Hindawi
Mathematical Problems in Engineering
Volume 2021, Article ID 6653586, 15 pages
https://doi.org/10.1155/2021/6653586

Research Article

A Flexible Reinforced Bin Packing Framework with Automatic Slack Selection

Ting Yang,[1] Fei Luo,[1] Joel Fuentes,[2] Weichao Ding,[1] and Chunhua Gu[1]

[1]*School of Information and Engineering, East China University of Science and Technology, Shanghai 200237, China*
[2]*Joel Fuentes is with Department of Computer Science and Information Technologies, Universidad del Bío-Bío, Chillán, Chile*

Correspondence should be addressed to Fei Luo; luof@ecust.edu.cn

Received 21 December 2020; Revised 9 April 2021; Accepted 10 May 2021; Published 19 May 2021

Academic Editor: José García

The slack-based algorithms are popular bin-focus heuristics for the bin packing problem (BPP). The selection of slacks in existing methods only consider predetermined policies, ignoring the dynamic exploration of the global data structure, which leads to nonfully utilization of the information in the data space. In this paper, we propose a novel slack-based flexible bin packing framework called reinforced bin packing framework (RBF) for the one-dimensional BPP. RBF considers the RL-system, the instance-eigenvalue mapping process, and the reinforced-MBS strategy simultaneously. In our work, the slack is generated with a reinforcement learning strategy, in which the performance-driven rewards are used to capture the intuition of learning the current state of the container space, the action is the choice of the packing container, and the state is the remaining capacity after packing. During the construction of the slack, an instance-eigenvalue mapping process is designed and utilized to generate the representative and classified validate set. Furthermore, the provision of the slack coefficient is integrated into MBS-based packing process. Experimental results show that, in comparison with fit algorithms, MBS and MBS', RBF achieves state-of-the-art performance on BINDATA and SCH_WAE datasets. In particular, it outperforms its baseline MBS and MBS', averaging the number increase of optimal solutions of 189.05% and 27.41%, respectively.

1. Introduction

As a classical discrete combinatorial optimization problems [1, 2], the bin packing problem (BPP) [3, 4] aims to minimize the number of used bins to pack items and it is NP-hard [5, 6]. In the past few decades, four main approaches have been extensively studied to resolve the BPP, such as exact approaches [7–9], approximation algorithms [4, 10], heuristic algorithms, and metaheuristic algorithms [11, 12]. The exact algorithms typically prune the lower bound information to address the BPP, which is suitable for small-scale instances. When the scale of datasets increases, the BPP becomes challenging to the approximation algorithms. The implementation of the metaheuristic algorithms is difficult due to the rigorous requirements on parameter adjustment and calculation complexity [13]. In the contrast, the heuristic algorithm is a popular bin packing method due to its efficiency on solving NP-hard problems.

As one of typical heuristic algorithms, the minimum bin slack (MBS) is particular useful to problems where an optimal solution requires most of the bins, if not all, to be exactly filled [14]. It is also useful for solving the problems where the sum of requirements of items is less than or equal to twice the bin capacity. In MBS, the selection of the packing sequence of the items is based on a predetermined strategy, which ignores the sampling deviation between the data of the items to be packed and cannot explore the global data space.

Therefore, the MBS algorithm may quickly fall into local optimal solutions ignoring the exploration of global item space in the training process. In the stage of iterative training, the deviation of the locally optimal solution is accumulated continuously, and the global optimal solution space is shifted steadily. It may result in a significant difference between the algorithm's packing result and the

optimal solution, which may lead to the failure to achieve the desired performance [14].

In order to solve the problems of MBS described above, we propose a reinforced bin packing framework, dubbed RBF, to resolve the BPP, where a reinforcement learning (RL) method, i.e., the Q-learning algorithm, is exploited to select a high-quality slack for the packing process. The RBF treats Q-learning as a prior data spatial information detector. To ingeniously select data samples as representatives of the datasets, it explores an intrinsic spatial distribution of sample bins by interacting with the environment and estimating the optimal slack of the global bins. The learned slacks are finally exploited in the improved MBS algorithm to pack items.

The proposed RBF can be distinguished from previous work in terms of the following characteristics:

(1) The reinforced learning algorithm is exploited to generate the slack automatically, which is further integrated to the MBS algorithm. With high-quality slacks, automatically, rather than manual design or empirical speculation, our method prevents the bin packing process from falling into a local optimal solution, which is a quite challenging problem especially for the large-scale dataset.

(2) The instance-eigenvalue mapping function is introduced to efficiently select representative and classified validate set of the input instances based on their similarity. This enables RBF to reduce the learning cost while generating a dynamic slack during the packing process.

The rest of this paper is organized as follows. Related work is presented in Section 2. The formulation of the BPP is depicted in Section 3. In Section 4, we briefly overview the design of the RBF and then detail its key components, such as the RL-system, the reinforced-MBS strategy, and the instance mapping process. Experimental results and theoretical analyses are presented in Section 5. Finally, conclusions are drawn in Section 6.

2. Related Work

Existing methods that address the BPP can be roughly classified into four major categories: the exact approaches [15–20], the approximation algorithms [4, 21, 22], the heuristic algorithms [14, 23–27], and the metaheuristic algorithms [11–13, 28–30]. In recent years, the RL-based methods [31–35] have also been proposed to resolve the BPP.

2.1. Exact Approaches. The exact approaches establish mathematical model and obtain the optimal solution of the problem by solving the mathematical model through optimization algorithms. CPLEX [36] solved the problem with mixed integer programming. Polyakovskiy and M'Hallah [15] characterized the properties of the two-dimensional nonoriented BPP with due dates, which packed a set of rectangular items, and experimentally proved that a tight lower bound enhanced an existing bound on maximum lateness for 24.07% of the benchmark instances. Since the quality of the solution depends on whether the model is reasonable or not, they are only applicable to small-scale instances.

Subsequent improvements focused on reconsideration about constraints in a novelty manner. Chitsaz et al. [18] proposed an algorithm to separate the subcontour elimination constraints of fractional solutions to solve production, inventory, and inbound transportation decision problems. The inequalities and separation procedures were used in a branch-and-cut algorithm. A similar idea was proposed in Mara's work [20], where an exact algorithm was proposed based on the classic ϵ-constraint method. The method addressed N single-objective problems by using reduction with test sets instead of an optimizer. Besides, one classic method belonging to this group was the arc-flow formulation method [9] which represented all the patterns in a very compact graph based on an arc-flow formulation with side constraints and can be solved exactly by general-purpose mixed integer programming solvers. Generally, when the scale of the problem becomes larger, the phenomenon of "combinatorial explosion" will lead to heavy computational overhead in the optimization process. It is difficult for the exact algorithm to be applied to large-scale combinatorial optimization problems.

2.2. Approximation Algorithms. Approximation algorithms are popular because their time complexity is polynomial, while they do not guarantee to find the optimal solution. Typical approximation algorithms include greedy algorithms and local search. Based on the observation and arrangement of Earth observation satellites, the authors in [21] proposed an index-based multiobjective local search to solve multiobjective optimization problems. Kang and Park [37] considered the problem of variable-size bin packaging and described two greedy algorithms. The objective was to minimize the total cost of used bins when the unit size cost per bin did not increase as the bin size increased. Moreover, the survey [38] presented an overview of approximation algorithms for the classical BPP and pointed out that although the approximation algorithms are universal, there is always a gap between the solution and the optimal solution under the polynomial time complexity. However, approximation algorithms are commonly subjective to polynomial time and cannot give guarantees of solutions.

2.3. Heuristic Algorithms. Heuristic algorithms are based on the intuitive and empirical design. Several new heuristics for solving the one-dimensional bin packing problem are presented [39]. Coffman and Garey [10] reviewed various heuristic algorithms, such as NF (Next Fit), FF (First Fit), BF (Best Fit), and WF (Worst Fit) [23]. These are typical online packing algorithms [40, 41] and are called as fit algorithms. Their corresponding offline packing algorithms are NFD [24], FFD, BFD, and WFD [23], which differ significantly from online packing algorithms in which offline algorithms rely on overall information for sorting. The fit algorithms,

for example, FF, WF, and BF, give priority for further packing to the bins that have already been packed with items, and a new bin will be activated only when there are no suitable nonempty bins for the current item. The strategy adopted by the fit algorithms ensures that each arriving item can always find a bin to be accommodated. However, it cannot guarantee that the item is the target item for optimum solution under the current situation. To address the issue, Cupta and Ho [14] proposed MBS, which mainly centered on bins and tried its best to find the collection of items that fill the bins. One problem with this method is that its sequence selection strategy often falls in the local region of the input space, which makes it hard for accurate estimation of the slack. Thus, it may result in a locally optimal solution. To solve the above problems, some methods have been proposed. Fleszar and Hindi [42] found that one effective hybrid method integrated perturbation MBS' and a good set of lower bounds into variable neighbourhood search (VNS), so as to improve its ability in reasonably short processing times. However, due to the complexity and uncertainty of combinatorial optimization problems, heuristic algorithms that rely on empirical criteria are not always reliable.

2.4. Metaheuristic Algorithms. Metaheuristic algorithms are widely used to find optimal solutions for solving problem of BPP. Early typical representatives include genetic algorithms [28] and simulated annealing algorithms [29]. The former is a promising tool for the BPP and one significant improvement is mainly used: grouping genetic algorithms (GGAs). Dokeroglu and Cosar [43] proposed a set of robust and scalable hybrid parallel algorithms. In GGA-CGT (grouping genetic algorithm with controlled gene transmission) [44], the transmission of the best genes in the chromosomes was promoted while keeping the balance between selection pressure and population diversity. Kucukyilmaz and Kiziloz proposed island-parallel GGA (IPGGA) in [45]. It realized the choice of communication topology, determined the migration and assimilation strategies, adjusted the migration rate, and exploited diversification technologies. Crainic et al. [46] proposed a two-level tabu search for the three-dimensional BPP by reducing the size of the solution space. Kumar and Raza [47] incorporated the concept of Paretos optimality for the BPP with multiple constraints and then proposed a family of solutions along the trade-off surface. However, due to the lack of particle diversity in the later stage of genetic algorithms as well as PSO algorithms, premature convergence always occurs [28].

2.5. RL-Based Methods. Machine learning has been extensively studied to resolve the NP-hard BPP by scholars in recent years. Ruben Solozabal's model tackled the BPP with RL. It trained multistacked long short-term memory cells to perform a recurrent neural network agent, which could embed information from the environment. The performance of the model was just comparable to the FF algorithm when introducing neural network overhead. Inspired by Pointer Network [48], a deep learning technology was successfully applied to learn and optimize the placing order of items [32], solved the classic TSP problem [33], and tackled the 3D BPP. These methods utilized RL to ensure the solution would not converge to local optimum, while they attempted to exploit neural networks [49] in the RL to solve the BPP, which increased the computational cost and time complexity. Heuristic algorithms rely on empirical criteria to consider predetermined strategies and ignore the dynamic exploration of the global data space in BPP. RL-based methods can intelligently mine data information from the environmental space through trial and error. Perhaps, it can help the existing heuristic algorithms to fully explore the effective information in the sample space, which inspired our method.

3. Formulation of the BPP

The classic one-dimensional BPP is formalized as follows. It is assumed that there are n items to be packed into bins with equal capacity C. The general objective is to find a packing way to arrange all items $(J_1, J_2, \ldots, J_n)$ with the minimum number of bins, of which the formal mathematical description can be defined as z:

$$\min z = \sum_{i=1}^{n} y_i. \tag{1}$$

Therein, y_i represents the indicator whether the ith bin is used or not. A value of 1 indicates that the bin is used, and a value of 0 indicates that it is not used. Note that once the bin B_i is used, the total load of the items placed in B_i cannot exceed the capacity of C. Thus, we have

$$\text{s.t.} \sum_{j=1}^{n} w_j x_{ij} \le C y_i, \quad i = 1, \ldots, n, \tag{2}$$

where w_j means the load of the jth item and x_{ij} is an indicator whether the jth item is packed into the ith bin or not. Especially, $x_{ij} = 1$ if the jth item is placed into the ith container, otherwise $x_{ij} = 0$. Furthermore, an equally fundamental constraint is that each item is just placed into one bin:

$$\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1, \ldots, n. \tag{3}$$

The detailed explanation of parameters for the formalization is defined in Table 1.

4. Design of RBF

In this section, the design of the proposed RBF framework is presented. First, the overview of the RBF is outlined, and then, the details of its key components, such as the RL-system, the reinforced-MBS strategy, and the instance mapping process, are presented.

4.1. Overview. The classical MBS algorithm follows two steps:

(1) Utilize lexicographic search optimization procedure [14], also referred as the L algorithm, to find the item set J_j that should be allocated to the bin B_i

Table 1: Decision variables.

Variable	Type	Meaning
B_i	Discrete	The ith bin
J_j	Discrete	The jth item
z	Discrete	The objective function of the number of bins used
C	Constant	The capacity of the bin
w_j	Constant	The load of the jth item
x_{ij}	Binary	Indicator whether the jth item is packed into the ith bin or not
y_i	Binary	Indicator whether the ith bin is used or not

(2) Utilize Step 1 to traverse all items to be packed and the minimum bin slack is $C - \sum_{j=1}^{n} w_j$, where w_j is the load of the packed jth item

The steps above means that the slack is utilized to jump out of the optimal local trap randomly in the classical MBS algorithm, while the exact distribution of the sample space is ignored. To resolve the instability of the random slack, a new bin packing framework, RBF, is presented, where the slack is learnable and adjusted according to the samples' structure.

The framework of RBF is illustrated in Figure 1, which consists of a RL-system, a reinforced-MBS strategy, and an instance-eigenvalue mapping process, and defined as follows:

(1) RL-system: the RL-system is used to generate a suitable slack by a reinforcement learning strategy, where the best action selection strategy is controlled by Q-agent

(2) Reinforced-MBS strategy: with the provision of the slack coefficient from the RL-system, the reinforced-MBS strategy is exploited to resolve the packing process

(3) Instance-eigenvalue mapping: instead of using the whole dataset directly, the instance-eigenvalue mapping is utilized to generate the representative and classified validate set for the RL-system based on the similarity of the input instances

The main idea of RBF is to utilize the RL-system to learn the slack according to the spatial variation of the sample dataset, and then, the slack can be adapted to the distribution of bins and the remaining items in the data space during the iterative packing process. With the instance-eigenvalue mapping, the representative and classified validate set of the input instances is generated. The validate set is further integrated into the RL-system, where an adaptive slack is generated by the Q-agent. The coefficient of the slack is finally applied in the reinforced-MBS strategy for the packing process.

4.2. Instance-Eigenvalue Mapping. To reduce the amount of calculation for the slack, the representative items are selected for the Q-agent, which can learn the data space without traversing all instances. Here, an instance classification method, called as instance-eigenvalue mapping, is proposed and defined as

$$y_x^i = \frac{\widehat{x}^i - x_{\min^i}}{x_{\max^i} - x_{\min^i}}, \tag{4}$$

where x is an given instance, $\widehat{x}^i$ is the average value of the items in the ith instance in the dataset, $x_{\min}^i$ and $x_{\max}^i$, respectively, represent the minimum value and maximum value of the items in the ith instance, and y_x^i denotes the instances eigenvalue of the ith instance.

According to the value of the instance-eigenvalue, the whole instances are reordered. The dataset U can be divided into K different subsets $U_1, U_2, \ldots, U_k$. The last instance of each subset is taken to form a validation set. Then, the validation set is utilized to iteratively learn the slack. Therefore, at each time step t in RBF, instead of using the whole instances, Q-agent utilizes the validation set to reduce the repetitive work of the system.

4.3. RL-System. The validate set is integrated into the RL-system with a Q-learning algorithm [50], where Q-agent is utilized to learn the appropriate strategy and then improve the MBS strategy by selecting high-quality slack.

The process of the RL-system can be described as Markov decision processes (MDP) which is represented as a tuple (S, A, P, R, γ). In the decision-making process of MDP, S is the state set, A is an action set, P is the transition probability between states, R is the return value after taking a certain action to reach the next state, and γ is the discount factor. To be adaptive to the packing circumstances, for example, the current distribution of containers and the remaining items, we proposed a slack learning algorithm and the detailed process is shown in Algorithm 1, where the parameters are illustrated in Table 2. By observing the current state S_t of the environment, Q-agent selects one action A_t that maximizes the value of reward function R_t according to the state observed. With Q-agent continually interacting with the environment, we explore an suitable data selection strategy of the slack coefficient. The algorithm returns both a reward r and a new state S_{t+1} to Q-agent in each packing iteration, of which the change of states depends on the state transition probability $p(S_{t+1}|S_t, A_t)$:

$$p(S_{t+1}|S_t, A_t) = p_{ss'}^a = p(S_{t+1} = s'|S_t = s, A_t = a). \tag{5}$$

The agent receives the performance-driven reward R_t, and then, the sum of discount reward at time step t is represented as G_t:

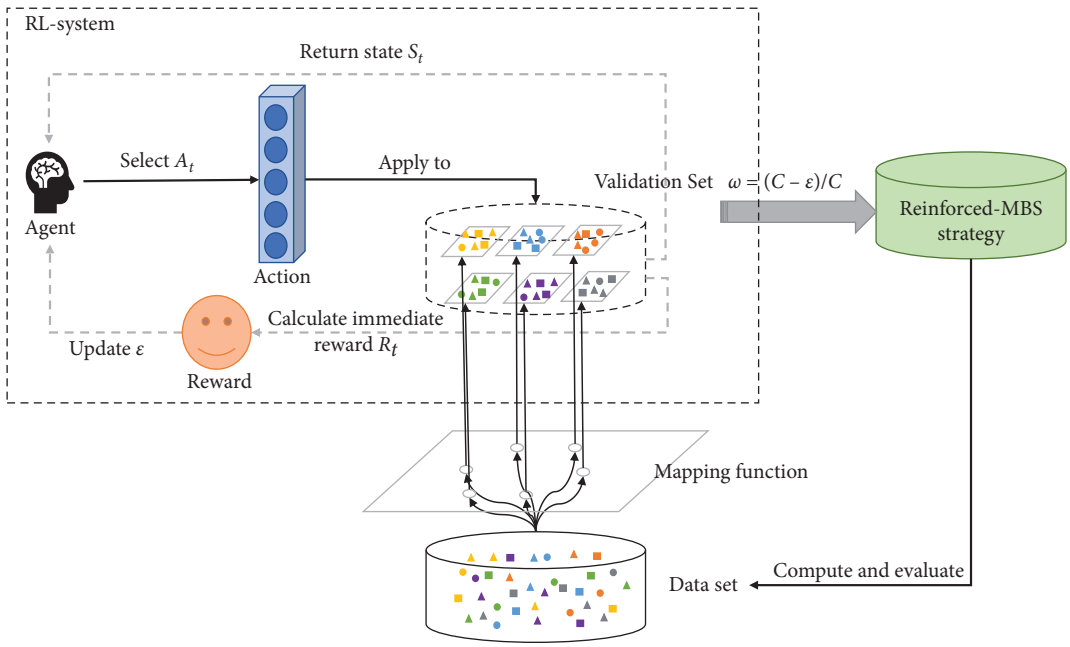

Figure 1: The framework of the RBF.

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{6}$$

Therein, $\gamma \in [0, 1)$, and it defines the weight of future reward and discount reward in the sum of reward. The closer γ is to 0, the more incentive is to consider short-term benefits. The closer γ is to 1, the more incentive is to consider long-term benefits.

The goal of the Q-agent at each time step t is to select an action A_t that can maximize future discount rewards G_t by finding an optimal policy π^*. Here, π^* is the strategy of taking the optimal action A_t at state S_t, while π is the strategy of taking action A_t at state S_t. Under the policy π, $Q^\pi(s, a)$ is defined as the expectation of the state-action value function. When the agent takes A_t at S_t, $Q^\pi(s, a)$ is represented as

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a], \tag{7}$$

where $\mathbb{E}_\pi$ is the expected function.

The maximum state-action function $Q^*(s, a)$ over all policies is represented in

$$Q^*(s, a) = \max_\pi Q^\pi(s, a). \tag{8}$$

The update rule of $Q(S_t, A_t)$ value is shown in

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha^* \left[R_{t+1} + \gamma^* \max_a Q(S_{t+1}, A_t) - Q(S_t, A_t) \right], \tag{9}$$

where $\alpha \in (0, 1)$ is the learning rate of the RL agent.

At each time step t, Q-agent observes the current state S_t and selects the action A_t from a discrete set of behaviors $A = \{1, 2, \ldots, k\}$, where the value of k is equal to the number of the items to be packed. At the beginning, the action A_t is randomly initialized, that is to say, the action A_t corresponding to the random number between 1 and k is selected. Then, the RL-system selects the action A_t that can maximize the $Q(S_t, a)$ value at each time step t:

$$A_t = \max_a Q(S_t, a). \tag{10}$$

The agent uses a greedy learning strategy [51] to choose actions. It selects actions according to the optimal value of Q table with $1 - \theta$ probability and randomly selects with θ probability.

The state is represented as the remaining space capacity of the bin after each round of packing. At each time step t, the remaining items prefer to be packed into as few bins as possible. When the bin is full, the agent is given a reward R_t. If the bin is overflowing, the agent is punished severely and it is told that state like this is not allowed.

The slack ϵ is defined as

$$\epsilon = (a + d/r), \tag{11}$$

Therein, d is a constant, r is the immediate reward achieved by Q-agent, and a represents the initial value in the first iteration process. By returning the reward value R_t, the slack ϵ is adjusted in each packing round accordingly. The slack ϵ can be changed in a range with the change of the reward value R_t. Ultimately, the new round of environment is updated as the subtraction of bin capacity C and slack ϵ.

Q-agent captures this intuition through performance-driven rewards R_t. At each time step t, the agent's reward is defined as t. Therein, count (n) is the number of bins that are exactly filled, count $(C - \epsilon)$ is the number of bins that are filled in the slack space, α is the weight coefficient of positive reward, Count (m) is the number of overflowing bins, β is the punishment coefficient of negative reward less than 0, and ι is a constant to regulate the value of the entire reward function R_t.

Input: training data itemList with n items, container list BinList with capacity C, remaining capacity Residual Capacity of the bin, learning rate α, discount factor γ, and the iterative number MAX_EPISODE.

(1) Initialize Q-table;
(2) **for** episode in range MAX_EPISODE **do**
(3) $S_t = 0$
(4) Initialize container list BinList [1, □, n];
(5) is_terminated = False;
(6) **while** not is_terminated do
(7) According to state S_t and Q-table, use epsilon greedy strategy to select actions A_t;
(8) Residual Capacity [1, i, n] = BinList [1, j, n]−itemList [1, k, n];
(9) Calculate immediate Reward R_{t+1} and get next State S_{t+1};
(10) get $Q_{\text{predict}}(S_t, A_t)$ from $Q(S_t, A_t)$;
(11) **if**S_{t+1} is not "terminal" then
(12) $Q_{\text{target}}(S_t, A_t) = R_{t+1} + \gamma^* \max_a Q(S_{t+1}, A_t)$;
(13) **else**
(14) $Q_{\text{target}}(S_t, A_t) = R_{t+1}$;
(15) is_terminated = True;
(16) **end if**
(17) Update ϵ;
(18) BinList [1, j, n] = BinList [1, j, n]−ϵ;
(19) **end while**
(20) $Q(S_t, A_t)$: $= Q(S_t, A_t) + \alpha^* [Q_{\text{target}}(S_t, A_t) - Q_{\text{predict}}(S_t, A_t)]$;
(21) $S_t = S_{t+1}$, $A_t = A_{t+1}$;
(22) **end for**
Output: Q-table, ϵ.

Algorithm 1: Slack learning algorithm combined with RL.

Table 2: Parameters and corresponding descriptions.

Parameter	Description
A_t	Action taken at time step t
S_t	State at time step t
R_t	Reward at time step t
A	a set of actions
S	a set of states
R	Instant reward
P	Transition probability between states
G_t	The sum of reward at time step t
α	Learning rate
γ	The discount factor
$Q^\pi(s, a)$	State-action value function
ϵ	Slack
SOL	Number of bins used by the concrete algorithm
OPT	Number of bins contained in the optimal solution of each bin packing instance
Unrealized SOL	Number of feasible optimal solution instances that the algorithm does not achieve
Gap	Deviation percentage between the solution reached by each algorithm and the optimal solution
CR_t	Competition ratio

4.4. Reinforced-MBS Algorithm. By introducing the slack learned by the RL agent into the MBS, we propose the Reinforced-MBS algorithm. Therein, slack is defined as

$$\text{slack} = \left(\min_{\text{value}}\right)^* \omega, \tag{12}$$

where ω is the slack parameter learned by the agent applying RL. It is calculated as $\omega = C - \epsilon/C$ by minimizing the number of used bins on the validation set. Then, the coefficient of slack is passed into our reinforced-MBS algorithm. In detail, the idea of the reinforced-MBS algorithm is shown in Algorithm 2.

In Algorithm 2, the improved L dictionary search procedure is utilized to find the set S_k of items that should be assigned to the bin B_k during the iterative process. The improved L dictionary search procedure is shown in Algorithm 3.

5. Experimental Evaluation

In this section, experiments are carried out to verify the effectiveness and robustness of the proposed RBF. First, experimental evaluation indexes are introduced and the

Input: training data itemList with n items, container list BinList with capacity C; set S, $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(s))$, for $t = 1, \ldots, n$, $s = n$
(1) Initialize $k = 1$, $\omega = C - \epsilon/C$;
(2) generate random Slack $\in\in (0, \min\_value^{*} \omega)$;
(3) **for***i* = 1 to |itemList|do
(4) Use the improved L dictionary search procedure to find the set of items that should be allocated to the bin of k
(5) **if**$\sigma = \varnothing$ **then** Pack $S = (S_1, S_2, \ldots, S_k)$ into the bins $(1, 2, \ldots, k)$;
(6) **else**
(7) $k = k + 1$
(8) **end if**
(9) **end for**
(10) CompetitionRatio = SOL/OPT
Output: bins $(1, 2, \ldots, k)$ with item set $(S_1, S_2, \ldots, S_k)$, CompetitionRatio.

Algorithm 2: Reinforced-MBS algorithm.

datasets used in the experiments are detailed. Afterwards, the experimental results are presented and analyzed. Finally, the robustness and stability of our method is discussed.

5.1. Evaluation Indexes and Datasets

5.1.1. Competition Ratio. The competition ratio [52, 53], CR_t, is defined as

$$CR_t = \frac{SOL}{OPT}, \tag{13}$$

where SOL represents the number of bins used by the concrete algorithm and OPT is the number of bins in the optimal solution for the packing instance. The competition ratio equal to 1 means that the algorithm has found the optimal solution.

Generally, $OPT(\sigma)$ has a lower limit, as shown in formula (14), where $\lceil\ \rceil$ is the ceiling function. Due to the limitation of bin packing conditions, the number of bins used in each bin packing iteration cannot be less than the ratio of the total load of the items to the capacity of a single bin:

$$OPT(\sigma) \geq \lceil \frac{\sum_{i=1}^{k} t_i}{C} \rceil. \tag{14}$$

5.1.2. FSOL. For a dataset, FSOL represents the number of feasible optimal solution instances achieved by the algorithm; in other words, the number of instances whose CR_t is 1. For the specialized algorithm alg and the dataset data, FSOL is specialized as $FSOL_{alg}(data)$.

5.1.3. Realization Rate. Realization rate (RT) is defined as formula (15), where INS is the number of instances in the packing dataset:

$$RT = \frac{FSOL}{INS}. \tag{15}$$

5.1.4. Gap. Gap is referred to the deviation between the number of used bins obtained by the algorithm and the optimal number of the packing. The relative Gap is exploited to evaluate the performance of the algorithms, which is calculated as

$$Gap = \frac{SOL - OPT}{OPT}. \tag{16}$$

The BINDATA [54] and SCH_WAE [55] datasets are used in the experiment for evaluation. Therein, BINDATA dataset includes three subsets, such as Bin1data, Bin2data, and Bin3data. The details of datasets are shown in Table 3, such as the number of instances, weight of items, capacity of bins, and the number of items in the instances.

5.2. Experimental Results and Analysis.

The performance of the RBF is compared with that of the classical Fit algorithms, the MBS algorithm, and the MBS' algorithm on BINDATA and SCH_WAE datasets shown in Table 3. For each instance of each dataset, the number of items in each category is the same. The experimental results reported in this paper are the average of ten runs under per hyperparameter settings.

5.2.1. Results on BINDATA. Table 4 lists the results of FSOL, RT, and CR_t of the algorithms involved in comparison on BINDATA, while Table 5 lists the value of Gap. In comparison with the classical heuristic algorithms, such as NFD, FFD, WFD, AWFD, BFD, MBS, and MBS', RBF obtains the maximum FSOL on BINDATA, while its CR_t and Gap are minimum. Furthermore, the improvement of $FSOL_{RBF}$, represented as $IMP_{RBF}(alg)$ and defined as formula (17), is further calculated, where $alg \in \{MBS, MBS'\}$ and $data \in \{Bin1data, Bin2data\}$. Especially, for Bin1data and Bin2data, $IMP_{RBF}(MBS)$ is 165.08% and 179.2%, respectively, while $IMP_{RBF}(MBS')$ is 5.53% and 41.3%, respectively. For the dataset Bin3data, RBF is the only one that can obtain 2 optimal solutions in the total 10 cases, while the others obtain zero optimal solutions:

$$IMP_{RBF}(alg) = \frac{FSOL_{RBF}(data) - FSOL_{alg}(data)}{FSOL_{alg}(data)}. \tag{17}$$

5.2.2. Results on SCH_WAE. The results of the compared algorithms on SCH_WAE are listed in Table 6. And, the

Input: t_i for $i = 1,\ldots,s$; C; $k = 1$, $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(j))$, where $t_{\sigma(1)} \geq t_{\sigma(2)} \geq \cdots \geq t_{\sigma(s)}$; $\alpha = 0$; $\pi = (\pi(1), \pi(2), \ldots, \pi(j)) = (\sigma(1), \sigma(2), \ldots, \sigma(j))$.

(1) Generate Slack $\in\in (0, \min\_value^{*}\omega)$.
(2) **Step 1**:
(3) **if** $0 \leq C - P_\pi \leq$ Slack **then**
(4) **if** $S_k = \pi$, go to Step 3.
(5) **else**
(6) Find the arrangement of the number of the last item in the temporary item list in the original item list, that is, to find q makes $\sigma_q = \pi_j$.
(7) **if** $P_\pi < C$ **then**
(8) $j = j + 1$
(9) **if** $P_\pi > \alpha$ **then**
(10) $\alpha = P_\pi$, prepare for packing $S_k = \pi$
(11) **end if**
(12) **Step 2**:
(13) **if** $q < s$ **then**
(14) $\pi_j = \sigma(q + 1)$, go to Step 1.
(15) **else**
(16) **if** $j = 1$ **then**
(17) go to Step 3
(18) **else**
(19) $j = j - 1$. Find q makes $\sigma(q) = \pi_j$, go to Step 2.
(20) **end if**
(21) **end if**
(22) **else**
(23) go to Step 2
(24) **end if**
(25) **end if**
(26) **Step 3**: Place the items in S_k into the Kth bin.

Output: $S = (S_1, S_2, \ldots, S_k)$.

Algorithm 3: Improved L dictionary search procedure.

Table 3: Information about the problem instances.

Bin1data	720	[1, 100]	{100, 120, 150}	{50, 100, 200, 500}
Bin2data	480	[1, 700]	1000	{50, 100, 200, 500}
Bin3data	10	[20000, 35000]	100000	200
SCH_WAE	200	[150, 200]	1000	{100, 120}

Table 4: Results on BINDATA.

Algorithm	Bin1data (n = 720)			Bin2data (n = 480)			Bin3data (n = 10)		
	FSOL	RT	CR_t	FSOL	RT	CR_t	FSOL	RT	CR_t
NFD [24]	0	0.00	1.3502	59	12.29	1.1271	0	0.00	1.1711
FFD [23]	546	75.83	1.0497	236	49.16	1.0315	0	0.00	1.0739
WFD [25]	442	61.38	1.0537	213	44.37	1.0336	0	0.00	1.0739
AWFD [25]	163	22.63	1.0663	1	0.20	1.0806	0	0.00	1.0865
BFD [23]	547	75.97	1.0497	236	49.16	1.0315	0	0.00	1.0739
MBS [14]	252	35.00	1.0645	125	26.04	1.0829	0	0.00	1.0594
MBS' [14]	633	87.91	1.0471	247	51.45	1.0264	0	0.00	1.0721
RBF	668	92.78	1.0468	349	72.71	1.0101	2	20.00	1.0198

detail results of RBF on SCH_WAE, that is, SOL, OPT, and the time cost (Runtime) on each instance, are shown in Tables 7 and 8. In the setting of Table 7, the number of items is 100 and the container capacity is 1000. Table 8 shows the statistical results when the number of items is 120. Meanwhile, Gap of each algorithm are shown in the last column of Table 5. It is shown that, in comparison with other algorithms on SCH_WAE, RBF achieved the minimum CR_t and

Table 5: Results of Gap.

Algorithm	Bin1data	Bin2data	Bin3data	SCH-WAE
NFD [24]	35.03%	12.71%	17.12%	6.22%
FFD [23]	4.98%	3.15%	7.39%	6.15%
WFD [25]	5.38%	3.37%	7.39%	6.15%
AWFD [25]	6.64%	8.06%	8.66%	10.69%
BFD [23]	4.98%	3.15%	7.39%	6.15%
MBS [14]	6.46%	3.08%	5.95%	5.17%
MBS' [14]	4.71%	2.98%	7.21%	5.10%
RBF	4.68%	1.01%	1.98%	1.39%

Table 6: Results on SCH_WAE ($n = 200$).

Algorithm	FSOL	RT	CR_t
NFD [24]	1	0.50	1.0622
FFD [23]	1	0.50	1.0614
WFD [25]	1	0.50	1.0614
AWFD [25]	0	0.00	1.1069
BFD [23]	1	0.50	1.0614
MBS [14]	25	12.50	1.0574
MBS' [14]	32	16.00	1.0513
RBF	143	71.50	1.0139

Table 7: Information of packing result for 200 instances of SCH_WAE.

Set	SOL	OPT	Runtime (s)
Number of items ($n = 100, C = 1000$)			
SCH_WAE1_1	18	18	1.462466
SCH_WAE1_2	18	18	2.727113
SCH_WAE1_3	18	18	3.150908
SCH_WAE1_4	18	18	3.70839
SCH_WAE1_5	18	18	0.539083
SCH_WAE1_6	18	18	0.758503
SCH_WAE1_7	18	18	0.664705
SCH_WAE1_8	18	18	0.679174
SCH_WAE1_9	19	18	4.522568
SCH_WAE1_10	18	18	4.374016
SCH_WAE1_11	18	18	0.982381
SCH_WAE1_12	18	18	1.691106
SCH_WAE1_13	18	18	4.604945
SCH_WAE1_14	19	18	10.755302
SCH_WAE1_15	18	18	1.797069
SCH_WAE1_16	18	18	1.450974
SCH_WAE1_17	19	18	1.246981
SCH_WAE1_18	18	18	0.722101
SCH_WAE1_19	18	18	0.713857
SCH_WAE1_20	18	18	0.709936
SCH_WAE1_21	18	18	0.998908
SCH_WAE1_22	19	18	3.028952
SCH_WAE1_23	18	18	0.557798
SCH_WAE1_24	18	18	0.976983
SCH_WAE1_25	18	18	1.074089
SCH_WAE1_26	19	18	0.31875
SCH_WAE1_27	18	18	3.446602
SCH_WAE1_28	18	18	6.492802
SCH_WAE1_29	18	18	4.080608
SCH_WAE1_30	18	18	1.483333
SCH_WAE1_31	18	18	0.731631
SCH_WAE1_32	18	18	1.545062
SCH_WAE1_33	18	18	1.73632
SCH_WAE1_34	18	18	0.730045
SCH_WAE1_35	18	18	1.060398
SCH_WAE1_36	18	18	0.987255
SCH_WAE1_37	19	18	13.783422

Table 7: Continued.

Set	SOL	OPT	Runtime (s)
SCH_WAE1_38	18	18	0.861974
SCH_WAE1_39	18	18	3.244424
SCH_WAE1_40	18	18	2.785588
SCH_WAE1_41	18	18	1.43709
SCH_WAE1_42	18	18	1.009547
SCH_WAE1_43	18	18	1.022882
SCH_WAE1_44	18	18	0.58083
SCH_WAE1_45	18	18	4.300251
SCH_WAE1_46	18	18	1.063874
SCH_WAE1_47	18	18	7.621793
SCH_WAE1_48	18	18	0.681381
SCH_WAE1_49	18	18	0.96076
SCH_WAE1_50	18	18	0.520412
SCH_WAE1_51	18	18	0.706839
SCH_WAE1_52	18	18	1.22345
SCH_WAE1_53	18	18	1.450091
SCH_WAE1_54	18	18	0.876361
SCH_WAE1_55	18	18	0.839967
SCH_WAE1_56	18	18	0.522451
SCH_WAE1_57	19	18	9.148892
SCH_WAE1_58	18	18	1.304132
SCH_WAE1_59	18	18	0.91717
SCH_WAE1_60	18	18	0.860201
SCH_WAE1_61	18	18	11.420425
SCH_WAE1_62	18	18	0.818042
SCH_WAE1_63	18	18	1.024627
SCH_WAE1_64	18	18	0.630548
SCH_WAE1_65	18	18	1.120638
SCH_WAE1_66	18	18	7.398161
SCH_WAE1_67	18	18	1.001821
SCH_WAE1_68	18	18	0.979633
SCH_WAE1_69	18	18	0.681692
SCH_WAE1_70	19	18	3.835019
SCH_WAE1_71	18	18	4.387344
SCH_WAE1_72	18	18	1.003916
SCH_WAE1_73	18	18	3.558745
SCH_WAE1_74	19	18	5.927648
SCH_WAE1_75	18	18	0.779189
SCH_WAE1_76	18	18	1.061736
SCH_WAE1_77	18	18	1.168255
SCH_WAE1_78	19	18	0.446685
SCH_WAE1_79	18	18	1.053857
SCH_WAE1_80	18	18	1.261393
SCH_WAE1_81	18	18	3.997979
SCH_WAE1_82	18	18	0.953343
SCH_WAE1_83	18	18	0.918264
SCH_WAE1_84	18	18	0.965771
SCH_WAE1_85	18	18	0.714233
SCH_WAE1_86	18	18	0.950277
SCH_WAE1_87	18	18	1.366335
SCH_WAE1_88	18	18	12.544721
SCH_WAE1_89	18	18	0.918441
SCH_WAE1_90	18	18	1.302644
SCH_WAE1_91	18	18	2.591703
SCH_WAE1_92	18	18	1.554301
SCH_WAE1_93	18	18	1.055307
SCH_WAE1_94	18	18	1.010704
SCH_WAE1_95	18	18	1.482197
SCH_WAE1_96	18	18	1.351396
SCH_WAE1_97	18	18	1.305057
SCH_WAE1_98	18	18	0.872971
SCH_WAE1_99	18	18	3.715139
SCH_WAE1_100	18	18	1.34348

Table 8: Information of packing result for 200 instances of SCH_WAE.

Set	SOL	OPT	Runtime (s)
Number of items ($n = 120, C = 1000$)			
SCH_WAE2_1	22	22	7.881069
SCH_WAE2_2	22	22	3.576123
SCH_WAE2_3	22	21	3.484807
SCH_WAE2_4	22	21	2.672799
SCH_WAE2_5	22	22	1.696537
SCH_WAE2_6	22	22	6.829253
SCH_WAE2_7	22	22	10.166636
SCH_WAE2_8	21	21	4.537143
SCH_WAE2_9	22	22	4.785806
SCH_WAE2_10	22	21	1.873941
SCH_WAE2_11	22	22	1.246602
SCH_WAE2_12	22	22	1.538963
SCH_WAE2_13	22	22	6.513991
SCH_WAE2_14	22	21	3.337582
SCH_WAE2_15	22	21	23.913958
SCH_WAE2_16	22	22	1.695519
SCH_WAE2_17	22	21	2.759149
SCH_WAE2_18	22	22	2.037806
SCH_WAE2_19	22	22	1.961526
SCH_WAE2_20	23	22	7.537802
SCH_WAE2_21	22	21	1.318665
SCH_WAE2_22	22	22	1.634718
SCH_WAE2_23	22	22	1.795601
SCH_WAE2_24	22	22	1.40357
SCH_WAE2_25	22	22	19.901876
SCH_WAE2_26	22	21	10.612043
SCH_WAE2_27	22	21	3.323103
SCH_WAE2_28	22	21	2.773831
SCH_WAE2_29	22	21	2.13123
SCH_WAE2_30	22	21	2.424299
SCH_WAE2_31	22	22	1.590401
SCH_WAE2_32	22	22	1.371827
SCH_WAE2_33	22	22	1.966954
SCH_WAE2_34	22	21	3.440176
SCH_WAE2_35	22	22	1.483907
SCH_WAE2_36	22	22	6.622695
SCH_WAE2_37	22	22	1.249159
SCH_WAE2_38	22	22	1.299636
SCH_WAE2_39	22	22	12.406897
SCH_WAE2_40	22	22	2.000733
SCH_WAE2_41	22	21	5.219861
SCH_WAE2_42	22	21	16.346722
SCH_WAE2_43	22	22	2.031538
SCH_WAE2_44	22	22	2.05005
SCH_WAE2_45	22	22	2.064586
SCH_WAE2_46	22	21	1.883535
SCH_WAE2_47	22	21	3.715486
SCH_WAE2_48	22	21	4.25449
SCH_WAE2_49	22	22	8.43341
SCH_WAE2_50	22	21	2.382179
SCH_WAE2_51	22	22	2.021937
SCH_WAE2_52	22	22	1.854207
SCH_WAE2_53	22	22	5.216751
SCH_WAE2_54	22	22	18.804251
SCH_WAE2_55	22	22	7.729899
SCH_WAE2_56	22	21	2.015478
SCH_WAE2_57	22	21	2.654803
SCH_WAE2_58	22	22	2.015065
SCH_WAE2_59	22	21	15.788445

Table 8: Continued.

Set	SOL	OPT	Runtime (s)
SCH_WAE2_60	22	22	12.14978
SCH_WAE2_61	22	22	6.24467
SCH_WAE2_62	22	21	2.718076
SCH_WAE2_63	22	21	10.584626
SCH_WAE2_64	22	21	2.177987
SCH_WAE2_65	22	22	4.06024
SCH_WAE2_66	22	22	1.741747
SCH_WAE2_67	22	22	2.653687
SCH_WAE2_68	22	22	1.437786
SCH_WAE2_69	22	21	3.402605
SCH_WAE2_70	22	22	3.326743
SCH_WAE2_71	22	21	2.248744
SCH_WAE2_72	22	22	16.249829
SCH_WAE2_73	22	22	1.297735
SCH_WAE2_74	22	21	17.573503
SCH_WAE2_75	22	21	1.374113
SCH_WAE2_76	22	22	2.612285
SCH_WAE2_77	22	21	3.444593
SCH_WAE2_78	22	21	2.393924
SCH_WAE2_79	22	21	2.894696
SCH_WAE2_80	23	22	14.637149
SCH_WAE2_81	23	22	11.182882
SCH_WAE2_82	22	22	3.12165
SCH_WAE2_83	22	21	1.875829
SCH_WAE2_84	22	22	10.238209
SCH_WAE2_85	22	21	8.727173
SCH_WAE2_86	22	22	4.11194
SCH_WAE2_87	22	21	4.925202
SCH_WAE2_88	22	22	1.467682
SCH_WAE2_89	22	22	1.679621
SCH_WAE2_90	22	21	7.822758
SCH_WAE2_91	22	21	2.573397
SCH_WAE2_92	22	21	2.676046
SCH_WAE2_93	22	21	3.51098
SCH_WAE2_94	22	21	1.595649
SCH_WAE2_95	22	21	2.751163
SCH_WAE2_96	22	21	2.599518
SCH_WAE2_97	22	22	1.551431
SCH_WAE2_98	22	21	2.829111
SCH_WAE2_99	22	22	1.861101
SCH_WAE2_100	22	21	2.520166

Gap, but the maximum was FSOL and RT. Especially, for SCH_WAE, $\mathrm{IMP}_{\mathrm{RBF}}(\mathrm{MBS})$ is 472%, while $\mathrm{IMP}_{\mathrm{RBF}}(\mathrm{MBS}')$ is 346.88%.

5.2.3. Cumulative Results. For all the instances in BINDATA and SCH_WAE, the cumulative packing results of the compared algorithms, such as the cumulative FSOL, the average RT, the average CR_t, and the average Gap, are shown in Table 9. It is shown that RBF obtained 1162 cumulative FSOL and the RT was 82.41%, which greatly overwhelmed other compared algorithms. Especially, according to formula (17), the cumulative improvement of RBF to MBS and MBS', correspondingly, $\mathrm{IMP}_{\mathrm{RBF}}(\mathrm{MBS})$ and $\mathrm{IMP}_{\mathrm{RBF}}(\mathrm{MBS}')$, is 189.05% and 27.41%, respectively. Figure 2 graphically shows the statistical FSOL of the comparison algorithm on each dataset. It is noted that, from the radar in Figure 2, the quantitative curve of RBF for FSOL is at the outermost. It means that RBF achieved the largest number of optimal solutions in the test cases. Also, RBF achieved the minimum CR_t and Gap. Results indicate that, in comparison with the typical heuristic algorithms, RBF has a stronger global optimal performance. Overall, the proposed RBF obtains first ranks in all metrics according to the results of all compared algorithms listed in Tables 4–6. In detail, the improvement in FSOL of uncomplicated datasets, Bin1data, is limited among all compared methods. RBF achieves best results and wins with few advantages. However, RBF works much better than other methods on difficult datasets, bin2data, bin3data, and SCH_WAE. Besides, RBF achieves huge advantages in Gap, and it wins almost all other methods on used datasets.

5.3. Robustness and Stability. The construction of the validation set is a key procedure of RBF. This experiment is carried out to verify the validity of the eigenvalue mapping function on the Bin1data. Since 10 instances of

Table 9: Average results.

Algorithm	NFD [24]	FFD [23]	WFD [25]	AWFD [25]	BFD [23]	MBS [14]	MBS' [14]	RBF
Cumulative FSOL	60	783	656	164	784	402	912	1162
Average RT	4.26%	55.53%	46.52%	11.63%	55.60%	28.51%	64.68%	82.41%
Average CR_t	1.1776	1.0541	1.0556	1.0850	1.0541	1.0660	1.0492	1.0226
Average Gap	17.77%	5.42%	5.57%	8.51%	5.42%	5.17%	5.00%	2.27%

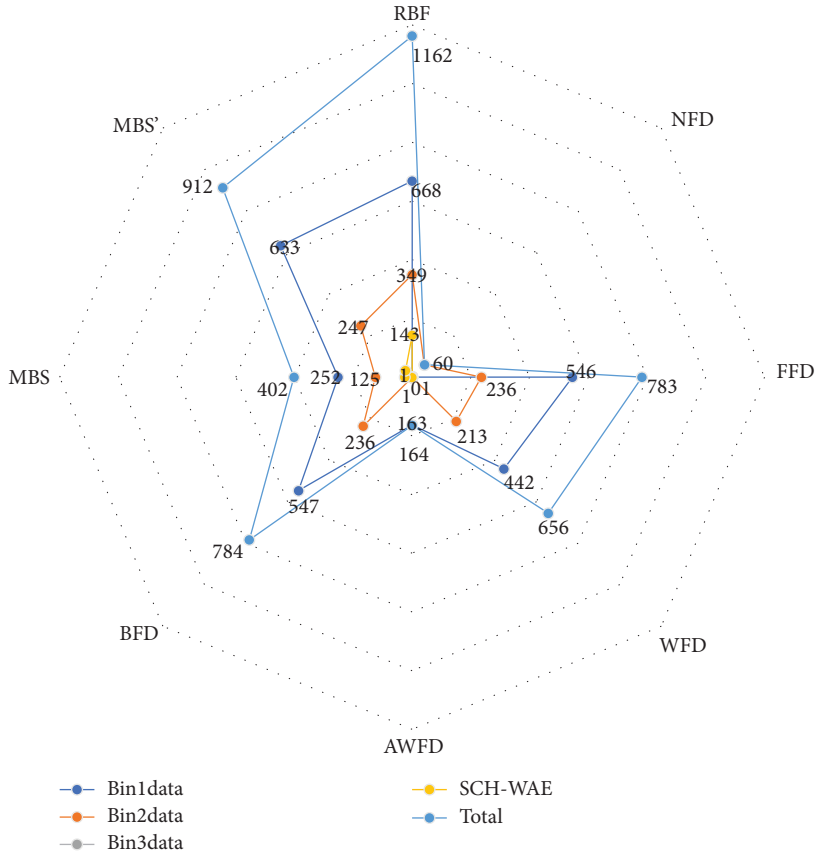

Figure 2: The number of instances to optimal solution by each algorithm.

Table 10: Statistical results with different composition of validation sets.

Validation set	Parameter ω	FSOL	RT (%)	CR_t	Gap (%)
Top 10 cases	0.1817	189	26.25	1.0532	5.32
The last 10 cases	0.8710	232	32.22	1.0620	5.87
Random 10 cases	0.4532	307	42.64	1.0492	4.93
Mapping 10 cases	0.5813	668	92.78	1.0468	4.68

the Bin1data are selected to form the validation set by the eigenvalue mapping function, here different selection policy are applied for comparison in the packing process. The first policy is that the first 10 instances of the Bin1data are selected, the second policy is that the last 10 instances of the Bin1data are selected, and the third policy is that 10 random instances of the Bin1data are selected to form the validation set. The packing results with different selection policies are depicted in Table 10. It can be seen that the value of slack learned by Q-agent is different with different selection policies. Especially, with the selection policy of the eigenvalue mapping function, RBF achieved the maximum FSOL and RT, while the minimum was CR_t and Gap. The results verified the validity of the eigenvalue mapping function, which helped RBF achieved better performance.

6. Conclusion and Future Work

In this paper we propose reinforced bin packing framework (RBF) to tackle the one-dimensional BPP. The proposed RBF consists of three main components: the RL-system, the instance-eigenvalue mapping process, and the reinforced-MBS strategy. The RL-system is designed to construct a slack selection policy automatically by Q-agent to select high-quality slack for the heuristic algorithm integrated in RBF. The instance-eigenvalue mapping process is utilized to generate the representative and classified the validate set based on the similarity of the input instances, which greatly eliminates the computational overhead and improves the generalization performance of the model. Finally, with the provision of the slack coefficient from the RL-system, the reinforced-MBS strategy is exploited to resolve the packing process. We evaluate our models on BPP tasks, where RBF exhibits excellent packing ability and experimental results validate its superior performance compared to state-of-the-art proposals on BINDATA and SCH_WAE datasets. Compared to its baseline methods, MBS and MBS', the average number of optimal solutions achieved by RBF increases by 189.05% and 27.41%, respectively.

For future work, we plan to investigate slack selection policies and new mechanisms to learn them automatically. We also foresee the extension of our method to more complex multiagent reinforcement learning frameworks, where the use of new aspects of the multiagent communication environment is crucial to boost the packing performance.

Data Availability

The datasets used in this paper contain one-dimensional bin packing datasets, such as BINDATA and SCH_WAE datasets, which can be found in http://people.brunel.ac.uk/~mastjjb/jeb/orlib/binpackinfo.html.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (no. 61472139) and the Shanghai 2020 Action Plan of Technological Innovation (no. 20dz1201400).

References

[1] H. Tang, K. C. Tan, and Z. Yi, "A columnar competitive model for solving combinatorial optimization problems," *IEEE Transactions on Neural Networks*, vol. 15, no. 6, pp. 1568–1573, 2004.

[2] D. Zhang, Z. Fu, and L. Zhang, "Joint optimization for power loss reduction in distribution systems," *IEEE Transactions on Power Systems*, vol. 23, no. 1, pp. 161–169, 2008.

[3] F. Eisenbrand, D. Pálvölgyi, and T. Rothvoß, "Bin packing via discrepancy of permutations," *ACM Transactions on Algorithms*, vol. 9, no. 3, pp. 24-25, 2013.

[4] H. I. Christensen, A. Khan, S. Pokutta, and P. Tetali, "Approximation and online algorithms for multidimensional bin packing: a survey," *Computer Science Review*, vol. 24, pp. 63–79, 2017.

[5] K. Lehmann, A. Grastien, and P. Van Hentenryck, "Ac-feasibility on tree networks is np-hard," *IEEE Transactions on Power Systems*, vol. 31, no. 1, pp. 798–801, 2015.

[6] R. Li, Y. Wang, S. Hu, J. Jiang, D. Ouyang, and M. Yin, "Solving the set packing problem via a maximum weighted independent set heuristic," *Mathematical Problems in Engineering*, vol. 2020, Article ID 3050714, 11 pages, 2020.

[7] S. Martello, D. Pisinger, and P. Toth, "New trends in exact algorithms for the 0-1 knapsack problem," *European Journal of Operational Research*, vol. 123, no. 2, pp. 325–332, 2000.

[8] S. Martello and D. Vigo, "Exact solution of the two-dimensional finite bin packing problem," *Management Science*, vol. 44, no. 3, pp. 388–399, 1998.

[9] F. Brandão and J. P. Pedroso, "Bin packing and related problems: general arc-flow formulation with graph compression," *Computers & Operations Research*, vol. 69, pp. 56–67, 2016.

[10] J. D. S. Coffman, M. R. Garey, and D. S. Johnson, "Approximation algorithms for bin packing: a survey," in *Approximation Algorithms for NP-Hard Problems*, D. S. Hochbaum, Ed., vol. 1, pp. 46–93, PWS Publishing Co., Boston, MA, USA, 1996.

[11] K. Sim, E. Hart, and B. Paechter, "A lifelong learning hyper-heuristic method for bin packing," *Evolutionary Computation*, vol. 23, no. 1, pp. 37–67, 2015.

[12] H. Wang, W. Wang, H. Sun, and S. Rahnamayan, "Firefly algorithm with random attraction," *International Journal of Bio-Inspired Computation*, vol. 8, no. 1, pp. 33–41, 2016.

[13] M. Abdel-Basset, G. Manogaran, L. Abdel-Fatah, and S. Mirjalili, "An improved nature inspired meta-heuristic algorithm for 1-d bin packing problems," *Personal and Ubiquitous Computing*, vol. 22, no. 5-6, pp. 1117–1132, 2018.

[14] J. N. D. Gupta and J. C. Ho, "A new heuristic algorithm for the one-dimensional bin-packing problem," *Production Planning & Control*, vol. 10, no. 6, pp. 598–603, 1999.

[15] S. Polyakovskiy and R. M'Hallah, "A hybrid feasibility constraints-guided search to the two-dimensional bin packing problem with due dates," *European Journal of Operational Research*, vol. 266, no. 3, pp. 819–839, 2018.

[16] R. Villasana, L. Garver, and S. Salon, "Transmission network planning using linear programming," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-104, no. 2, pp. 349–356, 1985.

[17] A. Richards, T. Schouwenaars, J. P. How, and E. Feron, "Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 4, pp. 755–764, 2002.

[18] M. Chitsaz, J.-F. Cordeau, and R. Jans, "A branch-and-cut algorithm for an assembly routing problem," *European Journal of Operational Research*, vol. 282, no. 3, pp. 896–910, 2020.

[19] Y. Zhang, X. Sun, and B. Wang, "Efficient algorithm for k-barrier coverage based on integer linear programming," *China Communications*, vol. 13, no. 7, pp. 16–23, 2016.

[20] M. I. Hartillo-Hermoso, H. Jiménez-Tafur, and J. M. Ucha-Enríquez, "An exact algebraic ϵ-constraint method for bi-objective linear integer programming based on test sets," *European Journal of Operational Research*, vol. 282, no. 2, pp. 453–463, 2020.

[21] P. Tangpattanakul, N. Jozefowiez, and P. Lopez, "A multi-objective local search heuristic for scheduling earth

observations taken by an agile satellite," *European Journal of Operational Research*, vol. 245, no. 2, pp. 542–554, 2015.

[22] D. S. Johnson, "Approximation algorithms for combinatorial problems," *Journal of Computer and System Sciences*, vol. 9, no. 3, pp. 256–278, 1974.

[23] D. S. Johnson, "Near-optimal bin packing algorithms," Ph. D. dissertation, Massachusetts Institute of Technology, Department of Mathematics, Cambridge, UK, 1973.

[24] M. Hofri and S. Kamhi, "A stochastic analysis of the nfd bin-packing algorithm," *Journal of Algorithms*, vol. 7, no. 4, pp. 489–509, 1986.

[25] N. G. Hall, S. Ghosh, R. D. Kankey, S. Narasimhan, and W. T. Rhee, "Bin packing problems in one dimension: heuristic solutions and confidence intervals," *Computers & Operations Research*, vol. 15, no. 2, pp. 171–177, 1988.

[26] R. Ren, X. Tang, Y. Li, and W. Cai, "Competitiveness of dynamic bin packing for online cloud server allocation," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1324–1331, 2016.

[27] F. F. Boctor, "Some efficient multi-heuristic procedures for resource-constrained project scheduling," *European Journal of Operational Research*, vol. 49, no. 1, pp. 3–13, 1990.

[28] J. F. Gonçalves and M. G. C. Resende, "A biased random key genetic algorithm for 2d and 3d bin packing problems," *International Journal of Production Economics*, vol. 145, no. 2, pp. 500–510, 2013.

[29] Y. Wu, M. Tang, and W. Fraser, "A simulated annealing algorithm for energy efficient virtual machine placement," in *Proceedings of the IEEE international Conference on Systems, Man, and Cybernetics*, pp. 1245–1250, Seoul, Korea, October 2012.

[30] F. Luo, I. D. Scherson, and J. Fuentes, "A novel genetic algorithm for bin packing problem in jmetal," in *Proceedings of the 2017 IEEE International Conference on Cognitive Computing (ICCC)*, pp. 17–23, Honolulu, HI, USA, June 2017.

[31] W. Gao and Z.-P. Jiang, "Adaptive dynamic programming and adaptive optimal output regulation of linear systems," *IEEE Transactions on Automatic Control*, vol. 61, no. 12, pp. 4164–4169, 2016.

[32] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," 2016, http://arxiv.org/abs/1611.09940.

[33] H. Hu, X. Zhang, X. Yan, L. Wang, and Y. Xu, "Solving a new 3d bin packing problem with deep reinforcement learning method," 2017, http://arxiv.org/abs/1708.05930.

[34] A. Mirhoseini, H. Pham, Q. V. Le et al., "Device placement optimization with reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning*, pp. 2430–2439, Sydney, Australia, August 2017.

[35] B. J. Hellstrom and L. N. Kanal, "Knapsack packing networks," *IEEE Transactions on Neural Networks*, vol. 3, no. 2, pp. 302–307, 1992.

[36] A. Savi A, J. Kratica, M. Milanovi, and D. Dugošija, "A mixed integer linear programming formulation of the maximum betweenness problem," *European Journal of Operational Research*, vol. 206, no. 3, pp. 522–527, 2010.

[37] J. Kang and S. Park, "Algorithms for the variable sized bin packing problem," *European Journal of Operational Research*, vol. 147, no. 2, pp. 365–372, 2003.

[38] E. G. Coffman Jr, J. Csirik, G. Galambos, S. Martello, and D. Vigo, "Bin packing approximation algorithms: survey and classification," *Handbook of Combinatorial Optimization*, vol. 1, no. 2, pp. 455–531, 2013.

[39] K. Fleszar and C. Charalambous, "Average-weight-controlled bin-oriented heuristics for the one-dimensional bin-packing problem," *European Journal of Operational Research*, vol. 210, no. 2, pp. 176–184, 2011.

[40] S. S. Seiden, "On the online bin packing problem," *Journal of the ACM*, vol. 49, no. 5, pp. 640–671, 2002.

[41] L. Epstein and R. Van Stee, "Online bin packing with resource augmentation," *Discrete Optimization*, vol. 4, no. 3-4, pp. 322–333, 2007.

[42] K. Fleszar and K. S. Hindi, "New heuristics for one-dimensional bin-packing," *Computers & Operations Research*, vol. 29, no. 7, pp. 821–839, 2002.

[43] T. Dokeroglu and A. Cosar, "Optimization of one-dimensional bin packing problem with island parallel grouping genetic algorithms," *Computers & Industrial Engineering*, vol. 75, pp. 176–186, 2014.

[44] M. Quiroz-Castellanos, L. Cruz-Reyes, J. Torres-Jimenez, C. Gómez S., H. J. F. Huacuja, and A. C. F. Alvim, "A grouping genetic algorithm with controlled gene transmission for the bin packing problem," *Computers & Operations Research*, vol. 55, pp. 52–64, 2015.

[45] T. Kucukyilmaz and H. E. Kiziloz, "Cooperative parallel grouping genetic algorithm for the one-dimensional bin packing problem," *Computers & Industrial Engineering*, vol. 125, pp. 157–170, 2018.

[46] T. G. Crainic, G. Perboli, and R. Tadei, "TS2PACK: a two-level tabu search for the three-dimensional bin packing problem," *European Journal of Operational Research*, vol. 195, no. 3, pp. 744–760, 2009.

[47] D. Kumar and Z. Raza, "A pso based vm resource scheduling model for cloud computing," in *Proceedings of the IEEE International Conference on Computational Intelligence & Communication Technology*, pp. 213–219, Ghaziabad, India, February 2015.

[48] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proceedings of the Advances in Neural Information Processing Systems*, pp. 2692–2700, Montreal, Canada, December 2015.

[49] K. H. Jin, M. T. McCann, E. Froustey, and M. Unser, "Deep convolutional neural network for inverse problems in imaging," *IEEE Transactions on Image Processing*, vol. 26, no. 9, pp. 4509–4522, 2017.

[50] B. Luo, D. Liu, T. Huang, and D. Wang, "Model-free optimal tracking control via critic-only q-learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 10, pp. 2134–2144, 2016.

[51] Q. Wei, F. L. Lewis, Q. Sun, P. Yan, and R. Song, "Discrete-time deterministic q-learning: a novel convergence analysis," *IEEE Transactions on Cybernetics*, vol. 47, no. 5, pp. 1224–1237, 2016.

[52] G. Gutin, T. Jensen, and A. Yeo, "Batched bin packing," *Discrete Optimization*, vol. 2, no. 1, pp. 71–82, 2005.

[53] E. F. Grove, "Online bin packing with lookahead," in *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 430–436, San Francisco, CA, USA, December 1995.

[54] A. Scholl, R. Klein, and C. Jürgens, "Bison: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem," *Computers & Operations Research*, vol. 24, no. 7, pp. 627–645, 1997.

[55] P. Schwerin and G. Wäscher, "The bin-packing problem: a problem generator and some numerical experiments with ffd packing and mtp," *International Transactions in Operational Research*, vol. 4, no. 5-6, pp. 377–389, 1997.