

Packing Rectangles into a Fixed Size Circular Container: Constructive and Metaheuristic Search Approaches

Mouaouia Cherif Bouzid¹ and Said Salhi²

¹*LTI-Laboratory, Department of Logistics and Transport Engineering, ENST, Dergana, Algiers, Algeria.*

²*Centre for Logistics & Heuristic Optimisation, Kent Business School, The University of Kent, Canterbury, Kent CT2 7PE, UK.*

(3rd March 2020)

Abstract

We investigate the orthogonal packing of rectangular objects into a circular container of fixed radius. We propose a new constructive heuristic called *pack* which builds a feasible packing starting from an ordered list of rectangles. This decoding procedure is polynomial and permits to move from the permutations search space to the packings search space by means of simple combinatorial moves combined with powerful geometrical analytical forms. The *pack* procedure is integrated into two well known metaheuristics, namely, a variable neighbourhood search (VNS) and a simulated annealing (SA). Two variants, namely *xVNS* and *xSA*, which stand as accelerated versions of VNS and SA are also presented. The proposed methodology produces 32 new best solutions out of the 54 benchmark instances while requiring less computational effort than the state-of-the-art method. In addition, we conduct experiments on newly generated larger instances which we have made publicly available alongside their respective results obtained from the proposed metaheuristics.

Keywords: Rectangle packing, fixed size circular container, constructive heuristic, variable neighbourhood search, simulated annealing.

1 Introduction and literature review

A packing problem can be defined as the optimisation of the choice and the location of a subset of objects into a container in such a way that no two objects overlap with each other and every object lies entirely within the container's perimeter while satisfying possible additional constraints. Several variants of the problem exist depending on features such as the dimensionality, the objects and container's shapes, the objective, etc (see Wäscher et al. (2007) for a typology of cutting and packing problems).

The complexity of packing problems have been tackled by several authors. For instance, Li and Cheng (1994) showed that deciding whether a set of squares can be packed into a larger rectangle is strongly NP-complete. Leung et al. (1990) also proved the same result when the target shape is a square.

Email addresses: m.cherif.bouzid@enst.dz (M.C. Bouzid, Corresponding author), s.salhi@kent.ac.uk (S. Salhi)

Two decades later, Demaine et al. (2010) showed that deciding whether a given set of circles can be packed into a rectangle, an equilateral triangle, or a unit square are NP-hard problems. To the best of our knowledge, the complexity of deciding whether a set of rectangles can be packed into a circular container of fixed size has not been tackled yet in the literature.

Table 1 summarises the references that addressed the packing of rectangular objects into containers having either a circular or an arbitrary convex shape. The first column indicates the reference. The next four columns describe the features of the problem considered while the last two columns indicate eventual additional constraints and the method adopted by the authors. Two main applications arise, one in the timber industry (Hinostroza et al., 2013) and the other in the satellite module layout design denoted SMLD (see for instance Zhong et al. (2019)). The latter consists in packing, inside a satellite module, a set of weighted objects on one or two bearing plates having a circular shape while optimising the inertia performance of the whole system and respecting some additional constraints such as ensuring a minimal equilibrium. This problem, which can be considered as a sophisticated multi-level two-dimensional packing problem where each level corresponds to a packing problem into a circular container, received some attention in the last two decades. Starting with Feng et al. (1999), the authors proposed to solve the problem by means of graph theory, group theory and global optimisation. Later, Teng et al. (2001) adopted a two-phases heuristic method which first solves a two-dimensional packing problem then translates bearing plates to improve the centre of mass of the satellite. Also, Sun and Teng (2003) proposed a two-phases heuristic based on non-linear programming which first distributes the objects on the bearing plates by means of a genetic algorithm and then solves several two-dimensional packing problems using ant colony optimisation. Xu et al. (2007) designed a particle swarm-based method while Liu and Teng (2008) put forward an evolutionary algorithm with expert knowledge. The latter formulated the problem as a non-linear program where the violation of the constraints is penalised into the objective function. Wang and Teng (2009) adopted a similar approach in a slightly different problem. Zhang et al. (2008) proposed a two-phases heuristic while Xu et al. (2010) designed a constructive heuristic based on an ordering of rectangles which is then embedded into a genetic algorithm. An interesting heuristic approach which relies heavily on physics notions such as energy and forces is put forward by Li et al. (2014). Their heuristic is applied on five SMLD examples with interesting results. Zhao et al. (2014) tackled the problem using particle swarm optimisation and immune algorithms with expert knowledge while Li et al. (2016) proposed an approach based on ant colony and particle swarm optimisation with expert knowledge. Fakoor et al. (2017) considered an extension of SMLD that limits the resonance phenomenon due to interferences while packing the objects into the module. Their approach relies on particle swarm optimisation and gradient-based sequential quadratic programming. Zhong et al. (2019) also extended SMLD by forbidding a circular region at the origin of the plates which represents a standing column.

Their heuristic is based on two main parts, namely, the assignment/reassignment of components to various bearing plates, and the layout optimisation which places the components on the surfaces using a differential evolution algorithm.

An interesting application of packing rectangles into a circle without rotation is found in the timber industry where the cutting of boards from logs takes place. Hinojosa et al. (2013) considered this problem and formulated it as a mixed-integer non-linear program (MINLP). The exact resolution of that formulation using a commercial solver turned out to be limited to instances involving 9 rectangles at most. The authors proposed an ordering heuristic and a simulating annealing (SA) to tackle larger instances. This SA algorithm will be revisited in this paper as this forms a suitable basis for benchmarking purposes.

When it comes to the packing of objects into containers having an arbitrary convex shape, the resolution is usually based on non-linear programming and the use of a generic solver. Birgin et al. (2006a) proposed a general approach based on so-called "sentinel sets" and non-linear programming for packing objects into an Euclidean n -dimensional space. The authors restricted their approach to address the two-dimensional packing of rectangles and rectangle-like objects into arbitrary convex regions. Using a solver, they could pack up to only 40 objects. Birgin et al. (2006b) tackled a less general problem formulated as a non-linear program using a solver in a multi-start fashion where they were able to pack up to 64 rectangles in over 2 hours on an AMD Opteron 244/1.8 GHz. Birgin and Lobato (2010) considered the problem of packing a maximum number of equal rectangles into an arbitrary convex region while allowing a common rotation angle for all objects and a 90° rotation for some of them. They formulated the problem as a MINLP using a branch & bound technique coupled with active set strategies. Similarly to Birgin et al. (2006b), Cassioli and Locatelli (2011) formulated the orthogonal packing of identical rectangles into a convex region as an unconstrained mixed integer global optimisation problem and solved it using an iterated local search combined with the L-BFGS algorithm of Liu and Nocedal (1989).

López and Beasley (2018b) considered the orthogonal packing of unequal rectangles into a circular container of fixed size. They formulated the problem as a mixed integer non-linear program and solved it using a third party solver, called SCIP, which is limited to solving small instances only. As we address the same problem, we will revisit their formulation in the next section. The authors then proposed an interesting implementation of formulation search space algorithm (FSS) to solve the problem which relies heavily on the use of SCIP. Instances involving up to 30 rectangles were generated with interesting results though requiring a relatively large amount of CPU. FSS is an interesting concept designed by Mladenović et al. (2005) where the change in neighbourhoods is replaced by the formulation space neighbourhood instead. For more information on non-linear programming and packing problems, the recent review by Birgin (2016) makes an interesting and informative read.

Table 1

The packing of rectangles into circular or arbitrary convex regions in literature

Reference	Container's shape ¹	Container's radius ²	Rotation angle ³	Objective function ⁴	Objects' shapes ⁵	Additional constraints	Method used ⁶
Feng et al. (1999)	○	F	θ_i	I	□	-	Graph theory, group theory, global optimisation
Teng et al. (2001)	○	F	θ_i	I	○, □	Equilibrium	Constructive heuristic
Sun and Teng (2003)	○	F	$\frac{\pi}{2}$	I	○, □	Equilibrium, minimal clearance	GA and ACO
Birgin et al. (2006a)	⊗	-	θ_i	#	unit □, U □	Forbidden region	Non-linear programming, sentinel sets
Birgin et al. (2006b)	⊗	-	$\frac{\pi}{2}$	#	unit □	-	Non-linear program solved by multi-start local solver
Xu et al. (2007)	○	V	θ_i	R, I	□	-	PSO
Liu and Teng (2008)	○	V	θ_i	R, I	○, □	Equilibrium, limited container's radius	EA with expert knowledge
Zhang et al. (2008)	○	F	$\frac{\pi}{2}$	I	○, □	Equilibrium	NN, GA, PSO, Quasi-principal component analysis
Wang and Teng (2009)	○	F	$\frac{\pi}{2}$	I, C, AC	○, □	Equilibrium, minimal clearance	EA with expert knowledge
Birgin and Lobato (2010)	⊗	-	$\theta, \frac{\pi}{2}$	#	unit □	-	MINLP tackled by B&B and active set strategies
Xu et al. (2010)	○	V	$\frac{\pi}{2}$	R, I	□	-	Constructive heuristic embedded into a GA
Cassoli and Locatelli (2011)	⊗	-	$\frac{\pi}{2}$	#	unit □	-	MINLP tackled by ILS and L-BFGS
Hinojosa et al. (2013)	○	F	0	A	□	-	Ordering heuristic embedded into a SA
Li et al. (2014)	○	V	$\frac{\pi}{2}$	#	□	Equilibrium, limited container's radius	Quasiphysical and Dynamic Adjustment Approach
Zhao et al. (2014)	○	F	θ_i	#	○, □	Equilibrium	PSO and Immune Algorithms with expert knowledge
Li et al. (2016)	○	F	θ_i	I	○, □	Equilibrium	ACO and PSO with expert knowledge
Fakoor et al. (2017)	○	F	$\frac{\pi}{2}$	I	○, □	Equilibrium, resonance	PSO and gradient-based method
López and Beasley (2018b)	○	F	$\frac{\pi}{2}$	A, #	□	-	FSS
Zhong et al. (2019)	○	F	$\frac{\pi}{2}$	I	○, □	Equilibrium, forbidden region	2-phases heuristic with EA
This paper	○	F	$\frac{\pi}{2}$	A, #	□	-	Constructive heuristic embedded into VNS and SA

¹ ○: Circular, ⊗: Arbitrary convex region;² F: Fixed, V: Variable;³ 0: No rotation allowed, $\frac{\pi}{2}$: Rotation by an angle of $\frac{\pi}{2}$ allowed, θ : Common rotation angle for all objects, θ_i : Arbitrary rotation angle for all objects;⁴ #: Maximise the number of objects packed, A: Optimise over the area packed or wasted; R: Minimise the radius of the container (applicable only for circular containers), I: Optimise the inertia performance of the packing, C: Optimise the clustering of the objects, AC: Optimise the accessibility to the objects;⁵ □: Rectangular objects with arbitrary dimensions, ○: Circular objects, U □: Union of equal rectangles (polygons);⁶ GA: Genetic algorithm, ACO: Ant colony optimisation, PSO: Particle swarm optimisation, EA: Evolutionary algorithm, NN: Neural network, B&B: Branch and bound, ILS: Iterated local search, L-BFGS: Limited memory Broyden-Fletcher-Goldfarb-Shanno algorithm, SA: Simulated annealing, FSS: Formulation search space, VNS: Variable neighbourhood search.

The above literature review suggests that, even though non-linear programming-based approaches developed for packing problems allow many features (arbitrary convex container, rectangular-like objects, arbitrary or uniform rotation, forbidden regions, etc), they still suffer from limitations due to the prohibiting computing time they require. Moreover, it is worth mentioning that in the presented literature, heuristic approaches which take benefit of the underlying combinatorial aspects of the considered packing problems have been seldom explored.

As mentioned in Table 1, this study addresses the orthogonal packing of unequal rectangles into a circular container of fixed size with the objective of maximising either the number of objects or the total area packed. To the best of our knowledge, the only reference that tackled this specific problem is due to López and Beasley (2018b). Given the two-folds complexity of a solution structure which is to determine the subset of rectangles to pack and their locations, we introduce a decoding procedure called `pack` which receives initially an ordered list of rectangles then returns a feasible packing. This combinatorial heuristic permits to move from the permutations search space to the packings search space. It is based on the concept of "border" that we introduce, a number of initial configurations which indicate how to pack the first objects in the list, and several moves which allow the packing of new objects. As the use of the constructive mechanism of `pack` on a single list is unlikely to provide a global optimum, we propose to integrate this procedure into two powerful metaheuristics, namely, a variable neighbourhood search and a simulated annealing. Moreover, exploiting the past placements of some items in the search permit us to propose two accelerated variants of those metaheuristics. To assess the performance of the proposed methodology, experiments are conducted on two datasets, one being a literature benchmark and the other which we newly generated and made publicly available for further benchmarking.

The contributions of this study are:

- to address the orthogonal packing of unequal rectangles into a circular fixed size container;
- to propose and determine the complexity of a flexible and efficient constructive heuristic 'pack';
- to provide and justify an algorithm 'buildBorder' which determines the surrounding border of an already known packing;
- to efficiently integrate `pack` into a variable neighbourhood search and a simulated annealing and to provide two accelerated version by means of `buildBorder`;
- to show the superiority of our methodology against the state-of-the-art method using benchmark instances and to assess the performance of the proposed metaheuristics on a new dataset that we generate and made publicly available.

The remainder of the paper is organised as follows: the next section is devoted to the detailed presentation of the procedure `pack`. In Section 3, the integration of `pack` into VNS and SA schemes is described.

Section 4 is dedicated to computational experiments including the generation of the newly constructed datasets. Finally, conclusions and some suggestions are given in Section 5.

2 The procedure pack

In this section, we first define formally the packing problem and we recall the formulation proposed by López and Beasley (2018b). We then introduce a decoding procedure which we refer to as 'pack'. The time complexity of the procedure 'pack' is also determined here. Finally, we present an algorithm to construct the border surrounding an already known packing.

2.1 Problem definition and formulation

Let $[n] = \{1, \dots, n\}$ be a set of rectangular objects of dimensions $(L_i, W_i)_{i \in [n]}$ where L_i and W_i denote the horizontal and vertical lengths of the rectangles respectively. Let v_i be a value associated with rectangle i and assume a container having a circular shape of fixed radius R . We define a packing p as the placement of a subset of rectangles into the container such that no two rectangles overlap and no rectangle passes through the container's perimeter. Formally, p is a 4-tuple $(\alpha_i, x_i, y_i, \theta_i)_{i \in [n]}$ such that α_i equals 1 if the rectangle i is packed and 0 otherwise, (x_i, y_i) are the coordinates of the centre of rectangle i and θ_i its rotational angle which value is considered in this study to be either 0 or $\frac{\pi}{2}$. In case rectangle i is not packed, (x_i, y_i) and θ_i are null. Let $V_i^1 = (x_i + L_i/2, y_i + W_i/2)$, $V_i^2 = (x_i - L_i/2, y_i + W_i/2)$, $V_i^3 = (x_i - L_i/2, y_i - W_i/2)$ and $V_i^4 = (x_i + L_i/2, y_i - W_i/2)$ be the coordinates of the upper-right, upper-left, lower-left and lower-right vertices of rectangle i respectively.

Some definitions related to the connectedness of a packing:

- Two different rectangles i and j overlap if $|x_i - x_j| - (L_i + L_j)/2 < 0$ and $|y_i - y_j| - (W_i + W_j)/2 < 0$.
- The sides AB and CD of two rectangles are *contiguous* if AB and CD are colinear and $C \in AB$, $D \in AB$ or $A \in CD$.
- A partial packing p is a 4-tuple $p = (\alpha_i, x_i, y_i, \theta_i)_{i \in E}$ such that $E \subsetneq [n]$ and $|E| > 0$.
- A partial packing p is *empty* if $\alpha_i = 0$ for all i in E .
- Two partial packings $p_1 = (\alpha_i, x_i, y_i, \theta_i)_{i \in E_1}$ and $p_2 = (\alpha_i, x_i, y_i, \theta_i)_{i \in E_2}$ are *disjoints* if $E_1 \cap E_2 = \emptyset$.
- Two disjoints partial packings p_1 and p_2 *overlap* if there exists a rectangle packed in p_1 and another one packed in p_2 such that the two rectangles overlap.
- Two partial packings are *compatible* if they are disjoints and do not overlap.
- The union of two compatible partial packings p_1 and p_2 is defined by $p_1 \cup p_2 = (\alpha_i, x_i, y_i, \theta_i)_{i \in E_1 \cup E_2}$.

- Two compatible partial packings p_1 and p_2 are *contiguous* if a rectangle packed in p_1 and a rectangle packed in p_2 have contiguous sides.
- A packing p is said to be *connected* if there exists no two non-empty compatible partial packings which are not contiguous and which union equals p .

The quality of a packing p can be measured by $f(p) = \sum_{i \in [n]} v_i \alpha_i$ as follows:

- (i) if $v_i = L_i W_i$ then $f(p)$ corresponds to the total area packed and,
- (ii) if $v_i = 1$ then $f(p)$ becomes the total number of rectangles packed.

The packing problem considered in this study is to find a packing p^* having a maximum value $f(p^*)$.

López and Beasley (2018b) formulated the problem of packing n rectangles of dimensions $(L_i, W_i)_{i \in [n]}$ into a circular container of fixed radius R without rotation as follows:

$$(P) \quad \max \sum_{i=1}^n v_i \alpha_i \quad (1)$$

s.t.

$$-\alpha_i(\sqrt{R^2 - W_i^2/4} - L_i/2) \leq x_i \leq \alpha_i(\sqrt{R^2 - W_i^2/4} - L_i/2), i = 1, \dots, n \quad (2)$$

$$-\alpha_i(\sqrt{R^2 - L_i^2/4} - W_i/2) \leq y_i \leq \alpha_i(\sqrt{R^2 - L_i^2/4} - W_i/2), i = 1, \dots, n \quad (3)$$

$$\|V_i^k\|^2 \leq \alpha_i R^2 + (1 - \alpha_i) R_i^2, \quad i = 1, \dots, n; k = 1, \dots, 4 \quad (4)$$

$$\alpha_i \alpha_j [\max\{|x_i - x_j| - (L_i + L_j)/2, |y_i - y_j| - (W_i + W_j)/2\}] \geq 0, \\ i = 1, \dots, n; j = 1, \dots, n; j > i \quad (5)$$

$$\alpha_i \in \{0, 1\}, i = 1, \dots, n \quad (6)$$

where $\|V_i^k\|$ is the euclidean norm of V_i^k and R_i is the radius of the smallest circle encompassing the i -th rectangle in case it is positioned at the origin. In the formulation (P), depending on the value of v_i , (1) maximises either the total area or the total number of rectangles packed. Constraints (2)-(3) make sure that in case a rectangle is packed, its coordinates lie in the appropriate range, otherwise its coordinates are forced to be zero. Constraints (4) ensure that, in case rectangle i is packed, all its vertices lie within the perimeter of the container. Constraints (5) prevent the overlapping between rectangles that are packed. The binary nature of the variables α_i is guaranteed by the constraints (6). Even though this formulation does not consider rotations, the authors proposed an elegant extension to allow orthogonal rotations of the rectangles. Suppose that a rectangle i can be rotated by a right angle, it is sufficient to add a new rectangle j that represents rectangle i if it is rotated, so that $L_j = W_i, W_j = L_i, v_j = v_i$ then, to add to the formulation:

$$\alpha_i + \alpha_j \leq 1 \quad (7)$$

Constraint (7) ensures that not both the original rectangle and its rotated equivalent are used. The authors used the solver SCIP to tackle (P) but could not solve any proposed instance to proven global optimality within the time limit imposed.

The number of solutions in the considered problem is dramatically large as any solution requires to identify which subset of rectangles to pack and also where to pack them. The first decision involves $\sum_{i=1}^n \binom{n}{i} = O(2^n)$ possibilities while the second offers an infinite number of choices given the decision is made over a subset of \mathbb{R}^2 . The main idea of our approach is to consider a much smaller search space which consists of all the ordered lists of the n rectangles that we will simply call lists. This is possible with a "decoding" procedure which permits to move from the lists search space to the packings search space.

2.2 Overview of the procedure pack

The procedure pack makes an intensive use of the notion of "border". The border associated with a packing p is a data structure, preferably implemented as a circular chained list, which contains the corner points of the rectangles that fall on the external side of p . This array is updated during the procedure by adding and removing corner points whenever a new rectangle is packed. To avoid redundancy, the handling of a border follows two rules:

- (i) the border must not contain any sequence of three aligned points and,
- (ii) any two successive points in the border must be different (see for instance Figure 1).

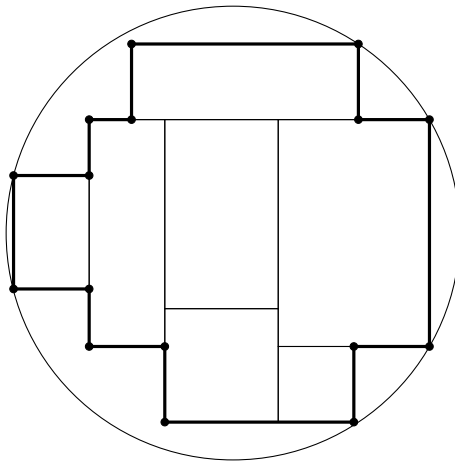


Fig. 1. Example of a packing delimited by its border

The procedure pack (see Algorithm 1) requires as input a list l and returns a packing p_{best} . Initially, no rectangle is packed in p_{best} (see line 1). The quality of the packing obtained after applying the procedure

pack on a list depends highly on the way the first one or two objects are placed into the container. Therefore, we apply several initial configurations to maximise the chances of obtaining a good packing. For each possible initial configuration, an initial packing is built and its border is initialised (see line 4). Next, for every remaining rectangle in the list, a promising feasible placement is sought around the border among all possible placements (see line 6). The determination of a promising feasible placement is presented in Section 2.4. If such a placement exists, the incumbent rectangle is packed at that position and the border is updated (see line 8). Then, the procedure continues with the next rectangle in the list (see line 10). When the end of the list is reached, if the incumbent packing is better than the best one, the best packing is updated (see line 12). The procedure terminates when all initial configurations have been considered and the best packing is returned.

Algorithm 1: pack

Input : list l

Output: packing p_{best}

```

1 set  $p_{best} := \emptyset$ ;
2 Construct the set of initial configurations  $E$ ;
3 foreach possible initial configuration in  $E$  do
4   Build a packing  $p$  based on the incumbent configuration and initialise its border;
5   while end of the list  $l$  not reached do
6     Find on the border a promising feasible placement for the incumbent rectangle (see
7     Subsection 2.4);
8     if feasible placement exists then
9       Pack the rectangle and update the border;
10    end if
11    Move to the next rectangle in the list  $l$ ;
12  end while
13  if  $p$  is better than  $p_{best}$  then  $p_{best} := p$ ;
14 end foreach
15 return  $p_{best}$ 

```

2.3 Initial configurations (step 2 of Algorithm 1)

We apply five initial configurations denoted by C (centre), R (right), T (top), RS (right-shifted), RA (right-aligned). We describe the initial configurations, the way to calculate the coordinates of the objects they pack and the composition of the initial border. For the sake of clarity, assume without loss of generality that the first two rectangles in the list are indexed 1 and 2.

In the C configuration, the centre of Rectangle 1 is simply placed at the origin. In the R configuration, Rectangle 1 is placed in such a way that its lower right and upper right corners lie on the circle. In this case, the centre (x, y) of Rectangle 1 must satisfy the following system (S_1) :

$$(S_1) \begin{cases} (x + \frac{L}{2})^2 + (y - \frac{W}{2})^2 = R^2 & \dots \text{ (lower-right corner on the circle)} \\ (x + \frac{L}{2})^2 + (y + \frac{W}{2})^2 = R^2 & \dots \text{ (upper-right corner on the circle)} \\ x \geq 0 & \dots \text{ (rectangle must fit into the circle)} \end{cases}$$

The solution of (S_1) is $(x, y) = (\sqrt{R^2 - \frac{W^2}{4}} - \frac{L}{2}, 0)$. This configuration is applicable only if $R^2 - \frac{W^2}{4} \geq 0$.

In the T configuration, Rectangle 1 is placed in such a way that its upper right and upper left corners are on the circle. Applying similar calculations to the ones applied for R configuration results in the coordinates $(0, \sqrt{R^2 - \frac{L^2}{4}} - \frac{W}{2})$.

In the three configurations C, R and T, the initial border is given by $(V_1^1, V_1^2, V_1^3, V_1^4)$.

In the RS configuration, Rectangle 1 is first placed as in the R one. Rectangle 2 is put on the left of the first one then shifted upward in a way that the upper-left corner of the second rectangle lies on the circle (see Figure 2). Let (x_1, y_1) and (x, y) be the coordinates of Rectangles 1 and 2 respectively. The location of Rectangle 1 is known a priori by the R configuration. Thus, (x, y) must satisfy the following system:

$$\begin{cases} (x_1 - \frac{L_1}{2} - L_2)^2 + (y + \frac{W_2}{2})^2 = R^2 & \dots \text{ (upper-left corner of Rectangle 2 on the circle)} \\ y \geq 0 & \dots \text{ (Rectangle 2 must fit into the circle)} \end{cases}$$

leading to $(x, y) = (x_1 - \frac{L_1}{2} - L_2, \sqrt{R^2 - (x_1 - \frac{L_1}{2} - L_2)^2} - \frac{W_2}{2})$. This placement is feasible only if $R^2 - (x_1 - \frac{L_1}{2} - L_2)^2 \geq 0$. To ensure that the packing remains connected, the condition $y \leq \frac{W_1 + W_2}{2}$ is added. The border is defined by $(V_1^1, V_1^2, V_2^1, V_2^2, V_2^3, V_2^4, V_1^3, V_1^4)$ on condition that $V_1^2 \neq V_2^1$ and $V_2^4 \neq V_1^3$, in which case, the redundant nodes are omitted.

In the RA configuration, Rectangle 1 is placed on Rectangle 2 in such a way that their left sides are

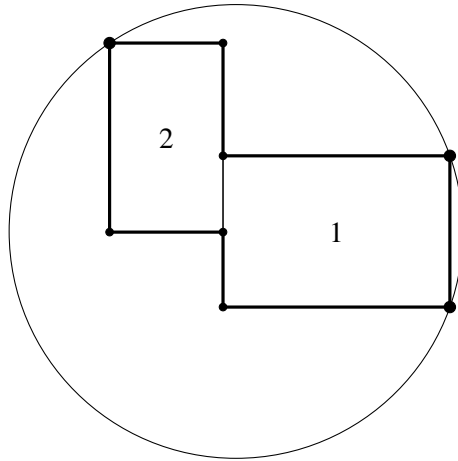


Fig. 2. Illustration of the RS initial configuration and its border

aligned and, the upper-right corner of Rectangle 1 and the lower-right corner of Rectangle 2 lie on the circle (see Figure 3). The coordinates (x, y) of the centre of Rectangle 1 must satisfy the following set of conditions (S_2) .

$$(S_2) \begin{cases} (x + \frac{L_1}{2})^2 + (y + \frac{W_1}{2})^2 = R^2 & \dots \text{ (upper-right corner of Rectangle 1 on the circle)} \\ (x - \frac{L_1}{2} + L_2)^2 + (y - \frac{W_1}{2} - W_2)^2 = R^2 & \dots \text{ (lower-right corner of Rectangle 2 on the circle)} \\ x \geq 0, y \geq 0 & \dots \text{ (Rectangle 1 must fit into the circle)} \end{cases}$$

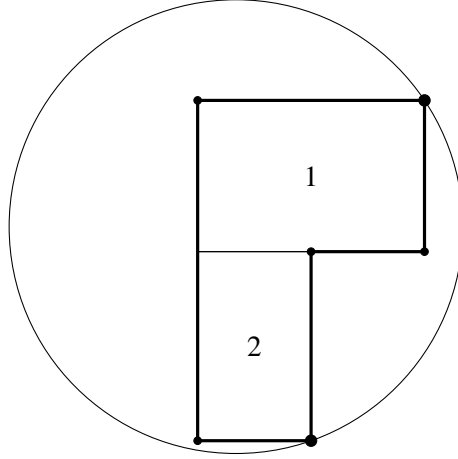


Fig. 3. Illustration of the RA initial configuration and its border

The system (S_2) consists of two bivariate polynomials of second degree. The problem of solving a system of multivariate quadratic equations over finite fields is NP-complete (Garey and Johnson, 1979). However, the resolution of (S_2) is possible given its small dimensions using the Buchberger's algorithm (Buchberger, 1976). Building over the initial set of polynomials, this algorithm constructs an equivalent system of polynomials in the form of a Gröbner basis which is usually easier to solve than the original system. Making use of the SageMath system for symbolic calculus (SageMath, Inc., 2018), the whole process results in the following solution

$$x = -\frac{1}{2} L_2 + \frac{1}{2} (W_1 + W_2) \sqrt{\frac{4 R^2}{(L_1 - L_2)^2 + (W_1 + W_2)^2} - 1}$$

$$y = \frac{1}{2} W_2 - \frac{1}{2} (L_1 - L_2) \sqrt{\frac{4 R^2}{(L_1 - L_2)^2 + (W_1 + W_2)^2} - 1}$$

The coordinates of rectangle 2 are then $(x - \frac{L_1}{2} + \frac{L_2}{2}, y - \frac{W_1}{2} - \frac{W_2}{2})$.

Obviously, the RA configuration is applicable only if $(\frac{L_1}{2} - \frac{L_2}{2})^2 + (\frac{W_1}{2} + \frac{W_2}{2})^2 \leq R^2$. If the two rectangles have different widths then the border is $(V_1^1, V_1^2, V_2^3, V_2^4, V_2^1, V_1^4)$. It is worth mentioning that if Rectangles 1 and 2 have the same widths and their stacking is considered as one rectangle then the formula reduces to the one obtained for the R configuration. Also, as Rectangles 1 and 2 are contiguous in this configuration, the resulting packing is connected.

2.4 Determining a promising feasible placement (step 6 of Algorithm 1)

A placement of the incumbent rectangle is feasible if no overlapping occurs between this rectangle and a rectangle already packed and no over-crossing happens with the container's perimeter. We determine a promising feasible placement for the incumbent rectangle around the border as follows. Let A , B and C be three consecutive corner points on the border. As no redundancy is allowed, the three points are disjoint and not aligned. We consider five cases when placing the incumbent rectangle on the border. The dashed area in Figure 4 represents the inner side of the border. The case (a) is when the point C lies on the right side of the line (AB) in which case the incumbent rectangle is stuck to the corner point B . When C is on the left side of (AB) , we consider two possible placements of the rectangle: either it sticks to the edge segment AB (denoted case (b)), or, it sticks to the edge segment BC (denoted case (c)). If the rectangle overlaps with the container in cases (b) or (c), the rectangle is translated according to \vec{BA} until its upper right corner lies on the circle (denoted case (d)), or, it is translated according to \vec{BC} until its upper right corner lies on the circle (denoted case (e)) respectively. All these placements are considered for every corner point on the border. Among all these placements, the feasible placement having a minimum distance to the centre of the container is returned. The rectangle is packed if and only if such a feasible placement exists.

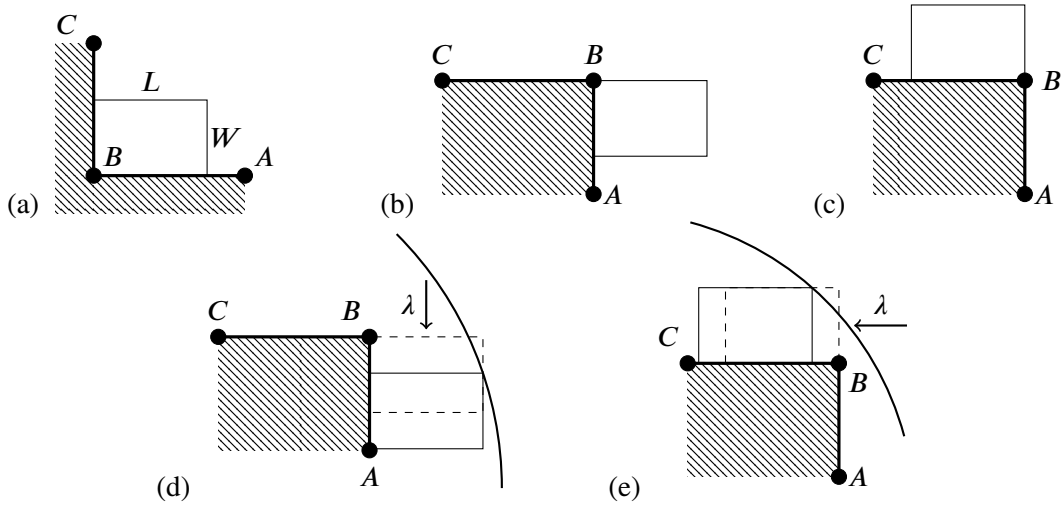


Fig. 4. Five ways of placing a rectangle on the border.

In order to determine whether the point C lies on the right or left side of (AB) , we test the sign of the dot product $N_{(AB)} \cdot \vec{BC}$ where $N_{(AB)}$ is the normal vector to the line (AB) . $N_{(AB)}$ is given by the gradient of the equation of (AB) that is $N_{(AB)} = (y_A - y_B, x_B - x_A)$. It is easy to see that:

$$N_{(AB)} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \cdot \vec{AB}$$

Therefore, $N_{(AB)}$ points always to the left side of (AB) , precisely the inner side of the border. As a result,

if the dot product $N_{(AB)} \cdot \vec{BC}$ is negative then $N_{(AB)}$ and \vec{BC} are pointing in opposite directions which corresponds to case (a). If this product is positive then it corresponds to the remaining cases.

Let $(x_i, y_i)_{i=a, \dots, e}$ be the coordinates of the centre of the incumbent rectangle for cases (a)-(e) respectively. Using basic mathematical manipulations, we get the following:

$$(x_a, y_a) = B + \frac{L}{2\|\vec{BA}\|} \vec{BA} + \frac{W}{2\|\vec{BC}\|} \vec{BC} \quad (8)$$

$$(x_b, y_b) = B + \frac{W}{2\|\vec{BA}\|} \vec{BA} + \frac{L}{2\|\vec{CB}\|} \vec{CB} \quad (9)$$

$$(x_c, y_c) = B + \frac{L}{2\|\vec{BC}\|} \vec{BC} + \frac{W}{2\|\vec{AB}\|} \vec{AB} \quad (10)$$

Regarding the cases (d) and (e), the problem is to find a step size λ which satisfies:

$$\left(\left\| B + \lambda \frac{\vec{BA}}{\|\vec{BA}\|} + L \frac{\vec{CB}}{\|\vec{CB}\|} \right\| \right)^2 = R^2 \quad (\text{case (d)}) \quad (11)$$

$$\left(\left\| B + \lambda \frac{\vec{BC}}{\|\vec{BC}\|} + W \frac{\vec{AB}}{\|\vec{AB}\|} \right\| \right)^2 = R^2 \quad (\text{case (e)}) \quad (12)$$

In case (d), in addition to (11), λ must satisfy the following two conditions in order to maintain the connectedness of the packing:

$$\lambda > 0 \Rightarrow \lambda \leq \|\vec{BA}\| \quad (13)$$

$$\lambda < 0 \Rightarrow |\lambda| \leq W \quad (14)$$

Whatever the sense in which the rectangle is shifted, the conditions (13) and (14) ensure that it remains contiguous to the edge BA . Similarly, in case (e), in addition to (12), the following two conditions must hold:

$$\lambda > 0 \Rightarrow \lambda \leq \|\vec{BC}\| \quad (15)$$

$$\lambda < 0 \Rightarrow |\lambda| \leq L \quad (16)$$

where the conditions (15) and (16) ensure the rectangle remains contiguous to the edge BC .

The expressions (11) and (12) are second degree equations in λ . If no overlap occurs, the smallest root in absolute value, when the discriminant is non negative, is chosen in order to keep the rectangle inside the container.

In all cases (a)-(e), if the line (AB) is horizontal in spite of being vertical or vice-versa, it is sufficient to swap the two values L and W in all the expressions (8)-(12), (14) and (16).

As described above, all the initial configurations and the moves used for the placements of the rectangles in the procedure `pack()` are designed in such a way that the resulting packing is connected as it is

stated in the following proposition.

Proposition 1. *pack returns a connected packing.*

2.5 Updating the border (step 8 of Algorithm 1)

When a rectangle is packed on a border, the latter needs to be updated. Assume that A , B and C are in the first quadrant of the plane and that i is the index of the incumbent rectangle. The border update for the other possibilities can be deduced from the following descriptions. In case (a), the sequence (V_i^4, V_i^1, V_i^2) is inserted in place of the corner point B in the border. In case (b) (resp. (e)), the sequence (V_i^3, V_i^4, V_i^1) (resp. $(V_i^3, V_i^4, V_i^1, V_i^2)$) replaces (resp. precedes) B in the border, otherwise, in case (c) (resp. (d)), the sequence (V_i^1, V_i^2, V_i^3) (resp. $(V_i^4, V_i^1, V_i^2, V_i^3)$) replaces (resp. succeeds) B in the border. Once the rectangle is packed, a post-processing is operated on the border to keep the corner points only.

2.6 Flexibility of the procedure pack

Applying small changes can broaden the applicability of pack considerably and, consequently the applicability of our methodology. As an example, the following two possible extensions to pack can be easily implemented.

- (a) **Enable the rotation of rectangles by 90°** - In case the considered instance allows the rotation of rectangles by 90° , any rectangle for which no feasible placement could be found is rotated by 90° and a feasible placement is sought. If the latter exists, the rotated rectangle is packed. Otherwise, the procedure pack moves to the next rectangle.
- (b) **Forbid a region** - It is possible to forbid a connected region during the packing process. To do so, the undesired region is simply considered as an initial configuration defined by a border in our procedure. Then, the procedure packs the rectangles around the border using the moves described above.

2.7 Complexity of pack

The following result gives the complexity of pack.

Theorem 1. *pack runs in $O(kn^2)$ where k is the number of initial configurations and n the number of rectangles.*

Proof. The complexity of pack reduces to the one of line 6 in Algorithm 1 times the number of initial configurations as all the other lines are constant in time. Each time a rectangle is packed, the number of new corners points in the border is 4 in the worst case (see cases (d) and (e) in Fig. 4). Assuming the container's area is sufficient to pack all the rectangles, the number of placements considered in the worst

case for the second rectangle is 4, 8 for the third one, 12 for the fourth one, ... and $4(n - 1)$ for the last rectangle. This sums up to $4(1 + 2 + \dots + (n - 1)) = 4 \cdot \frac{n(n-1)}{2}$ which is $O(n^2)$. Therefore, the complexity of pack is $O(kn^2)$. \square

2.8 An illustrative example

We illustrate the procedure pack on the instance rect1 proposed by López and Beasley (2018b) which consists in packing 10 rectangles into a circular container of radius 3.62 without rotation. Details of the instance together with three lists of rectangles l^1 , l^2 and l^3 which are given as inputs are presented in the upper-left part of Figure 5. The list l^1 (resp. l^2) is sorted in the decreasing (resp. increasing) order of the rectangles' areas. The list l^3 is obtained by means of a more sophisticated search described in Section 3. In Figure 5, the parts (i), (ii) and (iii) illustrate the packings returned by the pack procedure when applied on the lists l^1 , l^2 and l^3 respectively. The last part presents the results for the various packings. For a given list, the procedure pack builds a packing for every possible initial configuration then returns the best packing obtained (see Section 2.3). The best initial configuration for l^1 turns to be R implying that rectangle 10 is initially packed so that V_{10}^1 and V_{10}^4 lie on the circle. The initial border is $(V_{10}^1, V_{10}^2, V_{10}^3, V_{10}^4)$. The next five rectangles in the list which are 9, 8, 7, 6 and 5 can not be packed by any predefined move. The best placement for rectangle 4 with regard to the border is to place this rectangle on the left of rectangle 10 according to the move (e) (see Section 2.4). Then, rectangle 3 is placed under rectangle 4 by the move (a). No more rectangle can be packed in this case. The resulting border is $(V_{10}^1, V_{10}^2, V_4^1, V_4^2, V_4^3, V_3^2, V_3^3, V_3^4, V_{10}^3, V_{10}^4)$. The best initial configuration for l^2 is C showing that rectangle 1 is packed at the centre. Rectangle 2 which is the next in the list is packed on rectangle 1 by the move (c). Then, rectangle 3 is packed under rectangle 2 by the move (a). The closest placement to the centre for rectangle 4 lies on the right of rectangle 1 by means of the move (c). No feasible placement is found for rectangle 5 but rectangle 6 can be packed under rectangle 1 by a little shift as allowed by the move (e). No more rectangles in l^2 can be packed. As for the list l^1 , the best initial configuration for l^3 is R implying that rectangle 7 is packed on the right of the circle. The next rectangles 6, 4, 2, 3 and 1 are packed according to the moves (c), (a), (e), (a) and (a) respectively. No more rectangle can be packed by the predefined moves. In part (iv), when comparing the lists l^1 and l^2 , the packing produced with l^1 covers a larger area than the one of l^2 , whereas the packing generated with l^2 contains a larger number of items than the one of l^1 . However, the list l^3 which is neither sorted in the decreasing nor the increasing order of rectangles' areas produces the packing with both the largest area and number of items packed.

As presented in this example, the quality of the packing obtained depends highly on the list of rectangles given as input to pack, *a fortiori*, on the first two rectangles in the list.

Instance and lists

$n=10, R=3.62,$

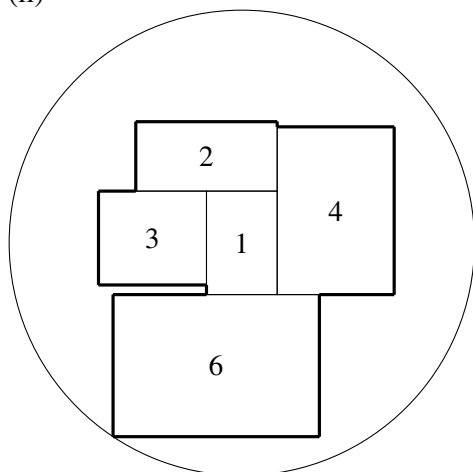
$l^1 = (10, 9, 8, 7, 6, 5, 4, 3, 2, 1),$

$l^2 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10),$

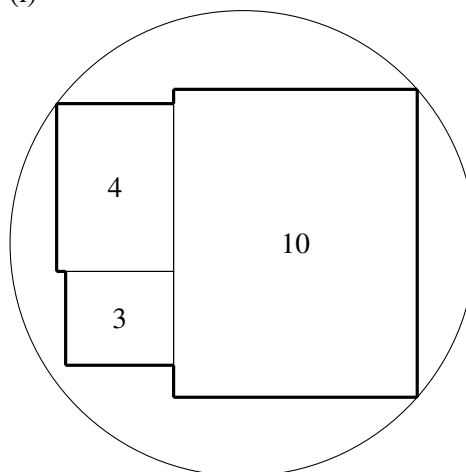
$l^3 = (7, 6, 4, 2, 3, 1, 5, 8, 9, 10).$

i	L_i	W_i
1	1.10	1.61
2	2.20	1.08
3	1.68	1.46
4	1.82	2.61
5	2.70	2.57
6	3.21	2.21
7	2.99	3.51
8	3.68	3.42
9	4.62	3.36
10	3.79	4.79

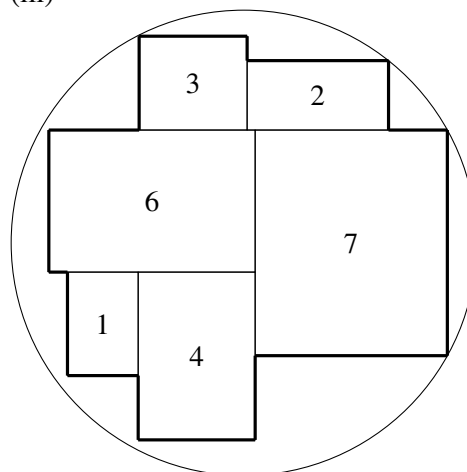
(ii)



(i)



(iii)



(iv)

l^i	Order	Packed area	Number of items
l^1	\searrow	25.3571	3
l^2	\nearrow	18.4441	5
l^3	-	28.9390	6

Fig. 5. Illustration of pack on the instance rect 1 of López and Beasley (2018b) without rotation.

2.9 Building the border of an already known packing

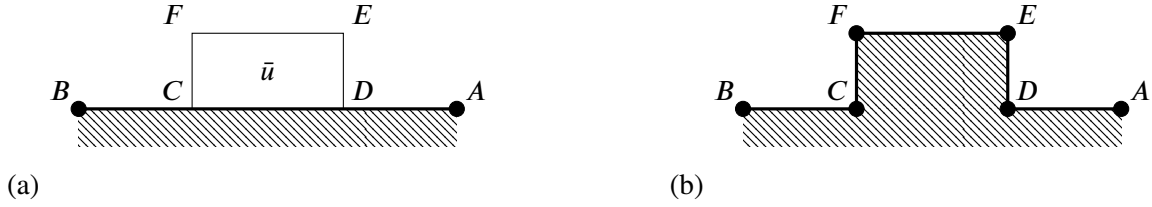
In this section, we present algorithm 2 which builds and returns the border surrounding a connected packing p given as input. During the process, the algorithm maintains a set S of rectangles which are packed in p but not surrounded by the border $b(p)$ yet. Initially, $b(p)$ receives the corners of a rectangle u_1 which is then removed from S . Then, while S is not empty, the algorithm examines every edge $[A, B]$ forming $b(p)$ in order to find if there is a rectangle $\bar{u} = (C, D, E, F)$ with a side $[C, D]$ contiguous to $[A, B]$. If the rectangle \bar{u} exists, its corners are integrated into $b(p)$ without forming duplicated nodes and \bar{u} is removed from S (see Figure 6). The algorithm terminates when S is empty.

Algorithm 2: buildBorder**Input** : connected packing p .**Output:** border $b(p)$.// Let q be the number of objects packed in p and $S = \{u_1, \dots, u_q\}$ be the set of rectangles which are packed and not integrated into the border yet.

```

1  $b(p) := (V_{u_1}^1, V_{u_1}^2, V_{u_1}^3, V_{u_1}^4)$ ;
2  $S := S \setminus \{u_1\}$ ;
3 while  $S \neq \emptyset$  do
4   Find an edge  $[A, B]$  in  $b(p)$  and a rectangle  $\bar{u} = (C, D, E, F)$  in  $S$  such that the side  $[C, D]$  is
   contiguous to  $[A, B]$ ;
5   Insert the relevant corners of  $\bar{u}$  into  $b(p)$ ;
6    $S := S \setminus \{\bar{u}\}$ ;
7 end while
8 return  $b$ 

```

**Fig. 6.** Inserting the corners of rectangle \bar{u} into the border.

Theorem 2. Let p be a connected packing containing q rectangles then Algorithm 2 finds the border $b(p)$ surrounding p in $O(q^3)$.

Proof. Given the connectedness of the packing p , every rectangle which is packed but not integrated yet into the border $b(p)$ will be reached and surrounded by the latter in the algorithm by means of the operations at lines 4 and 5. As all rectangles are integrated into $b(p)$ and every integrated rectangle is removed from S (see line 6), the set S becomes empty and the algorithm converges.

The time complexity of the algorithm equals the complexity of the while loop. The worst case occurs when the last edge in $b(p)$ is contiguous to the last rectangle examined in S and when four new corners are inserted at every iteration. Depending on the positions of A and B , we can determine in constant time whether the edge $[A, B]$ is a right, top, left or bottom edge of $b(p)$. Furthermore, depending on the orientation of $[A, B]$, we can limit the search for a contiguous rectangle's side $[C, D]$ to those which are confronting $[A, B]$. For instance, if $[A, B]$ is a right edge, we are looking for a left rectangle's side $[C, D]$ contiguous to $[A, B]$ and so on. As testing whether two segments are contiguous is constant in time, the number of operations performed at line 4 is $4k(q - k)$ where k is the number of rectangles already integrated into $b(p)$. The insertion and the removal at lines 5-6 are constant in time if $b(p)$ and S are implemented as chained lists. As a consequence, the complexity of the algorithm which equals the while loop's complexity corresponds to the number of pairs (edge $[A, B]$, rectangle's side $[C, D]$)

examined during the whole process which is given in the worst case by:

$$4(q-1) + 8(q-2) + \dots + 4(q-1) = \sum_{k=1}^{q-1} 4k(q-k) = \frac{2}{3}q(q^2-1) = O(q^3)$$

□

It is worth mentioning that Algorithm 2 can be easily adapted to determine the decomposition of a packing into connected components. Indeed, starting from a rectangle u given as an input, the algorithm determines the connected partial packing to which u belongs. If all the rectangles have been inserted into $b(p)$ then the packing is connected. Otherwise, the algorithm restarts from another rectangle which does not belong to the connected component of u to determine its connected component and so on.

3 Integration of pack into two metaheuristics

The application of pack on one single list is a simple constructive heuristic which may obviously lead to poor local optima. One way forward to increase the chances in providing a globally optimal packing is then to incorporate pack into a metaheuristic. As an illustration only, Hifi and Yousef (2019) provides a successful integration of a constructive heuristic into an effective metaheuristic scheme for the packing of spheres into a cuboid. In this section, we first review briefly some applications of two powerful metaheuristics, namely, variable neighbourhood search (VNS) and simulated annealing (SA), to packing problems. We then present two integrations of the procedure pack to these two metaheuristics. Moreover, based on Algorithm 2, we propose two accelerated variants of these methods. The purpose of these integrations is to visit a broad set of lists in order to maximise the chances to reach a global optimum. For completeness, we terminate this section by a discussion on the nature of our methodology and how it is situated in the field of metaheuristics.

3.1 Applications of VNS and SA to packing problems in the literature

VNS is a metaheuristic developed by Mladenović and Hansen (1997) which consists in applying systematic change of neighbourhood within a local search algorithm. VNS (and its variants) have been applied successfully to several packing problems. For instance, Mladenović et al. (2005; 2007) applied reformulation descent and FSS to the packing of equal circles into a unit circle respectively, enabling a significant speed-up over existing methods and packing up to 100 circles. López and Beasley (2011) proposed a competitive FSS approach for the packing of equal circles into a variety of containers and packing up to 500 circles. M'Hallah and Alkandari (2012) adopted a VNS for the packing of unit spheres into the smallest cube where the local search is performed by a non-linear programming solver. Later, M'Hallah et al. (2013) applied a similar approach to the packing of unit spheres into the smallest sphere and pro-

duced 29 new upper bounds out of 48 benchmark instances. López and Beasley (2013) extended their earlier work by tackling the problem of packing unequal circles having a variable size into a container of fixed size having various shapes. They proposed a FSS method coupled with a perturbation phase. Instances involving up to 35 objects were considered. In a subsequent study, López and Beasley (2016) addressed the problem of packing a subset of unequal circles into a circular container of fixed size where the choice of the subset of circles to pack is also a decision variable. They proposed a FSS approach that they test on instances they generated and which involve up to 40 circles. Zeng et al. (2016) combined tabu search and variable neighbourhood descent to tackle the packing of unequal circles into a circular container with the aim to minimise the radius. In their following study, Zeng et al. (2018) proposed a similar metaheuristic for the packing of unequal circles into a square of minimum side length.

Simulated annealing (SA) is a metaheuristic introduced by Kirkpatrick et al. (1983). It results from an analogy between optimisation and the annealing process found in the steel industry. SA has been applied successfully by several authors to the packing of rectangles into a larger containing rectangle (Dowsland, 1993; Leung et al., 2003; Soke and Bingul, 2006; Leung et al., 2011). Gomes and Oliveira (2006) considered the more general problem of packing irregular shapes into a rectangular container which they successfully addressed by hybridising SA and linear programming. Martins and Tsuzuki (2010) efficiently applied SA to another general problem which consists in packing irregular polygons into a container while allowing arbitrary rotations. Hinostrroza et al. (2013) developed a SA for a packing problem similar to the one we consider though no rotation is allowed in their algorithm. This latter relies heavily on an ordering heuristic which packs rectangles in a specified order. The dataset on which they assessed the performance of their method has not been made public. In this section, we present an alternative simulated annealing which integrates the procedure `pack`.

For further reading, Salhi (2017) provides an informative description of metaheuristics in general including VNS and SA. Very recently, Hansen et al. (2019) produces an updated chapter that presents the basic schemes and extensions as well as recent developments of VNS.

3.2 Integration of `pack` into a VNS scheme

Algorithm 3 is a variable neighbourhood search. It requires initially three parameters k_{\max} , it_{\max} and l_0 which represent the maximum number of neighbourhood structures, the maximum number of non-improving iterations allowed and an initial ordered list of the n rectangles respectively. It returns the best packing p_{best} found so far. As described above, our VNS explores the lists search space. The *neighbourhood structure* used is defined by $N_k(l)$ which constitutes the set of lists obtained by swapping k pairs of rectangles indices in the list l with $1 \leq k \leq k_{\max}$. These neighbourhoods are explored by the procedure `shake(l_{best} , k)` which returns a list obtained by swapping randomly k pairs of indices in the list l_{best} (see

line 7). In order to convert a list l into a packing, the algorithm relies on the *decoder* $\text{pack}(l)$ which constitutes a key component of our methodology (see line 8). Overall, our algorithm consists of two nested loops. The inner one explores the lists and packings search spaces by means of the two procedures shake and pack (see lines 5-18). At this step, the list l_{best} which permitted to find the best packing p_{best} so far is shaken to generate a new list l using the shake procedure. A new packing p is built from l using the procedure pack . If the value associated with p is better than the best value found so far, the search is recentred on l by reinitialising k and the outer loop iterator (see lines 9-13), otherwise, the search is enlarged around l_{best} (see line 15). The outer loop (lines 3-19) controls the overall process by limiting the number of non-improving iterations to it_{max} .

Algorithm 3: Variable Neighbourhood Search (VNS)

Input : integers k_{max} , it_{max} , list l_0

Output: packing p_{best} .

```

1  $l_{best} := l_0$ ;
2  $p_{best} := \text{pack}(l_{best})$ ;
3  $it := 1$ ;
4 repeat
5    $k := 1$ ;
6   repeat
7      $l := \text{shake}(l_{best}, k)$ ;
8      $p := \text{pack}(l)$ ;
9     if  $f(p) > f(p_{best})$  then
10       $l_{best} := l$ ;
11       $p_{best} := p$ ;
12       $k := 1$ ;
13       $it := 1$ ;
14     else
15       $k := k + 1$ ;
16       $it := it + 1$ ;
17     end if
18   until  $k > k_{max}$ ;
19 until  $it > it_{max}$ ;
20 return  $p_{best}$ 

```

3.3 Integration of pack into a SA scheme

Algorithm 4 is a simulated annealing metaheuristic that incorporates the procedure pack . It receives as input $T_0, T_f, \alpha, N_t, k, it_{max}$ and l_0 which are the initial and the final temperatures, the cooling factor, the number of iterations spent at every temperature, the number of rectangles to swap in the shake procedure, the maximum number of non-improving iterations allowed and the initial list respectively. It returns the

best packing found by the algorithm. Initially, the temperature T is set to T_0 and the incumbent packing p , which is considered as the best packing at this step, is generated from l_0 using `pack`. At every iteration of the main loop, a neighbouring list l' is generated by swapping randomly k rectangles in the incumbent list l . The resulting list is transformed into a candidate packing p' by using `pack` and the temperature is updated by `calcTemp(i, T)`. This function performs the classical geometric reduction $T := \alpha T$ every N_t iterations. If the value of the candidate packing is better than the incumbent one (see line 9), the latter is replaced by the former and a new test is made to check if the candidate packing improves the best packing found so far in which case the best list and packing are updated. In case the value of the candidate packing is not better than the incumbent one, it is accepted only if it satisfies the well known Boltzmann acceptance rule (see line 14). The main loop is performed until a stopping criterion is reached. In our case, this refers to the maximum number of iterations which is $N_t \lceil \frac{\ln T_f - \ln T_0}{\ln \alpha} \rceil$ or it_{\max} iterations without any improvement made to the objective function whichever comes first.

Algorithm 4: Simulated Annealing (SA)

Input : reals T_0, T_f, α, N_t ; integers k, it_{\max} , list l_0

Output: packing p_{best} .

```

1  $T := T_0$ ;
2  $l_{best} := l := l_0$ ;
3  $p_{best} := p := \text{pack}(l)$ ;
4  $i := 0$ ;
5 repeat
6    $l' := \text{shake}(l, k)$ ;
7    $p' := \text{pack}(l')$ ;
8    $T := \text{calcTemp}(i, T)$ ;
9   if  $f(p') > f(p)$  then
10     $l := l', p := p'$ ;
11    if  $f(p') > f(p_{best})$  then
12       $l_{best} := l', p_{best} := p'$ ;
13    end if
14  else if  $e^{\frac{f(p') - f(p)}{T}} > \text{random}[0, 1]$  then
15     $l := l', p := p'$ ;
16  end if
17   $i := i + 1$ ;
18 until stopping criterion;
19 return  $p_{best}$ 

```

3.4 Speed-up mechanisms

In heuristic search design, the number of iterations is usually very large and the solution configurations from one iteration to the next have a lot of in common. It is therefore crucial to avoid recomputing already processed parts of earlier configurations. Exploiting this aspect can reduce the computational

time considerably turning the search into a much faster one. Some of these powerful techniques are discussed in Salhi (2017, Chapter 6).

Let i be the lowest index such that l_i is relocated by the shake procedure at line 7 in VNS and line 6 in SA. When the pack procedure is called at line 8 in VNS, if the initial configuration of the packing p_{best} is preserved then the placement of the rectangles l_1 to l_{i-1} in p is the same as the one in p_{best} . Exploiting this idea for VNS and SA can considerably speed-up the search though the final solution quality may be deteriorated as a variation of the initial configuration is less likely to occur.

a) Case of VNS - We propose a variant of VNS called xVNS where the lines 7-8 are replaced by the two following lines:

$$\begin{aligned} \langle l, i \rangle &:= \text{xShake}(l_{best}, k); \\ p &:= \text{xPack}(l, i, p_{best}); \end{aligned}$$

The procedure $\text{xShake}(l_{best}, k)$ returns a copy l of l_{best} where k pairs of elements have been swapped randomly, i being the smallest index of an element relocated in l . This function returns both l and i . The procedure xPack is an extension of the procedure pack which copies from p_{best} into p the placements of the packed rectangles in the list $l_1 \dots l_{i-1}$, builds the border surrounding these rectangles by means of Algorithm 2 then pack the remaining rectangles in $l_i \dots l_n$ as performed in the while-loop (lines 5 to 11) of the procedure pack .

b) Case of SA - Similarly to case (a), we propose a variant of SA called xSA where the lines 6-7 are replaced by the two following lines:

$$\begin{aligned} \langle l', i \rangle &:= \text{xShake}(l, k); \\ p' &:= \text{xPack}(l', i, p); \end{aligned}$$

3.5 Some observations

Among the existing variants of VNS (see Hansen et al. (2010)), Algorithm 3 is closest to the *Reduced* VNS (RVNS) as it consists of a shaking procedure but no local search. From this point of view, the search space explored in Algorithm 3 is the one of permutations whereas the pack procedure plays the role of an *evaluation procedure* used to assess the ability of a given permutation in producing a good packing. The composite function $f(\text{pack}(l))$ used at line 9 of Algorithm 3 is then a non-constant time *cost function* (see Theorem 1) used during the *neighbourhood change* step of RVNS.

The idea of exploring an alternative search space instead of exploring directly the solution search space is present in the literature. By definition, evolutionary algorithms make intensive use of encoding procedure to convert solutions (phenotypes) into representations (genotypes) (Talbi, 2009, Section 1.4.1).

In the context of vehicle routing, Prins et al. (2014) reviews more than 70 references related to the class of so-called "order-first split-second" methods. These methods oscillate between two search spaces by means of a *decoding* and *encoding* procedures called `Split` and `Split-1` respectively. The encoding procedure permits to convert a routing into a permutation whereas the decoding procedure determines a routing by optimally partitioning a permutation.

Our methodology follows a simpler architecture as it uses only a perturbation procedure, namely shake, to explore the permutation search space, and a decoding procedure, namely pack, used to convert a permutation into a packing. The resulting algorithms (VNS and SA) are still trajectorial metaheuristics with the nuance that the path they describe is inscribed in the alternative search space; not in the solution search space. The search is thus focused on permutations and, in some sense, the packings visited during the search are only "evaluations" of the permutations encountered in the alternative search space. Based on the quality of the packings resulting from the visited permutations, the search is guided towards the "best" permutation; the one that results in a packing of best known quality when it is given as input to pack.

We demonstrate in the next section that our simplistic approach reveals highly effective and efficient when tested on benchmark instances.

4 Computational experiments

The experiments are conducted on an Intel Core i3-2330M @ 2.20 GHz with 3.9 Gbytes RAM running Linux and the algorithms are coded in Java. They consist of three main parts. First, we assess the performance of both the SA and the VNS algorithms with the algorithm of López and Beasley (2018b) (denoted FSS) on the dataset they proposed and which is available from López and Beasley (2018a). To the best of our knowledge, this is the only publicly available dataset for this packing problem. Second, we assess the statistical significance of the results obtained in the first part using the Friedman test. Finally, we generate larger instances to be used for a comparative study between SA, VNS and their respective accelerated variants xSA and xVNS.

4.1 Datasets and parameter calibration

4.1.1 Existing dataset

The dataset proposed by López and Beasley (2018b) consists of instances involving $n = 10, 20, 30$ rectangles or squares with randomly generated dimensions. For each instance, three different container radii are considered. They are calculated based on a formula involving a fraction ρ of the total area of the objects to pack. For all instances, both the maximisation of the total area packed and the maximisation

of the number of objects packed are considered. For instances involving rectangles, two supplementary cases are considered which are the possibility to rotate or not the rectangles. In total, this dataset consists of 54 instances.

4.1.2 Newly constructed large dataset

In addition and following the generating scheme proposed López and Beasley (2018b), we generated a new set of larger instances involving $n = 100, 150, 200$ rectangles/squares. Their dimensions were pseudo-randomly generated (to two decimal places) in the interval $[1, 5]$. For each instance, three container radii R are set so that the container's area represents $\frac{1}{3}$, $\frac{1}{2}$ and $\frac{2}{3}$ of the total area of the n objects to pack. This new dataset alongside all the new best known solutions obtained in this paper are made publicly available at the CLHO (2019) website. These informations can be useful for benchmarking purposes.

4.1.3 Parameter calibration for VNS & SA

To calibrate our metaheuristics, we carried out experiments on a pseudo-randomly generated instance which consists of 100 rectangles. For VNS, the three parameters k_{\max} , it_{\max} and l_0 need to be set, while, for SA there are seven parameters, namely, T_0 , T_f , α , N_t , k , it_{\max} and l_0 . Preliminary results showed clearly that when the objective is to maximise the area (resp. maximise the number of objects) packed, l_0 should be sorted in decreasing (resp. increasing) order of the object areas.

In the case of VNS, three values were tested for k (3,5 and 7) and two for it_{\max} (5×10^2 and 10^3). Five runs were performed for every combination. The best compromise in terms of solution quality and computation time turned out to be $k = 3$ and $it_{\max} = 5 \times 10^2$.

In the case of SA, the number of possibilities was much larger so we limited the number of values to two for all parameters except the number of runs to three per combination. The tested values were as follows: T_0 (10,20), T_f (0,01, 0.1), α (0.95, 0.98), N_t (5, 10), k (3, 7) and it_{\max} (10^3 and 10^4). The selected combination uses $T_0 = 10$, $T_f = 0.1$, $\alpha = 0.98$, $N_t = 5$, $k = 3$ and $it_{\max} = 10^4$.

Complementary data relevant to this preliminary experimentation are presented in Appendix A.

4.2 Results on the López and Beasley's dataset (2018a)

We first present the overall results and then analyse their statistical significance.

4.2.1 Overall results

Tables 2-4 and Tables 5-7 present the results for the cases where the objective is the maximisation of the number of objects packed and the maximisation of the area packed respectively. All Tables 2-7 follow

the same structure. The first two columns identify the instance. The next two columns report the results (solution and computation time) presented by López and Beasley (2018b) for FSS. The columns labelled $Val(I_{inc})$ and $Val(I_{dec})$ indicate the results obtained by applying pack on a list of the rectangles sorted in increasing and decreasing order of their areas respectively. Columns labelled SA and VNS present the results obtained by applying five runs of SA and VNS algorithms respectively. Both SA and VNS algorithms make an extensive use of the shake procedure which requires a pseudo-random generator when implemented. For every algorithm and for every instance, different runs are conducted in order to vary the initial random seed used in the search. This is performed to assess the robustness of SA and VNS. Columns labelled Best and Avg indicate the best solution value found and the average solution value over all the runs performed. Columns labelled T(s) correspond to the total time (in seconds) spent by the algorithms over all the runs performed. The computation time for a single run of the procedure pack is negligible. The last column (BKS) corresponds to the best known solution value including the ones found in this paper. Bold and underlined-bold values indicate a best solution value and a new best solution value respectively. The last rows provide average measures over the instances and the number of best values obtained. Average measures include the average difference calculated when the problem is to maximise the number of objects packed where the difference of a value Val is $Diff = Best - Val$, the average deviation calculated when the problem is to maximise the area packed where the deviation of a value Val is $Dev(\%) = 100 \times \frac{Best-Val}{Best}$, and the average computation time in seconds.

Table 2
Results in the case of maximising the number of rectangles packed without rotation

n	ρ	FSS		pack		SA			VNS			BKS
		Best	T(s)	$Val(I_{inc})$	$Val(I_{dec})$	Best	Avg	T(s)	Best	Avg	T(s)	
10	1	5	3058	4	2	5	5.00	1	5	4.40	1	5
		6	2862	5	4	6	6.00	2	6	6.00	1	6
		7	2966	6	6	7	7.00	2	7	7.00	1	7
20	2	7	6278	6	5	7	7.00	5	7	7.00	2	7
		10	4530	8	7	10	10.00	5	10	10.00	2	10
		11	7311	11	9	13	13.00	6	12	12.00	2	13
30	3	13	11514	13	7	14	14.00	10	14	13.80	7	14
		16	10029	15	10	18	18.00	11	17	17.00	5	18
		19	6966	17	15	21	21.00	13	21	20.20	8	21
Avg. diff. or T(s)		0.78	6168	1.78	4.00	0.00	0.00	6	0.22	0.40	3	
# Best		5		0	0	9	9		7	4		

In terms of *solution quality*, when the objective is to maximise the number of objects (area resp.) packed, the procedure pack provides the best results when the list is sorted in increasing (decreasing resp.) order of the rectangles' areas. It is clear that one single run of pack is outperformed by SA and VNS. When the objective is to maximise the number of objects packed, SA dominates both VNS and FSS while remaining very stable (the average difference is less or equal to 0.09). VNS is superior to FSS

Table 3

Results in the case of maximising the number of rectangles packed with rotation

n	ρ	FSS		pack		SA			VNS			BKS
		Best	T(s)	Val(l_{inc})	Val(l_{dec})	Best	Avg	T(s)	Best	Avg	T(s)	
10	1	5	9836	4	2	5	5.00	2	5	4.80	2	5
		6	10332	5	4	6	6.00	3	6	6.00	1	6
		7	12409	6	6	7	7.00	3	7	7.00	1	7
20	2	8	22759	6	5	8	7.60	9	8	7.80	4	8
		10	30682	8	7	10	10.00	10	10	10.00	4	10
		12	30823	11	9	13	13.00	11	12	12.00	4	13
30	3	14	49724	13	6	14	14.00	17	14	14.00	10	14
		17	45857	15	10	18	18.00	19	18	17.20	11	18
		20	57427	18	15	21	21.00	20	20	20.00	10	21
Avg. diff. or T(s)		0.33	29983	1.78	4.22	0.00	0.04	10	0.22	0.36	5	
# Best		6		0	0	9	8		7	4		

Table 4

Results in the case of maximising the number of squares packed

n	ρ	FSS		pack		SA			VNS			BKS
		Best	T(s)	Val(l_{inc})	Val(l_{dec})	Best	Avg	T(s)	Best	Avg	T(s)	
10	1	4	1123	3	1	4	4.00	1	4	4.00	1	4
		5	2761	4	4	5	5.00	1	5	5.00	1	5
		6	2275	5	4	6	6.00	2	6	6.00	1	6
20	2	11	5450	9	10	11	11.00	5	10	10.00	2	11
		12	6465	11	11	13	12.20	5	12	11.80	3	13
		14	6995	12	13	14	14.00	6	14	13.20	3	14
30	3	16	13552	14	13	16	16.00	10	15	15.00	6	16
		20	13457	18	13	20	20.00	12	20	19.60	8	20
		23	10427	21	17	23	23.00	13	22	22.00	7	23
Avg. diff. or T(s)		0.11	6945	1.67	2.89	0.00	0.09	6	0.44	0.60	3	
# Best		8		0	0	9	8		5	3		

when dealing with rectangle packing but FSS is able to provide better solutions when the problem is to pack squares. When the objective is to maximise the area packed, both SA and VNS dominate FSS but SA and VNS are not dominating each other. In total, 32 new best solutions out of 54 are found.

In terms of *computational effort*, SA and VNS are clearly faster than FSS though the computer used with FSS, the Intel Core i5-2400S @ 2.50 GHz, is ranked better than our computer in terms of CPU performance (PassMark, 2020). However, it is important to mention that FSS is a formulation search space approach which relies heavily on a third-party solver which slows the whole process.

We can conclude that, for this implementation of FSS and on this particular dataset, both VNS and SA are able, in the majority of cases, to find better solutions than FSS while being relatively much faster. There is however no obvious conclusion that can be made between SA and VNS.

Table 5

Results in the case of maximising the area of rectangles packed without rotation

n	ρ	FSS		pack		SA			VNS			BKS
		Best	T(s)	Val(l_{inc})	Val(l_{dec})	Best	Avg	T(s)	Best	Avg	T(s)	
10	1 2 3	18.4441	3292	11.35	17.8992	18.4441	18.4441	5	18.4441	18.3351	1	18.4441
		28.9390	2992	18.4441	25.3571	28.9390	28.9390	2	28.9390	28.9390	1	28.9390
		37.6878	4754	25.3831	36.5982	<u>39.4588</u>	38.9214	2	38.7870	38.7870	1	<u>39.4588</u>
20	1 2 3	43.3885	7227	31.5072	41.7542	<u>45.1727</u>	45.0616	5	45.1567	44.7210	2	<u>45.1727</u>
		63.1643	9791	42.7353	64.5214	<u>69.9263</u>	68.5124	6	68.8314	67.4388	3	<u>69.9263</u>
		84.4446	10601	70.1397	87.5713	<u>93.0556</u>	91.3516	6	91.6368	90.6159	3	<u>93.0556</u>
30	1 2 3	60.3570	14011	48.1206	59.8421	<u>65.2856</u>	64.5676	10	64.4689	63.7051	5	<u>65.2856</u>
		85.2113	19786	63.6659	95.3117	100.1839	98.4151	12	<u>102.1196</u>	99.3385	8	<u>102.1196</u>
		103.4802	19470	80.064	131.1013	135.9217	134.3052	13	<u>137.4149</u>	135.2876	7	<u>137.4149</u>
Avg. dev (%) or T(s)		8.46	10214	34.43	7.04	0.33	1.38	7	0.68	1.80	3	
# Best		2		0	0	7	2		4	1		

Table 6

Results in the case of maximising the area of rectangles packed with rotation

n	ρ	FSS		pack		SA			VNS			BKS
		Best	T(s)	Val(l_{inc})	Val(l_{dec})	Best	Avg	T(s)	Best	Avg	T(s)	
10	1 2 3	19.6702	8771	11.35	17.8992	19.6702	19.6702	3	19.6702	19.2633	1	19.6702
		29.5041	16093	18.4441	27.1281	<u>30.8746</u>	29.7782	3	<u>30.8746</u>	29.7208	1	<u>30.8746</u>
		37.9687	15526	25.3831	39.999	<u>41.1612</u>	40.8778	3	40.9063	40.3619	1	<u>41.1612</u>
20	1 2 3	43.6850	50558	31.2769	44.7796	<u>45.4200</u>	45.3034	9	<u>45.4200</u>	45.3303	5	<u>45.4200</u>
		63.5279	50013	42.7353	65.647	<u>71.2331</u>	69.7546	10	70.8221	68.7228	7	<u>71.2331</u>
		84.7008	63350	66.9917	87.7261	95.1127	94.0614	10	<u>95.2162</u>	93.4912	5	<u>95.2162</u>
30	1 2 3	57.9328	69565	48.1206	59.7641	<u>66.6947</u>	66.1056	18	66.6329	65.1248	14	<u>66.6947</u>
		84.3715	82101	63.4074	95.4395	<u>100.4537</u>	99.1734	20	100.3020	99.2145	11	<u>100.4537</u>
		110.3253	39564	88.8787	131.2393	135.2908	133.4534	20	<u>137.5277</u>	136.1031	12	<u>137.5277</u>
Avg. dev (%) or T(s)		9.64	43949	35.75	6.78	0.19	1.43	11	0.16	1.99	6	
# Best		1		0	0	7	1		5			

4.2.2 Statistical significance

The Friedman test is a non-parametric statistical procedure for comparing more than two samples that are related. When it leads to significant results, it rejects the hypothesis that there is no difference between the samples studied. Corder and Foreman (2009) provides a comprehensive introduction to this topic.

In our case, the hypotheses are the following:

H_0 (null hypothesis) : "There is no difference between the results returned by FSS, SA and VNS"

H_a (alternative hypothesis): "There is a difference between the results returned by FSS, SA and VNS"

For every case, the number of rows is 9 (instances) and the number of columns is 3 (algorithms). The Friedman test starts by ranking the results returned by the various algorithms from 1 (for the smallest value) to 3 (for the highest one). In case of ties, equal values receive the average of their values had they

Table 7
Results in the case of maximising the area of squares packed

n	ρ	FSS		pack		SA			VNS			BKS
		Best	T(s)	Val(l_{inc})	Val(l_{dec})	Best	Avg	T(s)	Best	Avg	T(s)	
10	1	22.9485	2762	11.6589	22.4676	23.9878	23.9878	1	23.9878	23.3642	0	23.9878
		36.7126	3402	19.3318	34.1265	37.7471	37.3333	2	37.7471	37.3333	1	37.7471
		51.7583	4593	30.2879	42.9585	52.7555	52.7555	2	52.7555	51.9923	1	52.7555
20	1	54.1054	9412	31.0113	56.9714	63.7430	62.9581	5	63.7523	62.7630	4	63.7523
		85.2107	11304	57.5113	87.1782	94.7706	94.5368	6	94.7706	94.0801	3	94.7706
		109.8363	7636	70.9802	117.5935	126.7480	126.0535	6	132.4100	125.8077	5	132.4100
30	1	54.4941	16629	43.2045	62.4524	63.1167	62.8116	11	63.9965	63.4017	4	63.9965
		77.5814	14808	65.9387	95.1851	97.2366	95.8948	12	98.1142	97.1655	12	98.1142
		103.0963	15145	87.758	127.511	129.6979	128.3066	14	131.5472	129.5725	6	131.5472
Avg. dev (%) or T(s)		12.07	9521	42.05	8.09	0.89	1.55	7	0.00	1.76	4	
# Best		0		0	0	4	1		9			

been different. The Friedman test statistic, denoted by F_r , is then calculated according to the following formula:

$$F_r = \frac{n(k-1) \left[\sum_{i=1}^k \frac{R_i^2}{n} - C_f \right]}{\sum r_{ij}^2 - C_f}$$

where n is the number of rows, k is the number of columns, R_i is the sum of the ranks from column i , C_f is a ties correction factor equal to $(1/4)nk(k+1)^2$; and r_{ij} is the rank corresponding to the row i and column j .







In Table 8, the first three columns describe the case considered, the fourth column corresponds to the table taken from this paper as a basis for the calculation of F_r and the last column is the calculated value of F_r in the corresponding case. We consider an α value of 0.05. The critical value associated with $k = 3$, $n = 9$ and $\alpha = 0.05$ is 6.222. If F_r is greater than 6.222, the null hypothesis is rejected in favour of the alternative hypothesis. Otherwise, there is not enough evidence to reject the null hypothesis.

In Table 8, bold values correspond to values of F_r greater than 6.222. The values of F_r are greater than the critical value in five cases among six which is sufficient to reject the null hypothesis in those cases.



At this level, we conclude that in the majority of cases, we have enough statistical evidence to state that there is a difference between the results returned by the three algorithms. Together with the conclusion drawn at the previous section, we can say that the improvement made by SA and VNS over FSS is statistically significant in the majority of cases considered in the López and Beasley's dataset (2018a).

Table 8

Friedman test statistic values for the different cases

Objective function ¹	Objects' shapes ²	Rotation angle ³	Base Table	F_r
#		0	2	7.43
#		$\frac{\pi}{2}$	3	4.67
#		0	4	6.50
A		0	5	11.14
A		$\frac{\pi}{2}$	6	13.07
A		0	7	16.75

¹ #: Maximise the number of objects packed, A: Maximise the area packed.

² : Rectangular objects, : Square objects.

³ 0: No rotation allowed, $\frac{\pi}{2}$: Rotation by an angle of $\frac{\pi}{2}$ allowed.

4.3 Results on the newly constructed dataset

The four metaheuristics SA, xSA, VNS and xVNS are tested on the newly generated dataset (see Section 4.1.2). Detailed results are presented in Appendix B. Table 9 describes a summary of the results. The first three columns identify the type of instance addressed. For every metaheuristic, the columns "Dev. (%)" or "Diff." and "T(s)" correspond to the average deviation or difference obtained in the considered type of instance and the average computation time (in seconds) respectively.

In terms of solution quality, the variable neighbourhood search methodologies produce the best average deviations in all cases except when maximising the number of squares packed where SA performs better. Moreover, VNS and xVNS are able together to generate 44 best solutions including ties among 54 possible ones (more than 80%) against only 18 best solutions including ties for SA and xSA (almost 33%). Regarding the computation time, all algorithms are able to tackle these large instances using a reasonable amount of time. The accelerated versions xSA and xVNS are in average 2.71 and 2.91 times faster than their counterparts SA and VNS though their average effectiveness is slightly affected as one may have expected (see Section 3.4). xVNS is the best performer in obtaining the best trade-off between solution quality and computation time.

Two illustrative examples

As an illustration, we present in Figure 7 the new best solution found by VNS on the instance `rect3` where the objective is to maximise the area packed without rotation. In that case, FSS produced a solution with value 103.4802 in 19470 seconds whereas VNS provides a solution with value 137.4149 in 7 seconds only. Also, we present in Figure 8 the best solution found by SA in `R200` where the objective is to maximise the number of rectangles packed with rotation. In that case, SA is able to pack 148 rectangles in 1519 seconds.

Table 9

Summary results of the four metaheuristics on the newly generated dataset

Instance type			SA		xSA		VNS		xVNS		
Objects' shape	Objective function	Rotation angle	Dev.(%) or Diff.	T(s)	Dev.(%) or Diff.	T(s)	Dev.(%) or Diff.	T(s)	Dev.(%) or Diff.	T(s)	
□	A	0	1.53	403	1.63	149	0.12	324	0.16	97	
□	A	$\frac{\pi}{2}$	1.40	659	1.40	242	0.09	740	0.07	186	
□	A	-	1.03	446	0.95	153	0.04	391	0.12	102	
□	#	0	1.00	441	1.00	172	0.33	322	0.22	144	
□	#	$\frac{\pi}{2}$	0.89	692	1.89	254	0.33	601	0.89	221	
□	#	-	0.33	464	0.56	176	2.00	399	1.89	204	
Average time (s)			517		191		463		159		
Average deviation (%)			1.32		1.32		0.08		0.12		
Average difference			0.74		1.15		0.89		1.00		
Number of best solutions				18				44			

5 Conclusion & Suggestions

In this paper, we tackled the problem of packing unequal rectangles into a circular container of fixed size with the objective of maximising the area or the number of objects packed. Given the complexity of the solution structure of the problem, we reduced our search to the space of lists of rectangles. This was possible due to a decoding procedure called *pack* that we introduced and which permits to convert an ordered list of rectangles into a feasible packing for the problem. This procedure which receives initially an ordered list of the objects to pack, returns a powerful packing by means of a new data structure we introduced called "border", a number of initial configurations for the packing of the first objects in the list and several powerful moves that enable an efficient packing of new objects, the whole taking advantage from the power of geometry to provide simple analytical forms. The extensions to the *pack* procedure we introduced widened the applications of our methodology by enabling the rotation of rectangles and forbidding connected regions in the container. As the call of *pack* on one single list is a constructive heuristic, we integrate *pack* into two metaheuristic schemes, namely, variable neighbourhood search (VNS) and simulated annealing (SA). Also, exploiting previous placements of some items in the search enabled us to provide two accelerated variants of SA and VNS denoted xSA and xVNS respectively. Experiments conducted on benchmark instances showed that our methodology is superior to the state-of-the-art method and that is able to provide 32 new best solutions out of 54 (almost 60% of the total number of instances) while requiring less computational effort. Additional experiments carried on larger instances which we constructed and also made publicly available showed that, in terms of solution quality, the VNS methodologies proposed perform better than the SA ones in most of the cases except when maximising the number of squares packed where SA does better. In terms of computation time, the

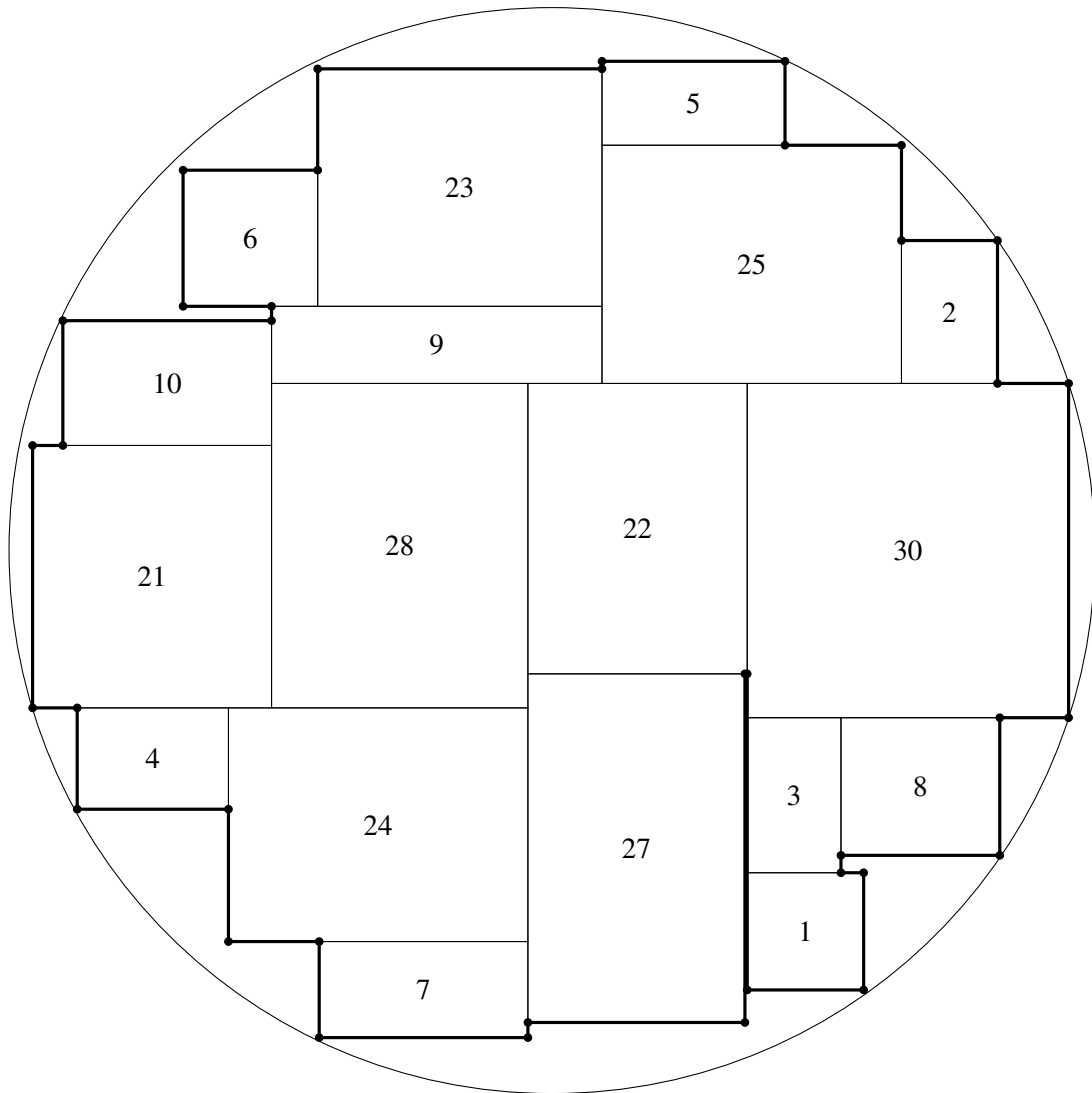


Fig. 7. Maximising the area of rectangles packed without rotation, $n = 30$, $\rho = \frac{2}{3}$.

four metaheuristics proposed are able to tackle large instances in a reasonable amount of time. The best trade-off between solution quality and computation time is achieved by xVNS.

Among the many interesting perspectives offered by this study, we mention the possibility to extend our methodology to tackle other packing problems involving more sophisticated forms of containers and forbidden regions. The metaheuristics proposed in this paper could be easily extended by hybridising them with large neighbourhood search making them even more powerful. The exciting area of adaptive or deep learning could be explored within these metaheuristics to efficiently address these classes of packing problems and other complex global optimisation problems.

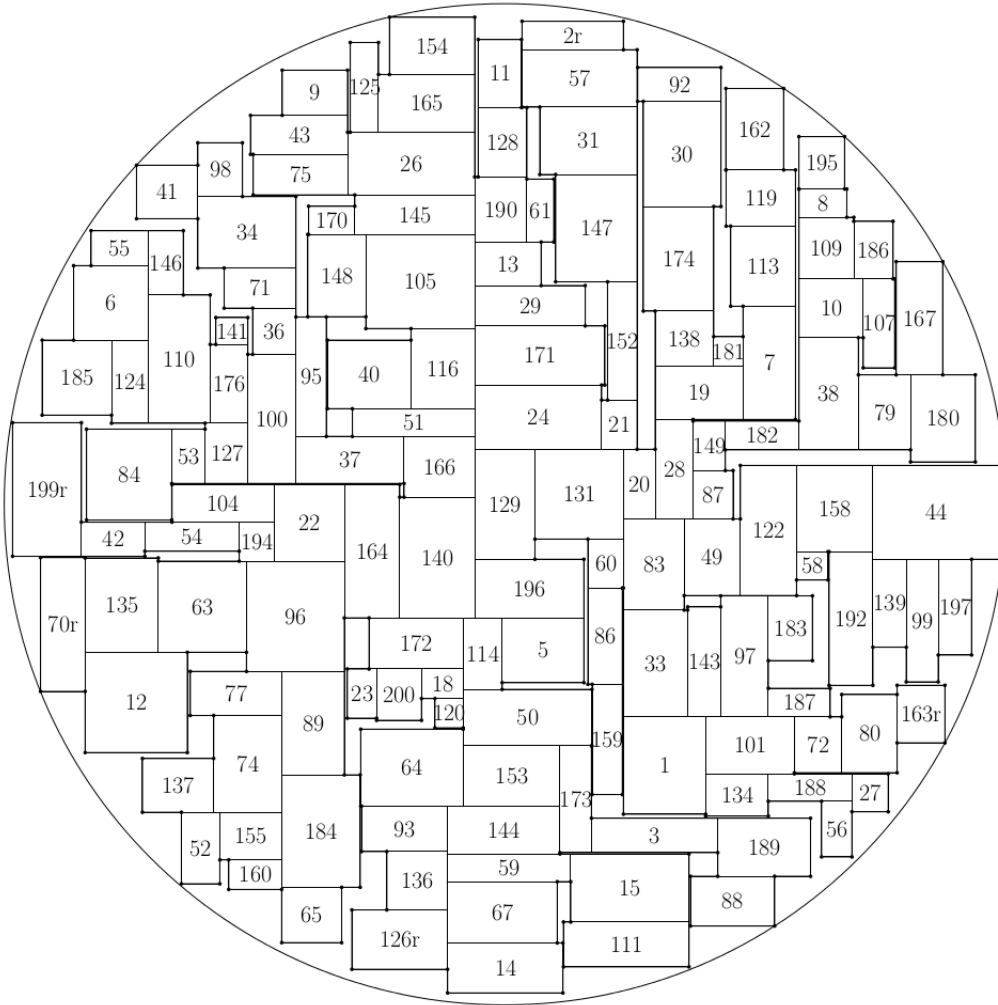


Fig. 8. Maximising the number of rectangles packed with rotation, $n = 200$, $\rho = \frac{2}{3}$

Acknowledgement

The authors would like to thank the referees for their constructive comments that improved the presentation as well as the content of the paper.

References

- Birgin, E. G. (2016). Applications of nonlinear programming to packing problems. In *Applications+ Practical Conceptualization+ Mathematics= fruitful Innovation*, pages 31–39. Springer.
- Birgin, E. G. and Lobato, R. D. (2010). Orthogonal packing of identical rectangles within isotropic convex regions. *Computers & Industrial Engineering*, 59:595–602.
- Birgin, E. G., Martínez, J. M., Mascarenhas, W. F., and Ronconi, D. P. (2006a). Method of sentinels for packing items within arbitrary convex regions. *Journal of the Operational Research Society*, 57:735–746.
- Birgin, E. G., Martínez, J. M., Nishihara, F. H., and Ronconi, D. P. (2006b). Orthogonal packing of

- rectangular items within arbitrary convex regions by nonlinear optimization. *Computers & Operations Research*, 33:3535–3548.
- Buchberger, B. (1976). A theoretical basis for the reduction of polynomials to canonical forms. *ACM SIGSAM Bulletin*, 10:19–29.
- Cassioli, A. and Locatelli, M. (2011). A heuristic approach for packing identical rectangles in convex regions. *Computers & Operations Research*, 38:1342–1350.
- CLHO (2019). Centre for Logistics & Heuristic Optimisation, University of Kent, Canterbury, UK. <https://research.kent.ac.uk/clho/>.
- Corder, G. W. and Foreman, D. I. (2009). *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*. John Wiley & Sons.
- Demaine, E. D., Fekete, S. P., and Lang, R. J. (2010). Circle packing for origami design is hard. arXiv:1008.1224.
- Dowsland, K. A. (1993). Some experiments with simulated annealing techniques for packing problems. *European Journal of Operational Research*, 68:389–399.
- Fakoor, M., Zadeh, P. M., and Eskandari, H. M. (2017). Developing an optimal layout design of a satellite system by considering natural frequency and attitude control constraints. *Aerospace Science and Technology*, 71:172–188.
- Feng, E., Wang, X., Wang, X., and Teng, H.-F. (1999). An algorithm of global optimization for solving layout problems. *European Journal of Operational Research*, 114:430–436.
- Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.
- Gomes, A. M. and Oliveira, J. F. (2006). Solving irregular strip packing problems by hybridising simulated annealing and linear programming. *European Journal of Operational Research*, 171(3):811–829.
- Hansen, P., Mladenović, N., Brimberg, J., and Moreno Pérez, J. A. (2019). Variable neighborhood search. In Gendreau, M. and Potvin, J., editors, *Handbook of metaheuristics*, pages 57–97. Springer.
- Hansen, P., Mladenović, N., and Moreno Pérez, J. A. (2010). Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175:367–407.
- Hifi, M. and Yousef, L. (2019). A local search-based method for sphere packing problems. *European Journal of Operational Research*, 274:482 – 500.
- Hinostroza, I., Pradenas, L., and Parada, V. (2013). Board cutting from logs: Optimal and heuristic approaches for the problem of packing rectangles in a circle. *International Journal of Production Economics*, 145:541–546.
- Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220:671–680.

- Leung, J. Y.-T., Tam, T. W., Wong, C. S., Young, G. H., and Chin, F. Y. L. (1990). Packing squares into a square. *Journal of Parallel and Distributed Computing*, 10:271–275.
- Leung, S. C. H., Zhang, D., and Sim, K. M. (2011). A two-stage intelligent search algorithm for the two-dimensional strip packing problem. *European Journal of Operational Research*, 215:57–69.
- Leung, T. W., Chan, C. K., and Troutt, M. D. (2003). Application of a mixed simulated annealing-genetic algorithm heuristic for the two-dimensional orthogonal packing problem. *European Journal of Operational Research*, 145:530–542.
- Li, K. and Cheng, K. H. (1994). *Interconnection Networks for High-performance Parallel Computers*, chapter Complexity of Resource Allocation and Job Scheduling Problems in Partitionable Mesh Connected Systems, pages 644–651. IEEE Computer Society Press, Los Alamitos, CA, USA.
- Li, Z., Wang, X., Tan, J., and Wang, Y. (2014). A quasiphysical and dynamic adjustment approach for packing the orthogonal unequal rectangles in a circle with a mass balance: satellite payload packing. *Mathematical Problems in Engineering*, 2014.
- Li, Z., Zeng, Y., Wang, Y., Wang, L., and Song, B. (2016). A hybrid multi-mechanism optimization approach for the payload packing design of a satellite module. *Applied Soft Computing*, 45:11–26.
- Liu, D. C. and Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45:503–528.
- Liu, Z. and Teng, H.-F. (2008). Human–computer cooperative layout design method and its application. *Computers & Industrial Engineering*, 55:735–757.
- López, C. O. and Beasley, J. E. (2011). A heuristic for the circle packing problem with a variety of containers. *European Journal of Operational Research*, 214:512–525.
- López, C. O. and Beasley, J. E. (2013). Packing unequal circles using formulation space search. *Computers & Operations Research*, 40:1276–1288.
- López, C. O. and Beasley, J. E. (2016). A formulation space search heuristic for packing unequal circles in a fixed size circular container. *European Journal of Operational Research*, 251:64–73.
- López, C. O. and Beasley, J. E. (2018a). Dataset for the packing of unequal rectangles into a circular container. <http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/> Accessed 15 July 2018.
- López, C. O. and Beasley, J. E. (2018b). Packing unequal rectangles and squares in a fixed size circular container using formulation space search. *Computers & Operations Research*, 94:106 – 117.
- Martins, T. C. and Tsuzuki, M. S. G. (2010). Simulated annealing applied to the irregular rotational placement of shapes over containers with fixed dimensions. *Expert Systems with Applications*, 37:1955–1972.
- M’Hallah, R. and Alkandari, A. (2012). Packing unit spheres into a cube using VNS. *Electronic Notes in Discrete Mathematics*, 39:201–208.

- M'Hallah, R., Alkandari, A., and Mladenović, N. (2013). Packing unit spheres into the smallest sphere using VNS and NLP. *Computers & Operations Research*, 40:603–615.
- Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24:1097 – 1100.
- Mladenović, N., Plastria, F., and Urošević, D. (2005). Reformulation descent applied to circle packing problems. *Computers & Operations Research*, 32:2419–2434.
- Mladenović, N., Plastria, F., and Urošević, D. (2007). Formulation space search for circle packing problems. In Stützle, T., Birattari, M., and Hoos, H., editors, *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics*, pages 212–216, Berlin, Heidelberg. Springer Berlin Heidelberg.
- PassMark (2020). CPU Comparison Intel i3-2330M vs Intel i5-2400S. <https://www.cpubenchmark.net/compare/Intel-i3-2330M-vs-Intel-i5-2400S/757vs794> Accessed 7 January 2020.
- Prins, C., Lacomme, P., and Prodhon, C. (2014). Order-first split-second methods for vehicle routing problems: A review. *Transportation Research Part C*, 40:179–200.
- SageMath, Inc. (2018). Cocalc collaborative computation online. <https://cocalc.com/> Accessed 27 November 2018.
- Salhi, S. (2017). *Heuristic Search: The Emerging Science of Problem Solving*. Springer.
- Soke, A. and Bingul, Z. (2006). Hybrid genetic algorithm and simulated annealing for two-dimensional non-guillotine rectangular packing problems. *Engineering Applications of Artificial Intelligence*, 19:557 – 567.
- Sun, Z.-G. and Teng, H.-F. (2003). Optimal layout design of a satellite module. *Engineering Optimization*, 35:513–529.
- Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*. John Wiley & Sons.
- Teng, H.-F., Sun, S., Liu, D., and Li, Y. (2001). Layout optimization for the objects located within a rotating vessel - a three-dimensional packing problem with behavioral constraints. *Computers & Operations Research*, 28:521–535.
- Wang, Y. and Teng, H.-F. (2009). Knowledge fusion design method: satellite module layout. *Chinese Journal of Aeronautics*, 22:32–42.
- Wäscher, G., Haußner, H., and Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183:1109 – 1130.
- Xu, Y., Xiao, R., and Amos, M. (2007). Particle swarm algorithm for weighted rectangle placement. In *Third International Conference on Natural Computation, 2007 (ICNC 2007)*, pages 728–732. IEEE.
- Xu, Y.-C., Dong, F.-M., Liu, Y., and Xiao, R.-B. (2010). Genetic algorithm for rectangle layout optimization with equilibrium constraints. *Pattern Recognition and Artificial Intelligence*, 23:794–801.

- Zeng, Z., Yu, X., He, K., and Fu, Z. (2018). Adaptive tabu search and variable neighborhood descent for packing unequal circles into a square. *Applied Soft Computing*, 65:196–213.
- Zeng, Z., Yu, X., He, K., Huang, W., and Fu, Z. (2016). Iterated tabu search and variable neighborhood descent for packing unequal circles into a circular container. *European Journal of Operational Research*, 250:615–627.
- Zhang, B., Teng, H.-F., and Shi, Y.-J. (2008). Layout optimization of satellite module using soft computing techniques. *Applied Soft Computing*, 8:507–521.
- Zhao, F., Li, G., Yang, C., Abraham, A., and Liu, H. (2014). A human–computer cooperative particle swarm optimization based immune algorithm for layout design. *Neurocomputing*, 132:68–78.
- Zhong, C.-Q., Xu, Z.-Z., and Teng, H.-F. (2019). Multi-module satellite component assignment and layout optimization. *Applied Soft Computing*, 75:148–161.

A Complementary data for the calibration of VNS and SA

Prior to the experiments conducted in Sections 4.2 and 4.3, we carried out a parameter calibration of our algorithms. This was conducted on the instance R100 that we generated and which is available at the CLHO (2019) website. The problem considered consists in maximising the area packed by 100 candidate rectangles into a circular container of radius 14.14 with rotation. Tables 10 and 11 present the results obtained during the calibration of VNS and SA respectively. In Table 10, the first two columns correspond to the combinations of parameter values tested while the two last columns are the average solution values and the average computing times in seconds over 5 runs respectively. The results presented in Table 11 correspond to the 15 combinations of parameters values which brought the best average solution values among 64 combinations tested. The first six columns correspond to the combinations tested while the last two columns are the average solution values and average computing times in seconds over 3 runs respectively. In both tables, results are sorted in the decreasing order of the average solution value obtained. Bold rows correspond to the selected combinations of parameters values.

Table 10

Results obtained during the calibration of VNS

k_{\max}	it_{\max}	Avg. Value	Avg. Time (s)
3	5×10^2	568.9470	51
3	10^3	568.8626	112
5	10^3	568.3557	86
5	5×10^2	567.1563	58
7	10^3	566.9445	91
7	5×10^2	564.6677	42

Table 11

Results obtained during the calibration of SA (15 best)

T_0	T_f	α	N_t	k	it_{\max}	Avg. Value	Time (s)
10	0.1	0.98	5	3	10^4	562.1693	71
10	0.1	0.98	10	3	10^4	561.2068	158
20	0.1	0.98	10	3	10^3	560.7677	50
20	0.1	0.95	5	3	10^3	560.0849	26
20	0.01	0.98	10	3	10^3	559.8920	62
20	0.1	0.95	10	3	10^3	559.3519	47
10	0.1	0.95	10	3	10^4	559.3456	62
10	0.01	0.95	5	3	10^4	559.0968	40
10	0.1	0.95	5	3	10^4	558.8763	30
10	0.01	0.98	5	3	10^4	558.8584	106
10	0.01	0.98	5	3	10^3	558.7867	57
10	0.01	0.95	10	3	10^4	558.7666	89
10	0.01	0.95	10	3	10^3	558.6839	55
10	0.1	0.95	5	7	10^3	558.6774	31
10	0.1	0.95	5	3	10^3	558.4810	27

B Detailed results on the newly generated dataset

Tables 12-17 present the results obtained by SA, xSA, VNS and xVNS on the dataset we have generated (see Section 4.1.2). The tables follow the same structure, that is, the first two columns identify the instance and the next columns present the results obtained by applying five runs of SA, xSA, VNS and xVNS respectively. The times presented in those columns correspond to the total time in seconds spent by each algorithm over the five runs for every instance. Bold values correspond to the best solution value found. The last rows provide average measures over the instances and the number of best values obtained.

Table 12

Results in the case of maximising the number of rectangles packed without rotation

n	ρ	SA			xSA			VNS			xVNS			BKS
		Best	Avg.	T(s)	Best	Avg.	T(s)	Best	Avg.	T(s)	Best	Avg.	T(s)	
100	1 3 3	45	44.20	101	45	43.80	33	45	44.00	66	44	43.20	29	45
		59	57.60	141	58	57.60	49	58	57.60	85	59	58.00	43	59
		70	69.80	181	70	69.80	64	69	69.00	95	70	69.60	51	70
150	1 3 3	70	68.80	272	69	68.20	96	72	71.20	217	72	71.20	84	72
		91	90.40	380	91	90.40	156	92	91.80	258	91	90.60	118	92
		110	108.80	501	110	108.60	208	110	108.40	286	110	108.40	175	110
200	1 3 3	95	94.00	534	96	94.40	212	99	97.00	468	99	97.40	221	99
		123	121.80	818	124	121.80	317	124	123.20	737	124	122.80	244	124
		146	144.80	1038	146	144.40	411	146	145.20	683	147	145.20	334	147
Avg. diff. or T(s)		1.00	1.98	441	1.00	2.11	172	0.33	1.18	322	0.22	1.29	144	
# Best		4			4			6			7			

Table 13

Results in the case of maximising the number of rectangles packed with rotation

n	ρ	SA			xSA			VNS			xVNS			BKS
		Best	Avg.	T(s)	Best	Avg.	T(s)	Best	Avg.	T(s)	Best	Avg.	T(s)	
100	1 3 3	46	45.2	185	45	43.8	56	45	44.4	110	44	43.8	44	46
		59	58.0	239	59	58.2	79	59	57.8	164	59	58.2	69	59
		71	70.4	275	71	69.8	97	70	69.4	149	70	69.0	65	71
150	1 3 3	70	69.4	458	69	68.8	167	73	71.6	463	72	71.2	120	73
		92	90.6	611	90	90.0	229	93	91.8	379	92	91.4	175	93
		110	109.0	745	109	108.0	287	110	108.8	570	109	108.4	209	110
200	1 3 3	95	94.6	927	95	94.4	321	98	97.4	805	98	97.0	267	98
		125	122.6	1267	122	121.2	469	126	124.4	1190	125	123.8	593	126
		148	146.0	1519	147	146.4	582	147	146.4	1579	147	145.8	451	148
Avg. diff. or T(s)		0.89	2.02	692	1.89	2.60	254	0.33	1.33	601	0.89	1.71	221	
# Best		5			2			6			2			

Table 14
Results in the case of maximising the number of squares packed

n	ρ	SA			xSA			VNS			xVNS			BKS
		Best	Avg.	T(s)	Best	Avg.	T(s)	Best	Avg.	T(s)	Best	Avg.	T(s)	
100	1	53	52.20	118	52	51.60	36	52	51.40	126	52	51.80	42	53
	1	67	65.60	153	66	64.80	52	64	63.40	145	64	63.60	55	67
	2	76	75.20	176	76	75.20	66	73	72.80	143	73	72.40	53	76
150	1	84	82.80	289	86	84.20	110	84	83.80	308	84	82.80	114	86
	1	103	101.80	429	103	102.00	156	100	99.80	319	101	100.40	156	103
	2	117	115.40	511	116	115.40	191	113	112.60	393	113	112.60	140	117
200	1	107	104.60	585	107	104.60	218	108	105.60	569	108	105.80	324	108
	1	132	130.60	832	131	128.60	323	130	129.20	687	130	129.20	371	132
	2	152	151.00	1082	152	151.60	430	152	149.80	895	152	149.60	575	152
Avg. diff. or T(s)		0.33	1.64	464	0.56	1.78	176	2.00	2.84	399	1.89	2.87	204	
# Best		7		4	4		2	2		2	2			

Table 15
Results in the case of maximising the area of rectangles packed without rotation

n	ρ	SA			xSA			VNS			xVNS			BKS
		Best	Avg.	T(s)	Best	Avg.	T(s)	Best	Avg.	T(s)	Best	Avg.	T(s)	
100	1	276.2503	274.7736	96	276.2814	274.9903	36	281.5851	279.3196	61	279.3878	278.0031	15	281.5851
	2	418.9820	418.9820	143	419.3924	419.0641	48	424.7806	424.3101	95	426.3633	424.4913	32	426.3633
	3	556.3578	555.6125	192	555.9007	555.4878	64	567.0707	563.8865	156	566.9142	564.9200	38	567.0707
150	1	382.7671	382.6841	250	382.6633	382.6633	75	387.9162	386.4728	144	388.2988	386.9971	30	388.2988
	2	576.9804	576.0199	324	575.7569	575.7569	127	583.5556	581.0678	258	586.5627	582.6336	76	586.5627
	3	769.4407	765.6101	436	765.4987	763.1941	178	780.1788	777.1628	510	778.7913	774.7563	111	780.1788
200	1	497.3100	497.3100	461	497.7669	497.5240	159	503.1195	502.1400	221	502.6467	501.5603	75	503.1195
	2	745.3243	745.0537	734	745.4894	745.0867	264	758.4439	754.7137	589	755.8380	753.8649	177	758.4439
	3	994.7474	990.0431	989	992.5025	987.2142	389	1003.5017	1001.2816	887	1004.4626	1002.1543	315	1004.4626
Avg. diff. or T(s)		1.53	1.74	403	1.63	1.80	149	0.12	0.52	324	0.16	0.55	997	
# Best		0			0			5			4			

Table 16
Results in the case of maximising the area of rectangles packed with rotation

n	ρ	SA			xSA			VNS			xVNS			BKS
		Best	Avg.	T(s)	Best	Avg.	T(s)	Best	Avg.	T(s)	Best	Avg.	T(s)	
100	1	275.9862	275.9862	167	277.5970	276.3659	57	282.3465	280.7830	129	281.9974	281.3749	34	282.3465
	2	422.5904	422.5904	221	422.5904	422.5904	73	426.5573	425.9241	145	427.7879	426.5267	37	427.7879
	3	564.5293	561.1405	281	562.1463	557.6321	96	571.6369	569.1376	336	572.0070	568.7478	84	572.0070
150	1	386.3599	386.0969	463	386.0312	386.0312	133	389.6020	388.5257	222	390.3642	388.9527	87	390.3642
	2	576.2670	574.8735	577	577.8812	575.4818	207	585.5947	583.7210	539	585.3371	583.9772	130	585.5947
	3	771.9016	769.5404	752	769.8563	768.8436	287	780.7081	778.2186	1073	778.4881	777.7283	284	780.7081
200	1	500.0271	498.8451	803	498.7330	498.3393	274	505.4126	503.2490	503	504.4356	503.6252	122	505.4126
	2	748.1296	748.1296	1215	752.3383	749.1079	429	757.9592	755.5614	1464	759.6069	757.6326	271	759.6069
	3	994.3341	992.4800	1448	990.3536	989.3667	624	1008.3091	1006.3442	2247	1009.0782	1004.5593	623	1009.0782
Avg. diff. or T(s)		1.40	1.58	659	1.40	1.66	242	0.09	0.43	740	0.07	0.37	186	
# Best		0			0			4			5			

Table 17
Results in the case of maximising the area of squares packed

n	ρ	SA			xSA			VNS			xVNS			BKS
		Best	Avg.	T(s)	Best	Avg.	T(s)	Best	Avg.	T(s)	Best	Avg.	T(s)	
100	1	293.5578	293.4949	106	294.0060	293.5846	33	297.8481	296.3685	72	297.5024	296.5671	20	297.8481
	2	446.0632	443.2703	145	447.1604	443.3074	48	449.0898	447.3728	138	448.5023	446.7429	33	449.0898
	3	587.8824	586.7831	194	588.6182	587.0086	62	600.2514	596.5111	208	597.5731	594.7559	58	600.2514
150	1	469.7621	468.9567	286	469.4241	468.7430	88	472.4785	470.7997	171	472.7061	471.3670	44	472.7061
	2	707.3694	705.7661	369	706.7174	705.8500	130	714.3385	712.3815	312	712.6763	711.3250	104	714.3385
	3	935.9346	935.0466	532	943.7229	939.3353	177	948.3443	946.8239	599	948.0180	944.3807	141	948.3443
200	1	593.9045	590.6651	572	591.9197	590.9765	176	596.4252	595.3754	237	595.6435	595.0209	59	596.4252
	2	882.0002	879.1137	748	882.5373	879.6888	272	889.4874	887.5525	457	890.1031	888.5546	150	890.1031
	3	1178.8631	1176.3317	1064	1175.9192	1174.5848	391	1185.9045	1182.5038	1330	1188.5528	1184.6662	308	1188.5528
Avg. diff. or T(s)		1.03	1.29	446	0.95	1.24	153	0.04	0.50	391	0.12	0.41	102	
# Best		0			0			6			3			