An effective evolutionary algorithm for packing rectangles into a fixed size circular container

Xiangjing Lai[1], Lei Wang[2], Jin-Kao Hao*[3], and Qinghua Wu[4]

[1]School of Business, Nanjing University of Information Science and Technology, Nanjing 210044, China.
laixiangjing@gmail.com (Xiangjing Lai)
[2]Institute of Advanced Technology, Nanjing University of Posts and Telecommunications, Nanjing 210023, P.R.China.
leiwang97@163.com (Lei Wang)
[3]LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France.
jin-kao.hao@univ-angers.fr (Jin-Kao Hao)
*Corresponding author
[4]School of Management, Huazhong University of Science and Technology, 430074 Wuhan, P.R.China.
qinghuawu1005@gmail.com (Qinghua Wu)

European Journal of Operational Research
https://doi.org/10.1016/j.ejor.2025.04.044

Abstract

We study the general problem of orthogonally packing rectangles in a fixed size circular container. This is a computationally challenging combinatorial optimization problem with important real-world applications and has recently received much attention from the operations research community. We propose an effective evolutionary algorithm for four variants of the problem, which integrates an improved decoding procedure and several dedicated search operators for population initialization and new solution generation. Computational results on 108 popular benchmark instances show that the proposed algorithm advances the state of the art in practically solving these four variants of the problem by finding 53 new best solutions (26 for the variants of maximizing the area of the packed items and 27 for the variants of maximizing the number of the packed items). We perform experiments to verify the design of key algorithmic components.

Keywords: Packing; rectangle packing; circular container; evolutionary algorithm; heuristics.

1 Introduction

Rectangle packing and cutting problems are a class of popular combinatorial optimization problems with important real-world applications, whose goal is to orthogonally pack a set of given rectangles into a container without overlapping, such that some criterion related to the packed items, such as the area of the packed items, is maximized. These packing and cutting problems have been studied extensively in the literature, most of which involve the rectangular or square containers and some additional constraints, such as the two-dimensional rectangular packing problem (Chen et al., 2019; He et al., 2012; dos Reis Arruda et al., 2024; Imahori et al., 2003; Wu et al., 2002; Fırat and Alpaslan, 2020), the rectangle packing area minimization problem (He et al., 2015; Wu et al., 2016; Bortfeldt, 2013), the problem of packing rectangles into the smallest square (Martello and Monaci, 2015), the orthogonal stock cutting problems (Burke et al., 2009; Delorme et al., 2017), the unconstrained two-dimensional non-guillotine cutting problem (Wei et al., 2018), and the two-dimensional variable-sized bin packing problem with guillotine constraints (Gardeyn and Wauters, 2022). To solve efficiently these problems, a large number of algorithms have been proposed in the literature, mainly including the exact algorithms (Iori et al., 2021; Wei et al., 2018) and the heuristic algorithms (Burke et al., 2004; Wei et al., 2009, 2011).

In addition to the rectangular container, there are some works dedicated to rectangle packing problems with a non-rectangular container. For example, Birgin et al. (2006); Birgin and Lobato (2010) and Cassioli and Locatelli (2011) respectively studied the problem of orthogonally packing identical rectangles into an arbitrary convex container. Forghani et al. (2024) studied the rectangle packing problem into a non-convex container. In particular, the rectangle packing problems with a circular container have recently received a great deal of attention due to their practical applications from various domains, such as the timber industry (Hinostroza et al., 2013) and the satellite module layout design (Li et al., 2014, 2016; Liu et al., 2017; Zhong et al., 2019).

This study focuses on the problem of orthogonally packing rectangles in a fixed size circular container (OPRCC), which has many real-world applications, such as pipeline packing (Zhang et al., 2024), lumber sawing in the forestry sector (Hinostroza et al., 2013), and sheet metal cutting (Luo et al., 2024).

Most packing and cutting problems, including the OPRCC problem, are NP-hard and thus computationally challenging. For example, Leung et al. (1990) proved that deciding whether a set of squares can be packed into a fixed size square container is strongly NP-hard. Li and Cheng (1994) showed the same result for the problem with a rectangular container. 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 is NP-hard. Recently, Zhang et al. (2024) pointed out that the OPRCC problem studied in this work is also NP-hard because it contains the one-dimensional bin packing problem.

Due to its practical and theoretical significance, the OPRCC problem has been intensively studied by several research groups. First, considering the geometry of the logs as regular cylinders of known radius and the boards as rectangular parallelepipeds, Hinostroza et al. (2013) formulate the lumber sawing problem as an OPRCC problem that maximizes the area of the packed items. Then, they designed an exact algorithm and two heuristic algorithms (an ordering heuristic and a simulated annealing heuristic) for solving the OPRCC problem.

López and Beasley (2018) gave a mixed-integer nonlinear programming (MINLP) formulation for the OPRCC problem and proposed a formulation space search (FSS) method using different formulations of the problem and minimizing the objective functions by means of a continuous MINLP solver. Moreover, they studied for the first time four variants of the OPRCC problem, including two variants of maximizing the number of the packed items and two variants of maximizing the area of the packed items.

To further solve the OPRCC problem, Bouzid and Salhi (2020) designed an ingenious data structure called the border of configuration and a corresponding decoding procedure called the *pack* procedure that converts a solution with an order representation into a packing configuration. By integrating the *pack* procedure into two meta-heuristic algorithms, they proposed two heuristic algorithms for the OPRCC problem, i.e., the border-based variable neighborhood search (VNS) algorithm and the border-based simulated annealing (SA) algorithm. Computational experiments on four variants of OPRCC show that their VNS and SA algorithms are very efficient and significantly outperform the FSS algorithm (López and Beasley, 2018). Furthermore, the VNS and SA algorithms improve the best-known solutions for 32 out of 54 benchmark instances.

Based on mixed-integer linear programming (MILP) models., Silva et al. (2022) proposed two exact algorithms, i.e., a cutting plane method (CPM) and a variant called the parallel enumeration algorithm (PEA). Computational results show that these two exact algorithms perform well on the small-scale instances with $N \leq 30$ and can find the optimal solutions for instances with $N \leq 20$ within a reasonable computation time. Moreover, for the variants of maximizing the number of the packed items, the PEA algorithm outperforms the VNS and SA algorithms (Bouzid and Salhi, 2020) on larger instances. Nevertheless, for the variants of maximizing the area of the packed items, the exact algorithms perform worse than the two heuristic algorithms on the large instances.

Recently (2024), based on a popular and efficient data structure called the skyline (Allen and Burke, 2012; Burke et al., 2004), Zhang et al. (2024) proposed a variable neighborhood search algorithm (denoted by SL-VNS) for the OPRCC problem. Taking advantage of the feature that the overlap constraints between the rectangles are automatically satisfied for the place operations on the skyline, the skyline-based decoding procedure of SL-VNS performs much faster compared to other decoding procedures, such as the border-based *pack* procedure of Bouzid and Salhi (2020). Thus, by integrating this skyline-based decoding procedure and several mutation operators, the SL-VNS algorithm attains a high performance compared to previous heuristic algorithms especially for the large instances. Experimental results show that the SL-VNS algorithm further improves the best-known solution for 51 out of 54 large instances with $100 \leq N \leq 200$.

Almost at the same time with the work of Zhang et al. (2024), Luo et al. (2024) proposed a hybrid-biased genetic algorithm (HGA) for the OPRCC problem by integrating a new border-based decoding procedure, which can be regarded as a variant of the *pack* procedure of (Bouzid and Salhi, 2020), an order crossover operator, and a mutation operator. Computational results on 108 popular benchmark instances show that the HGA algorithm outperforms its reference algorithms, especially for the small instances of maximizing the area of the packed items.

The following observations can be made from the previous studies. First, discrete optimization methods based on the decoding procedure significantly outperform continuous global optimization methods such as FSS of López and Beasley (2018). Second, exact algorithms such as CPM and PEA of Silva et al. (2022) perform well only on small instances with $N \leq 20$ especially for the variants of maximizing the area of the packed items. However, for the large-scale instances, the state-of-the-art heuristic algorithms significantly outperform the existing exact algorithms. Third, the performance of the decoding procedure depends largely on the data structures used, and at present it is not yet clear which data structures are best suited for rectangle packing problems. In fact, the decoding procedures of the state-of-the-art heuristic algorithms of the OPRCC problem mainly depend on an efficient data structure, i.e., the skyline (Burke et al., 2004; Zhang et al., 2024) or the border of packing configuration (Bouzid and Salhi, 2020). However, each of these two data structures has its advantages and limitations, and so it's still difficult to distinguish which data structure is the best for the decoding procedure. For example, the skyline-based decoding procedures (Burke et al., 2004; Zhang et al., 2024; Wei et al., 2011) are very fast due to the fact that no overlapping test between rectangles is needed in the decoding process, while their main disadvantage lies in their weak adaptability to other problems with a complicated container. For the border-based decoding procedures, it is not difficult to extend them to other problems with a complicated container, since little knowledge about the container is required in the process of decoding. However, their disadvantage is that they are generally much more time-consuming than the skyline-based decoding procedures due to the fact that it is inevitable to frequently check overlaps between the rectangles.

The recent study by Luo et al. (2024) show that the hybrid genetic algorithm equipped with a proper decoding procedure can achieve a high performance for solving the OPRCC problem. Thus, it is natural to ask whether it is possible to design a more efficient evolutionary algorithm for the OPRCC problem by improving the decoding procedure, the population initialization method, the evolution operators, and others.

Motivated by these observations, we aim to improve the performance of the border-based decoding procedure by reducing the number of overlap tests between the rectangles and designing a more efficient scoring function for the decoding procedure. Second, we aim to propose a highly efficient heuristic algorithm for the OPRCC problem by taking advantage of this improved decoding procedure and the powerful hybrid evolutionary framework.

The main contributions of this work to the packing research community can be summarized as follows. First, based on the concept of the border by Bouzid and Salhi (2020), a new scoring function and a speedup strategy, we propose a new decoding procedure for the studied OPRCC problem. Due to its flexibility, the proposed decoding procedure can be adapted to other rectangle packing problems, which could help to better solve these problems. Second, based on the new decoding procedure and several crossover and mutation operators, we propose a highly efficient evolutionary algorithm for the studied OPRCC problem. In particular, compared to the FSS method (López and Beasley, 2018), which uses discrete and continuous optimization to search, the proposed algorithm is a pure discrete optimization algorithm without using a time-consuming continuous solver. The high performance of the proposed algorithm suggests that discrete methods may be a better approach than continuous global optimization methods for this type of packing problems. Third, the analysis experiment shows that several algorithmic components (e.g., the population initialization method, the speedup strategy of the decoding procedure, and the scoring function) of the algorithm play an important role in the high performance of the algorithm, and these results provide insights for designing effective evolutionary algorithms for rectangle packing problems.

The rest of the paper is organized as follows. Section 2 gives a mathematical formulation of the problem. In section 3, we describe the proposed algorithm. In Section 4, the performance of the algorithm is evaluated based on benchmark instances. In Section 5, crucial algorithmic components are analyzed and discussed to show their impacts on the performance of the algorithm. In the last section, we summarize the main results of this study and provide several perspectives for further research.

2 Description and formulation of the problems

Given a set $M = \{r_1, r_2, \ldots, r_N\}$ of N rectangular items and a fixed size circular container C, the OPRCC problem aims to select a subset of M to be packed orthogonally into the container to maximize an objective function defined on the selected items, while satisfying the non-overlapping constraint between the items and the containment constraints of items. The non-overlapping constraint requires that no two rectangles in the packing configuration overlap, and the containment constraint requires that each packed rectangle is completely contained in the container. Fig. 1 gives an illustrative example

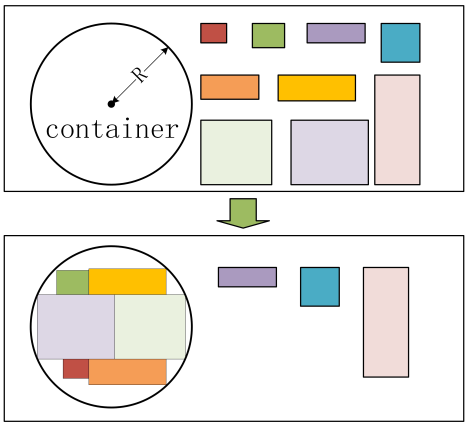


Figure 1: An illustrative example for the OPRCC.

for the OPRCC problem. There are two typical objective functions: one is to maximize the area of the packed items and the other is to maximize the number of the packed items. Thus, depending on the objective functions and whether the rectangles are allowed to be rotated orthogonally, the OPRCC problem has four variants. We study all these four variants in this work.

Given a fixed radius R of the circular container centering at the origin of the two-dimensional Cartesian coordinate system, and the dimensions (w_i, l_i) and value v_i of each item r_i $(1 \leq i \leq N)$ where w_i and l_i respectively denote the width and height of rectangular item r_i, the OPRCC problems that do not consider the orthogonal rotations of items can be formulated as a mixed-integer nonlinear programming problem (López and Beasley, 2018; Zhang et al., 2024):

$$\text{Maximize} \quad \sum_{i=1}^{N} \alpha_i v_i \tag{1}$$

$$\text{s.t.} \quad \alpha_i[\,(x_i - \frac{w_i}{2})^2 + (y_i - \frac{l_i}{2})^2] \leq R^2, 1 \leq i \leq N \tag{2}$$

$$\alpha_i[\,(x_i - \frac{w_i}{2})^2 + (y_i + \frac{l_i}{2})^2] \leq R^2, 1 \leq i \leq N \tag{3}$$

$$\alpha_i[\,(x_i + \frac{w_i}{2})^2 + (y_i - \frac{l_i}{2})^2] \leq R^2, 1 \leq i \leq N \tag{4}$$

$$\alpha_i[\,(x_i + \frac{w_i}{2})^2 + (y_i + \frac{l_i}{2})^2] \leq R^2, 1 \leq i \leq N \tag{5}$$

$$\alpha_i \alpha_j (max\{|x_i - x_j| - \frac{w_i + w_j}{2}, |y_i - y_j| - \frac{l_i + l_j}{2}\}) \geq 0, \quad 1 \leq i \neq j \leq N \tag{6}$$

where (x_i, y_i) represents the center of i-th rectangle r_i, α_i $(1 \leq i \leq N)$ is a 0–1 variable, and takes 1 if the i-th rectangle is selected from M, and 0 otherwise. The objective function (1) gives the objective value of the packed rectangles, where v_i $(1 \leq i \leq N)$ is the value of the i-th rectangle. When $v_i = w_i l_i$ or $v_i = 1$, the optimization objective is to maximize the total area of the packed items or the number of the packed items, respectively. Constraints (2)–(5) represent the containment constraints that ensure the four vertices of rectangular r_i $(1 \leq i \leq N)$ are contained in the circular container C, and constraint

(6) represents the non-overlapping constraint that ensures that there is no overlapping rectangles in the packing configuration. Thus, two variants of the problem without rotation can be consistently formulated by Eqs. (1)- (6), and the only difference between them lies in the objective function, where $v_i = w_i l_i$ corresponds to the variant of maximizing the area of packed items and $v_i = 1$ corresponds to the variant of maximizing the number of packed items.

In addition, for the two variants without rotation, we can respectively obtain a variant with rotation by allowing all rectangular items to be rotated orthogonally, while keeping other conditions unchanged. Furthermore, as mentioned by López and Beasley (2018), the model described by Eqs. (1)-(6) can be extended to the two variants with rotation by adding an orthogonally rotated copy $r_{i'}$ for each item $r_i \in M$ and a compatibility constraint of $\alpha_i + \alpha_{i'} \leq 1$ to ensure that at most one item between r_i and $r_{i'}$ is selected in the packing configuration, and thus leading to the extended models containing $2N$ items.

3 New evolutionary algorithm for the OPRCC

Evolutionary algorithms (Back et al., 1997; Martí et al., 2018) are a category of population-based search algorithms that mimic the natural evolution process by means of the evolution techniques, such as the inheritance, mutation, selection and crossover. The general procedure of an evolutionary algorithm can be briefly described as follows (see Algorithm 1). Starting from an initial population, the algorithm performs a number of generations until a stopping criterion is met. At each generation, one or more individuals are selected from the population for reproduction. Then, the selected individuals are used to produce offspring solutions by using crossover and mutation operators. Each offspring solution is then evaluated and used to update the population.

Algorithm 1: General procedure of evolutionary algorithms

```
  Input: Problem instance
  Output: The best solution found
1 Generate an initial population
2 Evaluate the fitness of each individual in the population
3 while Stopping criterion is not met do
4     Select parent individuals from the population for reproduction
5     Generate one or more offspring solutions through crossover and mutation operators
6     Evaluate the fitness of each offspring solution
7     Update the population by the offspring solutions
8 end
```

In this study, we propose an effective evolutionary algorithm named IDEA to solve the four variants of the OPRCC problem by integrating an improved decoding procedure, several mutation and crossover operators, and a greedy population update strategy.

3.1 Main framework of the proposed algorithm

The IDEA algorithm consists of seven components, including the population initialization, the constrained insertion mutation, the constrained k-swap mutation, two crossovers, the decoding procedure, and the population update strategy. The pseudo-code is given in Algorithm 2, where POP and S^* denote the current population and the best solution found so far, respectively. IDEA starts with the generation of an initial population by using the methods given in Section 3.2 (line 1, Algorithm 2).

Algorithm 2: Main framework of the evolutionary algorithm (IDEA)

```
Input: A set of rectangular items ($\{r_1, r_2, \ldots, r_N\}$), maximum time limit ($t_{max}$), maximum number of
       pairs of items ($k_{max}$) that can be swapped, parameters ($\alpha$, $\beta$ and $\gamma$)
Output: The best solution found ($S^*$)
1  $POP \leftarrow$ Initial_Population()
2  $S^* \leftarrow argmax\{f(S) : S \in POP\}$
3  while $time() \leq t_{max}$ do
4      $rd \leftarrow rand(0, 1)$        /* assign $rd$ a random number between 0 and 1 */
5      if $rd < \alpha$ then
6          $S \leftarrow Select(POP)$
7          $k \leftarrow rand[1, k_{max}]$
8          $S \leftarrow Swap(S, k)$     /* perturb the solution $S$ by performing $k$-swap
             mutation */
9      else if $rd < \alpha + \beta$ then
10         $S \leftarrow Select(POP)$
11         $S \leftarrow Insert\_Move(S)$     /* perturb the solution $S$ by the insertion
             mutation */
12     else if $rd < \alpha + \beta + \gamma$ then
13         $\{S_i, S_j\} \leftarrow Select(POP)$
14         $S \leftarrow$ Order_Crossover($S_i, S_j$)     /* generate an offspring solution with
             the order crossover */
15     else
16         $\{S_i, S_j\} \leftarrow Select(POP)$
17         $S \leftarrow$ Prefix_Crossover($S_i, S_j$) /* generate an offspring solution with the
             prefix crossover */
18     end
19     $f(S) \leftarrow IPACK(S)$      /* evaluate the quality of offspring $S$ by the
         decoding procedure given in Algorithm 5 */
20     if $f(S) > f(S^*)$ then
21         $S^* \leftarrow S$                      /* save the best solution found */
22     end
23     $POP \leftarrow$ Pool_Updating($POP$, $S$)              /* update the population */
24 end
```

Then, it enters a 'while' loop and performs a number of generations until the maximum time limit (t_{max}) is met (lines 3–24). At each generation, one or two parent solutions are randomly selected from the population, and two mutation or two crossover operators are used to generate an offspring solution S in a probabilistic way. Specifically, the constrained insertion mutation is applied with probability of α (lines 5–8), the constrained k-swap mutation is applied with a probability of β (lines 9–11), the randomized order crossover is applied with probability of γ (lines 12–14), and the prefix crossover is applied with probability of $1 - \alpha - \beta - \gamma$ (lines 15–17). Then, the decoding procedure is applied to the offspring solution S to evaluate its quality and obtain the corresponding packing configuration (line 19). Subsequently, the offspring solution is used to update the population using a greedy strategy, i.e., the worst individual in the population is replaced by the offspring solution S if it is worse than the offspring solution in terms of the objective value (line 23). Finally, when the stopping condition (a maximum allowed time) is met, the algorithm returns the best solution found S^* and stops.

Algorithm 3: Initialization of population for the variants maximizing the area of items packed

```
Input: A set of rectangular items ($\{r_1, r_2, \ldots, r_N\}$), size of population ($P$), maximal number of pairs
       of items that can be swapped ($k_{max}$)
Output: Initial population $POP$
1 Function Initial_Population()
2 $S_0 \leftarrow Sort(\{r_1, r_2, \ldots, r_N\})$    /* sort the items in a descending order
   according to their areas to obtain a seed solution $S_0$ */
3 $POP \leftarrow \emptyset$
4 for $i \leftarrow 1$ to $P$ do
5     $k \leftarrow rand[1, k_{max}]$    /* get a random number between 1 and $k_{max}$ */
6     $S_i \leftarrow Swap(S_0, k)$    /* perturb the seed sequence $S_0$ by consecutively
       performing $k$ random swap operations */
7     $f(S_i) \leftarrow IPACK(S_i)$    /* evaluate the quality of $S_i$ by the decoding
       procedure $IPACK$ (Algorithm 5) */
8     $POP \leftarrow POP \cup \{S_i\}$
9 end
```

Algorithm 4: Population initialization for the variants maximizing the number of items packed

```
Input: A set $\{r_1, r_2, \ldots, r_N\}$ of rectangular items, size of population ($P$)
Output: Initial population $POP$
1 Function Initial_Population()
2 $S_0 \leftarrow Sort(\{r_1, r_2, \ldots, r_N\})$    /* sort the items in a descending order
   according to their areas to obtain a seed solution $S_0$ */
3 $POP \leftarrow \emptyset$
4 for $i \leftarrow 1$ to $P$ do
5     $k \leftarrow 1 + (i \bmod N)$
6     $S_i \leftarrow Shuffle(S_0, k)$    /* mutate the seed sequence $S_0$ by performing
       two-stage $k$-shuffle mutation operation */
7     $f(S_i) \leftarrow IPACK(S_i)$    /* evaluate the quality of $S_i$ by the decoding
       procedure $IPACK$ (Algorithm 5) */
8     $POP \leftarrow POP \cup \{S_i\}$
9 end
```

3.2 Solution Representation and Initial Population

We use the order representation to code the solutions in the population. Specifically, given a set $\{r_1, r_2, \ldots, r_N\}$ of N items, a solution is a sequence of these N items $(r_{i_1}, r_{i_2}, \ldots, r_{i_N})$, where $(i_1, i_2, \ldots, i_N)$ is a permutation of $(1, 2, \ldots, N)$. Based on this solution representation, the initial population consists of P sequences of N items, where P is the size of population. Moreover, for the variants that maximize the area of items packed and the variants that maximize the number of items packed, we respectively propose a new initialization method to generate the initial population.

For the variants maximizing the area of the packed items, the initialization method, summarized in Algorithm 3, operates as follows. First, all rectangular items are sorted in a descending order according to their areas, and the resulting sequence of items is identified as a seed sequence S_0 that is used to generate P individuals of population. Then, the seed sequence S_0 is slightly changed by performing a random constrained k-swap mutation that is described in Section 3.3, and the resulting sequence is considered as an individual of the population and its quality is evaluated by the decoding procedure of

Section 3.4. The seed sequence S_0 is randomly perturbed P times by the k-swap mutation to generate P individuals of the initial population. This initialization method is designed to favor large items, based on the observation that large items are generally contained in high-quality packing configurations.

For the variants maximizing the number of items packed, the initialization method (see Algorithm 4) is slightly different from that in Algorithm 3. Based on the seed sequence S_0 mentioned above, the method uses a two-stage k-shuffle mutation to generate each individual, where the value of k varies between 1 and N. To illustrate the two-stage k-shuffle mutation operator, Fig. 2 shows an example. At the first stage, the k-shuffle mutation performs consecutively k 1-shuffle mutations that move the first item to the tail of the sequence. At the second stage, the order of the items that have been moved to the tail of the sequence is reversed to obtain the resulting solution. This initialization method is designed to favor small items, based on the observation that the largest items are generally absent in high-quality packing configurations for these variants.

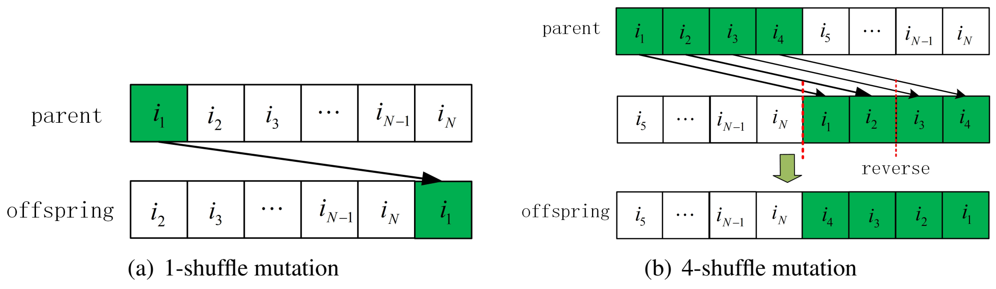

parent
offspring
i_1 i_2 i_3 ⋯ i_{N-1} i_N
(a) 1-shuffle mutation
parent
offspring
reverse
i_1 i_2 i_3 i_4 i_5 ⋯ i_{N-1} i_N
(b) 4-shuffle mutation

Figure 2: An illustrative example for two-stage k-shuffle mutation operator.

3.3 Mutation and Crossover

Our algorithm uses four operators to generate offspring solutions: the constrained insertion mutation, the constrained k-swap mutation, the randomized order crossover, and the prefix crossover, where the first three operators can be regarded as a new variant of the existing operators in the literature (Bouzid and Salhi, 2020; Zhang et al., 2024; Luo et al., 2024) and the prefix crossover is a new operator proposed in this work.

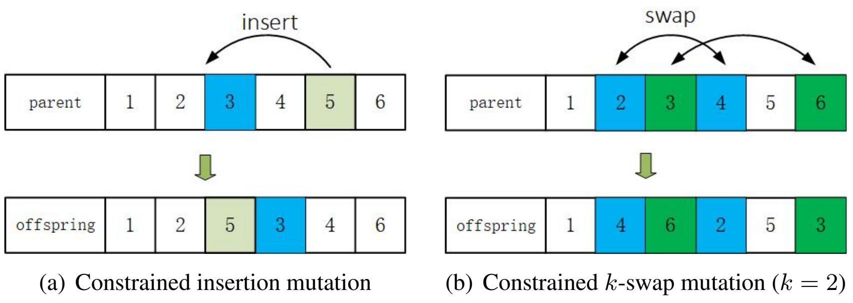


Figure 3: Illustrative examples for two mutation operators.

In the constrained insertion mutation, we first randomly select two items r_i and r_j from the given solution (i.e., a sequence of items), and then insert r_j in the front of r_i, where r_i and r_j must satisfy the condition of $|rank(r_i) - rank(r_j)| \leq \Delta$, where $rank(r_i)$ and $rank(r_j)$ respectively represent the

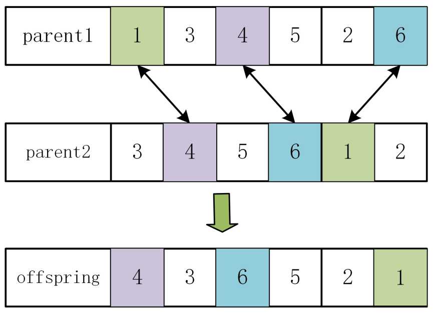


Figure 4: An illustrative example for the order crossover operator.

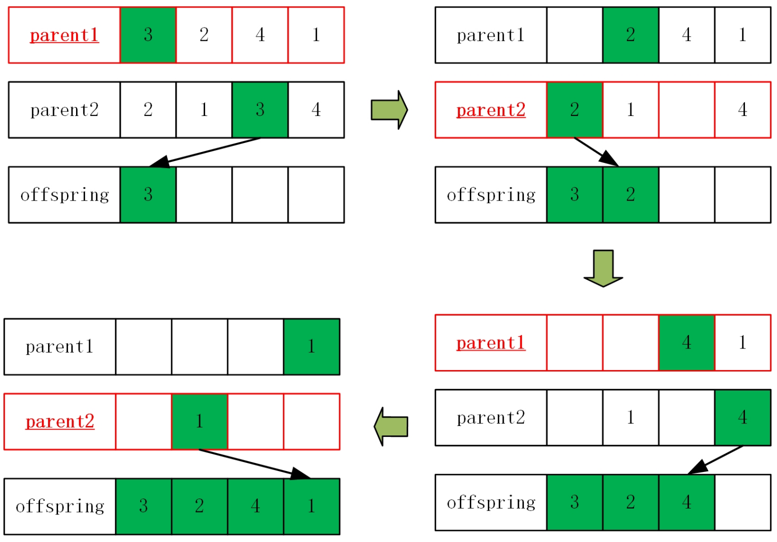


Figure 5: An illustrative example for the prefix crossover operator.

rankings of r_i and r_j with respect to the area of the items among all items, and Δ is a predetermined parameter.

In the constrained k-swap operator, we first randomly select k ($k \leq k_{max}$) pairs of similar items in terms of area, where k_{max} is a predetermined parameter that represents the maximal number of pairs of items that can be swapped, and two items r_i and r_j are considered to be similar if $|rank(r_i) - rank(r_j)| \leq \Delta$. Then, we exchange the positions of each pair of selected items to obtain a mutated solution. Fig. 3 gives an illustrative example for these two mutation operators.

Note that the above insertion and swap operations are restricted to two similar items r_i and r_j whose rank difference (i.e., $|rank(r_i) - rank(r_j)|$) is lower than or equal to Δ. The primary purpose of such a restriction is to avoid changing the packing configuration too much and degrading the quality of the solution too much.

The randomized order crossover works with two parent solutions $parent_1$ and $parent_2$. We first randomly select m (a parameter) items from the first parent, and then identify the order of these items in the second parent. Subsequently, the offspring solution inherits the identified order of these items, and then inherits the order of the first parent solution for the remaining items. The time complexity of this operator is $O(N)$, where N is the number of items in the problem instance. Fig. 4 gives an illustrative example. Note that this order crossover operator is slightly different from the classic order crossover operator (Luo et al., 2024), which first selects two crossover points in the parents and then directly copies the substring between the crossover points into the offspring solution.

The prefix crossover operator works as follows. Starting with an empty sequence as the offspring solution, the prefix crossover operator alternately chooses the first unused item (i.e., the prefix of sequence) from the two parents as the current item, and then adds it to the end of the offspring solution. Moreover, the chosen item is removed from two parents once it has been added to the offspring solution. The crossover operator terminates and returns the offspring solution when all items are added into the offspring. The time complexity of this crossover operator is also $O(N)$. Fig. 5 provides an illustrative example of our prefix crossover operator.

3.4 Decoding procedure

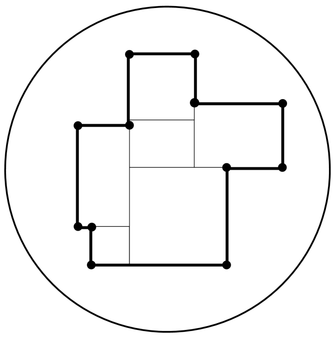


Figure 6: An example for the border of a packing configuration.

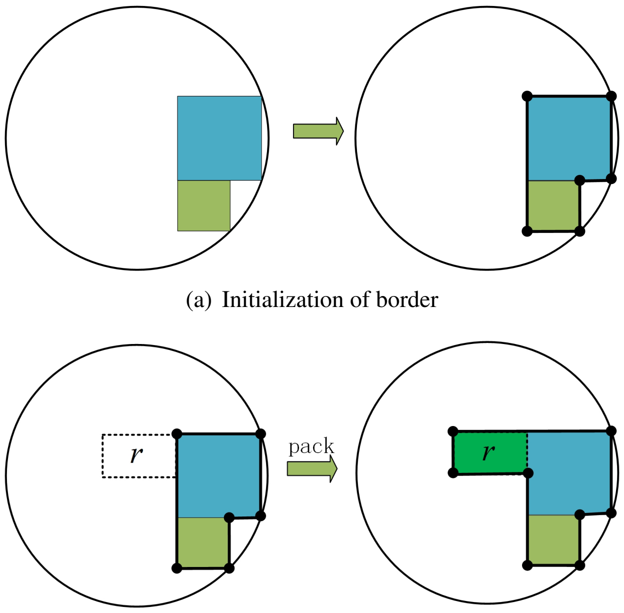


Figure 7: Initialization and update of border of a packing configuration.

The decoding procedure is a main component of the IDEA algorithm, whose goal is to convert a sequence of items to a packing configuration in order to obtain the positions of items packed and to

Algorithm 5: Decoding procedure (IPACK)

```
Input: A sequence $S$ of rectangular items
Output: A packing configuration $p$ of items packed
1  Function IPACK
2  Generate a partial packing $p$ with an initialization method
3  $S \leftarrow S \setminus \{p\}$                    /* Remove the items of $p$ from $S$ */
4  Build the border $B$ for the initial packing $p$
5  Construct a candidate list $CL(i)$ of nearby items packed for each vertex $i$ of $B$
6  while $S \neq \emptyset$ do
7      $r \leftarrow$ the first item in $S$
       /* Searching for a high-quality position on $B$ for $r$      */
8      $f_{best} \leftarrow 0$
9      for each position $POS$ on the border $B$ do
10         Pack tentatively $r$ on the current position $POS$
11         Containment test between $r$ and the container
12         Overlapping test between $r$ and the packed items in $CL(POS)$  /* $CL(POS)$ is
            associated with $POS$ */
13         if $POS$ is a feasible position for $r$ then
14             $f \leftarrow Score(POS)$  /* evaluate $POS$ by a scoring function */
15             if $f > f_{best}$ then
16                 $f_{best} \leftarrow f$
17                 $POS_{best} \leftarrow POS$        /* save the best position for $r$ */
18             end
19         end
20     end
       /* Performing the packing operation                          */
21     if there exists a feasible position on $B$ then
22         Pack rectangle $r$ on the position $POS_{best}$          /* $p \leftarrow p \cup \{r\}$ */
23         Update the border $B$ of $p$
24         Update the candidate list $CL(i)$ for each vertex $i$ of $B$
25     end
       /* Removing the rectangle $r$ from the sequence $S$          */
26     $S \leftarrow S \setminus \{r\}$
27 end
```

evaluate the packing quality. Our decoding procedure is an improved version of the *pack* procedure of Bouzid and Salhi (2020), thanks to the use of an ingenious data structure called the border of configuration to maintain the states of partial configurations. Specifically, the border of a packing is a circular list of vertices on the boundary of packing configuration, as shown in Fig. 6.

The decoding procedure is a constructive heuristic whose pseudo-code is given in Algorithm 5. Starting from an initial configuration generated by an initialization method, the decoding procedure first constructs the border of the configuration. Then, the procedure packs in order the remaining items on the positions of the border, and dynamically updates the packing configuration as well as its border.

As shown in Algorithm 5, after generating the initial configuration, the initial border, and the candidate list $CL(i)$ of items for each vertex i of the border (lines 2–5), the decoding procedure enters a 'while' loop to pack in order the remaining items until the current sequence S becomes an empty set (lines 6–27). At each loop, the first rectangular item in the current sequence S is first taken out as the current item r (line 7), then r is tentatively placed on each possible position of the border, and the feasibility of these positions is tested by checking whether r is completely contained in the container and overlaps with some items packed if it is placed on the position (lines 10-12). It should be noted that the number of overlapping tests between r and the items packed is generally large. To reduce the number of tests, we use a candidate list strategy to speed up the decoding process. Specifically, for each position POS on the border, only a subset $CL(POS)$ of the packed items is tested. Once a position is identified as feasible, a scoring function is used to evaluate its quality (line 14). Then, the position with the highest score is chosen to pack the item r when the feasibility of all positions has been checked and the set of feasible positions is not empty (lines 15-18, 21). After that, the packing configuration, the border of configuration, and the candidate lists ($CL(i)$) are accordingly updated (lines 22–24). Finally, the item r is removed from the sequence S (line 26). In particular, the initialization and the update of the border are illustrated in Fig. 7.

In our decoding procedure, the method of generating an initial packing configuration and the method of identifying the possible positions on the border are the same as in the *pack* procedure of Bouzid and Salhi (2020), and the main differences between our decoding procedure and the *pack* procedure of (Bouzid and Salhi, 2020) lie in our new scoring function to evaluate the positions on the border and a speedup strategy based on the candidate list strategy for the overlapping tests between the current item and the items packed. In the following subsections, we present the speedup strategy and the new scoring function in detail.

3.4.1 Speedup strategy for the overlapping tests between items

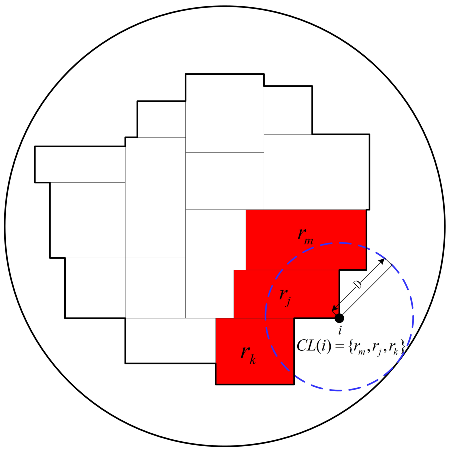


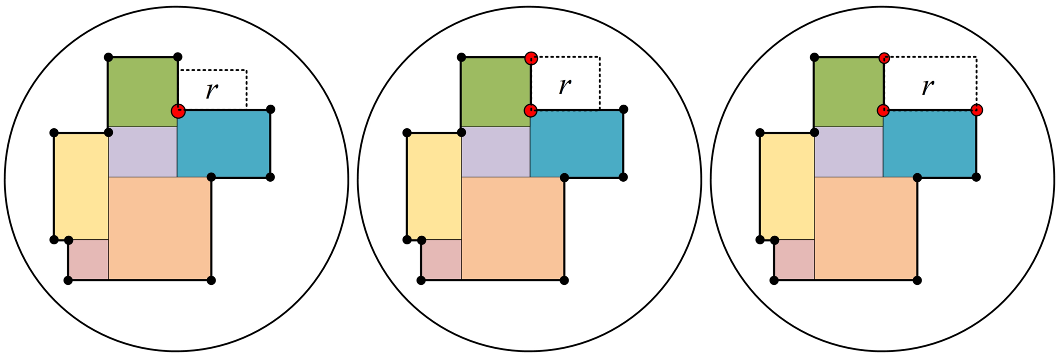
Figure 8: The candidate list strategy for the overlapping test of the decoding procedure.

In the border-based decoding procedure, the overlapping tests between the current item and the packed items is the most time-consuming part, due to the large number of the overlapping tests needed to check the feasibility of positions on the border. However, we observe that two sufficiently distant items are not overlapping and that it is not necessary to perform all the overlapping tests between the current item and the packed items if all the nearby items of the current position are given.

Thus, our decoding procedure employs a candidate list to record dynamically the nearby items packed for each vertex on the border. Specifically, for each vertex i, we generate a circle centering at the vertex i with a radius of D, where D represents the maximal diameter of N rectangular items. Then, all the packed items that overlaps with the circle are used to construct a candidate list $CL(i)$. With the help of a candidate list $CL(i)$, we only need to check the overlaps between the current item and those items in the list $CL(i)$ to check the feasibility of the current position, thus greatly reducing the computational complexity of the decoding procedure. Fig. 8 gives an illustrative example of our candidate list strategy, where the candidate list $CL(i)$ is composed of the red items.

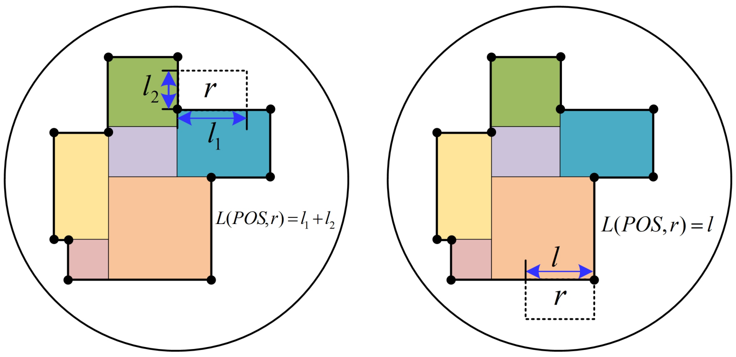
Assuming that the maximum number of items in the candidate lists $CL(i)$ ($i \in B$) is a constant Q, the time complexity of our decoding procedure is $O(N^2)$, which is much lower than $O(N^3)$ of the *pack* procedure of (Bouzid and Salhi, 2020). Thus, our decoding procedure is much more computationally efficient than the *pack* procedure of (Bouzid and Salhi, 2020) for handling large-scale instances.

3.4.2 Scoring function of the positions on the border



(a) $V(POS, r)$ =1 (b) $V(POS, r)$ =2 (c) $V(POS, r)$ =3

Figure 9: Illustrative examples for the fitness value V with respect to the current position of rectangular item r to be packed, where the coinciding vertices of r with the vertices of border are indicated in red.



(a) $L(POS, r) = l_1 + l_2$ (b) $L(POS, r) = l$

Figure 10: Two illustrative examples for the fitness value L with respect to the current position of rectangular item r to be packed.

Given a partial packing configuration, there may exist a number of feasible positions on the border of the configuration for the current item. A key task of the decoding procedure is then to find a suitable position for the current item. To do this, we generally need a scoring function to measure the goodness of each feasible position. And the design of the scoring function is very important because it strongly influences the performance of the decoding procedure. In order to achieve high performance, our decoding procedure uses a new scoring function, which is a variant of one used in (Wang et al., 2023), to efficiently evaluate the quality of positions on the border.

Given a position POS on the border and the current item r, the new scoring function denoted by $Score(POS, r)$ is composed of two parts and can be written as:

$$Score(POS, r) = \lambda \times V(POS, r) + L(POS, r) \tag{7}$$

where λ is a large coefficient such that the first term $V(POS, r)$ dominates the second term $L(POS, r)$.

The first term $V(POS, r)$ of Eq. (7) indicates the number of vertices of r that coincide with the vertices of border. Fig. 9 gives an illustrative example for the three possible situations. The second term $L(POS, r)$ of Eq. (7) measures the total overlapping length of sides of r that coincide with the border (see Fig. 10 for an example). Thus, for two positions POS_1 and POS_2 on the border, POS_1 is considered to be better than POS_2 if $Score(POS_1, r) > Score(POS_2, r)$. In other words, due to the large value of λ, POS_1 is superior to POS_2 if and only if one of the following conditions holds: 1) $V(POS_1, r) > V(POS_2, r)$, 2)$V(POS_1, r) = V(POS_2, r)$ and $L(POS_1, r) > L(POS_2, r)$. If multiple positions have the same best score, a random position among them is selected as the current position.

4 Experimental Evaluation and Computational Results

In this section, we evaluate the performance of the proposed IDEA algorithm based on the benchmark instances commonly used in the literature and make a comparison with several state-of-the-art algorithms.

4.1 Benchmark instances, parameter settings and experimental protocol

Table 1: Settings of parameters

Parameters	Section	Description	Values
P	3.1	size of population	200
α	3.1	probability of using the k-swap mutation	0.3
β	3.1	probability of using the insertion mutation	0.3
γ	3.1	probability of using the order crossover	0.2
m	3.1	parameter used in the order crossover	$N/4$
k_{max}	3.3	maximal number of pairs of items used in k-swap mutation	5
Δ	3.3	bound used in the mutation operators	$\{N, \frac{N}{5}\}$

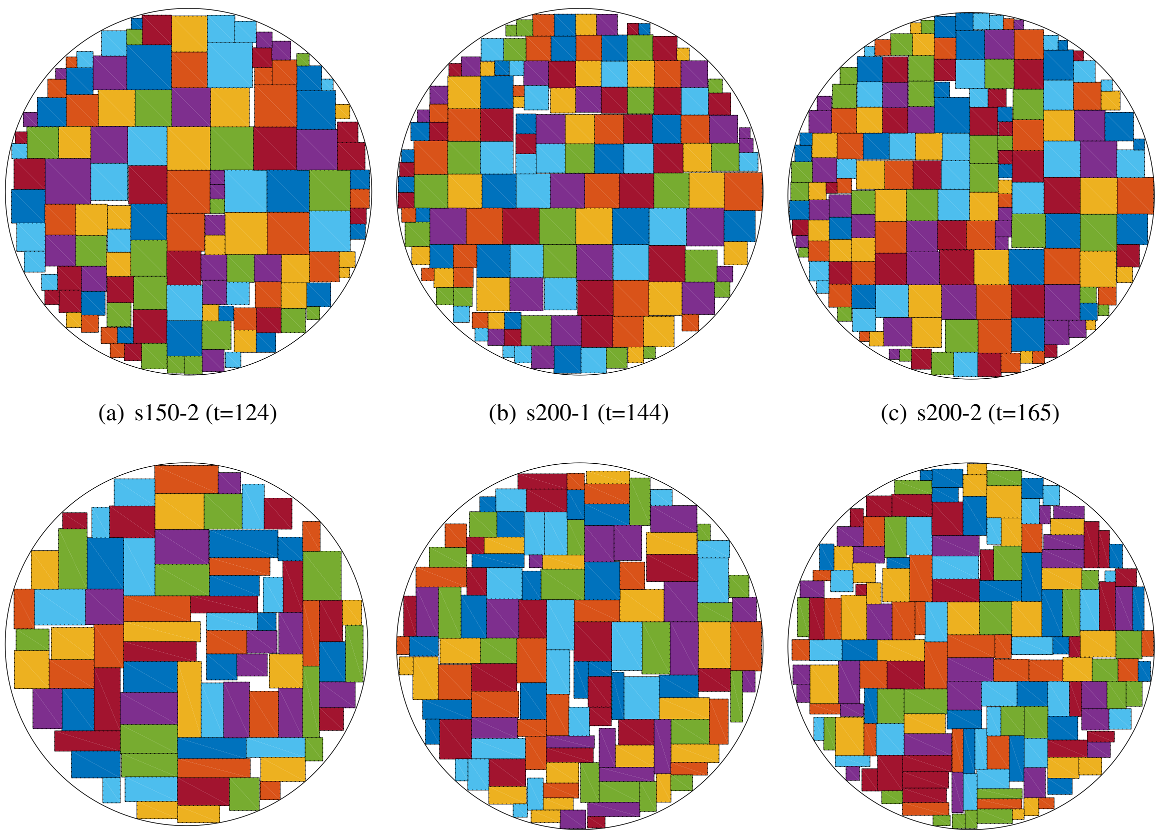
Based on the 36 basic benchmark instances generated by López and Beasley (2018)[1] and Bouzid and Salhi (2020)[2], we respectively obtain a set of benchmark instances for each of the four variants of the problem by considering the specific optimization objective and whether the rectangles can be rotated or not. More precisely, for the first variant (i.e., maximizing the number of items 'without' rotation), we obtain directly 36 instances. For the second variant (i.e., maximizing the number of items 'with' rotation), we obtain a set of 18 instances, because the other 18 basic instances are composed of squares and it makes no sense to rotate them. Thus, for the two variants of maximizing the number of items packed, we have a total of 54 ($= 36 + 18$) instances. Similarly, for the two variants of maximizing the area of items (with and without rotation), we obtain a set of 54 ($= 36 + 18$) benchmark instances. In total, we have 108 instances for all four variants of the problem.

[1]https://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/
[2]https://www.leandro-coelho.com/packing-circular-container

The IDEA algorithm uses several parameters whose default settings and descriptions are given in Table 1. These default parameter settings were determined via several preliminary experiments. The parameter P (the population size) was set to 200. The probabilities of applying k-swap and insertion mutations (i.e., α and β) were set to 0.3, and the probabilities of applying the two crossover operators (i.e., γ and $1-\alpha-\beta-\gamma$) were set to 0.2. The parameter k_{max} used in the k-swap mutation was to 5. The parameter Δ used in the constrained insertion and k-swap mutations was to N and $\frac{N}{5}$ respectively for the small instances with $N \leq 30$ and the large instances with $N \geq 100$.

The IDEA algorithm was implemented in the C language and all computational experiments were executed on a computer with a 2.3 GHz Intel(R) Xeon (R) Platinum 9242 CPU, running a Linux operating system. Moreover, due to the stochastic behavior of the algorithm, we ran the IDEA algorithm 10 times for each instance with different random seeds. The time limit t_{max} was set according to the size of instances: $t_{max} = 200$ seconds for the small instances with $N \leq 30$ and $t_{max} = 3000$ seconds for the large instances with $N \geq 100$.

4.2 Computational results for maximizing the number of items without rotation



(a) s150-2 (t=124) (b) s200-1 (t=144) (c) s200-2 (t=165)

(d) r150-0 (t=78) (e) r150-2 (t=118) (f) r200-2 (t=158)

Figure 11: Improved solutions found by our IDEA algorithm for 6 representative instances of the variant of maximizing the number of packed items without rotation, where t represents the number of items packed.

Table 2: Comparison of the IDEA algorithm with three previous algorithms on the small-scale instances of the variant of maximizing the number of items without rotation.

		FSS	PEA	SA		IDEA (this work)		
Instance	BKS	Best	Best	Best	Avg	Best	Avg	Δ_{BKS}
s10-0	**4**	**4**	**4***	**4**	4.00	**4**	4.00	0
s10-1	**5**	**5**	**5***	**5**	5.00	**5**	5.00	0
s10-2	**6**	**6**	**6***	**6**	6.00	**6**	6.00	0
s20-0	**11**	**11**	**11***	**11**	11.00	**11**	11.00	0
s20-1	**13**	12	**13***	**13**	12.20	**13**	13.00	0
s20-2	**15**	14	**15***	14	14.00	**15**	15.00	0
s30-0	**17**	16	**17**	16	16.00	**17**	17.00	0
s30-1	**21**	20	**21**	20	20.00	**21**	21.00	0
s30-2	**24**	23	**24**	23	23.00	**24**	24.00	0
r10-0	**5**	**5**	**5***	**5**	5.00	**5**	5.00	0
r10-1	**6**	**6**	**6***	**6**	6.00	**6**	6.00	0
r10-2	**7**	**7**	**7***	**7**	7.00	**7**	7.00	0
r20-0	**8**	7	**8***	7	7.00	**8**	8.00	0
r20-1	**11**	10	**11**	10	10.00	**11**	11.00	0
r20-2	**14**	11	**14**	13	13.00	**14**	13.30	0
r30-0	**15**	13	**15**	14	14.00	**15**	15.00	0
r30-1	**19**	16	**19**	18	18.00	**19**	19.00	0
r30-2	**22**	19	**22**	21	21.00	**22**	22.00	0
#Improve		0	0	0		0		
#Equal		7	18	8		18		
#Worse		11	0	10		0		

Table 3: Comparison of the IDEA algorithm with two recent heuristic algorithms on the small-scale instances of the variant of maximizing the number of items without rotation.

		HGA		SL-VNS		IDEA (this work)		
Instance	BKS	Best	Avg	Best	Avg	Best	Avg	Δ_{BKS}
s10-0	**4**	**4**	4.00	**4**	4.00	**4**	4.00	0
s10-1	**5**	**5**	5.00	**5**	5.00	**5**	5.00	0
s10-2	**6**	**6**	6.00	**6**	6.00	**6**	6.00	0
s20-0	**11**	**11**	11.00	10	10.00	**11**	11.00	0
s20-1	**13**	**13**	13.00	**13**	13.00	**13**	13.00	0
s20-2	**15**	**15**	15.00	**15**	15.00	**15**	15.00	0
s30-0	**17**	**17**	17.00	**17**	17.00	**17**	17.00	0
s30-1	**21**	**21**	21.00	**21**	21.00	**21**	21.00	0
s30-2	**24**	**24**	24.00	**24**	24.00	**24**	24.00	0
r10-0	**5**	**5**	5.00	4	4.00	**5**	5.00	0
r10-1	**6**	**6**	6.00	**6**	6.00	**6**	6.00	0
r10-2	**7**	**7**	7.00	6	6.00	**7**	7.00	0
r20-0	**8**	**8**	8.00	7	7.00	**8**	8.00	0
r20-1	**11**	**11**	11.00	**11**	10.80	**11**	11.00	0
r20-2	**14**	13	13.00	13	13.00	**14**	13.30	0
r30-0	**15**	**15**	15.00	14	14.00	**15**	15.00	0
r30-1	**19**	**19**	18.20	18	18.00	**19**	19.00	0
r30-2	**22**	**22**	21.75	21	21.00	**22**	22.00	0
#Improve		0		0		0		
#Equal		17		10		18		
#Worse		1		8		0		

This subsection aims to evaluate the performance of the proposed IDEA algorithm on the variant of maximizing the number of items without rotation, and the benchmark set used in the experiments consists of 36 instances and can be further divided into two subsets. The first subset contains 18 small-scale instances from López and Beasley (2018), where each instance is defined by a given container radius R and N ($N \in \{10, 20, 30\}$) squares or rectangles with randomly generated dimensions. The second set contains 18 large-scale instances from Bouzid and Salhi (2020), where each instance is defined by a container radius R and N ($N \in \{100, 150, 200\}$) squares or rectangles with their dimensions randomly generated in the interval $[1, 5]$.

The computational results of the IDEA algorithm are summarized in Tables 2-5 respectively for the small-scale instances and the large-scale instances, together with the results of six state-of-art algorithms in the literature, including the formulation space search (FSS) algorithm (López and Beasley, 2018), the parallel enumeration algorithm (PEA) (Silva et al., 2022) that is an exact algorithm, the simulated annealing (SA) algorithm (Bouzid and Salhi, 2020), the variable neighborhood search (VNS) algorithm (Bouzid and Salhi, 2020), the hybrid-biased genetic algorithm (HGA) (Luo et al., 2024), and the skyline-based variable neighborhood search (SL-VNS) algorithm (Zhang et al., 2024). The first two columns of Tables 2-5 give the names of instances and the best-known results or the optimal values (BKS) reported in the literature, where the instances with the letter 's' as their prefix consist of N squares. Columns 3 and 4 of Table 2 give the best objective value for the FSS and PEA algorithms, where the optimal values proved by the exact algorithm PEA are marked by the symbol '*'. For other four reference algorithms, the best objective value (Best), the average objective value (Avg) are given in the tables. The computational results of our IDEA algorithm are shown in the last three columns of the tables, where an improved result is underlined in terms of the best objective value, and the last column gives the difference (Δ_{BKS}) between the best result of IDEA and the best-known result, and a positive value of Δ_{BKS} means that an improved solution is found by our IDEA algorithm. In terms of the best objective value, the results matching the current best results are indicated in bold for all the algorithms. In addition, the last three rows '#Improve', '#Equal', and '#Worse' indicate the number of instances for which the corresponding algorithm obtains a better, equal or worse result than the best-known result reported in the literature in terms of the best objective value.

The computational results of the reference algorithms are taken from the literature, and the comparisons between the algorithms focus on solution quality. Due to the fact that the compared algorithms were implemented in different languages and run on different platforms, it is difficult to make a direct and fair comparison in terms of computation time. As an alternative, we provide the timing information of the compared algorithms in the online supplement for indicative purposes.

Table 2 shows that the IDEA algorithm performs very well for the small instances compared to the three reference algorithms FSS, SA, and PEA. Specifically, the IDEA algorithm significantly outperforms FSS and SA in terms of solution quality. The IDEA algorithm obtains the best-known result for all 18 small instances, while the FSS and SA algorithms obtain the best-known result for only 7 and 8 instances, respectively. Compared to the PEA algorithm that is an exact algorithm, the IDEA algorithm obtains an equal result for all instances in terms of the best objective value, where the result is proved to be optimal for 10 instances (indicated by '*').

Table 3 shows that IDEA also outperforms the recent HGA (Luo et al., 2024) and SL-VNS (Zhang et al., 2024) on these small instances. Compared to the SL-VNS algorithm, our IDEA algorithm obtains a better and equal result respectively for 8 and 10 out of 18 instances in terms of the best objective value. Moreover, the IDEA algorithm obtains a better result than the HGA algorithm for one instance in terms of the best objective value, and matches the result of the HGA algorithm for the remaining instances. Furthermore, IDEA outperforms HGA in terms of the average objective value for three instances.

Table 4: Comparison of the IDEA algorithm with two heuristic algorithms on the large-scale instances of the variant of maximizing the number of items without rotation.

		SA		VNS		IDEA (this work)		
Instance	BKS	Best	Avg	Best	Avg	Best	Avg	Δ_{BKS}
s100-0	57	53	52.20	52	51.40	**58**	57.10	1
s100-1	70	67	65.60	64	63.40	**71**	70.10	1
s100-2	80	76	75.20	73	72.80	**81**	80.90	1
s150-0	92	84	82.80	84	83.80	**93**	92.20	1
s150-1	110	103	101.80	100	99.80	**111**	110.10	1
s150-2	123	117	115.40	113	112.60	**124**	123.80	1
s200-0	117	107	104.60	108	105.60	**118**	117.90	1
s200-1	143	132	130.60	130	129.20	**144**	144.00	1
s200-2	164	152	151.00	152	149.80	**165**	164.30	1
r100-0	49	45	44.20	45	44.00	**50**	49.10	1
r100-1	63	59	57.60	58	57.60	**64**	63.90	1
r100-2	75	70	69.80	69	69.00	**76**	75.90	1
r150-0	77	70	68.80	72	71.20	**78**	78.00	1
r150-1	98	91	90.40	92	91.80	**101**	100.20	3
r150-2	116	110	108.80	110	108.40	**118**	118.00	2
r200-0	104	95	94.00	99	97.00	**107**	106.20	3
r200-1	131	123	121.80	124	123.20	**135**	134.90	4
r200-2	155	146	144.80	146	145.20	**158**	158.00	3
#Improve		0		0		18		
#Equal		0		0		0		
#Worse		18		18		0		

Table 5: Comparison of the IDEA algorithm with two recent heuristic algorithms on the large-scale instances of the variant of maximizing the number of items without rotation.

		HGA		SL-VNS		IDEA (this work)		
Instance	BKS	Best	Avg	Best	Avg	Best	Avg	Δ_{BKS}
s100-0	57	55	54.70	57	56.40	**58**	57.10	1
s100-1	70	68	67.45	70	69.40	**71**	70.10	1
s100-2	80	78	77.15	80	79.60	**81**	80.90	1
s150-0	92	91	89.40	92	91.40	**93**	92.20	1
s150-1	110	108	107.10	110	109.60	**111**	110.10	1
s150-2	123	121	120.20	123	122.60	**124**	123.80	1
s200-0	117	115	112.45	117	116.40	**118**	117.90	1
s200-1	143	140	138.20	143	142.80	**144**	144.00	1
s200-2	164	160	158.80	164	163.40	**165**	164.30	1
r100-0	49	49	47.65	47	46.60	**50**	49.10	1
r100-1	63	63	61.70	61	60.60	**64**	63.90	1
r100-2	75	75	73.60	73	72.60	**76**	75.90	1
r150-0	77	77	75.20	74	73.40	**78**	78.00	1
r150-1	98	98	96.70	95	94.60	**101**	100.20	3
r150-2	116	116	114.65	114	112.80	**118**	118.00	2
r200-0	104	104	101.80	101	100.80	**107**	106.20	3
r200-1	131	131	129.45	131	129.20	**135**	134.90	4
r200-2	155	155	153.10	154	153.20	**158**	158.00	3
#Improve		0		0		18		
#Equal		9		11		0		
#Worse		9		7		0		

Tables 4 and 5 clearly show that the IDEA algorithm significantly outperforms the four best-performing heuristic algorithms in the literature on the large-scale instances, i.e., SA (Bouzid and Salhi, 2020), VNS (Bouzid and Salhi, 2020), HGA (Luo et al., 2024) and SL-VNS (Zhang et al., 2024). We observe from Table 4 that our IDEA algorithm obtains a significantly better result than the SA and VNS algorithms for all the instances, both in terms of the best and average objective values. In particular, our IDEA algorithm improves the best-known result for all 18 large-scale instances. Furthermore, Table 5 shows that the IDEA algorithm also outperforms HGA and SL-VNS in terms of solution quality.

To have an intuitive impression on the best solutions from our IDEA algorithm, Fig. 11 shows the graphical representations of the improved solutions of 6 representative instances.

4.3 Computational results for maximizing the number of items with rotation

Table 6: Comparison of the IDEA algorithm with three previous algorithms on the small-scale instances of the variant of maximizing the number of items with rotation.

		FSS	PEA	SA		IDEA (this work)		
Instance	BKS	Best	Best	Best	Avg	Best	Avg	Δ_{BKS}
r10-0	**5**	**5**	$\mathbf{5}^*$	**5**	5.00	**5**	5.00	0
r10-1	**6**	**6**	$\mathbf{6}^*$	**6**	6.00	**6**	6.00	0
r10-2	**7**	**7**	$\mathbf{7}^*$	**7**	7.00	**7**	7.00	0
r20-0	**8**	**8**	$\mathbf{8}^*$	**8**	7.60	**8**	8.00	0
r20-1	**11**	10	**11**	10	10.00	**11**	11.00	0
r20-2	**14**	12	**14**	13	13.00	**14**	14.00	0
r30-0	**15**	14	**15**	14	14.00	**15**	15.00	0
r30-1	**19**	17	**19**	18	18.00	**19**	19.00	0
r30-2	**22**	20	**22**	21	21.00	**22**	22.00	0
#Improve		0	0	0		0		
#Equal		4	9	4		9		
#Worse		5	0	5		0		

Table 7: Comparison of the IDEA algorithm with two recent heuristic algorithms on the small-scale instances of the variant of maximizing the number of items with rotation.

		HGA		SL-VNS		IDEA (this work)		
Instance	BKS	Best	Avg	Best	Avg	Best	Avg	Δ_{BKS}
r10-0	**5**	**5**	5.00	**5**	4.60	**5**	5.00	0
r10-1	**6**	**6**	6.00	**6**	6.00	**6**	6.00	0
r10-2	**7**	**7**	7.00	**7**	7.00	**7**	7.00	0
r20-0	**8**	**8**	8.00	7	7.00	**8**	8.00	0
r20-1	**11**	**11**	11.00	**11**	10.80	**11**	11.00	0
r20-2	**14**	**14**	13.30	13	13.00	**14**	14.00	0
r30-0	**15**	**15**	15.00	**15**	15.00	**15**	15.00	0
r30-1	**19**	**19**	18.70	**19**	18.40	**19**	19.00	0
r30-2	**22**	**22**	21.95	**22**	21.60	**22**	22.00	0
#Improve		0		0		0		
#Equal		9		7		9		
#Worse		0		2		0		

This subsection aims to evaluate the performance of the IDEA algorithm on the variant of maximizing the number of items with rotation, where the items are allowed to be rotated orthogonally. The set of 18 benchmark instances can be further divided into two subsets, where the first subset contains 9 small instances with $N \in \{10, 20, 30\}$ from López and Beasley (2018) and the second subset contains 9 large instances with $N \in \{100, 150, 200\}$ generated by Bouzid and Salhi (2020). The computational results of the IDEA algorithm on the 9 small instances are summarized in Tables 6 and 7, together

Table 8: Comparison of the IDEA algorithm with two heuristic algorithms on the large-scale instances of the variant of maximizing the number of items with rotation.

		SA		VNS		IDEA (this work)		
Instance	BKS	Best	Avg	Best	Avg	Best	Avg	Δ_{BKS}
r100-0	48	46	45.20	45	44.40	**50**	49.20	2
r100-1	63	59	58.00	59	57.80	**64**	64.00	1
r100-2	75	71	70.40	70	69.40	**77**	76.10	2
r150-0	78	70	69.40	73	71.60	**79**	78.10	1
r150-1	100	92	90.60	93	91.80	**101**	100.90	1
r150-2	117	110	109.00	110	108.80	**118**	118.00	1
r200-0	104	95	94.60	98	97.40	**107**	107.00	3
r200-1	134	125	122.60	126	124.40	**136**	135.30	2
r200-2	158	148	146.00	147	146.40	**159**	158.80	1
#Improve		0		0		9		
#Equal		0		0		0		
#Worse		9		9		0		

Table 9: Comparison of the IDEA algorithm with two recent heuristic algorithms on the large-scale instances of the variant of maximizing the number of items with rotation.

		HGA		SL-VNS		IDEA (this work)		
Instance	BKS	Best	Avg	Best	Avg	Best	Avg	Δ_{BKS}
r100-0	48	48	47.95	48	48.00	**50**	49.20	2
r100-1	63	63	62.10	62	62.00	**64**	64.00	1
r100-2	75	75	74.00	75	74.20	**77**	76.10	2
r150-0	78	77	75.40	78	77.40	**79**	78.10	1
r150-1	100	99	97.00	100	99.20	**101**	100.90	1
r150-2	117	116	114.50	117	116.40	**118**	118.00	1
r200-0	104	103	102.25	104	103.80	**107**	107.00	3
r200-1	134	132	130.05	134	132.80	**136**	135.30	2
r200-2	158	155	153.40	158	157.00	**159**	158.80	1
#Improve		0		0		9		
#Equal		1		8		0		
#Worse		8		1		0		

with the results of five reference algorithms, i.e., FSS (López and Beasley, 2018), PEA (Silva et al., 2022), SA (Bouzid and Salhi, 2020), HGA (Luo et al., 2024) and SL-VNS (Zhang et al., 2024). The computational results of the IDEA algorithm on the 9 large instances are summarized in Tables 8 and 9, together with the results of four reference algorithms, i.e., SA, VNS (Bouzid and Salhi, 2020), HGA and SL-VNS. In these tables, the statistical information and symbols are the same as in the previous tables.

Table 6 shows that the IDEA algorithm performs very well and outperforms FSS and SA on the variant of maximizing the number of items with rotation. Specifically, the IDEA algorithm obtains the best-known result with a success rate of 100% for all instances, while FSS and SA obtain the best-known results for only 4 instances, respectively. Compared to the exact algorithm PEA, the IDEA algorithm obtains the same best solution for all instances, where the best solution is proved to be optimal for 4 instances (indicated by '*').

Table 7 shows that for the small instances the IDEA algorithm also outperforms the recent HGA and SL-VNS. In terms of the best objective value, the HGA and IDEA algorithms obtain the same result for all instances. However, the IDEA algorithm outperforms the HGA algorithm in terms of the average objective value. Moreover, compared to the SL-VNS algorithm, our IDEA algorithm obtains a better result for 2 instances in terms of the best objective value, matching the result of SL-VNS algorithm for the remaining instances. In terms of the average objective value, the IDEA algorithm obtains a better result for 5 out of 9 instances, and matches the result of SL-VNS algorithm for the remaining instances.

Tables 8 and 9 clearly show that in terms of the best objective value the proposed IDEA algorithm significantly outperforms four the state-of-art algorithms in the literature on the large-scale instances. It can be found from the tables that the IDEA algorithm improves the best-known results for all tested instances. Furthermore, the average objective value of the IDEA algorithm is superior to the best-known result for each tested instance, which shows a strong searching ability of IDEA algorithm.

In summary, the experimental results of this section show that the IDEA algorithm is very efficient and outperforms the state-of-the-art algorithms in the literature for the variant of maximizing the number of items with rotation.

4.4 Computational results for maximizing the area of items without rotation

This section evaluate the proposed IDEA algorithm on the variant of maximizing the area of items without rotation. The benchmark set used in the experiment consists of 36 instances and can be further divided into two subsets, including a set of 18 small-scale instances with $N \in \{10, 20, 30\}$ and a set of 18 large-scale instances with $N \in \{100, 150, 200\}$. The computational results of the IDEA algorithm on the small-scale instances are summarized in Tables 10 and 11, together with the results of five reference algorithms, i.e., the FSS algorithm (López and Beasley, 2018), the cutting plane method (CPM) (Silva et al., 2022) that is an exact algorithm, the VNS algorithm (Bouzid and Salhi, 2020), the HGA (Luo et al., 2024) , and the SL-VNS algorithm (Zhang et al., 2024). The computational results of the IDEA algorithm on the large-scale instances are summarized in Tables 12 and 13, together with the results of the SA, VNS, HGA and SL-VNS algorithms. The statistical information and symbols of the tables are the same as in the previous tables in Section 4.2.

Table 10: Comparison of the IDEA algorithm with three previous algorithms on the small-scale instances of the variant of maximizing the area of items without rotation.

		FSS	CPM	VNS		IDEA (this work)		
Instance	BKS	Best	Best	Best	Avg	Best	Avg	Δ_{BKS}
s10–0	**23.9878**	22.9485	**23.9878***	**23.9878**	23.3642	**23.9878**	23.5028	0.0000
s10–1	**37.7471**	36.7126	**37.7471***	**37.7471**	37.3333	**37.7471**	37.7471	0.0000
s10–2	**52.7555**	51.7583	**52.7555***	**52.7555**	51.9923	**52.7555**	52.7555	0.0000
s20–0	**64.7463**	54.1054	**64.7463***	63.7523	62.7630	63.7430	63.6483	-1.0033
s20–1	**95.9219**	85.2107	**95.9219***	94.7706	94.0801	95.7239	95.7239	-0.1980
s20–2	**137.2832**	109.8363	**137.2832***	132.4100	125.8077	131.9165	130.9737	-5.3667
s30–0	**65.1246**	54.4941	64.3817	63.9965	63.4017	64.7352	64.4022	-0.3894
s30–1	99.5590	77.5814	97.8908	98.1142	97.1655	**99.6053**	99.2590	0.0463
s30–2	134.5194	103.0963	131.6949	131.5472	129.5725	**134.6538**	133.7937	0.1344
r10–0	**18.4441**	**18.4441**	**18.4441***	**18.4441**	18.3351	**18.4441**	18.4441	0.0000
r10–1	**28.9390**	**28.9390**	**28.9390***	**28.9390**	28.9390	**28.9390**	28.9390	0.0000
r10–2	**39.4588**	37.6878	**39.4588***	38.7870	38.7870	**39.4588**	39.4588	0.0000
r20–0	**45.9961**	43.3885	**45.9961***	45.1567	44.7210	45.3621	45.3601	-0.6340
r20–1	**72.7850**	63.1643	**72.7850**	68.8314	67.4388	70.6065	70.6065	-2.1785
r20–2	**97.5007**	84.4446	96.7569	91.6368	90.6159	95.9012	95.4251	-1.5995
r30–0	67.4983	60.3570	67.1937	64.4689	63.7051	**67.6012**	67.2801	0.1029
r30–1	**104.7251**	85.2113	102.6709	102.1196	99.3385	104.0392	102.9556	-0.6859
r30–2	141.0049	103.4802	138.4558	137.4149	135.2876	**141.0108**	139.4474	0.0059
#Improve		0	0	0		4		
#Equal		2	15	5		6		
#Worse		16	3	13		8		

Table 10 shows that the IDEA algorithm outperforms both FSS and VNS on the small-scale instances in terms of solution quality. In terms of the best objective value, FSS and VNS obtain a worse result than the best-known result for 16 and 13 out of 18 instances, and match the best-known result for

Table 11: Comparison of the IDEA algorithm with two recent heuristic algorithms on the small-scale instances of the variant of maximizing the area of items without rotation.

		HGA		SL-VNS		IDEA (this work)		
Instance	BKS	Best	Avg	Best	Avg	Best	Avg	Δ_{BKS}
s10–0	**23.9878**	**23.9878**	23.9878	22.6150	22.6150	**23.9878**	23.5028	0.0000
s10–1	**37.7471**	**37.7471**	37.7471	36.7126	36.7126	**37.7471**	37.7471	0.0000
s10–2	**52.7555**	**52.7555**	52.7555	49.2838	49.0052	**52.7555**	52.7555	0.0000
s20–0	**64.7463**	**64.7463**	63.8517	59.6314	59.5471	63.7430	63.6483	-1.0033
s20–1	**95.9219**	**95.9219**	95.9044	88.9578	88.9578	95.7239	95.7239	-0.1980
s20–2	**137.2832**	**137.2832**	135.4750	130.9828	129.5658	131.9165	130.9737	-5.3667
s30–0	**65.1246**	**65.1246**	64.9994	63.9965	63.9965	64.7352	64.4022	-0.3894
s30–1	99.5590	99.5590	99.0206	97.7980	97.4868	**99.6053**	99.2590	0.0463
s30–2	134.5194	134.4015	133.11345	132.2788	131.1505	**134.6538**	133.7937	0.1344
r10–0	**18.4441**	**18.4441**	18.4441	17.8992	17.8992	**18.4441**	18.4441	0.0000
r10–1	**28.9390**	**28.9390**	28.9390	26.7643	26.7643	**28.9390**	28.9390	0.0000
r10–2	**39.4588**	**39.4588**	39.2676	37.3681	36.5987	**39.4588**	39.4588	0.0000
r20–0	**45.9961**	**45.9961**	45.7653	44.4296	44.4296	45.3621	45.3601	-0.6340
r20–1	**72.7850**	**72.7850**	71.6012	69.7481	69.5623	70.6065	70.6065	-2.1785
r20–2	**97.5007**	**97.5007**	95.7674	94.3265	93.7633	95.9012	95.4251	-1.5995
r30–0	67.4983	67.4983	66.77903	66.1060	66.1060	**67.6012**	67.2801	0.1029
r30–1	**104.7251**	**104.7251**	103.0498	101.7709	101.5139	104.0392	102.9556	-0.6859
r30–2	141.0049	141.0049	139.4206	137.5372	137.0940	**141.0108**	139.4474	0.0059
#Improve		0		0		4		
#Equal		14		0		6		
#Worse		4		18		8		

Table 12: Comparison of the IDEA algorithm with two heuristic algorithms on the large-scale instances of the variant of maximizing the area of items without rotation.

		SA		VNS		IDEA (this work)		
Instance	BKS	Best	Avg	Best	Avg	Best	Avg	Δ_{BKS}
s100–0	**301.3806**	293.5578	293.4949	297.8481	296.3685	300.5580	300.1525	-0.8226
s100–1	**455.4665**	446.0632	443.2703	449.0898	447.3728	453.1629	451.3734	-2.3036
s100–2	610.9711	587.8824	586.7831	600.2514	596.5111	**611.2694**	609.5736	0.2983
s150–0	**476.1826**	469.7621	468.9567	472.4785	470.7997	474.1083	473.0807	-2.0743
s150–1	**721.6048**	707.3694	705.7661	714.3385	712.3815	721.0362	719.8731	-0.5686
s150–2	**963.1189**	935.9346	935.0466	948.3443	946.8239	**963.5195**	960.9966	0.4006
s200–0	**602.4622**	593.9045	590.6651	596.4252	595.3754	601.8076	600.9707	-0.6546
s200–1	904.0008	882.0002	879.1137	889.4874	887.5525	**906.7465**	904.4098	2.7457
s200–2	1211.8357	1178.8631	1176.3317	1185.9045	1182.5038	**1212.3386**	1210.0431	0.5029
r100–0	285.3060	276.2503	274.7736	281.5851	279.3196	**286.8923**	284.7452	1.5863
r100–1	433.0382	418.9820	418.9820	424.7806	424.3101	**434.6132**	433.9410	1.575
r100–2	579.1004	556.3578	555.6125	567.0707	563.8865	**581.8476**	578.9688	2.7472
r150–0	393.1389	382.7671	382.6841	387.9162	386.4728	**394.9145**	393.9623	1.7756
r150–1	592.6726	576.9804	576.0199	583.5556	581.0678	**596.2875**	594.1016	3.6149
r150–2	793.9707	769.4407	765.6101	780.1788	777.1628	**796.9769**	795.2090	3.0062
r200–0	508.9918	497.3100	497.3100	503.1195	502.1400	**511.2864**	509.6866	2.2946
r200–1	766.7049	745.3243	745.0537	758.4439	754.7137	**770.7467**	769.0476	4.0418
r200–2	1021.4577	994.7474	990.0431	1003.5017	1001.2816	**1026.1222**	1023.7889	4.6645
#Improve		0		0		13		
#Equal		0		0		0		
#Worse		18		18		5		

Table 13: Comparison of the IDEA algorithm with two recent heuristic algorithms on the large-scale instances of the variant of maximizing the area of items without rotation.

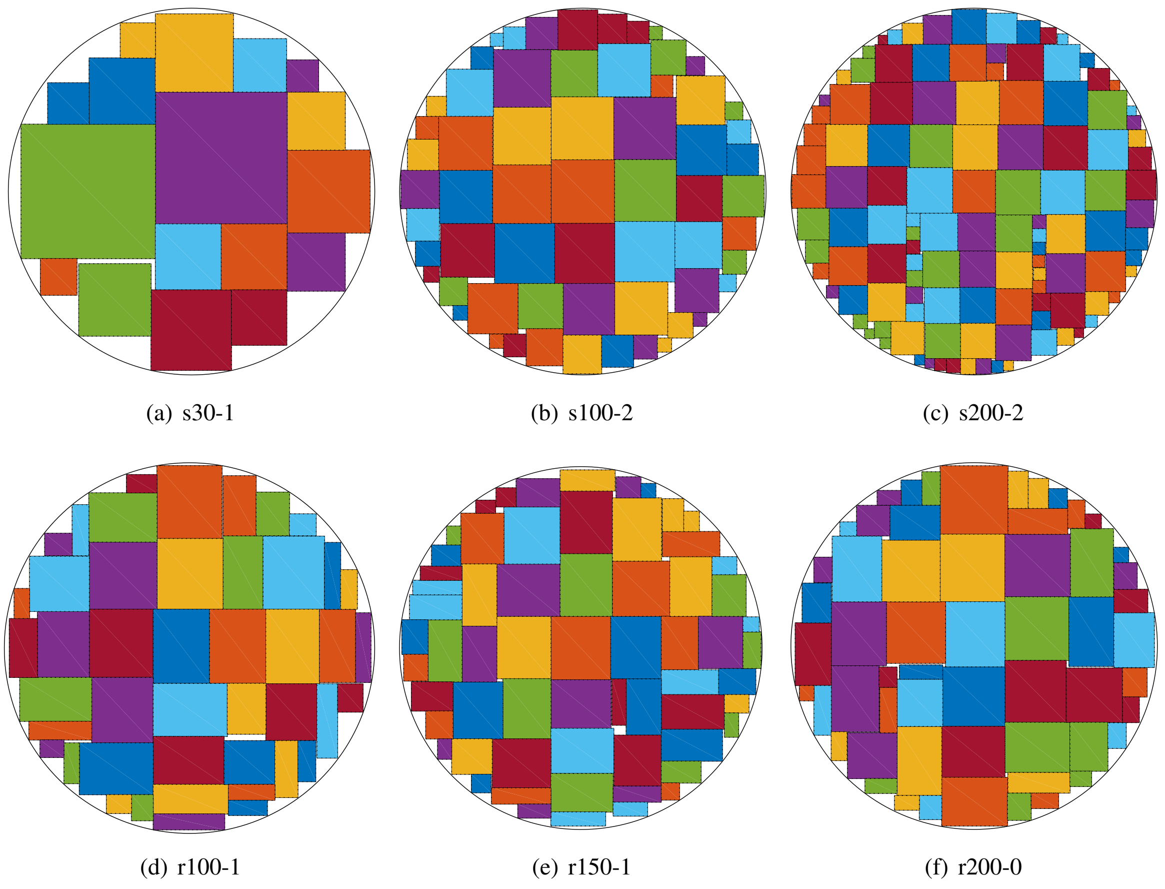
		HGA		SL-VNS		IDEA (this work)		
Instance	BKS	Best	Avg	Best	Avg	Best	Avg	Δ_{BKS}
s100–0	**301.3806**	**301.3806**	300.7750	297.5600	295.8297	300.5580	300.1525	-0.8226
s100–1	**455.4665**	**455.4665**	452.1387	451.0402	448.6063	453.1629	451.3734	-2.3036
s100–2	610.9711	610.9711	608.3166	602.1586	601.0954	**611.2694**	609.5736	0.2983
s150–0	**476.1826**	**476.1826**	474.2942	470.5036	469.5171	474.1083	473.0807	-2.0743
s150–1	**721.6048**	**721.6048**	720.1896	712.1517	711.2916	721.0362	719.8731	-0.5686
s150–2	963.1189	963.1189	961.9878	953.7799	952.2550	**963.5195**	960.9966	0.4006
s200–0	**602.4622**	**602.4622**	601.3726	597.6537	595.5345	601.8076	600.9707	-0.6546
s200–1	904.0008	904.0008	901.6450	898.7813	896.9252	**906.7465**	904.4098	2.7457
s200–2	1211.8357	1211.8357	1208.7768	1203.318	1200.9086	**1212.3386**	1210.0431	0.5029
r100–0	285.3060	285.3060	283.7283	283.5337	282.1808	**286.8923**	284.7452	1.5863
r100–1	433.0382	433.0382	430.8815	428.2655	426.8841	**434.6132**	433.9410	1.5750
r100–2	579.1004	579.1004	574.8603	569.5371	568.9367	**581.8476**	578.9688	2.7472
r150–0	393.1389	393.1389	391.2339	390.8242	390.4987	**394.9145**	393.9623	1.7756
r150–1	592.6726	592.6726	590.0955	590.7326	588.6003	**596.2875**	594.1016	3.6149
r150–2	793.9707	793.9707	789.5589	790.4961	787.3235	**796.9769**	795.2090	3.0062
r200–0	508.9918	508.9918	507.1107	508.4863	505.2021	**511.2864**	509.6866	2.2946
r200–1	766.7049	766.7049	764.3574	766.5361	763.5390	**770.7467**	769.0476	4.0418
r200–2	1021.4577	1021.4577	1016.3950	1018.2872	1017.1783	**1026.1222**	1023.7889	4.6645
#Improve		0		0		13		
#Equal		4		0		0		
#Worse		14		18		5		

the remaining instances. However, the IDEA algorithm obtains a worse result than the best-known result only for the 8 instances, and improves the best-known result for 4 instances. Compared to the exact algorithm CPM, the IDEA algorithm obtains a better and worse result for 6 instances, respectively, which means these two algorithms are comparable in terms of solution quality.

Table 11 shows that the IDEA algorithm significantly outperforms the SL-VNS algorithm but performs worse than the HGA algorithm on these small-scale instances. Specifically, the IDEA algorithm obtains a better result than the SL-VNS algorithm for all instances in terms of both the best and average objective values. On the other hand, the IDEA algorithm obtains a better and worse result than the HGA algorithm for 4 and 8 instances, respectively, in terms of the best objective value, which means that the HGA algorithm has a better performance than our IDEA algorithm on the small-scale instances. Since both the HGA and IDEA algorithms are evolutionary algorithms and use the same data structure in the decoding procedure, the difference in computational results between HGA and IDEA is mainly due to other algorithmic components, such as the number of initial configurations, the scoring function of the decoding procedure, the crossover and mutation operators, and the population updating strategy.

Tables 12 and 13 shows that in terms of solution quality the IDEA algorithm significantly outperforms four state-of-the-art algorithms on the large-scale instances, including SA, VNS, HGA, and SL-VNS. Compared to SA, VNS and SL-VNS, the IDEA algorithm obtains a better result for each instance tested in terms of both the best and average objective values. In particular, the IDEA algorithm improves the best-known result for 13 out of 18 instances. Moreover, compared to the HGA algorithm, the IDEA algorithm obtains a better result for 13 out of 18 instances in terms of the best objective value, and a worse result only for 5 instances.

To have an intuitive impression on the improved solutions, Fig. 12 provides the graphical representations for the best solutions of 6 representative instances.


(a) s30-1 (b) s100-2 (c) s200-2

(d) r100-1 (e) r150-1 (f) r200-0

Figure 12: Improved solutions found by the IDEA algorithm for 6 representative instances of the variant maximizing the area of packed items without rotation.

4.5 Computational results for maximizing the area of items with rotation

We now evaluate the performance of the IDEA algorithm on the variant of maximizing the area of items with rotation, based on the 18 benchmark instances which are further divided into a subset of 9 small-scale instances with $N \in \{10, 20, 30\}$ from López and Beasley (2018) and a subset of 9 large-scale instances with $N \in \{100, 150, 200\}$ from Bouzid and Salhi (2020). The computational results of the IDEA algorithm on the small-scale instances are given in Tables 14 and 15, together with the results of five reference algorithms in the literature. The results of the IDEA algorithm on the large-scale instances are given in Tables 16 and 17, together with the results of four state-of-the-art algorithms in literature.

We observe from Table 14 that the IDEA algorithm outperforms the FSS and VNS methods in terms of the best and average objective values except for two smallest instances. Compared to the exact CPM algorithm, the IDEA algorithm performs worse for 6 small instances $N \leq 20$, but performs better for the larger instances with $N = 30$, which means that the IDEA algorithm is more suitable than the CPM algorithm for the medium-sized and large-scale instances.

Table 15 shows that the IDEA algorithm outperforms the recent SL-VNS algorithm but performs worse than the HGA algorithm on these small-scale instances in terms of objective values. Specifically, the IDEA algorithm obtains a better result than the SL-VNS algorithm for almost all instances in terms

Table 14: Comparison of the IDEA algorithm with three previous algorithms on the small-scale instances of the variant maximizing the area of items with rotation.

		FSS	CPM	VNS		IDEA (this work)		
Instance	BKS	Best	Best	Best	Avg	Best	Avg	Δ_{BKS}
r10–0	**19.6702**	**19.6702**	**19.6702**	**19.6702**	19.2633	18.8619	18.8619	-0.8083
r10–1	**30.8746**	29.5041	**30.8746**	**30.8746**	29.7208	29.5041	29.5041	-1.3705
r10–2	**41.5246**	37.9687	**41.5246**	40.9063	40.3619	41.1612	41.1612	-0.3634
r20–0	**47.9336**	43.6850	**47.9336**	45.4200	45.3303	47.1228	46.9025	-0.8108
r20–1	**72.7850**	63.5279	**72.6945**	70.8221	68.7228	71.8253	71.7682	-0.9597
r20–2	**98.4189**	84.7008	**97.9712**	95.2162	93.4912	97.5326	97.4168	-0.8863
r30–0	**67.8840**	57.9328	67.2577	66.6329	65.1248	67.4140	67.3798	-0.4700
r30–1	**104.8784**	84.3715	103.1495	100.3020	99.2145	104.7157	103.3479	-0.1627
r30–2	**141.6376**	110.3253	138.6397	137.5277	136.1031	141.5478	140.3539	-0.0898
#Improve		0	0	0		0		
#Equal		1	8	2		0		
#Worse		8	1	7		9		

Table 15: Comparison of the IDEA algorithm with two recent heuristic algorithms on the small-scale instances of the variant maximizing the area of items with rotation.

		HGA		SL-VNS		IDEA (this work)		
Instance	BKS	Best	Avg	Best	Avg	Best	Avg	Δ_{BKS}
r10–0	**19.6702**	**19.6702**	19.6702	18.4441	18.4441	18.8619	18.8619	-0.8083
r10–1	**30.8746**	**30.8746**	30.8746	28.7839	27.7443	29.5041	29.5041	-1.3705
r10–2	**41.5246**	**41.5246**	41.3819	39.5569	39.5569	41.1612	41.1612	-0.3634
r20–0	**47.9336**	46.4452	46.0419	45.2503	45.2148	47.1228	46.9025	-0.8108
r20–1	**72.7850**	**72.7850**	71.6061	70.9040	70.4724	71.8253	71.7682	-0.9597
r20–2	**98.4189**	**98.4189**	97.5771	97.9176	96.8554	97.5326	97.4168	-0.8863
r30–0	**67.8840**	**67.8840**	67.1898	67.0593	66.8382	67.4140	67.3798	-0.4700
r30–1	**104.8784**	**104.8784**	103.7276	102.7266	102.2871	104.7157	103.3479	-0.1627
r30–2	**141.6376**	**141.6376**	140.0809	139.1076	138.0495	141.5478	140.3539	-0.0898
#Improve		0		0		0		
#Equal		8		0		0		
#Worse		1		9		9		

Table 16: Comparison of the IDEA algorithm with two heuristic algorithms on the large-scale instances of the variant of maximizing the area of items with rotation.

		SA		VNS		IDEA (this work)		
Instance	BKS	Best	Avg	Best	Avg	Best	Avg	Δ_{BKS}
r100–0	286.0224	275.9862	275.9862	282.3465	280.7830	**287.5666**	286.9704	1.5442
r100–1	434.7697	422.5904	422.5904	426.5573	425.9241	**435.3168**	434.4095	0.5471
r100–2	580.7641	564.5293	561.1405	571.6369	569.1376	**583.9064**	581.7455	3.1423
r150–0	393.9724	386.3599	386.0969	389.6020	388.5257	**395.5893**	394.5173	1.6169
r150–1	594.9417	576.2670	574.8735	585.5947	583.7210	**595.6288**	594.3948	0.6871
r150–2	796.1322	771.9016	769.5404	780.7081	778.2186	**799.1850**	795.5685	3.0528
r200–0	510.9300	500.0271	498.8451	505.4126	503.2490	**511.2842**	509.5505	0.3542
r200–1	768.3748	748.1296	748.1296	757.9592	755.5614	**771.3863**	769.5982	3.0115
r200–2	1026.7739	994.3341	992.4800	1008.3091	1006.3442	**1031.1575**	1027.0228	4.3836
#Improve		0		0		9		
#Equal		0		0		0		
#Worse		9		9		0		

Table 17: Comparison of the IDEA algorithm with two recent heuristic algorithms on the large-scale instances of the variant maximizing the area of items with rotation.

		HGA		SL-VNS		IDEA (this work)		
Instance	BKS	Best	Avg	Best	Avg	Best	Avg	Δ_{BKS}
r100–0	286.0224	285.2622	283.9407	286.0224	284.2914	**287.5666**	286.9704	1.5442
r100–1	434.7697	434.7697	432.9662	430.9151	429.6375	**435.3168**	434.4095	0.5471
r100–2	580.7641	580.7641	576.9916	579.1329	574.8210	**583.9064**	581.7455	3.1423
r150–0	393.9724	393.9724	392.2238	393.5067	391.5401	**395.5893**	394.5173	1.6169
r150–1	594.9417	594.9417	591.8375	594.0722	592.9682	**595.6288**	594.3948	0.6871
r150–2	796.1322	794.3447	791.3517	796.1322	794.7256	**799.1850**	795.5685	3.0528
r200–0	510.9300	509.2270	507.8536	510.9300	508.5306	**511.2842**	509.5505	0.3542
r200–1	768.3748	768.3748	766.4292	767.8849	765.3624	**771.3863**	769.5982	3.0115
r200–2	1026.7739	1026.7739	1021.1400	1026.3432	1024.2576	**1031.1575**	1027.0228	4.3836
#Improve		0		0		9		
#Equal		6		3		0		
#Worse		3		6		0		

of both the best and average objective values. On the other hand, our IDEA algorithm obtains a worse result than the HGA algorithm for 8 out of 9 instances in terms of the best objective value, and a better result for the remaining instance. The results of Tables 14 and 15 mean that for the small-scale instances the IDEA algorithm is the second best-performing heuristic algorithm after the HGA algorithm. It is worth noting that, according to an additional experiment (results not shown here), IDEA performs better than HGA in terms of the average objective value when both algorithms use the same initial configurations of Luo et al. (2024), which means that the initial configurations play an important role for the search ability of the algorithm on these small instances.

Tables 16 and 17 shows clearly that for the large-scale instances the IDEA algorithm significantly outperforms the four reference algorithms (SA, VNS, HGA and SL-VNS) in terms of solution quality. Compared to each of these reference algorithms, the IDEA algorithm produces a better result for all the tested instances in terms of both the best and average objective values. Moreover, the IDEA algorithm improves the best-known result for all the tested instances, which indicates a strong search ability of the algorithm.

4.6 A summary of comparisons between the proposed IDEA algorithm and the previous algorithms

Table 18: Summary for the number of instances where the proposed IDEA algorithm obtains a better, equal and worse result compared to the reference algorithms and the best-known result in terms of the best objective value.

			Maximizing Number				Maximizing Area			
Algorithm	Type	Reference	#Total	#Better	#Equal	#Worse	#Total	#Better	#Equal	#Worse
FSS	heuristic	López and Beasley (2018)	27	16	11	0	27	23	3	1
SA	heuristic	Bouzid and Salhi (2020)	54	32	12	0	54	44	8	2
VNS	heuristic	Bouzid and Salhi (2020)	54	44	10	0	54	45	5	4
PEA	exact	Silva et al. (2022)	54	27	27	0	54	37	6	11
CMP	exact	Silva et al. (2022)	27	9	18	0	27	12	6	9
SL-VNS	heuristic	Zhang et al. (2024)	54	37	17	0	54	53	0	1
HGA	heuristic	Luo et al. (2024)	54	28	26	0	54	26	6	22
BKS			54	27	27	0	54	26	6	22

To provide an overall impression of the comparisons between the proposed IDEA algorithm and seven previous algorithms from the literature, we summarize the comparative results in Table 18. This summary is based on the results from the previous subsections, as well as additional results from Bouzid and Salhi (2020) and Silva et al. (2022). The first three columns of the table give the name, type, and the reference of the algorithms, respectively. Columns 4-7 show the comparative results between the reference algorithms and our IDEA algorithm for the two variants of maximizing the number of items packed, including the total number (#Total) of instances used in the original paper in which the corresponding reference algorithm was proposed and tested, and the numbers (#Better, #Equal, and #Worse) of instances for which our IDEA algorithm achieves a better, equal, or worse result compared to the corresponding reference algorithm in terms of the best objective value. The last four columns of the table show the overall comparative results between our IDEA algorithm and the reference algorithms for the two variants of maximizing the area of packed items. In addition, the last row of the table compares the results of our IDEA algorithm with the best-known results in the literature.

We observe from Table 18 that the proposed IDEA algorithm significantly outperforms all the reference algorithms for the variants of maximizing the number of items packed. The best solution obtained by our IDEA algorithm is either better than or matches the best solution obtained by the reference algorithm for each instance. Furthermore, for the variants of maximizing the area of packed items, the IDEA algorithm also has a good performance compared to each reference algorithm in terms of the best objective value. The last row shows that the proposed IDEA algorithm improves the best-known solutions for 53 out of 108 instances, matching the best-known result for 33 instances. In summary, this outcome shows that the proposed algorithm significantly surpasses the state-of-the-art algorithms in the literature in terms of the solution quality.

5 Analysis of the Key Algorithmic Components

In this section, we analyze several key components of the proposed IDEA algorithm, including the population initialization method, the scoring function of decoding procedure, the parameters, and the speedup strategy of the decoding procedure.

5.1 Effectiveness of initial population

The population initialization method aims to provide a good initial population that contains some high-quality solutions. To check the effectiveness of the initialization method used in our IDEA algorithm, we carried out a comparative experiment based on 18 representative instances of the variant of maximizing the number of items without rotation. In this experiment, we created a variant IDEA* of our IDEA algorithm by replacing the initialization method with a random method, which generates each individual of the population randomly. We ran both IDEA* and IDEA 10 times on each instance, and report the computational results in Table 19, including the best objective value (Best), the average objective value (Avg), and the average computation time, where better results between the two compared algorithms are indicated in bold for each performance indicator. The last three rows '#Better', '#Equal', and '#Worse' indicate the numbers of instances for which the corresponding algorithm obtains a better, equal, or worse result than its competitor.

Table 19 shows that the IDEA algorithm significantly outperforms the IDEA* algorithm for all the considered performance indicators. First, in terms of the best objective value, the IDEA algorithm obtains a better result for 15 out of the 18 instances, while matching the results of IDEA* for the remaining instances. In terms of the average objective value, IDEA outperforms IDEA* for all the instances. In terms of computation time, the IDEA algorithm outperforms the IDEA* algorithm on

Table 19: Comparison between the population initialization methods on 18 representative instances of the variant maximizing the number of items without rotation. The IDEA* and IDEA algorithms were each run 10 times on each instance, and the dominant results between two algorithms are shown in bold for each considered performance indicator.

		Best		Avg		Time(s)	
Instance	BKS	IDEA*	IDEA	IDEA*	IDEA	IDEA*	IDEA
s100-0	57	57	**58**	56.60	**57.10**	325.20	**11.60**
s100-1	70	70	**71**	69.80	**70.10**	869.00	**107.30**
s100-2	80	80	**81**	79.20	**80.90**	331.40	**155.00**
s150-0	92	92	**93**	91.10	**92.20**	606.40	**407.00**
s150-1	110	109	**111**	108.80	**110.10**	1290.50	**18.40**
s150-2	123	122	**124**	121.90	**123.80**	1000.90	**594.70**
s200-0	117	116	**118**	115.50	**117.90**	1011.20	**517.10**
s200-1	143	142	**144**	141.50	**144.00**	1497.50	**43.30**
s200-2	164	161	**165**	160.80	**164.30**	943.80	**420.50**
r100-0	49	49	**50**	49.00	**49.10**	230.50	**27.70**
r100-1	63	64	64	63.20	**63.90**	**236.80**	478.70
r100-2	75	76	76	75.10	**75.90**	626.60	**257.80**
r150-0	77	78	78	77.30	**78.00**	587.60	**478.90**
r150-1	98	100	**101**	99.30	**100.20**	863.10	**628.50**
r150-2	116	117	**118**	116.40	**118.00**	1270.00	**1222.10**
r200-0	104	105	**107**	104.90	**106.20**	1313.10	**641.50**
r200-1	131	134	**135**	132.90	**134.90**	1728.90	**1199.50**
r200-2	155	156	**158**	155.50	**158.00**	1492.50	**540.20**
#Better		0	15	0	18	1	17
#Equal		3	3	0	0	0	0
#Worse		15	0	18	0	17	1

17 out of 18 instances. The result of this experiment clearly shows that the population initialization method plays an important role for the performance of the algorithm and that the initialization method used in our IDEA algorithm is much more efficient than the stochastic method.

5.2 Effectiveness of the scoring function

The scoring function of the decoding procedure is used to determine the position of the current rectangular item. To check whether the scoring function of decoding procedure plays an important role in the algorithm, we carried out an experiment based on the 18 representative instances used in Section 5.1. In this experiment, we first developed two IDEA variants, named IDEA$_1$ and IDEA$_2$, by using two alternative scoring functions, while maintaining the other components of the algorithm unchanged. In IDEA$_1$, the distance between the center of the item to be packed and the center of the container is used as a scoring function, as in Bouzid and Salhi (2020). In IDEA$_2$, the extended scoring function of Eq. (7) is used, i.e., the distance from a position of the current rectangular item to the boundary of the container is used as an additional criterion to further determine the scores of positions if multiple positions have the same score with respect to Eq. (7), and a shorter distance has a higher score. Then, the IDEA$_1$, IDEA$_2$ and IDEA algorithms were run 10 times on each instance, and the computational results are summarized in Table 20. The last two rows of the table respectively give the sum of the results in the corresponding column (i.e., sum) and the number of instances (i.e., #Best) where the corresponding algorithm obtains the best result between the compared algorithms in the performance indicator considered.

Table 20 shows that IDEA significantly outperforms IDEA$_1$ and slightly outperforms IDEA$_2$. Compared to the variant IDEA$_1$, IDEA achieves a better result in terms of the best objective value for 15 out of 18 instances, and matches the result of IDEA$_1$ for the remaining instances. For the average

Table 20: Comparison of the scoring functions on the 18 representative instances of the variant of maximizing the number of items packed without rotation. The IDEA$_1$, IDEA$_2$ and IDEA algorithms were respectively run 10 times for each instance and the best results between the algorithms are indicated in bold.

		Best			Avg			Time(s)		
Instance	BKS	IDEA$_1$	IDEA$_2$	IDEA	IDEA$_1$	IDEA$_2$	IDEA	IDEA$_1$	IDEA$_2$	IDEA
s100-0	57	57	**58**	**58**	57.00	**57.10**	**57.10**	45.70	107.50	**11.60**
s100-1	70	70	**71**	**71**	69.90	**70.20**	70.10	**57.60**	462.90	107.30
s100-2	80	80	**81**	**81**	80.00	80.70	**80.90**	69.80	**59.90**	155.00
s150-0	92	92	**93**	**93**	92.00	**92.20**	**92.20**	282.10	**159.80**	407.00
s150-1	110	109	110	**111**	109.00	110.00	**110.10**	30.90	**19.20**	18.40
s150-2	123	123	**124**	**124**	122.90	123.60	**123.80**	645.40	**222.80**	594.70
s200-0	117	117	**118**	**118**	116.90	117.70	**117.90**	687.70	**387.90**	517.10
s200-1	143	143	**144**	**144**	143.00	**144.00**	**144.00**	569.00	**42.1**	43.3
s200-2	164	163	**165**	**165**	162.60	164.10	**164.30**	677.80	**30.3**	420.5
r100-0	49	49	49	**50**	49.00	49.00	**49.10**	68.90	**22.50**	27.70
r100-1	63	**64**	**64**	**64**	63.80	**63.90**	**63.90**	893.40	**209.50**	478.70
r100-2	75	**76**	**76**	**76**	75.20	75.80	**75.90**	300.00	**245.10**	257.80
r150-0	77	**78**	**78**	**78**	77.60	**78.00**	**78.00**	705.30	**411.20**	478.90
r150-1	98	100	100	**101**	99.90	100.00	**100.20**	945.10	**138.3**	628.5
r150-2	116	117	**118**	**118**	117.00	117.60	**118.00**	**254.70**	542.50	1222.10
r200-0	104	106	106	**107**	105.90	106.00	**106.20**	797.70	**244.10**	641.50
r200-1	131	134	**135**	**135**	133.90	**134.90**	**134.90**	**850.70**	1649.50	1199.50
r200-2	155	157	**158**	**158**	156.90	**158.00**	**158.00**	697.60	703.40	**540.20**
sum		1835	1848	**1852**	1832.50	1842.80	**1844.60**	8579.16	**5658.49**	7749.77
#Best		3	14	18	0	8	17	3	13	2

objective value, IDEA outperforms IDEA$_1$ on all tested instances. On the other hand, compared to the variant IDEA$_2$, the IDEA algorithm obtains 4 better and 14 equal results in terms of the best objective value. For the average objective value, the IDEA and IDEA$_2$ algorithm obtain the best result among the three compared algorithms on 8 and 17 instances, respectively. However, the IDEA$_2$ algorithm generally outperforms the IDEA algorithm in terms of the computation time. This experiment confirms that the scoring function of our decoding procedure has a great impact on the performance of the IDEA algorithm and that the present scoring function is very efficient, especially for the variants of maximizing the number of items packed. Finally, we also tested the combined scoring function, which mixes the two scoring functions of IDEA$_2$ and IDEA. The preliminary results on instances of maximizing the number of items without rotation do not show an improvement over IDEA's results. However, the search for more meaningful scoring functions that take into account more problem features remains an interesting direction for future research.

5.3 Effectiveness of the speedup strategy of the decoding procedure

Recall that the overlapping test between items is the most frequent operation in the decoding process of a solution and that we use a candidate list strategy to reduce the number of the overlapping tests between items during the decoding process and greatly speed up the decoding process. To show the effect of this speedup strategy, we performed an experiment based on 18 representative instances mentioned in sections 5.1 and 5.2, where the IDEA algorithms with and without the speedup strategy were run 10 times on each instance and the average computation times of the underlying decoding procedures were recorded for each of the two algorithms. The computational results are plotted in Fig. 13, where the X-axis shows the instances and the Y-axis shows the average computation times of each run of the decoding procedures.

Fig. 13 shows that the speedup strategy plays an important role in enhancing the speed of the decoding procedure and that the effect of the speedup is closely related to the size of instances to be solved. For the medium-sized instances with $N = 100$, the decoding procedure with the speedup

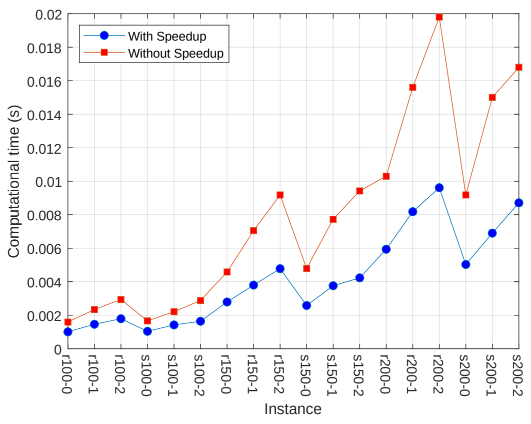


Figure 13: Comparison between the decoding procedures with and without the speedup in terms of the average computation time.

strategy leads to a speedup ratio of about 1.5. As the problem size increases, the effect of the speedup strategy is more and more obvious for most instances. In particular, the speedup ratio reaches about 2.0 for some large-scale instances with $N = 200$. Moreover, we can observe that the speed of the decoding procedure depends also on the instance to be solved. This experiment shows clearly that the candidate list strategy plays a key role in speeding up the decoding procedure especially for the large-scale instances.

5.4 Sensitivity analysis of the parameters

Table 21: Computational results of the proposed algorithm with eight parameter combinations of (α, β, γ) in terms of the average objective values (Avg).

	Avg							
Instance/(α, β, γ)	$(0, 0.1, 0.45)$	$(0, 0.5, 0.25)$	$(0.6, 0, 0.2)$	$(0.6, 0.1, 0.15)$	$(0.6, 0.1, 0.3)$	$(0.6, 0.1, 0)$	$(0.3, 0.3, 0.2)$	$(0.3, 0.1, 0.3)$
s100-0	57.0	57.0	57.0	57.1	57.0	57.0	57.1	57.0
s100-1	70.1	70.2	70.0	70.1	70.0	70.1	70.1	70.0
s100-2	80.6	80.8	80.7	80.9	80.8	80.8	80.9	80.4
s150-0	92.0	92.0	92.0	92.2	92.0	92.5	92.2	92.1
s150-1	110.0	110.0	110.0	110.0	110.0	110.0	110.1	110.0
s150-2	123.4	123.4	123.5	123.4	123.5	123.5	123.8	123.4
s200-0	117.6	117.7	117.8	117.5	117.2	118.0	117.9	117.7
s200-1	144.0	144.0	144.0	144.0	144.0	144.0	144.0	144.0
s200-2	164.1	164.2	164.0	164.2	164.0	164.2	164.3	164.1
r100-0	49.0	49.0	49.0	49.0	49.0	49.0	49.1	49.0
r100-1	63.6	64.0	63.5	63.8	63.8	63.7	63.9	63.7
r100-2	75.4	75.9	75.5	75.9	75.3	75.8	75.9	75.5
r150-0	77.9	78.0	78.0	78.1	77.9	77.9	78.0	78.0
r150-1	100.0	100.0	100.0	100.0	100.0	100.1	100.2	100.0
r150-2	117.6	118.0	117.6	117.7	117.7	117.8	118.0	117.9
r200-0	106.0	106.2	106.0	106.0	106.0	106.2	106.2	106.0
r200-1	134.9	135.0	134.4	134.8	134.5	134.6	134.9	134.8
r200-2	158.0	158.0	158.0	158.0	157.9	157.9	158.0	157.9
sum	1841.2	1843.4	1841.0	1842.7	1840.6	1843.1	1844.6	1841.5

To show the effect of the mutation and crossover operators on the performance of the algorithm and to determine an appropriate setting for the parameters α, β, γ, which are used to control the probabilities of applying these operators, we performed an experiment based on the 18 representative instances mentioned above. In this experiment, due to the high dependence between the parameters, the IDEA algorithm was run 10 times on each instance with each combination of (α, β, γ) given in Table 21, and the average objective values (Avg) obtained over 10 runs are summarized in columns 2-9 of Table 21 for each parameter combination. The last row of the table shows the sum of the computational results in the corresponding column.

The last row of Table 21 shows that the setting of the parameters α, β, γ has a slight influence on the performance of the algorithm. First, the second column of the table shows that the algorithm obtains a relatively poor performance with the setting of $(\alpha, \beta, \gamma) = (0, 0.1, 0.45)$, which means that the k-swap mutation operator is applied with a probability of 0 and both crossover operators are used with a large probability of 0.45. This result implies that the k-swap mutation plays a more important role than the crossover operators. Second, for the parameter combination of $(\alpha, \beta, \gamma) = (0, 0.5, 0.25)$, which means that the insertion mutation operator is used with a large probability of 0.5 and the k-swap mutation operator is applied with a probability of 0, the algorithm reaches a desirable performance (i.e., $sum = 1843.4$), and this outcome indicates that the insertion operator also plays an essential role for the performance of the algorithm. The conclusion is further confirmed by the worsen result (i.e., $sum = 1841$) in the fourth column in which the insertion operator is applied with a probability of 0. Third, with the combination of $(\alpha, \beta, \gamma) = (0.6, 0.1, 0.3)$, which means that the prefix crossover operator is disabled, the performance of the algorithm is obviously deteriorated (i.e., $sum = 1840.6$). This result implies that the prefix crossover operator plays a more important role compared to the order crossover operator, and this conclusion is also confirmed by the excellent results with respect to the parameter combination $(\alpha, \beta, \gamma) = (0.6, 0.1, 0)$ where the order crossover operator is disabled. Finally, for the combination $(\alpha, \beta, \gamma) = (0.3, 0.3, 0.2)$, which means that the two mutation operators are applied with a large probability of 0.3 and the two crossover operators are applied with a smaller probability of 0.2, the algorithm reaches a desirable performance with $sum = 1844.6$. In summary, this experiment shows that the mutation operators play a more important role than the crossover operators and thus should be applied with a large probability and that the crossover operators should be applied with a smaller probability to ensure a high performance of the algorithm.

6 Conclusions and Future Work

We study the problem of orthogonally packing rectangles in a fixed size circular container (OPRCC), which is a very challenging combinatorial optimization problem with many real-world applications. To efficiently solve the different variants of the OPRCC problem, we present an effective evolutionary algorithm that uses an improved decoding procedure, new initialization methods and a set of probabilistically applied mutation and crossover operators. The algorithm also integrates a new scoring function and an effective speedup strategy to accelerate the decoding procedure.

The performance of the proposed algorithm is evaluated on 108 benchmark instances widely used in the literature and compared with several state-of-the-art algorithms. The computational results on these benchmark instances show that the algorithm competes very favorably with the reference algorithms for the variants of maximizing the number of the packed items, improving the best-known result for 27 out of 54 instances, and matching the best-known result for the remaining instances. For the variants of maximizing the area of packed items, the algorithm improves the best-known result for 26 out of 54 instances. Experimental analysis shows that the population initialization methods, the scoring function and the speedup strategy of the decoding procedure all play a key role in the high performance of the algorithm.

The proposed algorithm has some limitations, which can be summarized as follows. First, compared to the algorithms using the skyline-based data structure (Wei et al., 2011; Zhang et al., 2024), the proposed algorithm is slower in the computational speed due to the fact that our decoding procedure must perform a large number of overlapping tests between the rectangles to find a feasible position for a rectangle to be packed. Second, the performance of the proposed algorithm is still worse than the state-of-the-art HGA algorithm (Luo et al., 2024) for the small-scale instances of the variants of maximizing the area of packed items, and thus the further improvements of the algorithm are still needed.

The present study can be extended in several directions. First, with some suitable modifications, the border-based decoding procedure with an appropriate scoring function can be applied to packing problems with a convex or non-convex container. In this sense, it can be considered as a general-purpose decoding procedure for rectangle packing problems, and it would be interesting to test it on rectangle packing problems with convex or non-convex containers, such as those encountered in the wood industry. Second, the proposed algorithm may be further improved by designing alternative methods of generating initial configurations of the underlying decoding procedure for the variants of maximizing the area of packed items. Third, it would be useful to study other efficient scoring functions for the decoding procedure, crossover operators, and population updating strategies. Finally, the proposed algorithm can be applied to other variants of the OPRCC problem, such as rectangle packing problems where the container has a number of defects.

Acknowledgments. We are grateful to the reviewers for their valuable comments and suggestions, which helped us to improve the paper. We thank the Beijing Beilong Super Cloud Computing Co., Ltd for providing HPC resources that have contributed to the computational experiments reported in this work. This work was partially supported by the National Natural Science Foundation of China (Grant No. 61703213).

References

S. D. Allen and E. K. Burke. Data structures for higher-dimensional rectilinear packing. *INFORMS Journal on Computing*, 24(3):457–470, 2012.

T. Back, D. B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation*. IOP Publishing Ltd., 1997.

E. G. Birgin and R. D. Lobato. Orthogonal packing of identical rectangles within isotropic convex regions. *Computers & Industrial Engineering*, 59(4):595–602, 2010.

E. G. Birgin, J. M. Martínez, F. Nishihara, and D. P. Ronconi. Orthogonal packing of rectangular items within arbitrary convex regions by nonlinear optimization. *Computers & Operations Research*, 33 (12):3535–3548, 2006.

A. Bortfeldt. A reduction approach for solving the rectangle packing area minimization problem. *European Journal of Operational Research*, 224(3):486–496, 2013.

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

E. K. Burke, G. Kendall, and G. Whitwell. A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research*, 52(4):655–671, 2004.

E. K. Burke, G. Kendall, and G. Whitwell. A simulated annealing enhancement of the best-fit heuristic for the orthogonal stock-cutting problem. *INFORMS Journal on Computing*, 21(3):505–516, 2009.

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

M. Chen, C. Wu, X. Tang, X. Peng, Z. Zeng, and S. Liu. An efficient deterministic heuristic algorithm for the rectangular packing problem. *Computers & Industrial Engineering*, 137:106097, 2019.

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

E. D. Demaine, S. P. Fekete, and R. J. Lang. Circle packing for origami design is hard. *arXiv preprint arXiv:1008.1224*, 2010.

V. P. dos Reis Arruda, L. G. B. Mirisola, and N. Y. Soma. Rectangle packing with a recursive pilot method. *Computers & Operations Research*, 161:106447, 2024.

H. Fırat and N. Alpaslan. An effective approach to the two-dimensional rectangular packing problem in the manufacturing industry. *Computers & Industrial Engineering*, 148:106687, 2020.

K. Forghani, M. Carlsson, P. Flener, M. Fredriksson, J. Pearson, and D. Yuan. Maximizing value yield in wood industry through flexible sawing and product grading based on wane and log shape. *Computers and Electronics in Agriculture*, 216:108513, 2024.

J. Gardeyn and T. Wauters. A goal-driven ruin and recreate heuristic for the 2D variable-sized bin packing problem with guillotine constraints. *European Journal of Operational Research*, 301(2): 432–444, 2022.

K. He, W. Huang, and Y. Jin. An efficient deterministic heuristic for two-dimensional rectangular packing. *Computers & Operations Research*, 39(7):1355–1363, 2012.

K. He, P. Ji, and C. Li. Dynamic reduction heuristics for the rectangle packing area minimization problem. *European Journal of Operational Research*, 241(3):674–685, 2015.

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

S. Imahori, M. Yagiura, and T. Ibaraki. Local search algorithms for the rectangle packing problem with general spatial costs. *Mathematical Programming*, 97:543–569, 2003.

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

J. Y. Leung, T. W. Tam, C. S. Wong, G. H. Young, and F. Y. Chin. Packing squares into a square. *Journal of Parallel and Distributed Computing*, 10(3):271–275, 1990.

K. Li and K. H. Cheng. Complexity of resource allocation and job scheduling problems in partitionable mesh connected systems. In *Interconnection Networks for High Performance Parallel Computers*, pages 644–651. IEEE, 1994.

Z. Li, X. Wang, J. Tan, and Y. Wang. 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(1):657170, 2014.

Z. Li, Y. Zeng, Y. Wang, L. Wang, and B. Song. A hybrid multi-mechanism optimization approach for the payload packing design of a satellite module. *Applied Soft Computing*, 45:11–26, 2016.

J. Liu, J. Li, Z. Lü, and Y. Xue. A quasi-human strategy-based improved basin filling algorithm for the orthogonal rectangular packing problem with mass balance constraint. *Computers & Industrial Engineering*, 107:196–210, 2017.

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

Q. Luo, Y. Rao, P. Yang, and X. Zhao. Hybrid-biased genetic algorithm for packing unequal rectangles into a fixed-size circle. *Computers & Operations Research*, 169:106716, 2024.

S. Martello and M. Monaci. Models and algorithms for packing rectangles into the smallest square. *Computers & Operations Research*, 63:161–171, 2015.

R. Martí, P. M. Pardalos, and M. G. C. Resende, editors. *Handbook of Heuristics*. Springer, 2018.

A. Silva, L. C. Coelho, M. Darvish, and J. Renaud. A cutting plane method and a parallel algorithm for packing rectangles in a circular container. *European Journal of Operational Research*, 303(1): 114–128, 2022.

L. Wang, X. Lai, and Z. Lin. An efficient heuristic algorithm for the rectangular packing problem. In *2023 China Automation Congress (CAC)*, pages 2585–2589. IEEE, 2023.

L. Wei, D. Zhang, and Q. Chen. A least wasted first heuristic algorithm for the rectangular packing problem. *Computers & Operations Research*, 36(5):1608–1614, 2009.

L. Wei, W.-C. Oon, W. Zhu, and A. Lim. A skyline heuristic for the 2D rectangular packing and strip packing problems. *European Journal of Operational Research*, 215(2):337–346, 2011.

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

L. Wu, L. Zhang, W.-S. Xiao, Q. Liu, C. Mu, and Y. Yang. A novel heuristic algorithm for two-dimensional rectangle packing area minimization problem with central rectangle. *Computers & Industrial Engineering*, 102:208–218, 2016.

Y.-L. Wu, W. Huang, S.-c. Lau, C. Wong, and G. H. Young. An effective quasi-human based heuristic for solving the rectangle packing problem. *European Journal of Operational Research*, 141(2): 341–358, 2002.

T. Zhang, R. Wang, H. Zhang, Q. Liu, and L. Wei. A skyline-based heuristic for orthogonal packing rectangles in a circle. *Computers & Operations Research*, 167:106664, 2024.

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

Online supplement of the paper “An effective evolutionary algorithm for packing rectangles into a fixed size circular container”

Xiangjing Lai[1], Lei Wang[2], Jin-Kao Hao*[3], and Qinghua Wu[4]

[1]School of Business, Nanjing University of Information Science and Technology, Nanjing 210044, China.
`laixiangjing@gmail.com (Xiangjing Lai)`
[2]Institute of Advanced Technology, Nanjing University of Posts and Telecommunications, Nanjing 210023, P.R.China.
`leiwang97@163.com (Lei Wang)`
[3]LERIA, Université d’Angers, 2 Boulevard Lavoisier, 49045 Angers, France.
`jin-kao.hao@univ-angers.fr (Jin-Kao Hao)`
*Corresponding author
[4]School of Management, Huazhong University of Science and Technology, 430074 Wuhan, P.R.China.
`qinghuawu1005@gmail.com (Qinghua Wu)`

April 20, 2025

1 Detailed Computational Results

In this online supplement, we present the detailed computational results of the IEDA algorithm and the reference algorithms in the literature, where ’Best’ denotes the best objective values obtained by the corresponding algorithm, ’Avg’ denotes the average objective values obtained over multiple runs of algorithms, and the computation times refer to the running time of the algorithms or their valid computation time (i.e., the elapsed time from the start of the program to the last update of the best solution found). The reference algorithms include the formulation space search (FSS) algorithm (López and Beasley, 2018), the cutting plane method (CPM) (Silva et al., 2022), the parallel enumeration algorithm (PEA) (Silva et al., 2022), the simulated annealing algorithm (SA) (Bouzid and Salhi, 2020), the variable neighborhood search algorithm (VNS) (Bouzid and Salhi, 2020), the hybrid-biased genetic algorithm (HGA) (Luo et al., 2024), and the skyline-based variable neighborhood search algorithm (SL-VNS) (Zhang et al., 2024). It should be noted that the computation times given in the tables are only indicative, since the algorithms compared were run on different computing platforms. In addition, for our IDEA algorithm, we give the average number of configurations (#pack) evaluated from the start of the program to the last update of the best solution found. In the following subsections, the results of the IEDA algorithm and the reference algorithms are given for each of the four problem variants.

1.1 Computational results for maximizing the number of items without rotation

Table 1: Comparison of the proposed algorithm with three previous algorithms (i.e., FSS (López and Beasley, 2018), PEA (Silva et al., 2022) and SA (Bouzid and Salhi, 2020)) on the small-scale instances of the variant of maximizing the number of items without rotation.

		FSS		PEA		SA			IDEA (this work)				
Instance	BKS	Best	Time(s)	Best	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Δ_{BKS}	#pack
s10-0	**4**	**4**	1123.0	**4**	0.2	**4**	4.00	1.0	**4**	4.00	0.00	0	228
s10-1	**5**	**5**	2761.0	**5**	0.3	**5**	5.00	1.0	**5**	5.00	0.00	0	215
s10-2	**6**	**6**	2275.0	**6**	0.6	**6**	6.00	2.0	**6**	6.00	0.00	0	213
s20-0	**11**	**11**	5450.0	**11**	236.7	**11**	11.00	5.0	**11**	11.00	0.01	0	207
s20-1	**13**	12	6465.0	**13**	358.4	**13**	12.20	5.0	**13**	13.00	0.02	0	204
s20-2	**15**	14	6995.0	**15**	28801.0	14	14.00	6.0	**15**	15.00	0.18	0	3497
s30-0	**17**	16	13552.0	**17**	28803.3	16	16.00	10.0	**17**	17.00	0.09	0	1051
s30-1	**21**	20	13457.0	**21**	28802.7	20	20.00	12.0	**21**	21.00	0.05	0	202
s30-2	**24**	23	10427.0	**24**	28801.3	23	23.00	13.0	**24**	24.00	0.07	0	202
r10-0	**5**	**5**	3058.0	**5**	0.2	**5**	5.00	1.0	**5**	5.00	0.00	0	211
r10-1	**6**	**6**	2862.0	**6**	0.2	**6**	6.00	2.0	**6**	6.00	0.00	0	210
r10-2	**7**	**7**	2966.0	**7**	0.2	**7**	7.00	2.0	**7**	7.00	0.00	0	229
r20-0	**8**	7	6278.0	**8**	3441.7	7	7.00	5.0	**8**	8.00	8.50	0	391862
r20-1	**11**	10	4530.0	**11**	28800.1	10	10.00	5.0	**11**	11.00	0.26	0	7906
r20-2	**14**	11	7311.0	**14**	28801.3	13	13.00	6.0	**14**	13.30	2.68	0	64724
r30-0	**15**	13	11514.0	**15**	28801.8	14	14.00	10.0	**15**	15.00	0.34	0	5192
r30-1	**19**	16	10029.0	**19**	28801.5	18	18.00	11.0	**19**	19.00	3.16	0	36849
r30-2	**22**	19	6966.0	**22**	28801.0	21	21.00	13.0	**22**	22.00	0.28	0	2262
#Improve		0		0		0			0				
#Equal		7		18		8			18				
#Worse		11		0		10			0				

Table 2: Comparison of the proposed algorithm with two recent heuristic algorithms (i.e., HGA (Luo et al., 2024) and SL-VNS (Zhang et al., 2024)) on the small-scale instances of the variant of maximizing the number of items without rotation.

		HGA			SL-VNS			IDEA (this work)				
Instance	BKS	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Δ_{BKS}	#pack
s10-0	**4**	**4**	4.00	0.03	**4**	4.00	0.15	**4**	4.00	0.00	0	228
s10-1	**5**	**5**	5.00	0.06	**5**	5.00	0.20	**5**	5.00	0.00	0	215
s10-2	**6**	**6**	6.00	0.08	**6**	6.00	0.25	**6**	6.00	0.00	0	213
s20-0	**11**	**11**	11.00	0.18	10	10.00	0.50	**11**	11.00	0.01	0	207
s20-1	**13**	**13**	13.00	0.14	**13**	13.00	0.50	**13**	13.00	0.02	0	204
s20-2	**15**	**15**	15.00	1.78	**15**	15.00	0.55	**15**	15.00	0.18	0	3497
s30-0	**17**	**17**	17.00	42.36	**17**	17.00	1.15	**17**	17.00	0.09	0	1051
s30-1	**21**	**21**	21.00	67.97	**21**	21.00	1.10	**21**	21.00	0.05	0	202
s30-2	**24**	**24**	24.00	84.8	**24**	24.00	1.15	**24**	24.00	0.07	0	202
r10-0	**5**	**5**	5.00	0.03	4	4.00	0.20	**5**	5.00	0.00	0	211
r10-1	**6**	**6**	6.00	0.04	**6**	6.00	0.20	**6**	6.00	0.00	0	210
r10-2	**7**	**7**	7.00	0.05	6	6.00	0.25	**7**	7.00	0.00	0	229
r20-0	**8**	**8**	8.00	3.51	7	7.00	0.45	**8**	8.00	8.50	0	391862
r20-1	**11**	**11**	11.00	0.5	**11**	10.80	0.80	**11**	11.00	0.26	0	7906
r20-2	**14**	13	13.00	36.79	13	13.00	0.75	**14**	13.30	2.68	0	64724
r30-0	**15**	**15**	15.00	38.7	14	14.00	1.00	**15**	15.00	0.34	0	5192
r30-1	**19**	**19**	18.20	53.78	18	18.00	1.15	**19**	19.00	3.16	0	36849
r30-2	**22**	**22**	21.75	68.64	21	21.00	1.25	**22**	22.00	0.28	0	2262
#Improve		0			0			0				
#Equal		17			10			18				
#Worse		1			8			0				

Table 3: Comparison of the proposed algorithm with two heuristic algorithms (i.e., SA (Bouzid and Salhi, 2020) and VNS (Bouzid and Salhi, 2020)) on the large-scale instances of the variant of maximizing the number of items without rotation.

		SA			VNS			IDEA (this work)				
Instance	BKS	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Δ_{BKS}	#pack
s100-0	57	53	52.20	118.0	52	51.40	126.0	**58**	57.10	11.6	1	11923
s100-1	70	67	65.60	153.0	64	63.40	145.0	**71**	70.10	107.3	1	81927
s100-2	80	76	75.20	176.0	73	72.80	143.0	**81**	80.90	155.0	1	104364
s150-0	92	84	82.80	289.0	84	83.80	308.0	**93**	92.20	407.0	1	162361
s150-1	110	103	101.80	429.0	100	99.80	319.0	**111**	110.10	18.4	1	5075
s150-2	123	117	115.40	511.0	113	112.60	393.0	**124**	123.80	594.7	1	153639
s200-0	117	107	104.60	585.0	108	105.60	569.0	**118**	117.90	517.1	1	109983
s200-1	143	132	130.60	832.0	130	129.20	687.0	**144**	144.00	43.3	1	6718
s200-2	164	152	151.00	1082.0	152	149.80	895.0	**165**	164.30	420.5	1	58229
r100-0	49	45	44.20	101.0	45	44.00	66.0	**50**	49.10	27.7	1	31116
r100-1	63	59	57.60	141.0	58	57.60	85.0	**64**	63.90	478.7	1	364156
r100-2	75	70	69.80	181.0	69	69.00	95.0	**76**	75.90	257.8	1	158865
r150-0	77	70	68.80	272.0	72	71.20	217.0	**78**	78.00	478.9	1	180291
r150-1	98	91	90.40	380.0	92	91.80	258.0	**101**	100.20	628.5	3	164351
r150-2	116	110	108.80	501.0	110	108.40	286.0	**118**	118.00	1222.1	2	265266
r200-0	104	95	94.00	534.0	99	97.00	468.0	**107**	106.20	641.5	3	109738
r200-1	131	123	121.80	818.0	124	123.20	737.0	**135**	134.90	1199.5	4	152346
r200-2	155	146	144.80	1038.0	146	145.20	683.0	**158**	158.00	540.2	3	59062
#Improve		0			0			18				
#Equal		0			0			0				
#Worse		18			18			0				

Table 4: Comparison of the proposed algorithm with two recent heuristic algorithms (i.e., HGA (Luo et al., 2024) and SL-VNS (Zhang et al., 2024)) on the large-scale instances of the variant of maximizing the number of items without rotation.

		HGA			SL-VNS			IDEA (this work)				
Instance	BKS	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Δ_{BKS}	#pack
s100-0	57	55	54.70	323.0	57	56.40	9.70	**58**	57.10	11.6	1	11923
s100-1	70	68	67.45	513.0	70	69.40	8.70	**71**	70.10	107.3	1	81927
s100-2	80	78	77.15	725.0	80	79.60	10.10	**81**	80.90	155.0	1	104364
s150-0	92	91	89.40	813.0	92	91.40	18.45	**93**	92.20	407.0	1	162361
s150-1	110	108	107.10	1353.0	110	109.60	24.90	**111**	110.10	18.4	1	5075
s150-2	123	121	120.20	1944.0	123	122.60	25.55	**124**	123.80	594.7	1	153639
s200-0	117	115	112.45	1276.0	117	116.40	35.85	**118**	117.90	517.1	1	109983
s200-1	143	140	138.20	2235.0	143	142.80	40.85	**144**	144.00	43.3	1	6718
s200-2	164	160	158.80	3445.0	164	163.40	60.95	**165**	164.30	420.5	1	58229
r100-0	49	49	47.65	299.0	47	46.60	10.20	**50**	49.10	27.7	1	31116
r100-1	63	63	61.70	490.0	61	60.60	9.60	**64**	63.90	478.7	1	364156
r100-2	75	75	73.60	711.0	73	72.60	9.25	**76**	75.90	257.8	1	158865
r150-0	77	77	75.20	690.0	74	73.40	19.35	**78**	78.00	478.9	1	180291
r150-1	98	98	96.70	1201.0	95	94.60	35.95	**101**	100.20	628.5	3	164351
r150-2	116	116	114.65	1793.0	114	112.80	33.70	**118**	118.00	1222.1	2	265266
r200-0	104	104	101.80	1263.0	101	100.80	40.65	**107**	106.20	641.5	3	109738
r200-1	131	131	129.45	2146.0	131	129.20	51.55	**135**	134.90	1199.5	4	152346
r200-2	155	155	153.10	3220.0	154	153.20	52.95	**158**	158.00	540.2	3	59062
#Improve		0			0			18				
#Equal		9			11			0				
#Worse		9			7			0				

1.2 Computational results for maximizing the number of items with rotation

Table 5: Comparison of the proposed algorithm with three previous algorithms (i.e., FSS (López and Beasley, 2018), PEA (Silva et al., 2022), and SA (Bouzid and Salhi, 2020))on the small-scale instances of the variant of maximizing the number of items with rotation.

		FSS		PEA		SA			IDEA (this work)				
Instance	BKS	Best	Time(s)	Best	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Δ_{BKS}	#pack
r10-0	**5**	**5**	9836.0	**5**	0.3	**5**	5.00	2.0	**5**	5.00	0.00	0	211
r10-1	**6**	**6**	10332.0	**6**	0.3	**6**	6.00	3.0	**6**	6.00	0.01	0	224
r10-2	**7**	**7**	12409.0	**7**	0.4	**7**	7.00	3.0	**7**	7.00	0.01	0	220
r20-0	**8**	**8**	22759.0	**8**	9896.2	**8**	7.60	9.0	**8**	8.00	0.02	0	202
r20-1	**11**	10	30682.0	**11**	28802.2	10	10.00	10.0	**11**	11.00	0.12	0	1802
r20-2	**14**	12	30823.0	**14**	28802.4	13	13.00	11.0	**14**	14.00	14.41	0	211175
r30-0	**15**	14	49724.0	**15**	28802.8	14	14.00	17.0	**15**	15.00	0.29	0	2566
r30-1	**19**	17	45857.0	**19**	28802.9	18	18.00	19.0	**19**	19.00	0.68	0	4268
r30-2	**22**	20	57427.0	**22**	28801.7	21	21.00	20.0	**22**	22.00	0.28	0	1130
#Improve		0		0		0			0				
#Equal		4		9		4			9				
#Worse		5		0		5			0				

Table 6: Comparison of the proposed algorithm with two recent heuristic algorithms (i.e., HGA (Luo et al., 2024) and SL-VNS (Zhang et al., 2024)) on the small-scale instances of the variant of maximizing the number of items with rotation.

		HGA			SL-VNS			IDEA (this work)				
Instance	BKS	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Δ_{BKS}	#pack
r10-0	**5**	**5**	5.00	0.03	**5**	4.60	0.30	**5**	5.00	0.00	0	211
r10-1	**6**	**6**	6.00	0.04	**6**	6.00	0.40	**6**	6.00	0.01	0	224
r10-2	**7**	**7**	7.00	0.05	**7**	7.00	0.40	**7**	7.00	0.01	0	220
r20-0	**8**	**8**	8.00	0.15	7	7.00	0.80	**8**	8.00	0.02	0	202
r20-1	**11**	**11**	11.00	29.92	**11**	10.80	1.20	**11**	11.00	0.12	0	1802
r20-2	**14**	**14**	13.30	36.97	13	13.00	1.75	**14**	14.00	14.41	0	211175
r30-0	**15**	**15**	15.00	40.19	**15**	15.00	1.95	**15**	15.00	0.29	0	2566
r30-1	**19**	**19**	18.70	56.18	**19**	18.40	2.55	**19**	19.00	0.68	0	4268
r30-2	**22**	**22**	21.95	69.25	**22**	21.60	3.00	**22**	22.00	0.28	0	1130
#Improve		0			0			0				
#Equal		9			7			9				
#Worse		0			2			0				

Table 7: Comparison of the proposed algorithm with two heuristic algorithms (i.e., SA (Bouzid and Salhi, 2020) and VNS (Bouzid and Salhi, 2020)) on the large-scale instances of the variant of maximizing the number of items with rotation.

		SA			VNS			IDEA (this work)				
Instance	BKS	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Δ_{BKS}	#pack
r100-0	48	46	45.20	185.0	45	44.40	110.0	**50**	49.20	66.0	2	38947
r100-1	63	59	58.00	239.0	59	57.80	164.0	**64**	64.00	41.3	1	16260
r100-2	75	71	70.40	275.0	70	69.40	149.0	**77**	76.10	282.3	2	92984
r150-0	78	70	69.40	458.0	73	71.60	463.0	**79**	78.10	155.0	1	28754
r150-1	100	92	90.60	611.0	93	91.80	379.0	**101**	100.90	839.0	1	111109
r150-2	117	110	109.00	745.0	110	108.80	570.0	**118**	118.00	54.5	1	5781
r200-0	104	95	94.60	927.0	98	97.40	805.0	**107**	107.00	1281.9	3	116066
r200-1	134	125	122.60	1267.0	126	124.40	1190.0	**136**	135.30	547.3	2	36916
r200-2	158	148	146.00	1519.0	147	146.40	1579.0	**159**	158.80	932.3	1	54280
#Improve		0			0			9				
#Equal		0			0			0				
#Worse		9			9			0				

Table 8: Comparison of the proposed algorithm with two recent heuristic algorithms (i.e., HGA (Luo et al., 2024) and SL-VNS (Zhang et al., 2024)) on the large-scale instances of the variant of maximizing the number of items with rotation.

		HGA			SL-VNS			IDEA (this work)				
Instance	BKS	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Δ_{BKS}	#pack
r100-0	48	48	47.95	311.0	48	48.00	17.20	**50**	49.20	66.0	2	38947
r100-1	63	63	62.10	502.0	62	62.00	21.40	**64**	64.00	41.3	1	16260
r100-2	75	75	74.00	719.0	75	74.20	41.35	**77**	76.10	282.3	2	92984
r150-0	78	77	75.40	677.0	78	77.40	79.95	**79**	78.10	155.0	1	28754
r150-1	100	99	97.00	1173.0	100	99.20	45.90	**101**	100.90	839.0	1	111109
r150-2	117	116	114.50	1756.0	117	116.40	58.15	**118**	118.00	54.5	1	5781
r200-0	104	103	102.25	1253.0	104	103.80	76.70	**107**	107.00	1281.9	3	116066
r200-1	134	132	130.05	2197.0	134	132.80	168.20	**136**	135.30	547.3	2	36916
r200-2	158	155	153.40	3346.0	158	157.00	102.85	**159**	158.80	932.3	1	54280
#Improve		0			0			9				
#Equal		1			8			0				
#Worse		8			1			0				

1.3 Computational results for maximizing the area of items without rotation

Table 9: Comparison of the proposed algorithm with three previous algorithms (i.e., FSS (López and Beasley, 2018), CPM (Silva et al., 2022) and VNS (López and Beasley, 2018)) on the small-scale instances of the variant of maximizing the area of items without rotation.

		FSS		CPM		VNS			IDEA (this work)				
Instance	BKS	Best	Time(s)	Best	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Δ_{BKS}	#pack
s10–0	**23.9878**	22.9485	2762	**23.9878**	0.2	**23.9878**	23.3642	0.0	**23.9878**	23.5028	0.12	0.0000	32166
s10–1	**37.7471**	36.7126	3402	**37.7471**	0.4	**37.7471**	37.3333	1.0	**37.7471**	37.7471	0.07	0.0000	11864
s10–2	**52.7555**	51.7583	4593	**52.7555**	2.3	**52.7555**	51.9923	1.0	**52.7555**	52.7555	0.00	0.0000	482
s20–0	**64.7463**	54.1054	9412	**64.7463**	33.6	63.7523	62.7630	4.0	63.7430	63.6483	0.94	-1.0033	37671
s20–1	**95.9219**	85.2107	11304	**95.9219**	114.4	94.7706	94.0801	3.0	95.7239	95.7239	14.11	-0.1980	477067
s20–2	**137.2832**	109.8363	7636	**137.2832**	1055.4	132.4100	125.8077	5.0	131.9165	130.9737	58.23	-5.3667	1419112
s30–0	**65.1246**	54.4941	16629	64.3817	28802.0	63.9965	63.4017	4.0	64.7352	64.4022	9.48	-0.3894	216055
s30–1	99.5590	77.5814	14808	97.8908	28800.3	98.1142	97.1655	12.0	**99.6053**	99.2590	67.18	0.0463	1069698
s30–2	134.5194	103.0963	15145	131.6949	28814.1	131.5472	129.5725	6.0	**134.6538**	133.7937	46.72	0.1344	528448
r10–0	**18.4441**	**18.4441**	3292	**18.4441**	0.2	**18.4441**	18.3351	1.0	**18.4441**	18.4441	0.00	0.0000	162
r10–1	**28.9390**	**28.9390**	2992	**28.9390**	0.6	**28.9390**	28.9390	1.0	**28.9390**	28.9390	0.00	0.0000	94
r10–2	**39.4588**	37.6878	4754	**39.4588**	1.2	38.7870	38.7870	1.0	**39.4588**	39.4588	0.00	0.0000	213
r20–0	**45.9961**	43.3885	7227	**45.9961**	104.3	45.1567	44.7210	2.0	45.3621	45.3601	22.30	-0.6340	1350650
r20–1	**72.7850**	63.1643	9791	**72.7850**	28805.0	68.8314	67.4388	3.0	70.6065	70.6065	2.30	-2.1785	100572
r20–2	**97.5007**	84.4446	10601	96.7569	28811.9	91.6368	90.6159	3.0	95.9012	95.4251	66.60	-1.5995	2081882
r30–0	67.4983	60.3570	14011	67.1937	28821.1	64.4689	63.7051	5.0	**67.6012**	67.2801	61.76	0.1029	1458596
r30–1	**104.7251**	85.2113	19786	102.6709	28806.1	102.1196	99.3385	8.0	104.0392	102.9556	84.90	-0.6859	1473801
r30–2	141.0049	103.4802	19470	138.4558	28803.5	137.4149	135.2876	7.0	**141.0108**	139.4474	76.84	0.0059	1046130
#Improve		0		0		0			4				
#Equal		2		15		5			6				
#Worse		16		3		13			8				

Table 10: Comparison of the proposed algorithm with two recent heuristic algorithms (i.e., HGA (Luo et al., 2024) and SL-VNS (Zhang et al., 2024)) on the small-scale instances of the variant of maximizing the area of items without rotation.

		HGA			SL-VNS			IDEA (this work)				
Instance	BKS	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Δ_{BKS}	#pack
s10–0	**23.9878**	**23.9878**	23.9878	0.25	22.6150	22.6150	0.25	**23.9878**	23.5028	0.12	0.0000	32166
s10–1	**37.7471**	**37.7471**	37.7471	0.17	36.7126	36.7126	0.25	**37.7471**	37.7471	0.07	0.0000	11864
s10–2	**52.7555**	**52.7555**	52.7555	0.13	49.2838	49.0052	0.30	**52.7555**	52.7555	0.00	0.0000	482
s20–0	**64.7463**	**64.7463**	63.8517	26.05	59.6314	59.5471	0.75	63.7430	63.6483	0.94	-1.0033	37671
s20–1	**95.9219**	**95.9219**	95.9044	10.02	88.9578	88.9578	0.65	95.7239	95.7239	14.11	-0.1980	477067
s20–2	**137.2832**	**137.2832**	135.4750	33.02	130.9828	129.5658	0.80	131.9165	130.9737	58.23	-5.3667	1419112
s30–0	**65.1246**	**65.1246**	64.9994	29.16	63.9965	63.9965	0.95	64.7352	64.4022	9.48	-0.3894	216055
s30–1	99.5590	99.5590	99.0206	38.22	97.7980	97.4868	2.10	**99.6053**	99.2590	67.18	0.0463	1069698
s30–2	134.5194	134.4015	133.11345	48.09	132.2788	131.1505	1.70	**134.6538**	133.7937	46.72	0.1344	528448
r10–0	**18.4441**	**18.4441**	18.4441	0.03	17.8992	17.8992	0.20	**18.4441**	18.4441	0.00	0.0000	162
r10–1	**28.939**	**28.939**	28.9390	0.06	26.7643	26.7643	0.25	**28.939**	28.939	0.00	0.0000	94
r10–2	**39.4588**	**39.4588**	39.2676	7.58	37.3681	36.5987	0.30	**39.4588**	39.4588	0.00	0.0000	213
r20–0	**45.9961**	**45.9961**	45.7653	11.26	44.4296	44.4296	0.55	45.3621	45.3601	22.30	-0.6340	1350650
r20–1	**72.785**	**72.785**	71.6012	21.47	69.7481	69.5623	0.70	70.6065	70.6065	2.30	-2.1785	100572
r20–2	**97.5007**	**97.5007**	95.7674	26.55	94.3265	93.7633	0.90	95.9012	95.4251	66.60	-1.5995	2081882
r30–0	67.4983	67.4983	66.7790	30.41	66.106	66.1060	1.00	**67.6012**	67.2801	61.76	0.1029	1458596
r30–1	**104.7251**	**104.7251**	103.0498	36.56	101.7709	101.5139	2.15	104.0392	102.9556	84.90	-0.6859	1473801
r30–2	141.0049	141.0049	139.4206	44.71	137.5372	137.094	2.25	**141.0108**	139.4474	76.84	0.0059	1046130
#Improve		0			0			4				
#Equal		14			0			6				
#Worse		4			18			8				

Table 11: Comparison of the proposed algorithm with two heuristic algorithms (i.e., SA (Bouzid and Salhi, 2020) and VNS (Bouzid and Salhi, 2020)) on the large-scale instances of the variant of maximizing the area of items without rotation.

		SA			VNS			IDEA (this work)				
Instance	BKS	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Δ_{BKS}	#pack
s100–0	**301.3806**	293.5578	293.4949	106.0	297.8481	296.3685	72.0	300.5580	300.1525	664.8	-0.8226	1483757
s100–1	**455.4665**	446.0632	443.2703	145.0	449.0898	447.3728	138.0	453.1629	451.3734	1146.8	-2.3036	1859588
s100–2	610.9711	587.8824	586.7831	194.0	600.2514	596.5111	208.0	**611.2694**	609.5736	1224.7	0.2983	1213440
s150–0	**476.1826**	469.7621	468.9567	286.0	472.4785	470.7997	171.0	474.1083	473.0807	1891.5	-2.0743	2174031
s150–1	**721.6048**	707.3694	705.7661	369.0	714.3385	712.3815	312.0	721.0362	719.8731	1184.7	-0.5686	790107
s150–2	**963.1189**	935.9346	935.0466	532.0	948.3443	946.8239	599.0	**963.5195**	960.9966	1622.9	0.4006	818294
s200–0	**602.4622**	593.9045	590.6651	572.0	596.4252	595.3754	237.0	601.8076	600.9707	975.6	-0.6546	603029
s200–1	904.0008	882.0002	879.1137	748.0	889.4874	887.5525	457.0	**906.7465**	904.4098	1454.8	2.7457	544019
s200–2	1211.8357	1178.8631	1176.3317	1064.0	1185.9045	1182.5038	1330.0	**1212.3386**	1210.0431	1833.3	0.5029	435898
r100–0	285.3060	276.2503	274.7736	96.0	281.5851	279.3196	61.0	**286.8923**	284.7452	1249.3	1.5863	3238794
r100–1	433.0382	418.9820	418.9820	143.0	424.7806	424.3101	95.0	**434.6132**	433.9410	1615.3	1.5750	2588311
r100–2	579.1004	556.3578	555.6125	192.0	567.0707	563.8865	156.0	**581.8476**	578.9688	840.9	2.7472	879791
r150–0	393.1389	382.7671	382.6841	250.0	387.9162	386.4728	144.0	**394.9145**	393.9623	1986.0	1.7756	2416976
r150–1	592.6726	576.9804	576.0199	324.0	583.5556	581.0678	258.0	**596.2875**	594.1016	1771.4	3.6149	1273424
r150–2	793.9707	769.4407	765.6101	436.0	780.1788	777.1628	510.0	**796.9769**	795.2090	2299.7	3.0062	994613
r200–0	508.9918	497.3100	497.3100	461.0	503.1195	502.1400	221.0	**511.2864**	509.6866	1869.3	2.2946	1242239
r200–1	766.7049	745.3243	745.0537	734.0	758.4439	754.7137	589.0	**770.7467**	769.0476	1953.3	4.0418	702961
r200–2	1021.4577	994.7474	990.0431	989.0	1003.5017	1001.2816	887.0	**1026.1222**	1023.7889	2268.8	4.6645	492533
#Improve		0			0			13				
#Equal		0			0			0				
#Worse		18			18			5				

Table 12: Comparison of the proposed algorithm with two recent heuristic algorithms (i.e., HGA (Luo et al., 2024) and SL-VNS (Zhang et al., 2024)) on the large-scale instances of the variant of maximizing the area of items without rotation.

		HGA			SL-VNS			IDEA (this work)				
Instance	BKS	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Δ_{BKS}	#pack
s100–0	**301.3806**	**301.3806**	300.7750	126.0	297.5600	295.8297	11.9	300.5580	300.1525	664.8	-0.8226	1483757
s100–1	**455.4665**	**455.4665**	452.1387	207.0	451.0402	448.6063	15.7	453.1629	451.3734	1146.8	-2.3036	1859588
s100–2	610.9711	610.9711	608.3166	293.0	602.1586	601.0954	20.9	**611.2694**	609.5736	1224.7	0.2983	1213440
s150–0	**476.1826**	**476.1826**	474.2942	260.0	470.5036	469.5171	31.8	474.1083	473.0807	1891.5	-2.0743	2174031
s150–1	**721.6048**	**721.6048**	720.1896	422.0	712.1517	711.2916	54.0	721.0362	719.8731	1184.7	-0.5686	790107
s150–2	963.1189	963.1189	961.9878	543.0	953.7799	952.2550	54.2	**963.5195**	960.9966	1622.9	0.4006	818294
s200–0	**602.4622**	**602.4622**	601.3726	374.0	597.6537	595.5345	52.3	601.8076	600.9707	975.6	-0.6546	603029
s200–1	904.0008	904.0008	901.6450	674.0	898.7813	896.9252	58.9	**906.7465**	904.4098	1454.8	2.7457	544019
s200–2	1211.8357	1211.8357	1208.7768	1056.0	1203.3180	1200.9086	119.7	**1212.3386**	1210.0431	1833.3	0.5029	435898
r100–0	285.306	285.306	283.7283	137.0	283.5337	282.1808	15.2	**286.8923**	284.7452	1249.3	1.5863	3238794
r100–1	433.0382	433.0382	430.8815	199.0	428.2655	426.8841	9.2	**434.6132**	433.9410	1615.3	1.5750	2588311
r100–2	579.1004	579.1004	574.8603	329.0	569.5371	568.9367	14.1	**581.8476**	578.9688	840.9	2.7472	879791
r150–0	393.1389	393.1389	391.2339	249.0	390.8242	390.4987	19.4	**394.9145**	393.9623	1986.0	1.7756	2416976
r150–1	592.6726	592.6726	590.0955	432.0	590.7326	588.6003	31.5	**596.2875**	594.1016	1771.4	3.6149	1273424
r150–2	793.9707	793.9707	789.5589	706.0	790.4961	787.3235	42.5	**796.9769**	795.2090	2299.7	3.0062	994613
r200–0	508.9918	508.9918	507.1107	407.0	508.4863	505.2021	45.0	**511.2864**	509.6866	1869.3	2.2946	1242239
r200–1	766.7049	766.7049	764.3574	762.0	766.5361	763.5390	72.1	**770.7467**	769.0476	1953.3	4.0418	702961
r200–2	1021.4577	1021.4577	1016.3950	1495.0	1018.2872	1017.1783	67.4	**1026.1222**	1023.7889	2268.8	4.6645	492533
#Improve		0			0			13				
#Equal		4			0			0				
#Worse		14			18			5				

1.4 Computational results for maximizing the area of items with rotation

Table 13: Comparison of the proposed algorithm with three previous algorithms (i.e., FSS (López and Beasley, 2018), CPM (Silva et al., 2022), and VNS (Bouzid and Salhi, 2020)) on the small-scale instances of the variant maximizing the area of items with rotation.

		FSS		CPM		VNS			IDEA (this work)				
Instance	BKS	Best	Time(s)	Best	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Δ_{BKS}	#pack
r10–0	**19.6702**	**19.6702**	8771.0	**19.6702**	0.2	**19.6702**	19.2633	1.0	18.8619	18.8619	0.07	-0.8083	7551
r10–1	**30.8746**	29.5041	16093.0	**30.8746**	1.1	**30.8746**	29.7208	1.0	29.5041	29.5041	0.13	-1.3705	10879
r10–2	**41.5246**	37.9687	15526.0	**41.5246**	2.6	40.9063	40.3619	1.0	41.1612	41.1612	0.06	-0.3634	3027
r20–0	**47.9336**	43.6850	50558.0	**47.9336**	28802.5	45.4200	45.3303	5.0	47.1228	46.9025	68.81	-0.8108	2260765
r20–1	**72.7850**	63.5279	50013.0	**72.6945**	28817.4	70.8221	68.7228	7.0	71.8253	71.7682	38.95	-0.9597	914689
r20–2	**98.4189**	84.7008	63350.0	**97.9712**	28805.4	95.2162	93.4912	5.0	97.5326	97.4168	16.16	-0.8863	275148
r30–0	**67.8840**	57.9328	69565.0	67.2577	28808.2	66.6329	65.1248	14.0	67.4140	67.3798	45.92	-0.4700	606523
r30–1	**104.8784**	84.3715	82101.0	103.1495	28808.3	100.3020	99.2145	11.0	104.7157	103.3479	86.7	-0.1627	834274
r30–2	**141.6376**	110.3253	39564.0	138.6397	28802.2	137.5277	136.1031	12.0	141.5478	140.3539	52.51	-0.0898	373245
#Improve		0		0		0			0				
#Equal		1		8		2			0				
#Worse		8		1		7			9				

Table 14: Comparison of the proposed algorithm with two recent heuristic algorithms (i.e., HGA (Luo et al., 2024) and SL-VNS (Zhang et al., 2024)) on the small-scale instances of the variant maximizing the area of items with rotation.

		HGA			SL-VNS			IDEA (this work)				
Instance	BKS	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Δ_{BKS}	#pack
r10–0	**19.6702**	**19.6702**	19.6702	0.13	18.4441	18.4441	0.08	18.8619	18.8619	0.07	-0.8083	7551
r10–1	**30.8746**	**30.8746**	30.8746	0.12	28.7839	27.7443	0.08	29.5041	29.5041	0.13	-1.3705	10879
r10–2	**41.5246**	**41.5246**	41.3819	10.72	39.5569	39.5569	0.09	41.1612	41.1612	0.06	-0.3634	3027
r20–0	**47.9336**	46.4452	46.0419	23.46	45.2503	45.2148	0.23	47.1228	46.9025	68.81	-0.8108	2260765
r20–1	**72.7850**	**72.7850**	71.6061	25.41	70.9040	70.4724	0.38	71.8253	71.7682	38.95	-0.9597	914689
r20–2	**98.4189**	**98.4189**	97.5771	28.19	97.9176	96.8554	0.43	97.5326	97.4168	16.16	-0.8863	275148
r30–0	**67.8840**	**67.8840**	67.1898	34.19	67.0593	66.8382	0.56	67.4140	67.3798	45.92	-0.4700	606523
r30–1	**104.8784**	**104.8784**	103.7276	39.12	102.7266	102.2871	0.97	104.7157	103.3479	86.7	-0.1627	834274
r30–2	**141.6376**	**141.6376**	140.0809	46.94	139.1076	138.0495	0.86	141.5478	140.3539	52.51	-0.0898	373245
#Improve		0			0			0				
#Equal		8			0			0				
#Worse		1			9			9				

Table 15: Comparison of the proposed algorithm with two heuristic algorithms (i.e., SA (Bouzid and Salhi, 2020) and VNS (Bouzid and Salhi, 2020)) on the large-scale instances of the variant of maximizing the area of items with rotation.

		SA			VNS			IDEA (this work)				
Instance	BKS	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Δ_{BKS}	#pack
r100–0	286.0224	275.9862	275.9862	167.0	282.3465	280.7830	129.0	**287.5666**	286.9704	1181.9	1.5442	1619532
r100–1	434.7697	422.5904	422.5904	221.0	426.5573	425.9241	145.0	**435.3168**	434.4095	1759.9	0.5471	1569765
r100–2	580.7641	564.5293	561.1405	281.0	571.6369	569.1376	336.0	**583.9064**	581.7455	1163.9	3.1423	624290
r150–0	393.9724	386.3599	386.0969	463.0	389.6020	388.5257	222.0	**395.5893**	394.5173	2017.6	1.6169	1343666
r150–1	594.9417	576.2670	574.8735	577.0	585.5947	583.7210	539.0	**595.6288**	594.3948	1852.4	0.6871	644533
r150–2	796.1322	771.9016	769.5404	752.0	780.7081	778.2186	1073.0	**799.1850**	795.5685	2433.9	3.0528	479858
r200–0	510.9300	500.0271	498.8451	803.0	505.4126	503.2490	503.0	**511.2842**	509.5505	1833.7	0.3542	560675
r200–1	768.3748	748.1296	748.1296	1215.0	757.9592	755.5614	1464.0	**771.3863**	769.5982	2141.1	3.0115	343765
r200–2	1026.7739	994.3341	992.4800	1448.0	1008.3091	1006.3442	2247.0	**1031.1575**	1027.0228	2181.8	4.3836	236466
#Improve		0			0			9				
#Equal		0			0			0				
#Worse		9			9			0				

Table 16: Comparison of the proposed algorithm with two recent heuristic algorithms (i.e., HGA (Luo et al., 2024) and SL-VNS (Zhang et al., 2024)) on the large-scale instances of the variant maximizing the area of items with rotation.

		HGA			SL-VNS			IDEA (this work)				
Instance	BKS	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Δ_{BKS}	#pack
r100–0	286.0224	285.2622	283.9407	175.0	286.0224	284.2914	36.55	**287.5666**	286.9704	1181.9	1.5442	1619532
r100–1	434.7697	434.7697	432.9662	238.0	430.9151	429.6375	43.50	**435.3168**	434.4095	1759.9	0.5471	1569765
r100–2	580.7641	580.7641	576.9916	384.0	579.1329	574.8210	38.90	**583.9064**	581.7455	1163.9	3.1423	624290
r150–0	393.9724	393.9724	392.2238	327.0	393.5067	391.5401	80.90	**395.5893**	394.5173	2017.6	1.6169	1343666
r150–1	594.9417	594.9417	591.8375	543.0	594.0722	592.9682	76.80	**595.6288**	594.3948	1852.4	0.6871	644533
r150–2	796.1322	794.3447	791.35165	849.0	796.1322	794.7256	67.65	**799.1850**	795.5685	2433.9	3.0528	479858
r200–0	510.9300	509.2270	507.8536	567.0	510.9300	508.5306	149.00	**511.2842**	509.5505	1833.7	0.3542	560675
r200–1	768.3748	768.3748	766.4292	1008.0	767.8849	765.3624	116.25	**771.3863**	769.5982	2141.1	3.0115	343765
r200–2	1026.7739	1026.7739	1021.1400	1770.0	1026.3432	1024.2576	141.10	**1031.1575**	1027.0228	2181.8	4.3836	236466
#Improve		0			0			9				
#Equal		6			3			0				
#Worse		3			6			0				

References

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

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

Q. Luo, Y. Rao, P. Yang, and X. Zhao. Hybrid-biased genetic algorithm for packing unequal rectangles into a fixed-size circle. *Computers & Operations Research*, 169:106716, 2024.

A. Silva, L. C. Coelho, M. Darvish, and J. Renaud. A cutting plane method and a parallel algorithm for packing rectangles in a circular container. *European Journal of Operational Research*, 303(1): 114–128, 2022.

T. Zhang, R. Wang, H. Zhang, Q. Liu, and L. Wei. A skyline-based heuristic for orthogonal packing rectangles in a circle. *Computers & Operations Research*, 167:106664, 2024.