





SOLVING RECTANGULAR STRIP PACKING PROBLEM WITH REINFORCEMENT LEARNING: A COMPARATIVE CASE STUDY

XUSHENG ZHAO^{1,2}, YUNQING RAO^{1,2,*}, SIRUI WANG^{1,2} AND NAI LI^{1,2}

Abstract. This paper examines the value-based reinforcement learning method applied to the optimization of the rectangular strip packing problem with three different approaches to define the state and the action. The episode in the reinforcement learning is defined as a round of placement of all the given pieces. We analyze the drawbacks of two previously designed approaches and propose that the state is defined by the stage along with the selected piece. We also record the fitness value of the placement during the packing and design a fitness-based reward. The three methods are evaluated on a group of random packing problems in terms of time consumption by searching for the particular state, memory consumption by recording the past state, and the efficiency of packing optimization. The results show that the proposed reinforcement learning with fitness-based reward delivers a good comprehensive performance. The proposed method is also tested on a few well-known benchmark problems, and the results indicate that the proposed method could be an effective tool. We discuss the similarities and differences between the reinforcement learning method and the local search method.

Mathematics Subject Classification. 68T20, 90-05.

Received November 26, 2023. Accepted March 24, 2025.

1. INTRODUCTION

Packing problems arise in the industrial context where one or more large objects are supposed to be divided into smaller ones so that the material utilization is maximized [36]. They share a common structure that the assignment between the large objects and small pieces should follow the constraints: the small pieces lie within the large object, and the small pieces do not overlap [19].

A typical classification that distinguishes the packing problems into refined types is relied on the shape of the small pieces [41], namely regular pieces (rectangles, boxes, circles, etc.) or irregular ones. In the regular packing problems, the pieces are commonly assumed to be placed orthogonally, while the geometric structure of the irregular packing problems is much more complex than that of the regular packing problems [24]. Since the large object is continuous and the placement position is infinite, determining the most appropriate position for a piece in an irregular packing problem implies higher computational complexity, and simply modifying a

Keywords. Packing problem, reinforcement learning, Q -learning, optimization, fitness-based reward.

¹ School of Mechanical Science & Engineering, Huazhong University of Science and Technology, Wuhan 430074, Hubei, P.R. China.

² State Key Laboratory of Intelligent Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan 430074, Hubei, P.R. China.

*Corresponding author: ryq@hust.edu.cn

successful approach for a regular packing problem may not be sufficient to solve an irregular packing problem [5].

Rectangular packing problems make up a substantial part of the regular packing problems. They were proposed in the 1950s [13] and have been researched over the years regarding positioning techniques and piece ordering approaches. A significant number of problem variants with problem-specific objectives and settings have been raised, among which one of the most documented variant types is the rectangular packing problem with multiple bins, where a set of bins with fixed sizes is provided for the packing of the pieces. The objective can either be the minimization of the number of used bins where the bins have adequate supplies [39] or the maximization of profit where the number of bins is predefined and the pieces with different values have to be selected and then placed [35]. Another popularly studied variant type is the rectangular strip packing problem, in which the bin is initialized with an infinite dimension that has to be minimized through the proper arrangement of the pieces [25].

In recent years, various attempts to automate the design of algorithms have been driven by many real-world problems. As reinforcement learning becomes an effective tool for decision-making [40], it receives an increasing amount of attention, and many reinforcement learning-based approaches have been proposed. Kollar *et al.* [23] formulated the robot planning to explore an unknown environment as a constrained optimization and used reinforcement learning to improve the accuracy of the built map. Mishra *et al.* [31] focused on building an online resource allocation mechanism to maximize the social welfare of the provider. They proposed a novel reinforcement learning algorithm that adopts a novel monotonic reward function.

Furthermore, as an integration of reinforcement learning with deep neural networks, deep reinforcement learning (DRL) has been adopted in certain fields to make appropriate solution predictions. Viquerat *et al.* [37] proposed the first application of DRL to airfoil shape optimization. They proved that adequate reward and a trained neural network could generate the optimal shape automatically. Walraven *et al.* [38] proposed a reinforcement learning-based method to optimize traffic flow, in which they used Q -learning to decide the maximum allowed driving speed on a highway to reduce traffic congestion. Mocanu *et al.* [32] explored the benefits of DRL in the smart grid for the first time, using a hybrid method combined with reinforcement learning and deep learning to optimize the energy management system scheduling. The other fields of application include automated optimization of chemical reactions [49], optimization of inherent stochasticity of bioprocess system and plant-model mismatches [34], near-optimal control of continuous-time system for autonomous ground vehicle [28], dynamic sleeping control for reduction of mobile network energy consumption [26].

While reinforcement learning has a range of applications and achievements, we notice that there are relatively few literature reports about its application to the packing problems. In this paper, we studied the methods of reinforcement learning applied to the packing optimization with different definitions of the state and the action. We focus on the two-dimensional rectangular strip packing problem and propose a reinforcement learning method with fitness-based reward.

The remaining part of this paper is organized as follows. We review the related work in the literature about the rectangular packing problems in Section 2, especially for the heuristic methods. In Section 3, we present the decoding algorithm based on the best fit placement rule. In Section 4, we propose a new approach to define the state and the action in the reinforcement learning method after analyzing two relevant approaches, and we design a fitness-based reward in the reinforcement learning method. Then, we present the case studies on the reinforcement learning methods and test the proposed method on the benchmark problems in Section 5, and we draw the conclusions in Section 6.

2. RELATED WORK

Most solutions to the rectangular packing problems fall into either one of the two classes: exact methods [18], where mathematic programming is the predominant underlying tool to calculate optimal solutions, and heuristic methods [33], where human expert experience and skills are concretized for the generation of better packing schemes, usually achieving near-optimal solutions.

Exact methods normally formulate the rectangular packing problems as linear programming problems and solve them by branch-and-bound, column generation, or dynamic programming [29]. Macedo *et al.* [29] extended an arc-flow model used for a one-dimensional bin packing problem and explored its behavior on the two-dimensional rectangular cutting stock problem. Côté *et al.* [11] proposed an innovative exact method based on Benders' decomposition. The pieces are cut into unit-width slices and packed contiguously on the sheet in the master problem, and they are reconstructed by fixing the vertical positions in the slave problem. Kenmochi *et al.* [22] studied an exact branch-and-bound algorithm for the perfect rectangular packing problem, where the pieces are required to perfectly fill the sheet without waste. Then, they transformed the algorithm to solve the rectangular strip packing problem.

Although the optimal solutions can be guaranteed, exact methods usually require a large amount of computation. Thus, heuristic methods are predominantly adopted to solve the rectangular packing problems. A well-known example is the sequential placement heuristic method, which formulates the packing by the placement of rectangles piece by piece, selecting the next piece to be packed and determining the exact position for the placement according to a given evaluation rule [27]. In this way, the solution to a particular rectangular packing problem is encoded as the sequence of the pieces. Then, through a decoding algorithm with a designed placement rule, the sequence of the pieces can be transformed into the corresponding layout.

Sliding the piece from the top right corner to the bottommost and leftmost available position (BL) is a classical placement rule proposed by Jakobs [20]. Then, the placement rule was enhanced by Hopper *et al.* [15] by filling the empty regions beyond the bottom left corner. As preordering the pieces in a decreasing size manner enlarges the chance of prioritizing the placement of the most valuable piece, it has been extensively adopted and frequently used as a preprocessing step before the packing procedure. Based on the preordering, two placement rules, namely the first fit decreasing (FFD) and the best fit decreasing (BFD), were proposed by Crainic *et al.* [12]. FFD places the piece in the first bin. If the bin does not have enough accommodation, a new bin is then added and used. In contrast, BFD places the piece in the most suitable bin. Burke *et al.* [8] proposed a best fit (BF) placement rule. In their algorithm, the large object is defined by a set of linear elements, and each placement examines the lowest linear element and evaluates the best piece to be placed. Similar to the evaluation mechanism in BF, another placement rule assesses the appropriateness of a placement by the smooth degree of the residual space left by the placement [14, 39]. The region in a bin is represented by a group of empty maximal spaces, which indicate the available positions and are continually relocated with each placement.

Several sequential placement heuristics choose local search to improve solution qualities. Simulated annealing [9, 44] and genetic algorithm [6, 21] are the most reported local search approaches. The local search is usually implemented in an iterative way of perturbations on the current solution. Then, regenerated solutions are transformed by the decoding algorithm and subsequently determined whether to be retained or to be discarded through given rules. Wauters *et al.* [42] proposed a shaking procedure, its basic principle is to switch between the placement on the bottom left corner and the placement on the top right corner. Wei *et al.* [43] proposed an iterative doubling binary search, the algorithm allocates more computational effort to the latter phase of the searching to increase the possibility of identifying better solutions.

As machine learning presents considerable achievements in pattern recognition and natural language processing, recent research has been conducted with the motivation of replacing the heuristic policy with the deep neural network or learning the best behavior automatically. Zhao *et al.* [46] formulated the online 3D bin packing problem as a constrained Markov Decision Process (MDP). They proposed a constrained reinforcement learning method to solve the problem with a prediction-and-projection scheme. Hottung *et al.* [17] proposed a deep learning heuristic tree search for the container pre-marshalling problem, using a deep neural network to learn strategies and lower bounds by analyzing existing optimal solutions to similar instances. An exploratory study was presented by Zhao *et al.* [47], designing a lowest centroid placement rule and applying Q -learning as the optimization tool to improve the rectangular packing quality. Zhao *et al.* [48] modeled the consecutive piece packing as a MDP, and they proposed the reverse updating of the Q -value and used Q -learning to optimize the sequence and orientation of the rectangles.

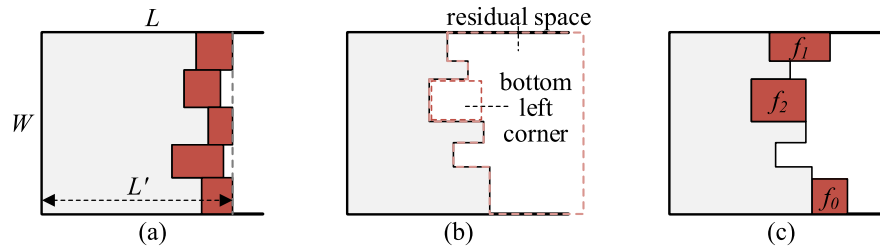


FIGURE 1. Problem definition and placement rule with fitness value. (a) Problem definition. (b) Placement position. (c) Fitness value examples.

3. DEFINITION AND RECTANGULAR PLACEMENT

We briefly introduce the description of the rectangular strip packing problem and provide the decoding algorithm with the best fit (BF) placement rule in this section.

In this paper, the definition of the rectangular strip packing problem is the same as that in [48]. A rectangular sheet is provided with its two dimensions denoted by W and L , respectively. W is a predefined constant, and L represents an infinite value, as shown in Figure 1a. A group of rectangular pieces are supposed to be placed inside the rectangular sheet without overlap. The number of the pieces is denoted by n , and the packing length along the L direction after the placement of all the pieces is denoted by L' . The objective of the rectangular strip packing problem is the minimization of L' . In this paper, each piece is placed with a fixed orientation (*i.e.*, rotation is not considered).

A typical spatial state of the rectangular sheet during packing is presented in Figure 1b. The placement starts from the left side of the sheet and leaves the residual space on the sheet as a polygon region, which is subsequently used for the placement of the remained pieces. We define that the bottom left (the leftmost and the bottommost) corner of the residual space is chosen as the placement position.

We denote the length and width of each piece by l_i and w_i , respectively, where l_i is the dimension along the L direction, and w_i is the dimension along the W direction. The BF placement rule [8, 43] is adopted, which selects the piece through an evaluation rule considering the size matching between the piece and the residual space. We present an example in Figure 1c to illustrate the evaluation rule. If the piece fully matches a particular corner of the residual space, the placement would have a fitness value of 2, such as the placement marked with f_2 in Figure 1c, in which both the dimensions of the piece match that of the corner (*i.e.*, the piece size (l_i, w_i) is exactly the same as the corner). Similarly, one dimension matching would generate a fitness value of 1, and none dimension matching would generate a fitness value of 0. The associated placements of fitness values 1 and 0 are marked with f_1 and f_0 in Figure 1c, respectively.

The piece in the placement of the maximum fitness value is selected by the BF placement rule. If there is a tie, the first piece in the sequence with the maximum fitness value is chosen. The decoding algorithm is presented in Algorithm 1. Let the selected piece in the placement be p and the associated fitness value of the placement be f , and we use a list denoted by L_{pf} to record the selected piece along with the fitness value.

The placement of each piece (lines 2–6) is executed through the position selection (line 3) and the piece selection (line 4) each time, and we record the piece in the order of the successive placement (line 5). The location of the bottom left corner of the residual space on the rectangular sheet is stored as the position of the piece. As each placement would reshape the residual space on the rectangular sheet, which determines the placement in the later steps, thus, the packing length L' and the residual space are updated after the placement of each selected piece (line 6). The packing length L' and the list L_{pf} are returned at the end of the decoding algorithm.

Algorithm 1: Decoding algorithm.

Input: sequence and sizes of the pieces
Output: packing scheme

- 1 Initialize the rectangular sheet
- 2 **for** $i = 1$ to n **do**:
- 3 Choose the bottom left corner
- 4 Select the piece corresponding to the maximum fitness value
- 5 Record the selected piece p and the fitness value f in L_{pf}
- 6 Place the piece and update the rectangular sheet
- 7 **return** L', L_{pf}

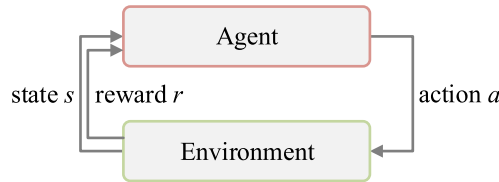


FIGURE 2. Interaction between agent and environment.

4. REINFORCEMENT LEARNING-BASED OPTIMIZATION

As introduced in Section 1, the application of reinforcement learning to the packing problems has seldom been studied, in this section, we apply reinforcement learning (RL) as a tool to optimize the sequence of the pieces in the rectangular strip packing problem. We analyze the characteristics of two different definitions of the state and the action in RL and summarize their limits. Then, we propose a reinforcement learning method with fitness-based reward (RL-FR).

4.1. Value-based reinforcement learning framework

Reinforcement learning formulates an interaction between the agent and the environment. The agent takes an action in the environment, after which the environment changes its state and sends the agent a reward as feedback. The interaction is shown in Figure 2. The best policy is iteratively learned by the agent to reach the maximum amount of reward.

The interaction is described as a Markov Decision Process (MDP), and value-based reinforcement uses the state-action value (Q -value) to define the expected reward stimulated by an action adopted at a given state [40]. We denote the state, action, and reward by s , a , and r , respectively. The update of the Q -value is computed as in equation (1):

$$Q(s, a) = Q(s, a) + \alpha * [r(s, a) + \gamma * \max_{a'} Q(s', a') - Q(s, a)], \quad (1)$$

where α represents the learning rate, and γ represents the discount factor. Both α and γ are within the range of $[0, 1]$.

An episode in RL is a round of exploration by the agent from a specified start state to a specified end state, and all the states, actions, and rewards experienced by the agent within an episode constitute the trajectory of the episode. As the rectangles are placed piece by piece and the result of the packing is evaluated by the packing length on the rectangular sheet, a natural idea is to define an episode in RL as a round of placement of the given n rectangles. Through episodes of exploration, the value of $Q(s, a)$ is updated, which in turn prompts the agent to take the action associated with the maximum $Q(s, a)$ at each state (ϵ -greedy exploration policy), leading the agent to achieve the maximal expected return. The framework of reinforcement learning applied to the packing optimization is illustrated in Figure 3.

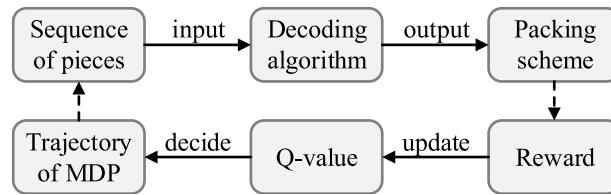


FIGURE 3. Procedure of value-based RL applied to the packing optimization.

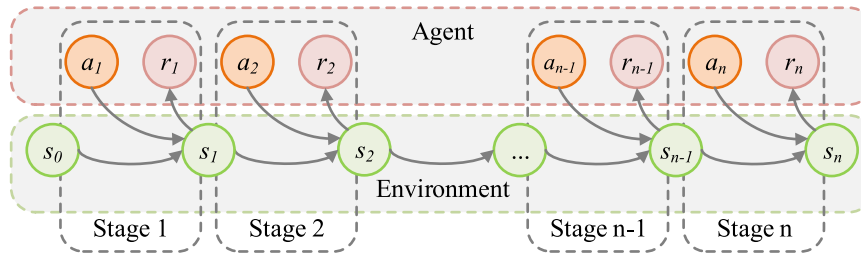


FIGURE 4. An episode in RL.

A trajectory indicates a sequence of the pieces in a round of placement. The Q -value determines the policy of RL, which further determines the states and the actions along the trajectory of the MDP. Then, the sequence of the pieces associated with the trajectory is transformed by the decoding algorithm into the packing scheme. Through a problem-specific reward mechanism, the reward is returned to the agent, updating the Q -value subsequently.

4.2. Optimization through Q -learning with fitness-based reward

We divide the episode into n stages, with each stage representing a transition of the state through the adoption of an action. We denote the state, action, and reward in stage i by s_i , a_i , and r_i , respectively, and we denote the trajectory of the Markov Decision Process (MDP) by τ . Then, $\tau = s_0, a_1, s_1, r_1, a_2, s_2, r_2, \dots, a_n, s_n, r_n$. The state s_0 represents the start of the episode, *i.e.*, the start of the generation of a piece sequence, and the state s_n defines the end of the episode, at which point the generation of the piece sequence is completed. The trajectory of one episode is illustrated in Figure 4. Note that each trajectory reflects a sequence of the n pieces, which is a solution to the rectangular strip packing problem.

We notice that when applied to solve the rectangular strip packing problem, the definition of the state and the action in RL plays an important role in the optimization performance. Previous studies (namely, [47, 48]) provide two approaches to define the state and the action.

The first approach defines both the action and the state as the piece (selected to construct the piece sequence). For example, suppose the piece p is selected in stage i . Then, $a_i = p$. When the action a_i is taken, the state is transformed from s_{i-1} to s_i , and $s_i = a_i = p$. At the end of the episode, the series of actions indicates a sequence of the n pieces. A non-zero reward is given only at the last state s_n . The reward is set to the reciprocal of the packing length L' associated with the piece sequence, since a smaller L' represents a more compact layout, and the agent should receive more favorable feedback.

From a statistical standpoint, all the potential trajectories might be employed by chance as the sequence of the pieces, with which the pieces are packed and the associated packing quality is returned. During the searching, the trajectory with the largest amount of return can be located, which leads to the result of the best packing layout. However, using the same representation of the state and the action (*i.e.*, both represented by a piece) may mislead the update of the value of $Q(s, a)$. We present an example to describe this drawback as follows.

We assume a packing problem with eight pieces to be solved. The pieces are denoted by p_1 – p_8 , the actions are denoted by a_1 – a_8 , and the states are denoted by s_0 – s_8 . Suppose a sequence of $[p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8]$ is generated in the first episode. Then, $Q(s_7, a_8)$, *i.e.*, $Q(p_7, p_8)$, will be updated into a positive value after the first episode, and $Q(s_{i-1}, a_i)$, $i \in [1, 7]$, will remain the value of zero. In the second episode, suppose the piece p_7 is selected in the first stage. Then, $s_1 = a_1 = p_7$, and the pieces allowed to be chosen are within the pieces p_1 – p_6 and p_8 in the second stage. Since $Q(p_7, p_8) > 0$, and $Q(p_7, p_i) = 0$, where $i \in [1, 7]$, to maximize the expected return, the agent guided by the value of $Q(s, a)$ tends to select the piece p_8 under the ϵ -greedy exploration policy. In this case, the priority of the selection of p_8 becomes unreasonable guidance to the agent, because the selection of piece p_8 after p_7 in the eighth stage of the first episode reflects a different spatial condition compared to the selection of piece p_8 after p_7 in the second stage of the second episode. In the eighth stage of the first episode, the selection of piece p_8 is implemented after the selection of seven pieces, which corresponds to the case that the rectangular sheet is already filled with seven pieces and the piece p_8 should be positioned in the residual space beyond the seven pieces. However, in the second stage of the second episode, the selection of piece p_8 is implemented after the selection of piece p_7 , which corresponds to the case that the rectangular sheet is only filled with piece p_7 .

The second approach shares the same definition of the action and the reward mechanism as the first approach. In the second approach, the state is initialized with a set of -1 and defined as the set of pieces already selected in the previous stages of the episode. We reuse the example of the eight pieces presented above to introduce the generation of the state. We also suppose the sequence of $[p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8]$ is generated in an episode. Then, the states along the trajectory of the episode are listed as follows: $s_0 = [-1, -1, \dots, -1]$, $s_1 = [p_1, -1, \dots, -1]$, $s_2 = [p_1, p_2, \dots, -1]$, \dots , $s_8 = [p_1, p_2, \dots, p_8]$.

The definition avoids the undesirable drawback of the first approach. However, as the state distinguishes each piece that has already been selected or to be selected, the number of the state might be extremely large, resulting in substantial memory consumption in the implementation of the second approach. We analyze its defect as follows. Since each piece can be selected in any stage of an episode, if we only consider the state at the end of each episode, the number of the states reaches $n!$ (*i.e.*, the number of permutations of the n pieces). While there are intermediate states during each episode, namely, the states in the form of $[p_x, p_y, \dots, -1]$, $x, y \in [1, n]$, the former part of which consists of the particular pieces and the latter part consists of the -1 values. Then, the number of all the possible states is larger than $n!$, which might limit the application of the second approach to large-scale problems due to the memory usage for storing these states. Furthermore, as the state representation contains n elements, it would take non-negligible time to find the given state among a large number of states before updating the value of $Q(s, a)$ (especially for the large-scale problems), which might be another defect of the second approach.

To overcome the limits of the two approaches of definition, we define that the state is represented by the stage along with the selected piece, *i.e.*, if we denote stage i by t_i and the piece selected in stage i by p_i , the state is then represented by (t_i, p_i) . The unreasonable guidance to the agent by the Q -value updated between stages could be avoided by recording the number of the stages, and this way of definition also reduces the substantial memory consumption for storing specific states.

We also notice that when the problem scale increases, the guidance of history experience of the agent will be weakened at the beginning of an episode if the reward is only given in the last stage of an episode, since the discount factor γ diminishes the expected return from the last stage to the first stage. We take the case where α is set to 1 as an example. Then, the update of the Q -value is simplified as in equation (2):

$$Q(s, a) = r(s, a) + \gamma * \max_{a'} Q(s', a'). \quad (2)$$

For simplicity, we use a variable r' to represent the reward given in the last stage of an episode, and we suppose the agent chooses the particular trajectory of the episode n times to explain the change of the value of $Q(s, a)$. Then, after the first episode, the value of $Q(s, a)$ in the last stage is presented in equation (3).

$$Q(s_{n-1}, a_n) = r(s_{n-1}, a_n) = r'. \quad (3)$$

After the second episode, the value of $Q(s, a)$ in the $n - 1$ th stage is presented in equation (4).

$$\begin{aligned} Q(s_{n-2}, a_{n-1}) &= r(s_{n-2}, a_{n-1}) + \gamma * \max Q(s_{n-1}, a_n) \\ &= \gamma * \max Q(s_{n-1}, a_n) \\ &= \gamma * r'. \end{aligned} \quad (4)$$

Similarly, when the n episodes are finished, the value of $Q(s, a)$ in the x th stage is presented in equation (5).

$$Q(s_{x-1}, a_x) = \gamma^{n-x} * r', \quad (5)$$

where $x \in [1, n - 1]$. Therefore, the value of $Q(s, a)$ gradually decreases to zero from the last stage to the first stage of the episode. For the same reason, all the values of $Q(s, a)$ are equal to zero at the beginning of each episode, where the ϵ -greedy exploration policy becomes ineffective, leading to a completely random selection of pieces at the beginning of each episode. Hence, we propose a fitness-based reward within the stages. We denote the reward associated with the fitness value in stage i by r_{fi} , and the reward computed through the packing length L' by $r_{L'}$. Note that $r_{L'}$ only exists in the last stage of each episode, while a non-zero reward r_{fi} can be returned in any stage of the episode if the associated placement generates a non-zero fitness value. Then, the reward in stage i is computed as in equation (6):

$$r_i = \begin{cases} r_{fi}, & \text{if } i \in [1, n - 1], \\ r_{fi} + r_{L'}, & \text{if } i = n. \end{cases} \quad (6)$$

The preordering [43] and the reverse updating of the $Q(s, a)$ value [48] are also adopted in this paper. The reverse updating is used to sufficiently employ the experience of each episode, and it is implemented in the form of first generating a trajectory and then reversely updating the value of $Q(s, a)$ along the trajectory at the end of each episode. The preordering is used to improve the quality of the initial solution and initialize the value of $Q(s, a)$. The preordering involves sorting the pieces in descending order of the piece length, piece width, piece perimeter, piece area, and the maximum value between piece length and piece width, respectively. For each of the five orders, we use it as the sequence of the pieces and pack the rectangles with the decoding algorithm (Algorithm 1), and we can obtain the list L_{pf} , which contains the packed pieces along with the fitness values. Then, we replace each piece in the states and actions along the trajectory sequentially with each packed piece in L_{pf} , and we compute each reward along the trajectory with formula (6). In this way, the trajectory reflects the sequence in which the pieces are truly placed on the rectangular sheet in the round of placement, and we reversely update the value of $Q(s, a)$ along the trajectory. When the five rounds of updates associated with the preordering are completed, the value of $Q(s, a)$ is initialized, after which the exploration of RL begins, and the value of $Q(s, a)$ updates continuously in each episode (in the same manner as in the preordering) while searching for the optimal solution.

The reinforcement learning with fitness-based reward (RL-FR) is presented in Algorithm 2. We denote the number of episodes by n_{epi} and the variable that controls the episode by $iter$, and we use a list denoted by L_s and another list denoted by L_a to store the states and the actions along the trajectory τ , respectively. The sequence of the pieces is denoted by S . The minimum packing length found during the implementation is denoted by L'_{min} , with the corresponding sequence denoted by S_{best} .

The decoding algorithm (Algorithm 1) is called as a tool to obtain the layout of the pieces and the list L_{pf} associated with a given piece sequence (lines 2, 12). The preordering is presented in lines 1–3, and the procedure of an episode in RL is presented in lines 4–16. Within each episode, first, we reset the state list L_s and action list L_a , and then we reinitialize the trajectory τ to the initial state s_0 (lines 5, 6). Each episode is composed of n stages (lines 7–10). As the list L_a represents the piece (selected to construct the piece sequence), at the end of the episode, a copy of L_a is made as the piece sequence S (line 11). Then, the list L_{pf} is returned by the decoding algorithm with sequence S as the input, and the pieces in the states and actions along the trajectory τ are updated in the same manner as in the preordering. The rewards along the trajectory τ are then computed

Algorithm 2: Reinforcement learning with fitness-based reward (RL-FR).

Input: sizes of the pieces**Output:** optimized packing scheme

```

1 Compute the preordering of the pieces
2 Implement the packing, obtain  $L'$  and  $L_{pf}$  (call Algorithm 1)
3 Initialize  $Q(s, a)$  and  $S_{best}$  with the results of the preordering
4 for  $iter = 1$  to  $n_{epi}$  do:
5   Reset the  $L_s$  and  $L_a$  to  $\emptyset$ 
6   Reset the current state to  $s_0$ 
7   for  $i = 1$  to  $n$  do:
8     Select the action  $a_i$  according to  $\epsilon$ -greedy exploration policy
9     Take  $a_i$  and change the current state to  $s_i$ 
10    Record  $s_i$  in  $L_s$  and record  $a_i$  in  $L_a$ 
11    Generate the piece sequence  $S$ 
12    Implement the packing, obtain  $L'$  and  $L_{pf}$  (call Algorithm 1)
13    Compute the reward  $r_i, i \in [1, n]$ , along the trajectory  $\tau$ 
14    Reversely update  $Q(s_i, a_i), i \in [1, n]$ , along the trajectory  $\tau$ 
15    if  $L' < L'_{min}$ :
16       $L'_{min} \leftarrow L', S_{best} \leftarrow S$ 
17 return  $L'_{min}$ 

```

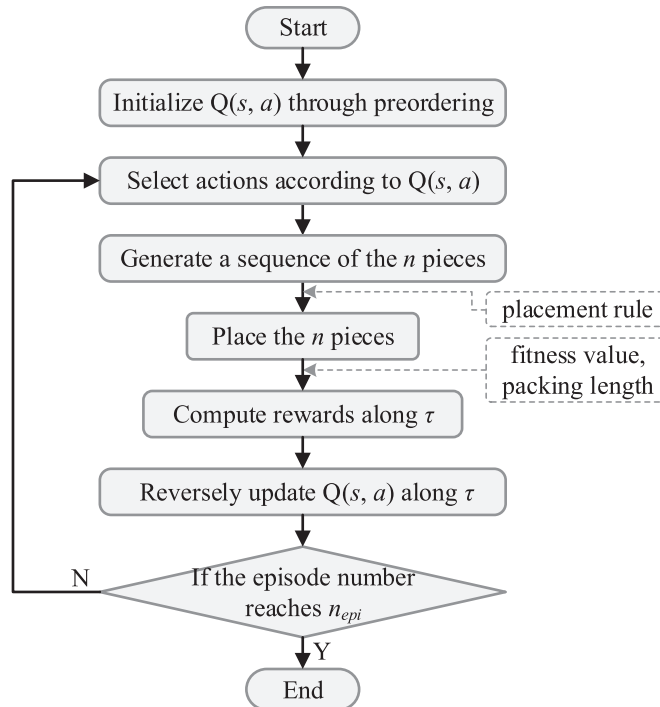


FIGURE 5. Flowchart of optimization through RL.

(line 13). All the states, actions, and rewards along the trajectory τ are used to reversely update the value of $Q(s, a)$ (line 14). The procedure of RL-FR is presented in Figure 5.

TABLE 1. Random problem data.

Group	n	Subgroup A	Subgroup B	Subgroup C
P1	50	$W = 50$, size $\in [6, 8]$	$W = 100$, size $\in [10, 18]$	$W = 500$, size $\in [60, 80]$
P2	100	$W = 50$, size $\in [5, 8]$	$W = 100$, size $\in [8, 18]$	$W = 500$, size $\in [40, 80]$
P3	200	$W = 50$, size $\in [4, 8]$	$W = 100$, size $\in [6, 18]$	$W = 500$, size $\in [20, 80]$
P4	500	$W = 50$, size $\in [3, 8]$	$W = 100$, size $\in [4, 18]$	$W = 500$, size $\in [10, 80]$

TABLE 2. Memory and time consumption of RL methods.

Problem	RL-1		RL-2		RL-3	
	Memory (GB)	Time (s)	Memory (GB)	Time (s)	Memory (GB)	Time (s)
P1-A	<0.1	0.06	<0.1	0.14	<0.1	0.07
P1-B	<0.1	0.05	<0.1	0.15	<0.1	0.05
P1-C	<0.1	0.06	<0.1	0.14	<0.1	0.07
P2-A	<0.1	0.11	0.2	0.35	<0.1	0.14
P2-B	<0.1	0.11	0.2	0.37	<0.1	0.16
P2-C	<0.1	0.10	0.2	0.34	<0.1	0.15
P3-A	<0.1	0.22	0.6	1.03	<0.1	0.51
P3-B	<0.1	0.19	0.6	0.99	0.1	0.50
P3-C	<0.1	0.21	0.6	1.01	0.1	0.47
P4-A	<0.1	0.69	3.9	4.43	0.7	1.72
P4-B	<0.1	0.52	4.0	4.28	0.8	1.87
P4-C	<0.1	0.63	3.8	4.36	0.9	1.91

5. COMPUTATIONAL CASE STUDIES AND COMPARATIVE ANALYSIS

In this section, we study the performance of the reinforcement learning (RL) method applied to the optimization of the packing. The performance of RL is evaluated in terms of memory consumption by the storage of the state, time consumption by searching for the given state, and the packing optimization efficiency. The methods presented in this section are coded by Python and tested on a PC with a 2.3 GHz CPU and 16 GB RAM.

5.1. Cases studies on the reinforcement learning

Four groups of random problems represented by P1–P4 with designed data properties are used to test the RL methods. The four groups are presented in Table 1, each of which is composed of three subgroups, namely A, B, and C. The width of the rectangular sheet is set to the value provided in the table, and both the two dimensions of each piece are randomly generated integers within the given size range.

From groups P1 to P4, we set the number of pieces to 50, 100, 200, and 500, respectively, representing the packing problems from small scale to large scale. We also enlarge the size range by lowering the minimum size limit, which indicates that the pieces might have a larger variety of shapes in group P4. For each group, from subgroup A to subgroup C, we increase both the width of the rectangular sheet and the endpoint values of the size range. Then, more large pieces with relatively less identical shapes might be generated in subgroup C of each group. As the pieces in these four groups are randomly generated, and the factors of problem scale and piece size have been considered within the settings, the four groups of random problems can represent a range of packing problems.

We denote the reinforcement learning method with the first approach of definition (described in the previous section) by RL-1, and that with the second approach of definition by RL-2. We set the number of episodes to 1000, and we record the memory cost for the running and the time consumed by searching for the given state in RL-1, RL-2, and RL-FR. The results are listed in Table 2. For simplicity, each random problem is recorded with an abbreviation (*e.g.*, P1-A represents the problem related to the subgroup A in P1).

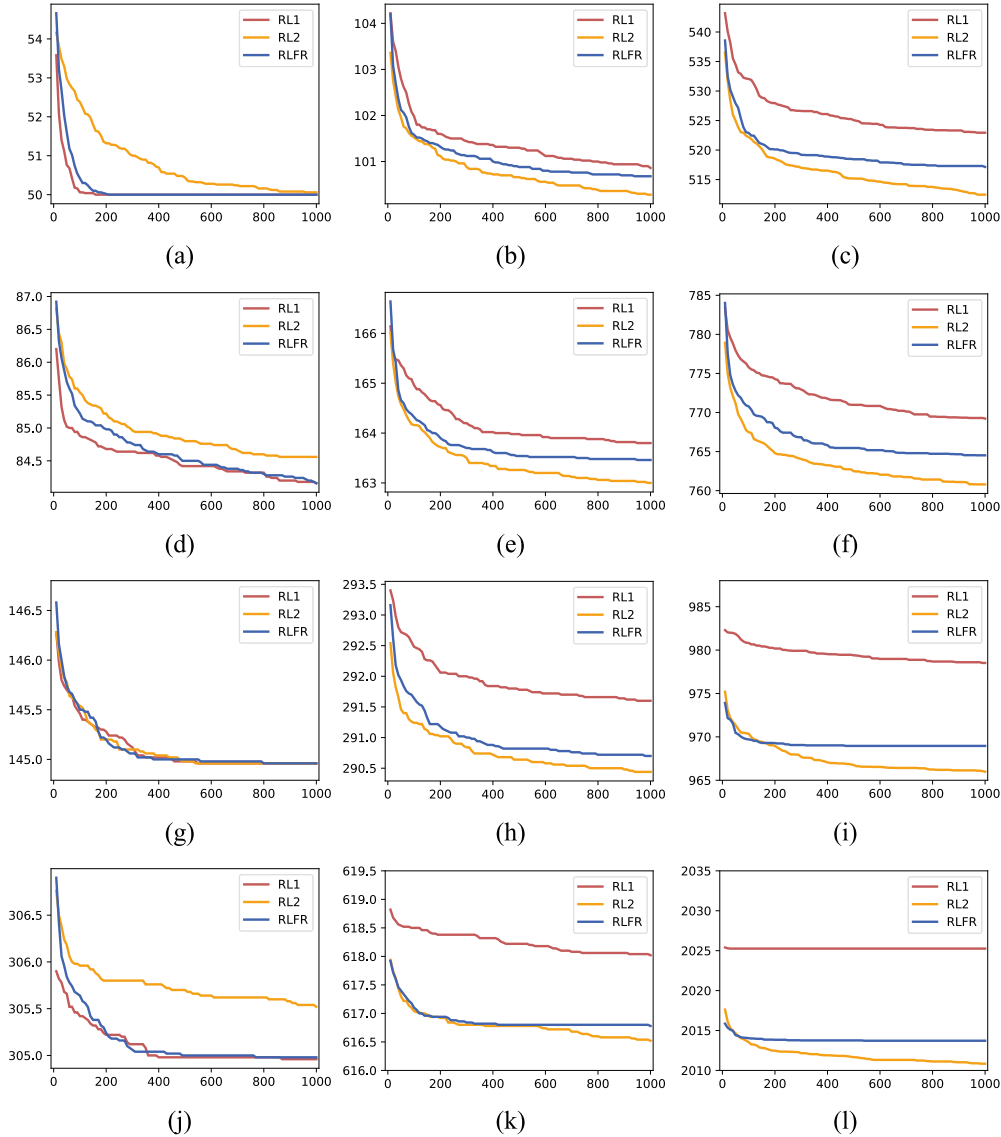


FIGURE 6. Performance of the RL methods. (a) P1-A. (b) P1-B. (c) P1-C. (d) P2-A. (e) P2-B. (f) P2-C. (g) P3-A. (h) P3-B. (i) P3-C. (j) P4-A. (k) P4-B. (l) P4-C.

We also record the minimum packing length found by the three RL methods during the 1000 episodes. The recorded performances are the average of 50 runs, and the results are presented in Figure 6, where the x -axis denotes the number of the episodes and the y -axis denotes the minimum packing length found by the RL method. The decreasing rate of the curve reflects the optimization efficiency of the relevant RL method.

5.2. Benchmark problem evaluation

A few well-known benchmark problems are taken for the evaluation of the reinforcement learning with fitness-based reward (RL-FR), including the zero-waste datasets C [16] and N [8], and the non-zero waste datasets cgcut [10], gcut [2], ngcut [3], beng [4], and Class [7, 30]. We run RL-FR 10 times, each running is set within a time limit of 60s, and we record the result by gap, $\text{gap} = (L' - LB) / LB$, where LB represents the lower bound.

TABLE 3. Results on problem C.

	Instance			Best gap (%) in setting time limit			
	n	W	LB	GRASP	SRA	IA	RL-FR
C11	16	20	20	0	0	0	0
C12	17	20	20	0	0	0	0
C13	16	20	20	0	0	0	0
C21	25	40	15	0	0	0	0
C22	25	40	15	0	0	0	0
C23	25	40	15	0	0	0	0
C31	28	60	30	0	0	0	0
C32	29	60	30	3.3	3.3	0	0
C33	28	60	30	0	0	0	0
C41	49	60	60	1.7	1.7	1.7	1.7
C42	49	60	60	1.7	1.7	1.7	1.7
C43	49	60	60	1.7	1.7	0	0
C51	73	60	90	1.1	0	0	1.1
C52	73	60	90	1.1	0	0	1.1
C53	73	60	90	1.1	1.1	0	1.1
C61	97	80	120	1.7	0.8	0	0.8
C62	97	80	120	0.8	0.8	0	0.8
C63	97	80	120	1.7	0.8	0	0.8
C71	196	160	240	1.7	0.4	0.4	0.4
C72	197	160	240	1.3	0.4	0.4	0.4
C73	196	160	240	1.3	0.4	0.4	0.4
Average gap				0.96	0.62	0.22	0.49

We compare the best performance found during the running of RL-FR with that of GRASP [1], SRA [45], and IA [44] in Tables 3–6, as these three algorithms have achieved quite excellent results for the rectangular strip packing problem with fixed piece orientation in the literature. The best results are marked in bold.

5.3. Analysis

From the results in Table 2, Figure 6, and Tables 3–6, we observe that the proposed reinforcement learning with fitness-based reward (RL-FR) could be an efficient tool applied to the optimization of the rectangular strip packing problem.

Comparison between three RL methods

As presented in Figure 6, the three RL methods, namely RL-1, RL-2, and RL-FR, show significantly different performance. RL-1 outperforms RL-2 on the problems P1-A, P2-A, and P4-A and appears to be equivalent to RL-2 on the problem P3-A, while it becomes obviously inferior to RL-2 on the other random problems. We consider that this is caused by the properties of the piece dimensions. The two dimensions of each piece in subgroup A of the four groups of problems are values generated in a limited interval, which is extended in subgroups B and C. Then, the pieces in subgroup C can be more heterogeneous than that in subgroups A and B. Since subgroup C represents the more general cases of packing, where the curve of RL-2 shows the fastest decreasing rate, RL-2 appears to be the most effective one among the three RL methods. The performance of RL-FR is superior to that of RL-2 on subgroup A and slightly inferior to that of RL-2 on subgroups B and C.

The results in Table 2 indicate the defects of RL-2. Since in RL-2, the state is defined by all the packed pieces, the number of the past states becomes extremely large as the problem scale increases, especially for the problems P4-A, P4-B, and P4-C. Therefore, the running of RL-2 takes a significant amount of memory for the storage and continual update of all the states. Additionally, Searching for a particular state among a large number of states results in a non-negligible time consumption compared with RL-1 and RL-FR.

TABLE 4. Results on problem N.

	Instance			Best gap (%) in setting time limit			
	n	W	LB	GRASP	SRA	IA	RL-FR
N1	10	40	40	0	0	0	0
N2	20	30	50	0	0	0	0
N3	30	30	50	2	0	0	0
N4	40	80	80	1.3	0	0	0
N5	50	100	100	2	0	0	0
N6	60	50	100	1	0	0	0
N7	70	80	100	1	0	0	0
N8	80	100	80	1.3	1.3	1.3	1.3
N9	100	50	150	0.7	0	0	0
N10	200	70	150	0.7	0	0	0
N11	300	70	150	0.7	0	0	0
N12	500	100	300	1.3	0.3	0.3	0.3
N13	3152	640	960	0.5	0.1	0.1	0.1
Average gap				0.96	0.13	0.13	0.13

From a comprehensive perspective, RL-FR avoids the drawbacks of RL-1 and RL-2, and we can draw the conclusion that RL-FR is an effective RL-based tool for the optimization of the packing quality.

Comparison between reinforcement learning and local search

Tables 4 and 5 prove that RL-FR is comparable to the excellent local search methods. RL-FR finds the optimal results of all the problems of N and beng, and it also finds the best results on the majority of the problems of cgcut, gcut, and ngcut. The average gap of RL-FR is considerably close to that of the local search. From Tables 3 and 6, we note that RL-FR performs slightly worse than the local search. However, its performance is still satisfactory. It provides the optimal solutions to the small-scale problems of C11–C33, and its performance is close to SRA and better than GRASP.

We consider that the minor inferiority of RL-FR might be caused by the architecture of the reinforcement learning and local search applied to the packing problem. The reinforcement learning method and the local search method share the same idea of stepwise improvement on the solution quality, achieving a balance between the exploitation and the exploration. The major difference between them might be the solution generation mechanism. RL models the generation of a solution as a multistage decision process, where the chosen action in each stage of the episode determines the direction of the trajectory and constitutes a part of the solution, thereby constructing the solution step by step. While the local search holds a predefined number of solutions, from which new solutions are generated through perturbations and used to replace the old ones. Additionally, the discrepancies between the solutions generated in successive iterations in RL might be greater than the discrepancies between the solutions generated in successive iterations in the local search. Since the ϵ -greedy exploration policy can theoretically choose any available action in any stage of an episode, the set of actions in the current episode might differ significantly from the set of actions in the previous episode, leading to the generation of a completely different solution in the current iteration in RL compared to the previous iteration. While in the local search, the discrepancies between the solutions generated in the current iteration and the solutions generated in the previous iteration might be minor, since these discrepancies are caused by perturbations. Therefore, compared with local search, it might be harder for RL to focus the exploration on the neighborhood of the best solution that has been found.

When applied to optimize the sequence of the pieces, RL might have a potential advantage over the local search in terms of preserving the exploration experience, since each episode of exploration updates the value of $Q(s, a)$, which resembles an experience storage and subsequently determines the exploration in the later

TABLE 5. Results on problems cgcut, gcut, ngcut, and beng.

	Instance			Best gap (%) in setting time limit			
	n	W	LB	GRASP	SRA	IA	RL-FR
cgcut1	16	10	23	0	4.3	4.3	0
cgcut2	23	70	63	3.2	3.2	3.2	3.2
cgcut3	62	70	636	3.9	3.9	3.9	3.9
gcut1	10	250	1016	0	0	0	0
gcut2	20	250	1133	5.1	4.8	4.8	5.1
gcut3	30	250	1803	0	0	0	0
gcut4	50	250	2934	2.3	2.4	2.2	2.4
gcut5	10	500	1172	8.6	8.6	8.6	8.6
gcut6	20	500	2514	4.5	4.3	4.3	4.7
gcut7	30	500	4641	1.1	1.1	1.1	1.1
gcut8	50	500	5703	3.7	3	2.8	3.4
gcut9	10	1000	2022	14.6	14.6	14.6	14.6
gcut10	20	1000	5356	11.4	11.4	11.4	11.4
gcut11	30	1000	6537	5.5	5	5	6
gcut12	50	1000	12 522	17.3	17.3	17.3	17.3
gcut13	32	3000	4772	4.7	4.1	4.2	4.5
ngcut1	10	10	23	0	0	0	0
ngcut2	17	10	30	0	3.3	3.3	0
ngcut3	21	10	28	0	0	0	0
ngcut4	7	10	20	0	0	0	0
ngcut5	14	10	36	0	0	0	0
ngcut6	15	10	29	6.9	6.9	6.9	6.9
ngcut7	8	20	20	0	0	0	0
ngcut8	13	20	32	3.1	6.3	6.3	9.4
ngcut9	18	20	49	2	4.1	4.1	2
ngcut10	13	30	80	0	0	0	0
ngcut11	15	30	50	4	4	4	4
ngcut12	22	30	87	0	0	0	0
beng1	20	25	30	0	3.3	3.3	0
beng2	40	25	57	0	0	0	0
beng3	60	25	84	0	0	0	0
beng4	80	25	107	0	0	0	0
beng5	100	25	134	0	0	0	0
beng6	40	40	36	0	0	0	0
beng7	80	40	67	0	0	0	0
beng8	120	40	101	0	0	0	0
beng9	160	40	126	0	0	0	0
beng10	200	40	156	0	0	0	0
Average gap				2.68	3.05	3.04	2.86

episodes. Another possible advantage might be the convenience of extension based on deep learning, as the expected return obtained by taking an action in a specific state can be forecasted by neural networks, trained neural networks might identify a better action to take at each state from a global perspective.

Future research

In this paper, the elements that define the state and the action in the MDP only include the episode stage and the selected piece, while we notice that the geometry structure of the packing problem still contains some critical information that has not been used in the modeling of the MDP, for example, the contour of the residual

TABLE 6. Results on problem Class.

	Instance			Best gap (%) in setting time limit			
	n	W	LB	GRASP	SRA	IA	RL-FR
Class1	20	10	60.3	1.8	1.7	1.7	1.7
	40	10	121.6	0.2	0.2	0.2	0.2
	60	10	187.4	0.6	0.6	0.6	0.6
	80	10	262.2	0.2	0.2	0.2	0.2
	100	10	304.4	0.2	0.2	0.1	0.2
Class2	20	30	19.7	0.5	0.5	0	0.5
	40	30	39.1	0	0	0	0
	60	30	60.1	0.3	0	0	0
	80	30	83.2	0.1	0	0	0
	100	30	100.5	0.1	0	0	0
Class3	20	40	157.4	3.9	3.9	3.9	4.2
	40	40	328.8	1.6	1.3	1.2	1.6
	60	40	500	1.3	1	1	1.2
	80	40	701.7	1.1	1	1	1.1
	100	40	832.7	0.9	0.5	0.5	0.6
Class4	20	100	61.4	3.1	3.4	2.9	3.7
	40	100	123.9	1.9	1.3	1	1.6
	60	100	193	1.9	0.9	0.8	1.2
	80	100	267.2	1.8	0.7	0.5	1.2
	100	100	322	1.6	0.5	0.3	0.9
Class5	20	100	512.2	4.2	4.2	4.3	4.4
	40	100	1053.8	2	1.9	1.9	2.1
	60	100	1614	2	1.7	1.7	2
	80	100	2268.4	1	0.9	0.8	0.9
	100	100	2617.4	1.3	1.1	1	1
Class6	20	300	159.9	4.6	4.9	4.9	4.9
	40	300	323.5	3.1	2.6	2.3	3.5
	60	300	505.1	2.9	2.2	2	3.4
	80	300	699.7	2.7	2	1.6	2.5
	100	300	843.8	2.5	1.7	1.4	2.5
Class7	20	100	490.4	2.3	2.3	2.3	2.3
	40	100	1049.7	0.9	0.9	0.9	0.9
	60	100	1515.9	0.9	0.9	0.9	0.9
	80	100	2206.1	0.7	0.7	0.7	0.7
	100	100	2627	0.6	0.7	0.6	0.7
Class8	20	100	434.6	5.5	5.2	5	5.4
	40	100	922	3.5	2.9	2.4	3.4
	60	100	1360.9	3.2	2.5	2.1	3.1
	80	100	1909.3	3.3	2.4	2	2.5
	100	100	2362.8	3.1	2	1.7	2.4
Class9	20	100	1106.8	0	0	0	0
	40	100	2189.2	0.1	0.1	0.1	0.1
	60	100	3410.4	0	0	0	0
	80	100	4578.6	0.2	0.2	0.2	0.2
	100	100	5430.5	0.1	0.1	0.1	0.1
Class10	20	100	337.8	3.8	3.5	3.5	3.9
	40	100	642.8	3.4	3	2.8	3.3
	60	100	911.1	2.6	2.3	2	2.4
	80	100	1177.6	2.7	2.1	1.8	2.1
	100	100	1476.5	2.4	1.8	1.5	1.8
Average gap				1.77	1.49	1.37	1.68

space on the rectangular sheet, the properties of the unpacked pieces during one episode, and the placement location of a specific piece. There exists the probability that a more precise and problem-related representation of the state and the action would better formulate the packing process and deliver better results. Future work could continue to explore the modeling of the MDP of the rectangular strip packing problems or the other problem variants.

6. CONCLUSIONS

We apply the value-based reinforcement learning method to the optimization of the rectangular strip packing problem and examine three different approaches to define the state and the action. In the reinforcement learning, the episode is defined as the placement procedure of all the given pieces. We note that the previously designed approach that defines the state only by the piece might mislead the update of the Q -value, and another proposed approach that defines the state by the set of already packed pieces can suffer from the large memory consumption for the state storage and non-negligible time consumption for searching for the given state. Then, we propose the reinforcement learning method with the state defined by the stage along with the selected piece, and we design a fitness-based reward. We evaluate the three reinforcement learning methods on four groups of random packing problems. The results show that the proposed method delivers a good comprehensive performance in terms of time consumption by searching for the particular state, memory consumption by recording the past state, and the efficiency of packing optimization. The computational results on the benchmark problems C, N, cgcut, gcut, ngcut, beng, and Class also indicate that the proposed method could be an effective tool.

FUNDING

This work was supported by the National Natural Science Foundation of China (Grant No. 51975231).

DATA AVAILABILITY STATEMENT

The research data associated with this article are included in the article.

REFERENCES

- [1] R. Alvarez-Valdés, F. Parreño and J.M. Tamarit, Reactive grasp for the strip-packing problem. *Comput. Oper. Res.* **35** (2008) 1065–1083.
- [2] J.E. Beasley, Algorithms for unconstrained two-dimensional guillotine cutting. *J. Oper. Res. Soc.* **36** (1985) 297–306.
- [3] J.E. Beasley, An exact two-dimensional non-guillotine cutting tree search procedure. *Oper. Res.* **33** (1985) 49–64.
- [4] B.-E. Bengtsson, Packing rectangular pieces – a heuristic approach. *Comput. J.* **25** (1982) 353–357.
- [5] J.A. Bennell and J.F. Oliveira, The geometry of nesting problems: a tutorial. *Eur. J. Oper. Res.* **184** (2008) 397–415.
- [6] J.A. Bennell, L.S. Lee and C.N. Potts, A genetic algorithm for two-dimensional bin packing with due dates. *Int. J. Prod. Econ.* **145** (2013) 547–560.
- [7] J.O. Berkey and P.Y. Wang, Two-dimensional finite bin-packing algorithms. *J. Oper. Res. Soc.* **38** (1987) 423–429.
- [8] E.K. Burke, G. Kendall and G. Whitwell, A new placement heuristic for the orthogonal stock-cutting problem. *Oper. Res.* **52** (2004) 655–671.
- [9] E.K. Burke, G. Kendall and G. Whitwell, A simulated annealing enhancement of the best-fit heuristic for the orthogonal stock-cutting problem. *Inf. J. Comput.* **21** (2009) 505–516.
- [10] N. Christofides and C. Whitlock, An algorithm for two-dimensional cutting problems. *Oper. Res.* **25** (1977) 30–44.
- [11] J.-F. Côté, M. Dell’Amico and M. Iori, Combinatorial benders’ cuts for the strip packing problem. *Oper. Res.* **62** (2014) 643–661.
- [12] T.G. Crainic, G. Perboli and R. Tadei, Extreme point-based heuristics for three-dimensional bin packing. *Inf. J. Comput.* **20** (2008) 368–384.
- [13] K. Eisemann, The trim problem. *Manage. Sci.* **3** (1957) 279–284.
- [14] J.F. Gonçalves and M.G.C. Resende, A biased random-key genetic algorithm for the unequal area facility layout problem. *Eur. J. Oper. Res.* **246** (2015) 86–107.
- [15] E. Hopper and B. Turton, A genetic algorithm for a 2D industrial packing problem. *Comput. Ind. Eng.* **37** (1999) 375–378.
- [16] E. Hopper and B.C.H. Turton, An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *Eur. J. Oper. Res.* **128** (2001) 34–57.
- [17] A. Hottung, S. Tanaka and K. Tierney, Deep learning assisted heuristic tree search for the container pre-marshalling problem. *Comput. Oper. Res.* **113** (2020) 104781.
- [18] M. Iori, V.L. De Lima, S. Martello, F.K. Miyazawa and M. Monaci, Exact solution techniques for two-dimensional cutting and packing. *Eur. J. Oper. Res.* **289** (2021) 399–415.

- [19] M. Iori, V.L. de Lima, S. Martello and M. Monaci, 2DPackLib: a two-dimensional cutting and packing library. *Optim. Lett.* **16** (2022) 471–480.
- [20] S. Jakobs, On genetic algorithms for the packing of polygons. *Eur. J. Oper. Res.* **88** (1996) 165–181.
- [21] K. Kang, I. Moon and H. Wang, A hybrid genetic algorithm with a new packing strategy for the three-dimensional bin packing problem. *Appl. Math. Comput.* **219** (2012) 1287–1299.
- [22] M. Kenmochi, T. Imamichi, K. Nonobe, M. Yagiura and H. Nagamochi, Exact algorithms for the two-dimensional strip packing problem with and without rotations. *Eur. J. Oper. Res.* **198** (2009) 73–83.
- [23] T. Kollar and N. Roy, Trajectory optimization using reinforcement learning for map exploration. *Int. J. Rob. Res.* **27** (2008) 175–196.
- [24] A.A.S. Leao, F.M.B. Toledo, J.F. Oliveira, M.A. Carravilla and R. Alvarez-Valdés, Irregular packing problems: a review of mathematical models. *Eur. J. Oper. Res.* **282** (2020) 803–822.
- [25] S.C.H. Leung and D. Zhang, A fast layer-based heuristic for non-guillotine strip packing. *Expert Syst. App.* **38** (2011) 13032–13042.
- [26] J. Liu, B. Krishnamachari, S. Zhou and Z. Niu, DeepNap: data-driven base station sleeping operations through deep reinforcement learning. *IEEE Int. Things J.* **5** (2018) 4273–4282.
- [27] E. López-Camacho, H. Terashima-Marin, P. Ross and G. Ochoa, A unified hyper-heuristic framework for solving bin packing problems. *Expert Syst. App.* **41** (2014) 6876–6889.
- [28] Y. Lu, W. Li, X. Zhang and X. Xu, Continuous-time receding-horizon reinforcement learning and its application to path-tracking control of autonomous ground vehicles. *Opt. Control App. Methods* **44** (2023) 1129–1147.
- [29] R. Macedo, C. Alves and J.M.V. De Carvalho, Arc-flow model for the two-dimensional guillotine cutting stock problem. *Comput. Oper. Res.* **37** (2010) 991–1001.
- [30] S. Martello and D. Vigo, Exact solution of the two-dimensional finite bin packing problem. *Manage. Sci.* **44** (1998) 388–399.
- [31] P. Mishra and A. Moustafa, Reinforcement learning based monotonic policy for online resource allocation. *Future Gener. Comput. Syst.* **138** (2023) 313–327.
- [32] E. Mocanu, D.C. Mocanu, P.H. Nguyen, A. Liotta, M.E. Webber, M. Gibescu and J.G. Slootweg, On-line building energy optimization using deep reinforcement learning. *IEEE Trans. Smart Grid* **10** (2018) 3698–3708.
- [33] Ó. Oliveira, D. Gamboa and E. Silva, An introduction to the two-dimensional rectangular cutting and packing problem. *Int. Trans. Oper. Res.* **30** (2023) 3238–3266.
- [34] P. Petsagkourakis, I.O. Sandoval, E. Bradford, D. Zhang and E.A. del Rio-Chanona, Reinforcement learning for batch bioprocess optimization. *Comput. Chem. Eng.* **133** (2020) 106649.
- [35] M. Russo, A. Sforza and C. Sterle, An improvement of the knapsack function based algorithm of gilmore and gomory for the unconstrained two-dimensional guillotine cutting problem. *Int. J. Prod. Econ.* **145** (2013) 451–462.
- [36] E. Silva, J.F. Oliveira and G. Wäscher, 2DCPackGen: a problem generator for two-dimensional rectangular cutting and packing problems. *Eur. J. Oper. Res.* **237** (2014) 846–856.
- [37] J. Viquerat, J. Rabault, A. Kuhnle, H. Ghraieb, A. Larcher and E. Hachem, Direct shape optimization through deep reinforcement learning. *J. Comput. Phys.* **428** (2021) 110080.
- [38] E. Walraven, M.T.J. Spaan and B. Bakker, Traffic flow optimization: a reinforcement learning approach. *Eng. App. Artif. Intell.* **52** (2016) 203–212.
- [39] Y. Wang and L. Chen, Two-dimensional residual-space-maximized packing. *Expert Syst. App.* **42** (2015) 3297–3305.
- [40] Q. Wang and C. Tang, Deep reinforcement learning for transportation network combinatorial optimization: a survey. *Knowl.-Based Syst.* **233** (2021) 107526.
- [41] G. Wäscher, H. Haußner and H. Schumann, An improved typology of cutting and packing problems. *Eur. J. Oper. Res.* **183** (2007) 1109–1130.
- [42] T. Wauters, J. Verstichel and G.V. Berghe, An effective shaking procedure for 2D and 3D strip packing problems. *Comput. Oper. Res.* **40** (2013) 2662–2669.
- [43] L. Wei, W.-C. Oon, W. Zhu and A. Lim, A skyline heuristic for the 2D rectangular packing and strip packing problems. *Eur. J. Oper. Res.* **215** (2011) 337–346.
- [44] L. Wei, H. Qin, B. Cheang and X. Xu, An efficient intelligent search algorithm for the two-dimensional rectangular strip packing problem. *Int. Trans. Oper. Res.* **23** (2016) 65–92.
- [45] S. Yang, S. Han and W. Ye, A simple randomized algorithm for two-dimensional strip packing. *Comput. Oper. Res.* **40** (2013) 1–8.

- [46] H. Zhao, Q. She, C. Zhu, Y. Yang and K. Xu, Online 3D bin packing with constrained deep reinforcement learning, in Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 35. PKP Publishing Services Network (2021) 741–749.
- [47] X. Zhao, Y. Rao and J. Fang, A reinforcement learning algorithm for the 2D-rectangular strip packing problem. *J. Phys. Conf. Ser.* **2181** (2022) 012002.
- [48] X. Zhao, Y. Rao, R. Meng and J. Fang, A Q-learning-based algorithm for the 2D-rectangular packing problem. *Soft Comput.* **27** (2023) 12057–12070.
- [49] Z. Zhou, X. Li and R.N. Zare, Optimizing chemical reactions with deep reinforcement learning. *ACS Cent. Sci.* **3** (2017) 1337–1344.



Please help to maintain this journal in open access!

This journal is currently published in open access under the Subscribe to Open model (S2O). We are thankful to our subscribers and supporters for making it possible to publish this journal in open access in the current year, free of charge for authors and readers.

Check with your library that it subscribes to the journal, or consider making a personal donation to the S2O programme by contacting subscribers@edpsciences.org.

More information, including a list of supporters and financial transparency reports, is available at <https://edpsciences.org/en/subscribe-to-open-s2o>.