**	**8**	**8**	8	19984.1	**8**	4509.7	**8***
		50	10	10	10	**11**	14	28800.4	**11**	28828.7	**11**
		67	12	13	12	13	16	28800.3	**14**	28835.1	**14**
	30	33	14	14	14	14	21	28800.3	**15**	28831.0	**15**
		50	17	18	18	17	23	28800.1	**19**	28837.4	**19**
		67	20	21	20	21	25	28800.1	**22**	28823.2	**22**
Square	10	33	**4**	**4**	**4**	**4**	4	0.1	**4**	1.1	**4***
		50	**5**	**5**	**5**	**5**	5	0.5	**5**	1.1	**5***
		67	**6**	**6**	**6**	**6**	6	4.1	**6**	1.0	**6***
	20	33	**11**	**11**	10	**11**	11	10.5	**11**	484.6	**11***
		50	12	**13**	12	**13**	13	180.7	**13**	28803.7	**13***
		67	14	14	14	**15**	15	8675.2	**15**	28800.5	**15**
	30	33	16	16	15	16	23	28800.6	**17**	28814.9	**17**
		50	20	20	20	19	25	28800.3	**21**	28818.9	**21**
		67	23	23	22	23	26	28800.2	**24**	28817.0	**24**

*: optimality proven; boldface: BKS; underlined: new BKS

or the best exact method for each objective function could be chosen considering the trade-off between time and quality. Moreover, while some results from the literature remain the best known ones when maximizing the area (Table 2), Table 1 shows that for the maximization of the number of items, our exact methods dominate the alternative algorithms on all instances.

Table 2: Best solutions found by state-of-the-art heuristics and our exact methods when maximizing the total area packed for set 1

Set	*n*	*ρ*	*FSS*	*SA*	*VNS*	*B&C*			*PEA*		*BKS*
						LB	*UB*	*T(s)*	*Sol*	*T(s)*	
Rectangular (no rotation)	10	33	**18.4441**	**18.4441**	**18.4441**	**18.4441**	18.4441	0.1	**18.4441**	0.8	**18.4441***
		50	**28.9390**	**28.9390**	**28.9390**	**28.9390**	28.9390	0.7	**28.9390**	1.3	**28.9390***
		67	37.6878	**39.4588**	38.7870	**39.4588**	39.4588	2.3	**39.4588**	1.6	**39.4588***
	20	33	43.3885	45.1727	45.1567	**45.9961**	46.0000	946.1	**45.9961**	220.7	**45.9961***
		50	63.1643	69.9263	68.8314	**72.7850**	107.5065	28801.7	**72.7850**	28800.1	**72.7850**
		67	84.4446	93.0556	91.6368	**96.5377**	147.1685	28803.1	95.9797	28800.1	**96.5377**
	30	33	60.3570	65.2856	64.4689	**66.8049**	152.9675	28804.0	66.3047	28800.3	**66.8049**
		50	85.2113	100.1839	102.1196	**103.5390**	223.6899	28800.8	98.2185	28800.1	**103.5390**
		67	103.4802	135.9217	**137.4149**	136.4249	230.3991	28800.4	126.9673	28800.4	**137.4149**
Rectangular (rotation)	10	33	**19.6702**	**19.6702**	**19.6702**	**19.6702**	19.6702	0.2	**19.6702**	0.8	**19.6702***
		50	29.5041	**30.8746**	**30.8746**	**30.8746**	30.8746	2.4	**30.8746**	1.2	**30.8746***
		67	37.9687	41.1612	40.9063	**41.5246**	41.5246	2.1	**41.5246**	2.1	**41.5246***
	20	33	43.6850	45.4200	45.4200	**47.8299**	81.0636	28804.0	**47.8299**	1382.9	**47.8299***
		50	63.5279	71.2331	70.8221	**72.7698**	125.6325	28802.1	72.0126	28800.1	**72.7698**
		67	84.7008	95.1127	95.2162	**97.7258**	159.8655	28800.8	94.8381	28800.2	**97.7258**
	30	33	57.9328	66.6947	66.6329	**66.8074**	164.7611	28802.5	66.1901	28800.2	**66.8074**
		50	84.3715	100.4537	100.3020	**102.2846**	223.6899	28800.5	95.0703	28800.1	**102.2846**
		67	110.3253	135.2908	**137.5277**	129.8499	230.3991	28800.3	127.6377	28803.5	**137.5277**
Square	10	33	22.9485	**23.9878**	23.9878	**23.9878**	23.9878	0.1	**23.9878**	0.7	**23.9878***
		50	36.7126	**37.7471**	**37.7471**	**37.7471**	37.7471	0.8	**37.7471**	1.1	**37.7471***
		67	51.7583	**52.7555**	**52.7555**	**52.7555**	52.7555	3.9	**52.7555**	1.7	**52.7555***
	20	33	54.1054	63.7430	63.7523	**64.7463**	64.7463	37.5	**64.7463**	7.8	**64.7463***
		50	85.2107	94.7706	94.7706	**95.9219**	95.9219	342.0	**95.9219**	24.6	**95.9219***
		67	109.8363	126.7480	132.4100	**137.2832**	137.2832	5903.2	**137.2832**	198.1	**137.2832***
	30	33	54.4941	63.1167	63.9965	**64.3278**	139.9398	28803.3	63.9141	28800.2	**64.3278**
		50	77.5814	97.2366	**98.1142**	97.8900	173.7835	28802.5	94.4836	28800.1	**98.1142**
		67	103.0963	129.6979	131.5472	**131.6949**	224.6918	28800.6	126.8909	28800.1	**131.6949**

*: optimality proven; boldface: BKS; underlined: new BKS

5.4.2. Using truncated PEA to solve set 2

Previous experiments showed that solutions converge very quickly to the best solutions found after a long run when using PEA for the maximization of the number of items packed, and that PEA finds all BKS in instances with $n \leq 30$. In this last round of experiments, we assess the competitiveness of PEA against the metaheuristics when solving larger instances for the same objective. We compare here the results obtained by PEA for a similar run time as the metaheuristics proposed by Bouzid and Salhi [5] for the instances of set 2. To do so, we run it with a time limit of five, 10, and 30 minutes for instances with 100, 150, and 200 items, respectively. The results are presented in Table 3. We compare our solutions against those obtained by SA, VNS, and accelerated versions of them (xSA and xVNS). As before, the best solutions are shown, now followed by the total time to perform all runs as reported in their papers. Values in boldface are the BKS, and underlined values are new BKS. We have also tested both B&C and PEA on the same instances for the objective function of maximizing the total area packed. As before, this objective leads to a more challenging problem and the results of our exact methods are not competitive for instances of that size. The results appear online.

Table 3: Comparison of state-of-the-art heuristics against PEA when maximizing the number of items packed for set 2

Set	*n*	ρ	*SA*		*xSA*		*VNS*		*xVNS*		*PEA*		*BKS*
			Best	*T(s)*	*Best*	*T(s)*	*Best*	*T(s)*	*Best*	*T(s)*	*Sol*	*T(s)*	
Rectangular (no rotation)	100	33	45	505	45	165	45	330	44	145	**48**	300	**48**
		50	59	705	58	245	58	425	59	215	**60**	300	**60**
		67	70	905	70	320	69	475	70	255	**71**	300	**71**
	150	33	70	1360	69	480	72	1085	72	420	**74**	600	**74**
		50	91	1900	91	780	92	1290	91	590	**93**	600	**93**
		67	**110**	2505	**110**	1040	**110**	1430	**110**	875	109	601	**110**
	200	33	95	2670	96	1060	**99**	2340	**99**	1105	**99**	1806	**99**
		50	123	4090	124	1585	124	3685	124	1220	**125**	1802	**125**
		67	146	5190	146	2055	146	3415	**147**	1670	146	1801	**147**
Rectangular (rotation)	100	33	46	925	45	280	45	550	44	220	**48**	300	**48**
		50	59	1195	59	395	59	820	59	345	**61**	300	**61**
		67	71	1375	71	485	70	745	70	325	**72**	300	**72**
	150	33	70	2290	69	835	73	2315	72	600	**74**	600	**74**
		50	92	3055	90	1145	93	1895	92	875	**94**	600	**94**
		67	**110**	3725	109	1435	**110**	2850	109	1045	**110**	601	**110**
	200	33	95	4635	95	1605	98	4025	98	1335	**100**	1801	**100**
		50	125	6335	122	2345	**126**	5950	125	2965	**126**	1801	**126**
		67	**148**	7595	147	2910	147	7895	147	2255	**148**	1804	**148**
Square	100	33	53	590	52	180	52	630	52	210	**55**	300	**55**
		50	**67**	765	66	260	64	725	64	275	**67**	300	**67**
		67	76	880	76	330	73	715	73	265	**77**	302	**77**
	150	33	84	1445	86	550	84	1540	84	570	**88**	601	**88**
		50	103	2145	103	780	100	1595	101	780	**105**	602	**105**
		67	**117**	2555	116	955	113	1965	113	700	**117**	606	**117**
	200	33	107	2925	107	1090	108	2845	108	1620	**110**	1801	**110**
		50	132	4160	131	1615	130	3435	130	1855	**135**	1802	**135**
		67	152	5410	152	2150	152	4475	152	2875	**154**	1802	**154**

boldface: BKS; underlined: new BKS

The solutions presented in Table 3 show that the truncated PEA finds all but two BKS, improving 20 BKS out of 27 instances. This is an impressive performance considering that PEA is a simple greedy algorithm compared to the very well designed metaheuristics used as comparison.

We attribute this advantage of PEA over the metaheuristics to the facility B&C has to prove feasibility of a packing when items fit too easily into the container. Analyzing the detailed logs, we see that PEA can very quickly find a solution with one or two items below the BKS for most instances. Then, it spends most of the remaining time searching for a feasible packing for the next couple of items. The same behavior was observed for the smaller instances (set 1).

6. Conclusions

In this paper, we solved the two-dimensional circular packing problem, which deals with packing rectangles into a circular container with the objective of maximizing the number of items or the total area packed. A literature review shows that only non-linear models exist to prove the optimality of solutions for this problem. In practice, these models are very limited since the largest instance with optimality proven contained only eight items.

We proposed a linear model to solve the 2D-CPP using a B&C method, where the problem is formulated considering a square container with the circle inscribed, and cuts are added iteratively to remove infeasible packings during the solution process. Alternatively, we transformed this model to consider the decision problem of whether a given set of items can be packed into the circular bin. Then, we used this model to solve the 2D-CPP with a two-phase algorithm. In the first phase, we select the items set using an explicit enumeration of all non-dominated subsets of items. Then, the second phase tests each subset's feasibility using the decision model. This approach is implemented using parallelism such that multiple models can be solved simultaneously.

Computational experiments show that these two exact methods can solve instances with up to 30 items, depending on the characteristics of the problem, such as bin size, objective function, and whether rotation of items is allowed. Smaller bin sizes allow larger problems to be solved. We also show that B&C converges faster to good solutions when maximizing the total area packed, while the parallel enumeration algorithm converges faster when maximizing the number of items. Two benchmark instance sets are used to compare our exact methods to heuristic approaches from the literature to the 2D-CPP. For smaller instances, with 10 to 30 items, our models prove optimality of half of them for the first time and improved the best known solution of almost as many instances. For larger instances, with 100 to 200 items, we show that a truncated execution of our parallel enumerative algorithm outperforms all of the state-of-the-art heuristics when maximizing the number of items packed, also finding new best known solutions for most of the instances.

References

[1] R. Baldacci and M. A. Boschetti. A cutting-plane approach for the two-dimensional orthogonal non-guillotine cutting problem. *European Journal of Operational Research*, 183(3):1136–1149, 2007.

[2] J. E. Beasley. An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research*, 33(1):49–64, 1985.

[3] J. A. Bennell and J. F. Oliveira. A tutorial in irregular shape packing problems. *Journal of the Operational Research Society*, 60(1):S93–S105, 2009.

[4] M. A. Boschetti, A. Mingozzi, and E. Hadjiconstantinou. New upper bounds for the two-dimensional orthogonal non-guillotine cutting stock problem. *IMA Journal of Management Mathematics*, 13(2):95–119, 2002.

[5] M. C. Bouzid and S. Salhi. Packing rectangles into a fixed size circular container: Constructive and metaheuristic search approaches. *European Journal of Operational Research*, 285(3):865–883, 2020.

[6] A. Caprara and M. Monaci. On the two-dimensional knapsack problem. *Operations Research Letters*, 32(1):5–14, 2004.

[7] A. Cassioli and M. Locatelli. A heuristic approach for packing identical rectangles in convex regions. *Computers & Operations Research*, 38(9):1342–1350, 2011.

[8] P. M. Castro and J. F. Oliveira. Scheduling inspired models for two-dimensional packing problems. *European Journal of Operational Research*, 215(1):45–56, 2011.

[9] C. S. Chen, S.-M. Lee, and Q. S. Shen. An analytical model for the container loading problem. *European Journal of Operational Research*, 80(1):68–76, 1995.

[10] G. F. Cintra, F. K. Miyazawa, Y. Wakabayashi, and E. C. Xavier. Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation. *European Journal of Operational Research*, 191(1):61–85, 2008.

[11] F. Clautiaux, J. Carlier, and A. Moukrim. A new exact method for the two-dimensional orthogonal packing problem. *European Journal of Operational Research*, 183(3):1196–1211, 2007.

[12] J.-F. Côté and M. Iori. The meet-in-the-middle principle for cutting and packing problems. *INFORMS Journal on Computing*, 30(4):646–661, 2018.

[13] J.-F. Côté, M. Gendreau, and J.-Y. Potvin. An exact algorithm for the two-dimensional orthogonal packing problem with unloading constraints. *Operations Research*, 62(5):1126–1141, 2014.

[14] M. Delorme, M. Iori, and S. Martello. Logic based Benders' decomposition for orthogonal stock cutting problems. *Computers & Operations Research*, 78:290–298, 2017.

[15] J. Egeblad and D. Pisinger. Heuristic approaches for the two-and three-dimensional knapsack packing problem. *Computers & Operations Research*, 36(4):1026–1049, 2009.

[16] Y. Fan, A. Arevalo, H. Li, and I. G. Foulds. Low-cost silicon wafer dicing using a craft cutter. *Microsystem Technologies*, 21(7):1411–1414, 2015.

[17] T. Fanslau and A. Bortfeldt. A tree search algorithm for solving the container loading problem. *INFORMS Journal on Computing*, 22(2):222–235, 2010.

[18] S. P. Fekete and J. Schepers. A new exact algorithm for general orthogonal d-dimensional knapsack problems. In *European Symposium on Algorithms*, pages 144–156. Springer, 1997.

[19] S. P. Fekete, J. Schepers, and J. C. Van der Veen. An exact algorithm for higher-dimensional orthogonal packing. *Operations Research*, 55(3):569–587, 2007.

[20] F. Furini, E. Malaguti, and D. Thomopulos. Modeling two-dimensional guillotine cutting problems via integer programming. *INFORMS Journal on Computing*, 28(4):736–751, 2016.

[21] M. R. Garey and D. S. Johnson. Computers and intractability: A guide to the theory of NP-completeness. In W. H. Freeman, editor, *A Series of Books in the Mathematical Sciences*, page 340. W. H. Freeman & Co., New York, 1979.

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

[23] E. Hadjiconstantinou and N. Christofides. An exact algorithm for general, orthogonal, two-dimensional knapsack problems. *European Journal of Operational Research*, 83(1):39–56, 1995.

[24] M. Hifi and R. M'hallah. A literature review on circle and sphere packing problems: Models and methodologies. *Advances in Operations Research*, 2009:150624, 2009.

[25] I. Hinostroza, L. Pradenas, and V. Parada. Board cutting from logs: Optimal and heuristic approaches for the problem of packing rectangles in a circle. *International Journal of Production Economics*, 145(2):541–546, 2013.

[26] M. Iori, V. L. de Lima, S. Martello, F. K. Miyazawa, and M. Monaci. Exact solution techniques for two-dimensional cutting and packing. *European Journal of Operational Research*, 289(2): 399–415, 2021.

[27] V. M. Kartak and A. V. Ripatti. The minimum raster set problem and its application to the d-dimensional orthogonal packing problem. *European Journal of Operational Research*, 271(1): 33–39, 2018.

[28] R. E. Korf, M. D. Moffitt, and M. E. Pollack. Optimal rectangle packing. *Annals of Operations Research*, 179(1):261–295, 2010.

[29] D. V. Kurpel, C. T. Scarpin, J. E. Pécora Jr., C. M. Schenekemberg, and L. C. Coelho. The exact solutions of several types of container loading problems. *European Journal of Operational Research*, 284(1):87–107, 2020.

[30] S. C. H. Leung, D. Zhang, C. Zhou, and T. Wu. A hybrid simulated annealing metaheuristic algorithm for the two-dimensional knapsack packing problem. *Computers & Operations Research*, 39(1):64–73, 2012.

[31] A. Lodi and M. Monaci. Integer linear programming models for 2-staged two-dimensional knapsack problems. *Mathematical Programming*, 94(2):257–278, 2003.

[32] C. O. López and J. E. Beasley. Packing unequal rectangles and squares in a fixed size circular container using formulation space search. *Computers & Operations Research*, 94:106–117, 2018.

[33] M. Martin, R. Morabito, and P. Munari. A bottom-up packing approach for modeling the constrained two-dimensional guillotine placement problem. *Computers & Operations Research*, 115: 104851, 2020.

[34] M. Mesyagutov, G. Scheithauer, and G. Belov. LP bounds in various constraint programming approaches for orthogonal packing. *Computers & Operations Research*, 39(10):2425–2438, 2012.

[35] J. P. Pedroso, S. Cunha, and J. N. Tavares. Recursive circle packing problems. *International Transactions in Operational Research*, 23(1-2):355–368, 2016.

[36] D. Pisinger and M. Sigurd. Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. *INFORMS Journal on Computing*, 19(1):36–51, 2007.

[37] E. Silva, J. F. Oliveira, and G. Wäscher. 2DCPackGen: A problem generator for two-dimensional rectangular cutting and packing problems. *European Journal of Operational Research*, 237(3): 846–856, 2014.

[38] G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, 2007.

[39] L. Wei, Q. Hu, A. Lim, and Q. Liu. A best-fit branch-and-bound heuristic for the unconstrained two-dimensional non-guillotine cutting problem. *European Journal of Operational Research*, 270 (2):448–474, 2018.

[40] C.-Q. Zhong, Z.-Z. Xu, and H.-F. Teng. Multi-module satellite component assignment and layout optimization. *Applied Soft Computing*, 75:148–161, 2019.