A Combination Feature-Based Reinforcement Learning Approach via Mathematical Optimization

Fengyuan Shi, Ying Meng, Jiyin Liu, and Lixin Tang, *Fellow, IEEE*

***Abstract*—Reinforcement learning is a promising method for solving decision problems, and its potential has been increasingly recognized for large-scale combinatorial optimization problems in recent years. However, the existing studies on reinforcement learning for cutting stock problems mostly rely on sequence-to-sequence or graph neural network approaches that use the learned experience to make decisions while neglecting the combination features of cutting stock problems. In this paper, we propose a novel reinforcement learning framework for cutting stock problems that integrate integer programming and monotone comparative statics to construct a Markov decision process with a high-quality action space. We start by constructing a new Markov decision process that considers the diagonal structure of the integer programming model for combinatorial optimization problems, and then use column generation to obtain each action by combining multiple decision variables. Furthermore, we design a bipartite graph and related bipartite graph convolutional network to find the solutions. The results show that the proposed reinforcement learning framework provides a high-quality action space, and the designed bipartite graph convolutional network can effectively select the best actions from the action set.**

***Note to Practitioners*—This article was motivated by the cutting stock problems that exist in various industrial scenarios such as the wood, steel, paper, and glass industries. We improve the reinforcement learning for the cutting stock problem that can be adopted in industrial scenarios, which can increase the profile and reduce the production cost of industrial enterprises. Our improvement can also be referred to when solving other combinatorial optimization problems that can promote making decisions in industrial production.**

***Index Terms*—Reinforcement learning, bipartite graph convolutional network, column generation, monotone comparative statics, cutting stock problem.**

Received 2 June 2024; revised 1 November 2024; accepted 26 January 2025. Date of publication 21 February 2025; date of current version 16 April 2025. This article was recommended for publication by Associate Editor F. Ju and Editor K. Liu upon evaluation of the reviewers' comments. This work was supported in part by the Major Program of National Natural Science Foundation of China under Grant 72192830 and Grant 72192831, in part by the National Natural Science Foundation of China under Grant 72002028, and in part by the 111 Project under Grant B16009. *(Corresponding author: Ying Meng.)*

Fengyuan Shi and Ying Meng are with the National Frontiers Science Center for Industrial Intelligence and Systems Optimization and the Key Laboratory of Data Analytics and Optimization for Smart Industry, Ministry of Education, Northeastern University, Shenyang 110819, China (e-mail: im.fengyuan.shi@outlook.com; mengying@ise.neu.edu.cn).

Jiyin Liu is with the School of Business Economics, Loughborough University, LE11 3TU Leicestershire, U.K. (e-mail: j.y.liu@lboro.ac.uk).

Lixin Tang is with the National Frontiers Science Center for Industrial Intelligence and Systems Optimization, Northeastern University, Shenyang 110819, China (e-mail: lixintang@mail.neu.edu.cn).

Digital Object Identifier 10.1109/TASE.2025.3544431

I. Introduction

REINFORCEMENT learning (RL) has demonstrated tremendous success in learning complex tasks and solving challenging problems in various fields, such as optimal control, operations research, and economics [1], [2]. RL algorithms learn to decide by selecting actions that minimize or maximize a desired goal. At each stage, the agent observes the system state and selects an action, leading to a transition from the current state to a new state. The applied action generates a reward as feedback to the agent. Accordingly, the agent learns to evaluate the state or action.

In most problems, the number of actions and states is increasing exponentially along with the dimension of the state and action. This phenomenon is called the "curse of dimensionality", which makes the actions and states hard to evaluate. The approximate function is adopted while evaluating states or actions to overcome the "curse of dimensionality". Monographs such as [3] and [4], as well as published articles, have focused on extracting features (e.g., Tetris [5]), designing approximate function structures (e.g., linear function [6], the piecewise linear function [7], the quadratic function [8]), and designing parameter optimization algorithms (e.g., least squares temporal differences [9], least squares policy iteration [10]). In 2013, DeepMind published an article [11] showing that the deep learning techniques enabled reinforcement learning to handle more complex tasks, which sparked a surge in research on deep reinforcement learning (DRL).

In DRL, features are extracted using a neural network. Researchers have mainly focused on two areas: improving DRL algorithms and designing appropriate neural networks. For example, an improved deep Q network was proposed in reference [12] to address the instability of the value network during training. To increase the estimation accuracy of the action value, the dueling deep Q network was proposed [13]. Prioritized experience replay was proposed by reference [14] to improve the batch-selecting phase and obtain important samples from the replay buffer. The actor-critic framework established in reference [15] reduces the variance while calculating the gradient of the policy network. To accelerate the training phase, parallel operations such as interacting with the environment, collecting samples, and updating the network parameters are used in the asynchronous advantage actor-critic algorithm proposed in reference [16]. Other algorithms proposed to improve DRL include the trust region policy optimization algorithm [17], proximal policy optimization algorithms [18], the deep deterministic policy gradient

algorithm [19], and the soft Q-learning algorithm [20]. To address the challenge of designing reward functions, reference [21] proposes the orientation-preserving rewards balancing method to balance the auxiliary rewards and the main reward. References [22] and [23] consider the use of prior human knowledge to design rewards and improve data efficiency.

Combinatorial optimization problems (COPs) are a crucial class of optimization problems that can be solved using RL. While classical techniques such as cutting plane, branch and bound, and benders decomposition can solve COPs optimally, they are time-consuming integer programming techniques. RL is considered an approximate and faster method for solving COPs, as most of them can be reformulated as a Markov decision process (MDP).

As an important instance of COPs, the cutting stock problem (CSP) widely exists in the real world, especially in the manufacturing sectors. The cutting stock problem is to cut standard-sized material stocks into items of specified sizes. Given the requirement for each item type (items with each specified size), the objective is to minimize the number of stocks needed. Research on CSP can reduce the cost and increase the profit in the industry such as the wood industry [24], steel industry [25], paper industry [26], and glass industry [27]. As described above, solving CSP using RL is worth studying.

Bin packing problem is similar to the CSP. It may be viewed as a special case of CSP with the demand for each item type being exactly one. The CSP can be reducible to the knapsack problem by limiting the number of stock to one. In the view of RL, the techniques of the knapsack problem and bin packing problem can be also adopted to the CSP. In the related works, reference [28] solves a CSP using a linear function to approximate the evaluation of the action. A widely used technique is the sequence-to-sequence approach. The pointer network [29] is a representative one. References [30], [31], and [32] solve the knapsack problem and 3D-bin packing problems using pointer network-based RL. The bin packing problem is also solved by Monte Carlo Tree Search (MCTS) [33]. The pointer network is a commonly used technique that can tackle the challenge of instance scale being changeable. Besides being used for solving CSP, bin packing, and knapsack problems, the pointer network also adopted by references [34], [35], [36], and [37] to solve different kinds of vehicle routing and traveling salesman problem. Graph neural networks such as the graph convolutional network (GCN) [38] are also widely adopted to solve various combinatorial optimization problems.

To summarize, two main techniques can be used to solve CSP: sequence-to-sequence approaches (such as the pointer network) and graph neural networks. They employ the same scheme, which defines the item set as the action set and outputs the sequence of items. In this scheme, the agent selects an item to allocate to the stock at each stage. When no item is available for the current stock, a new empty stock is created. In this way, the agent repeatedly selects the item until all items are allocated to stocks. Since only one item as an action is selected at each stage without considering the combinatorial relationships between different items, it is hard to find the optimal or a good solution.

Since CSP is a typical combinatorial optimization problem, we believe considering the combinatorial relationships between items will lead to a better solution. Accordingly, a nature idea is to define the combination of items as an action. However, taking the combinations of items as the action space is impractical due to the "curse of dimensionality". For example, enumerating all combinations of 100 items results in 2^{100} possible actions in the action space. Additionally, CSP often has various constraints that are difficult for RL to handle.

Drawing inspiration from [39] and [40], we propose an approach that combines reinforcement learning with optimization methods. Such an approach can improve the solution quality for CSP by considering the combination feature. Some original decisions are involved in the combinations of items, and then these combinations form the action space. To overcome the "curse of dimensionality", column generation is adopted to enumerate all promising actions implicitly. In addition, we design a monotonicity cut strategy based on monotone comparative statics to ensure action compactness.

In general, there are two types of COPs: grouping and sorting. Grouping problems involve grouping the elements of a set into several subsets, such as the CSP, bin packing problem, and knapsack problem. On the other hand, sorting problems involve sorting the elements within each subset, such as the vehicle routing problem and the traveling salesman problem. Our proposed method solves the CSP. However, it can be adopted for solving other grouping problems. Such a method can also be referred to when solving other COPs. What is the most important in our method is the idea to take advantage of the combination features in the learning process and the way of doing so. We consider the combination features using column generation. Any approaches that consider the combination features can also be adopted, such as other mathematical optimization approaches and methods directly based on the propositions of the problems. Our proposed MDP and the related graph convolutional neural network can be adopted directly after designing the action space. Moreover, if the monotone comparative statics holds in the specific problem, the monotonicity cut strategy can also be adopted.

Our contributions are as follows.

- We propose an MDP framework for COPs that constructs the action space based on the optimization method and the monotonicity cut strategy, thereby addressing the challenge of constraints in RL.
- We customize a bipartite graph to represent the structure of the MDP for grouping problems and design vertex and edge features according to the column generation method.
- We propose a bipartite graph convolutional layer for the customized graph and design actor and critic networks based on this layer. These networks are trained using the proximal policy optimization (PPO) algorithm. Experimental results demonstrate the effectiveness of our method.

The paper is structured as follows. In Section II, we provide some preliminary information and basic theoretical frameworks. In Section III, we present our proposed method, which includes the Markov decision process, the bipartite graph convolutional layer, the related actor/critic network, and

the monotonicity cut. Section IV reports on the experimental results, while Section V summarizes the conclusions of the paper.

II. Preliminaries

This section briefly introduces the basic concept related to our proposed method.

A. Cutting Stock Problem

We are given m item types, each having an integer weight w_j and an integer demand d_j ($j = 1, \cdots, m$), and a sufficiently large number of identical stocks of integer capacity z. Let u be any upper bound on the minimum number of stocks needed, and assume that the potential stocks are numbered as $1, \cdots, u$. The binary decision variable is introduced in (1)

$$y_i = \begin{cases} 1, & \text{if stock } i \text{ is used in the solution;} \\ 0, & \text{otherwise} \end{cases} \quad (i = 1, \cdots, u) \tag{1}$$

Let ξ_{ij} be the number of items of type j packed into stock i ($i = 1, \cdots, u$; $j = 1, \cdots, m$). The formulation of the CSP is as follows.

$$\begin{aligned} \min \quad & \sum_{i=1}^{u} y_i \\ \text{s.t.} \quad & \sum_{j=1}^{m} w_j \xi_{ij} \le z y_j \quad i = 1, \cdots, u \\ & \sum_{i=1}^{u} \xi_{ij} \ge d_j \quad j = 1, \cdots, m \\ & y_i \in \{0, 1\} \quad i = 1, \cdots, u \\ & \xi_{ij} \ge 0, \text{integer} \quad i = 1, \cdots, u; j = 1, \cdots, m \end{aligned}$$

B. Reinforcement Learning

RL is used to train an agent, which selects an action at each stage of the MDP. MDP is composed of five components: 1) state set $\mathcal{S}$; 2) action set $\mathcal{A}$; 3) transition function $\mathcal{F}$; 4) immediate reward function $\mathcal{R}$ and 5) discount factor γ. These components can be written using a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{R}, \gamma)$.

The goal of the agent is to find an action sequence for all stages in $\mathcal{T} = \{0, \cdots, \tau\}$ to maximize the accumulative reward, which is presented as the optimization problem (2).

$$\max_{a_t \in \mathcal{A}_t(s_t), t \in \mathcal{T}} \sum_{t \in \mathcal{T}} \mathcal{R}(s_t, a_t) \tag{2}$$

The agent selects the actions a_t from the action set $\mathcal{A}_{s_t}$ according to the value function $V_\theta(s_t)$ or the policy function $\pi_\theta(s_t)$. The most commonly used method to optimize value function parameters is to minimize Bellman error using the temporal difference method derived from the Bellman equation (3). As shown in (4), the Bellman error is minimized according to the set of transition data $\mathcal{D}$, which is sampled in the MDP. The transition data $d \in \mathcal{D}$ is formed by tuple $d = (s_t, a_t, r_t, s_{t+1})$, where $d_s = s_t$, $d_a = a_t$, $d_r = r_t$ and $d_{s'} = s_{t+1}$.

$$V_\theta(s_t) = \max_{a_t} \{\mathcal{R}(s_t, a_t) + V_\theta(s_{t+1})\}, \forall t \in \mathcal{T} \tag{3}$$

$$\min_\theta \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} [V_\theta(d_s) - (\mathcal{R}(d_s, d_a) + V_\theta(d_{s'}))]^2 \tag{4}$$

Regarding the policy function, the most commonly used method is policy gradient, as shown in (5), where $\bar{A}(d) = Q(d_s, d_a) - V(d_s)$

$$\min_\theta \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \log \pi_\theta(d_a | d_s) \bar{A}(d) \tag{5}$$

After the V_θ or the π_θ are well trained, action at each stage is selected using formulations $a_t = \arg\max_{a_t} \{\mathcal{R}(s_t, a_t) + V_\theta(s_{t+1})\}$ or $a_t = \pi_\theta(s_t)$ for a deterministic policy, else $a_t \sim \pi_\theta(s_t)$ for a stochastic policy.

C. Column Generation

Column generation is an efficient technique for solving large-scale (enormous variables or constraints) linear optimization problems. Consider a large-scale optimization problem, which is formulated as a linear programming formulation (6) with a block-diagonal structure.

$$Z = \min_x \left\{ \sum c^T x : Bx \ge b, Dx \ \le d, x \ge 0 \right\} \tag{6}$$

where c, x, b and d are vectors, B and D are matrix. We define $X = \{x | Dx \le d, x \ge 0\}$ to be a polytope. According to the Minkowski Resolution Theorem [41], any point in this polytope can be written as a convex combination of the extreme points of the polytope. Let A be the extreme points set of the polytope X. Then any point $\bar{x}$ in the polytope can be obtained by $\bar{x} = \sum_{a \in A} \lambda_a a, \sum_{a \in A} \lambda_a = 1$. By substituting $\bar{x}$ into Z, the resulting linear programming (LP) formulation (7) is obtained.

(LP)

$$Z_{\mathrm{L}} = \min_{\lambda_a} \left\{ \sum_{a \in A} r(a) \lambda_a : \sum_{a \in A} \mathcal{J}(a) \lambda_a \ge b, \lambda_a \ge 0, \sum_{a \in A} \lambda_a = 1 \right\} \tag{7}$$

where $r(a) = c^T a$ and $\mathcal{J}(a) = Ba$. As defined above, A is the extreme points set of the polytope $X = \{x | Dx \le d, x \ge 0\}$. To optimally solve the LP formulation 7, it is essential to obtain all extreme points of X. Enumerating all variables and columns is too costly when $|A|$ is huge. Instead, column generation solves LP by only considering part of the variables, which could avoid enumerating all decision variables. When adopting column generation, a restricted main problem (RMP) (8) and a pricing sub-problem (SP) (9) are established.

(RMP)

$$Z_{\mathrm{M}} = \min_{\lambda_a} \left\{ \sum_{a \in \mathcal{A}} r(a) \lambda_a : \sum_{a \in \mathcal{A}} \mathcal{J}(a) \lambda_a \ge b, \lambda_a \ge 0, \sum_{a \in A} \lambda_a = 1 \right\} \tag{8}$$

$$\text{(SP) } Z_{\mathrm{S}} = \min_a \{f(\alpha, a) : a \in \mathcal{A}\} \tag{9}$$

In RMP, only a subset of columns $\mathcal{A} \subseteq A$ are considered, columns $a \in \mathcal{A}$ are generated by the SP, the related constraints coefficient is $\mathcal{A}$, the right-hand side b is the limitation of the

constraints. In SP, the objective function $f(\alpha, a)$ is derived according to the dual theory, in which α are the dual variables of the RMP constraints. The decision variables of the SP are coefficients in the column that the RMP considers.

When adopting column generation to solve the linear program, the RMP with partial variables and columns is first solved, and the optimal values of dual variables α are obtained to update the pricing sub-problem. Then, the pricing sub-problem is optimally solved, whose optimal objective value Z_S is called reduced cost. If $Z_S < 0$, the obtained new column a by solving SP is added into $\mathcal{A}$ of the RMP, i.e., $\mathcal{A} := \mathcal{A} \cup \{a\}$. At the next iteration, RMP with the updated $\mathcal{A}$ is optimally solved again. Iteration stops when the objective value of SP is greater than or equal to 0, which implies no more column should be added into the RMP and the current solution of RMP is also the optimal one for the original linear program.

D. Bipartite Graph Convolutional Network

In this work, we use a bipartite graph to present the MDP and then design a bipartite graph convolutional network to learn knowledge from the MDP. A graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ is bipartite if and only if $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$, $\mathcal{V}_1 \cap \mathcal{V}_2 = \varnothing$, $v_1 \in \mathcal{V}_1$, and $v_2 \in \mathcal{V}_2$ for all $e_{(v_1,v_2)} = (v_1, v_2) \in \mathcal{E}$. The neighbors set of v_1 is $\mathcal{N}(v_1)$, a subset of $\mathcal{V}_2$, i.e., $\mathcal{N}(v_1) \subset \mathcal{V}_2$. The most commonly used graph filtering operations can be described as (10) and (11) where $h_{v_1}^{(l-1)}$ denotes the feature of v_1 at layer $h-1$, $h_{v_2}^{(l-1)}$ denotes the feature of v_2 at layer $h-1$, and $h_{e_{(v_1,v_2)}}$ denotes the feature of the edge $e_{(v_1,v_2)}$. *AGG* is used to denote a differentiable, permutation invariant function, e.g., sum, mean or max, the σ and Φ are used to denote differentiable functions such as multi-layer perceptions. The formulation (10) is used to obtain the node representation of node v_1 after the l-th layer. The meaning of (11) is similar to (10).

$$h_{v_1}^{(l)} = \sigma\left(h_{v_1}^{(l-1)}, AGG_{v_1}^{(l)}\left(\Phi\left(h_{v_1}^{(l-1)}, h_{v_2}^{(l-1)}, h_{e_{(v_1,v_2)}}\right)\right)\right), \quad v_2 \in \mathcal{N}(v_1) \tag{10}$$

$$h_{v_2}^{(l)} = \sigma\left(h_{v_2}^{(l-1)}, AGG_{v_2}^{(l)}\left(\Phi\left(h_{v_2}^{(l-1)}, h_{v_1}^{(l-1)}, h_{e_{(v_2,v_1)}}\right)\right)\right), \quad v_1 \in \mathcal{N}(v_2) \tag{11}$$

E. Monotone Comparative Statics

Monotone comparative statics, which is proposed by [42], is mainly concerned with the structural properties of these problems, i.e., the optimal solution set is monotonic. Moreover, some handy analytical tools are also developed in [43], i.e., supermodularity and complementarity. A parameterized optimization problem using state is obtained at each stage of MDP. The monotonicity can be used to compact the action set and further improve the solution quality. We call such an action set compact strategy monotonicity cut.

III. Solution Method

As described in Section I, the existing related works only consider one single decision as an action, disregarding the combinatorial effects of decisions. However, the objective value of RL can be improved by considering such effects.

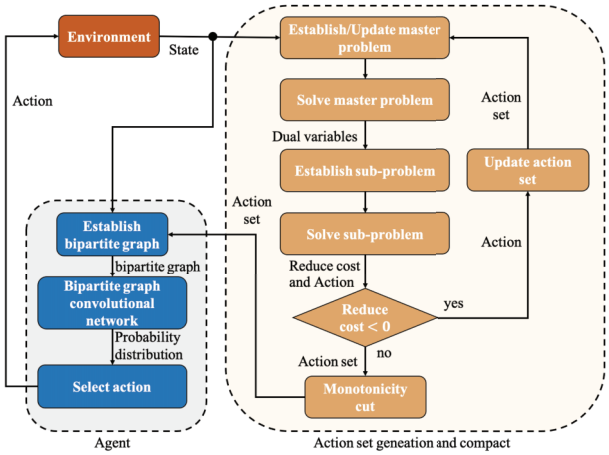

Fig. 1. This figure illustrates the overall process of our proposed method. The yellow processes represent the action set generation and compaction process. The column generation algorithm is run in a loop to generate new actions. As described in Section II-C, new actions are generated by establishing or updating the RMP, solving the RMP to obtain dual variables, establishing the SP based on the dual variables, solving the SP to obtain the new action and the reduced cost, and checking whether the new action should be added to the action space. This loop continues until a stopping criterion is met. Accordingly, the action set can be compacted using the monotonicity cut during the process. Once the action space is updated, the agent uses the state and action space to select an action to be implemented, as shown in blue. First, a bipartite graph is created using the state and action space. Then, a customized bipartite graph convolutional network is used to obtain the probability distribution of the selected action. Finally, the implemented action is chosen based on the probability distribution. After the action is implemented, the environment is updated, a new state is observed, and the process is repeated until the problem is solved.

Additionally, constraints are common in COPs, and they can be classified as cumulative or instantaneous constraints. The former restricts the action sequence, while the latter restricts each stage's action. RL inadequately addresses constraints caused by cumulative constraints.

In this paper, we propose a method that utilizes Dantzig-Wolfe decomposition and column generation to establish an MDP that considers the quality of actions and the constraints of subsets. The high-quality action space is established under the guidance of optimization methods that can capture the problem's mathematical structure. Our proposed method consists of three steps. Firstly, we establish the MDP using column generation. Secondly, we represent the MDP using a bipartite graph. Thirdly, we design two bipartite graph convolutional networks for actor and critic networks. Moreover, we introduce an action space compact strategy, namely the monotonicity cut, for appropriate problems to enhance the action space further. Figure 1 illustrates the overall process of our proposed method.

A. Markov Decision Process

In current RLs for combinatorial optimization problems, the actions are usually designed as the decision variable, that is, each element of x. According to the section II-C, any decision variable can be represented as the combinantion of the extreme points of the given polytope $X = \{x | Dx \leq d, x \in \mathbb{Z}_+^n\}$. In this work, we define the selection of extreme points, that is $a \in A$, as the action. However, due to the "curse of dimensionality",

it is impractical to enumerate all extreme points to obtain the entire action space. Column generation is an effective algorithm for solving linear programs with exponentially many variables. The main advantage of column generation is that not all possibilities need to be enumerated. Only the variables that have the potential to improve the objective function are generated. Hence, we adopt column generation algorithm to dynamically generate high-quality actions instead of all possible actions.

In column generation, the objective value of the RMP can be improved by adding new columns. The dual variables of the RMP constraints describe the urgency of satisfying them. The objective function of the SP is formed using the dual variables of the RMP constraints. Therefore, the new column generated by solving the SP can satisfy the constraints to the fullest extent possible. As mentioned early, each column corresponds to an extreme point of the given polytope. Then, the constraints that restrict the polytope can be tackled by column generation. This makes it easier for RL to handle constraints, while providing a high-quality action space. We define our MDP as follows:

- Stage: The entire decision process is divided into multiple stages according to the stock, that is, the cutting scenario for a stock is determined at each stage t. As described in Section II-A, u is the upper bound on the minimum number of stocks needed. Then, we have $t \in \mathcal{T} = \{1, 2, \cdots, u\}$. In this context, our MDP is a discrete event system, where the episode might complete at $t \le u$. For a given trajectory of actions, let τ define as its terminal stage, then the stage set is defined as $\mathcal{T} = \{1, 2, \cdots, \tau\}$.
- State: The action is selected according the information of the system. For a CSP, the information is the parameters of the CSP, that is, the item weight w, item demand d, and stock capacity z. w and z are constants while d is variable. Therefore, the state set is defined as $s \in \mathcal{S}$ as follows

$$\begin{aligned} s &= [z, w, s'] \\ &= [z, [w_1, w_2, \cdots, w_m], [s'_1, s'_2, \cdots, s'_m]] \end{aligned}$$

  where $s'_j \le d_j$ denotes the unfulfilled demands.
- Action: Action space A is defined as the set of potential cutting schemes, i.e., combinations of items, which satisfy the capacity constraint of stocks. Hence, we have $\mathcal{A} = \{a | a = [a_1, a_2, \cdots, a_m], a_i \le d_i\}$, where a_i implies the number of item i packed into the current stock.
- Transition Function: The state is updated using the transition function $\mathcal{F} : s'_{t+1} = s'_t - a_t$. In addition to the state, the action set should be updated after the implementation of a_t. At stage t, the RMP is given by (12).

$$Z^t_{\mathrm{M}} = \min_{\lambda_a} \sum_{a \in A} r(a)\lambda_a : \sum_{a \in A} a\lambda_a \ge s'_t, \lambda_a \ge 0 \tag{12}$$

  Once action a_t is implemented, the corresponding item demands are updated. Accordingly, we can get the updated RMP as (13).

$$Z^{t+1}_{\mathrm{M}} = \min_{\lambda_a} \sum_{a \in A} r(a)\lambda_a : \sum_{a \in A} a\lambda_a \ge s'_t - a_t, \lambda_a \ge 0 \tag{13}$$

  To obtain the new potential actions, the column generation is adopted to solve the linear relaxation of RMP (13) and the SP of (14).

$$Z_S = \min_a \left\{ \sum_{j=1}^{m} \pi_j : \sum_{j=1}^{m} a_j w_j \le z \right\} \tag{14}$$

  where π_j is the dual variable of the RMP constraints. Each a with negative reduced cost will be added into the column set of RMP. Meanwhile, such a is also added into the action set using $\mathcal{A} := \mathcal{A} + \{a\}$. After column generation stops, the action set is updated. At the next stage, a new action will be selected from the updated action set.
- Immediate Reward: We can obtain the reward $r(a)$ by referring to the objective function of the IP. In our MDP, reward is defined as $r(a) = 1$.

Algorithm 1 Obtain Action Set at Stage t

Input: state s_t, action set $\mathcal{A}$
Output: action set $\mathcal{A}$

```
1: initialize RMP and SP using s_t, set reduce cost δ := −1
2: while δ < 0 do
3:     update RMP using A
4:     solve RMP and obtain dual variables α
5:     establish SP using α
6:     solve SP, obtain objective value Z_S and solution a
7:     set δ := Z_S
8:     if δ < 0 then
9:         A := A + {a}
10:    end if
11: end while
12: return action set A
```

At stage t, the RL agent selects and performs action a_t from the action set $\mathcal{A}$. The state is updated, and the new state s_{t+1} is observed at stage $t + 1$. The operations repeat until the MDP is completed. The procedure to obtain action set at stage t is described in the Algorithm 1. Line 2–11 is the column generation procedure described in Section II-C.

B. Bipartite Graph Convolutional Network

We use PPO to train an agent to select action. A customized neural network architecture for the actor and critic networks is designed.

1) Bipartite Graph of Our MDP: The network receives the state and outputs the action evaluation in general RL. In our MDP, each action represents a combination of items. The definition of action set makes the general RL, such as sequence-to-sequence and graph neural network based approaches, cannot be adopted directly. In the approach based on sequence-to-sequence structure, such as the pointer network [29], the all input elements are sorted and output as a sequence. In our MDP, the output is a subset of the action set. Moreover, the cardinality of the action set varies across stages. Therefore, the sequence-to-sequence structure is not appropriate for our MDP. In addition, the graph neural networks in the published papers are unavailable for our MDP, although they can tackle

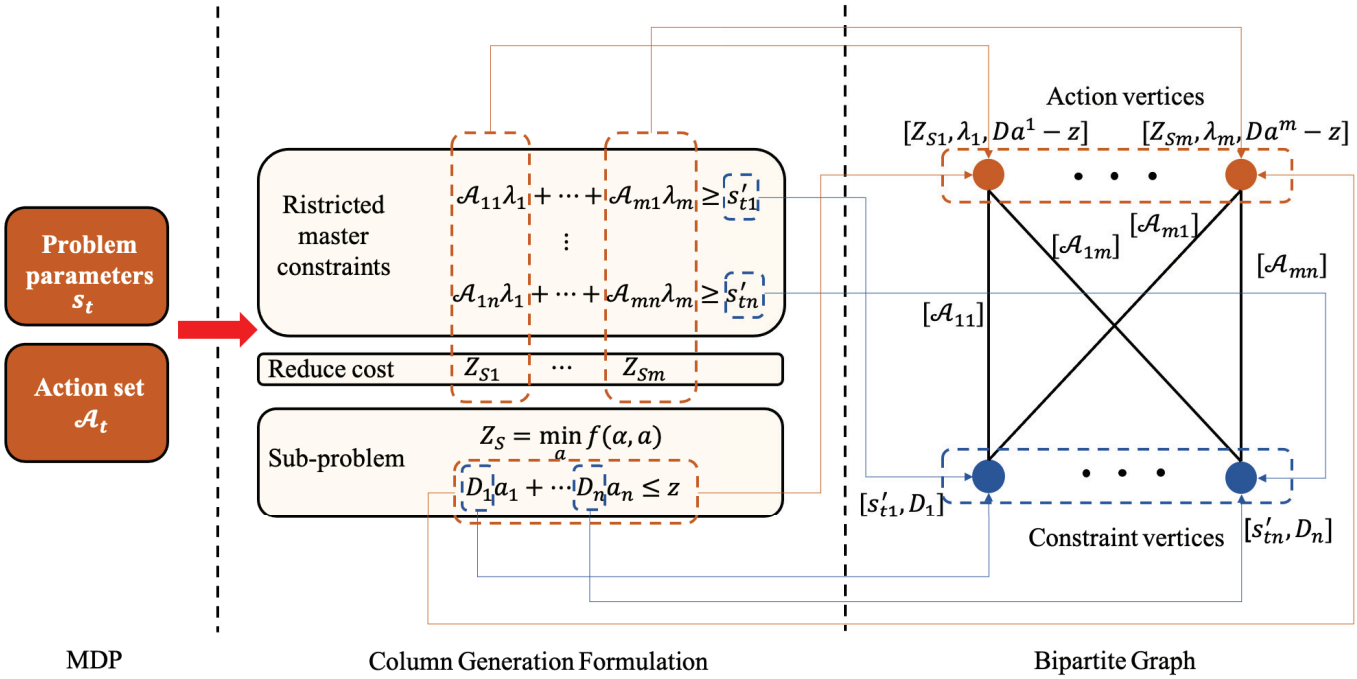

Fig. 2. General process of MDP.

action sets with changeable cardinality. In the published articles such as [44], the graphs are usually used to describe the relationship with a physical meaning. For example, in the VRP and the TSP, the vertices denote customers or cities. The vertices in the related articles are action candidates at each stage. In other words, these graphs only contain action vertices but not state vertices. Establishing a graph containing both action and state vertices is not processable by the published articles. In our MDP, the actions and the state should be considered while selecting action. Hence, this paper's graph and network structure must be customized according to our MDP framework. Moreover, a special structure of our MDP could be utilized to obtain well-performing features.

In our MDP, a state represents the remaining problem given the actions already taken so far, and an action corresponds to a column in the optimization formulation. It is challenging to represent the relationship between them in a graph. However, when we obtain the action set at each stage, we need to establish or update the optimization formulations, that is, the RMP and the SP. The state and action features as well as their connections are contained in the RMP and SP, so we can design the graph accordingly. This graph architecture can suit our MDP as described later.

Following the discussion above, we represent our MDP using a bipartite graph under the guide of column generation. The proposed bipartite graph can capture the structure of our MDP. The bipartite graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ for our MDP is defined as follows. The vertex set $\mathcal{V}$ is formed by the constraint vertex set $\mathcal{V}_c$ and the action vertex set $\mathcal{V}_a$. The edge set $\mathcal{E}$ describes the relationship between the constraint and action vertices. How to establish the bipartite graph according to our MDP is illustrated in Fig. 2.

Based on the established bipartite graph, the actor and critic networks should be designed to select the actions and evaluate the states. The actor-network selects an action based on the evaluation of the actions, while the critic network evaluates the state. To evaluate the actions, we need to consider the features of the action vertices in the bipartite graph, which should also consider the impact of the constraint vertices. When evaluating the state, we aim to estimate the accumulated reward from the current state to the terminal state, which can be represented as $V(s_t) = \sum_{k=t}^{\tau} r(a_k)$. As mentioned earlier, the state s_t is mapped to the bipartite graph $\mathcal{G}_t$ through Z_{M} and Z_{S}. Therefore, the value of the state can be expressed as $V(s_t) = V(\mathcal{G}_t)$.

While calculating the $V(s_t)$, if the terminal state is observed, which indicates that the problem has been solved, as illustrated in Fig. 4. For the RMP, all constraints are satisfied in the terminal state. For more details, each constraint is satisfied partially by performing an action. When all constraints are satisfied, then a complete feasible solution is obtained, that is, the problem is solved. Therefore, the value of the state could be evaluated by estimating the accumulated reward before all constraints are satisfied. To this end, we can estimate the accumulated reward of constraints before they are satisfied according to their embedding. After that, we can evaluate the value of the state by adding them up. Following this scheme, the reward at each stage is decomposed as the reward of the constraints when they are partially satisfied. Considering the RMP of the $\mathcal{G}_t$ has a constraint vertex set $\mathcal{V}_c^t$, the estimation of the v_c^t is denoted as $\hat{V}(v_c^t)$, the state value is represented as $V(s_t) = V(\mathcal{G}_t) = \sum_{v_c^t \in \mathcal{V}_c^t} \hat{V}(v_c^t)$. Fig. 3. illustrates the details of the evaluation of the state value more intuitively.

Based on the established bipartite graph, actor and critic networks are designed. These networks will incorporate the

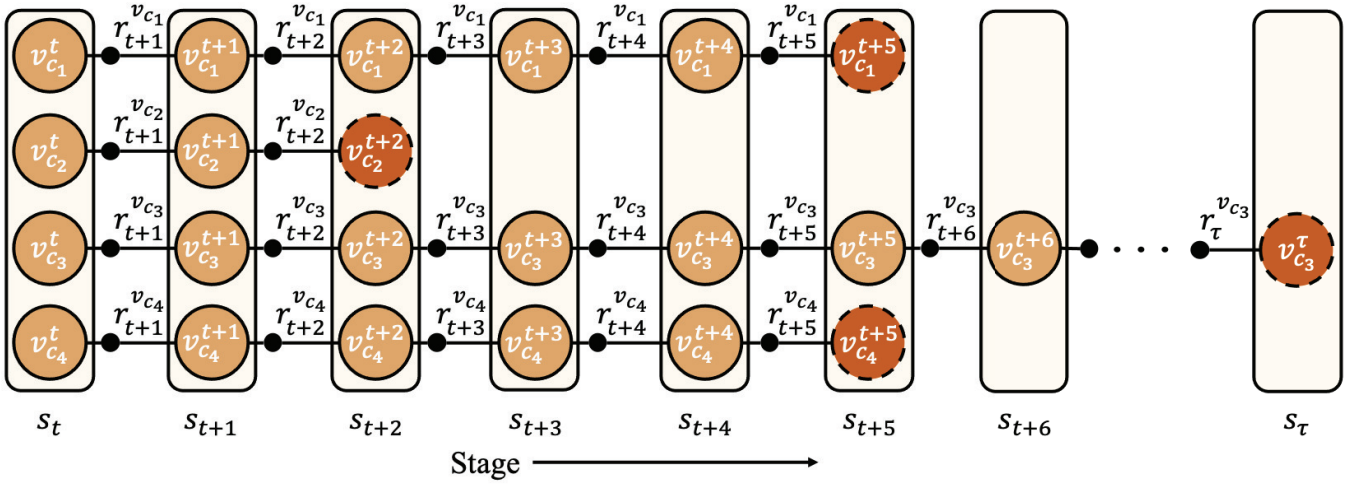

Fig. 3. The figure illustrates the graph representation of the MDP, where the light yellow rectangle covers the vertices representing the constraints of the RMP in the current stage. Each implemented action satisfies the constraints partially, and when all constraints are satisfied, the problem is solved, and the terminal state is observed. The constraint vertices labeled in yellow with a full line indicate the related constraints that are unsatisfied, while the ones labeled in red with a dotted line indicate the related constraints that are satisfied. For a problem with four constraints, the reward is implicitly decomposed into four parts, and hence, we have $r_{t+1} = \sum_{i\in 1,2,3,4} r_{t+1}^{v_{c_i}}$. If constraint c_2 is satisfied at stage $t+2$, the relationship of rewards at stage $t+3$ is $r_{t+3} = \sum_{i\in 1,3,4} r_{t+3}^{v_{c_i}}$. At stages $t+6$ and later, only constraint c_3 is unsatisfied, and the relationship of the reward is $r_k = r_k^{v_{c_3}}$, where $t+6 \le k \le \tau$. The value of the state can be formulated as $V(s_t) = \sum_{k=t+1}^{\tau} r_k = \sum_{k=t+1}^{\tau} \sum_{i\in I} r_k^{v_{c_i}}$. The accumulative reward of constraint c_i at stage t can be represented as $V(v_{c_i}^t) = \sum_{k=t+1}^{\tau} r_k^{v_{c_i}}$. For example, $V(v_{c_1}^t) = \sum_{k=t+1}^{t+5} r_k^{v_{c_1}}$. Consequently, the value of the state can be reformulated using $V(s_t) = \sum_{k=t+1}^{\tau} r_k = \sum_{v\in\mathcal{V}_c} V(v^t) = V(\mathcal{G}t)$, where v^t is the constraint vertex at stage t. Instead of estimating the value of the state directly, we can evaluate the value of the constraints using the approximate value function $\hat{V}(v_c)$ and add them up.

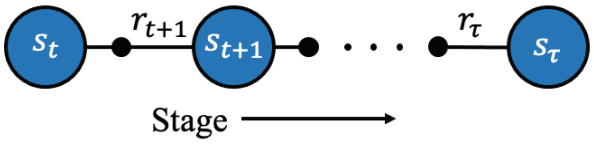

Fig. 4. In the MDP, the agent receives feedback as a reward r_t after executing the action a_t. This leads to the observation of a new state s_{t+1}, and this process continues until the terminal state is reached, indicating that the problem has been solved. The value of the state can be computed as $V(s_t) = \sum_{k=t+1}^{\tau} r_k$.

features of the bipartite graph, the bipartite graph convolution layer, and the output operator, as described earlier.

2) Feature of the Bipartite Graph: The feature of the constraint vertices is determined by the coefficients s'_t and D in the RMP and the SP. The coefficient s'_t is the right-hand side coefficient of the RMP constraints, while D is the coefficient of the constraints in the SP. The feature of each constraint vertex, denoted by F_{v_c}, is defined as $F_{v_c} = [s'_{tv_c}, D_{v_c}]$, where n is the number of constraints in the RMP. During the column generation algorithm, each generated column a has an objective value Z_S^a, representing the column's reduced cost. We use the generated columns to solve the updated RMP and obtain the decision variables λ. The value of the decision variables in the RMP represents the fractional solution, which indicates how much the columns are used. The feature of each action vertex, denoted by F_{v_a}, is defined as $F_{v_a} = [Z_S^a, \lambda^a, Da - z]$. In our MDP, the relationship between the constraint c and the column a is represented by a^c. Although there are at most $n \times |\mathcal{A}|$ edges, we only consider the edge $e(c,a)$ when $a^c \neq 0$. The feature of the edge $e_{(v_c,v_a)}$ is $F_{e(v_c,v_a)} = [a^c]$. We define the embedding operator as (15), where β denotes the LeakyReLU activation function and ζ denotes several FullConnection layers. The embedding of the vertices and edges, denoted by $h_{v_c}^{(0)}$, $h_{v_a}^{(0)}$, and $h_{e_{(v_c,v_a)}}$, can be obtained by adapting Emb as $h_{v_c}^{(0)} = \beta\left(\zeta_{v_c,1}^{Emb}\left(\zeta_{v_c,2}^{Emb}(F_{v_c})\right)\right)$, $h_{v_a}^{(0)} = \beta\left(\zeta_{v_a,1}^{Emb}\left(\zeta_{v_a,2}^{Emb}(F_{v_a})\right)\right)$, and $h_e = \beta\left(\zeta_{e,1}^{Emb}\left(\zeta_{e,2}^{Emb}(F_e)\right)\right)$.

$$\text{(Emb)} \quad h^{(0)} = \beta\left(\zeta\left(\zeta(F)\right)\right) \tag{15}$$

3) Bipartite Graph Convolutional Operator: With the embedding layer, we can calculate the embedding of the bipartite graph using the bipartite graph convolutional operator based on the message-passing framework. We design (16) and (17) to pass messages from layer $l-1$ to layer l, where $h_{v_a}^{(l)}$ and $h_{v_c}^{(l)}$ are the embeddings of vertices v_a and v_c at layer l, respectively.

$$\begin{aligned}&(\text{BGCov}_{v_c}^{(l)})\\ &h_{v_c}^{(l)} = \sigma\left(h_{v_c}^{(l-1)}, AGG_{v_c}^{(l)}\left(\Phi\left(h_{v_c}^{(l-1)}, h_{v_a}^{(l-1)}, h_{e_{(v_c,v_a)}}\right)\right)\right)\end{aligned} \tag{16}$$

$$\begin{aligned}&(\text{BGCov}_{v_a}^{(l)})\\ &h_{v_a}^{(l)} = \sigma\left(h_{v_a}^{(l-1)}, AGG_{v_a}^{(l)}\left(\Phi\left(h_{v_a}^{(l-1)}, h_{v_c}^{(l-1)}, h_{e_{(v_c,v_a)}}\right)\right)\right)\end{aligned} \tag{17}$$

In (16) and (17), ε represents the activation function of Sigmoid, Φ represents the message function, and σ represents the update function. The message operator and the update operator are defined in (18) and (19).

$$\begin{aligned}&\Phi\left(h_v^{(l-1)}, h_u^{(l-1)}, h_{e_{(v,u)}}\right)\\ &= \beta\left(\zeta_m^{(l)}\left(\zeta_v^{(l)}\left(h_v^{(l-1)}\right) + \varepsilon\zeta_e^{(l)}\left(h_{e_{(v,u)}}\right) \times \zeta_u^{(l)}\left(h_u^{(l-1)}\right)\right)\right)\end{aligned} \tag{18}$$

$$\sigma(h, h') = \beta\left(\zeta^{(l)}\left([h; h']\right)\right) \tag{19}$$

The aggregation operator uses different definitions for v_c and v_a, where $AGG^{(l)}v_c$ takes the mean and $AGG^{(l)}v_a$ takes summation. Domain knowledge was considered while designing the message operator Φ and the aggregation operator AGG. In the bipartite graph, all actions have the potential to impact the constraint vertex, but only one action will be implemented, and the message will be passed to the constraint vertex. To evaluate the comprehensive impact of actions, we set the constraint aggregation as "mean". Unlike constraint vertices, an action vertex is impacted by all constraint vertices. Thus, we set the action aggregation as "summation". When passing the message from previous layer vertices to current layer vertices, the edge embedding acts as an amplifier, amplifying the influence of neighbor vertices. Hence, we multiply the embedding of the edge by its neighbor vertices' embedding to obtain the neighbor vertices' message. Using the definitions above, we can obtain the output of the BGC operator in the actor network using $h_{v_a}^{(3)} = \text{BGCactor}(hv_c{}^{(0)}, h_{v_a}^{(0)}, h_{e_{(v_c,v_a)}})$, as described in (20).

$$
\begin{aligned}
&(\text{BGC}_{\text{actor}})\\
&h_{v_c}^{(1)} = \text{BGCov}_{v_c}^{(1)}(h_{v_c}^{(0)}, h_{v_a}^{(0)}, h_{e_{(v_c,v_a)}})\\
&h_{v_a}^{(1)} = \text{BGCov}_{v_a}^{(1)}(h_{v_c}^{(1)}, h_{v_a}^{(0)}, h_{e_{(v_c,v_a)}})\\
&h_{v_c}^{(2)} = \text{BGCov}_{v_c}^{(2)}(h_{v_c}^{(1)}, h_{v_a}^{(1)}, h_{e_{(v_c,v_a)}})\\
&h_{v_a}^{(2)} = \text{BGCov}_{v_a}^{(2)}(h_{v_c}^{(2)}, h_{v_a}^{(1)}, h_{e_{(v_c,v_a)}})\\
&h_{v_c}^{(3)} = \text{BGCov}_{v_c}^{(3)}(h_{v_c}^{(2)}, h_{v_a}^{(2)}, h_{e_{(v_c,v_a)}})\\
&h_{v_a}^{(3)} = \text{BGCov}_{v_a}^{(3)}(h_{v_c}^{(3)}, h_{v_a}^{(2)}, h_{e_{(v_c,v_a)}})
\end{aligned}
\tag{20}
$$

In the same way, we can obtain the output of the BGC operator in the critic network by using the equation $h_{v_c}^{(3)} = \text{BGC}_{\text{critic}}(h_{v_c}^{(0)}, h_{v_a}^{(0)}, h_{e_{(v_c,v_a)}})$, as described in equation (21).

$$
\begin{aligned}
&(\text{BGC}_{\text{critic}})\\
&h_{v_a}^{(1)} = \text{BGCov}_{v_a}^{(1)}(h_{v_c}^{(0)}, h_{v_a}^{(0)}, h_{e_{(v_c,v_a)}})\\
&h_{v_c}^{(1)} = \text{BGCov}_{v_c}^{(1)}(h_{v_c}^{(0)}, h_{v_a}^{(1)}, h_{e_{(v_c,v_a)}})\\
&h_{v_a}^{(2)} = \text{BGCov}_{v_a}^{(1)}(h_{v_c}^{(1)}, h_{v_a}^{(1)}, h_{e_{(v_c,v_a)}})\\
&h_{v_c}^{(2)} = \text{BGCov}_{v_c}^{(2)}(h_{v_c}^{(1)}, h_{v_a}^{(2)}, h_{e_{(v_c,v_a)}})\\
&h_{v_a}^{(3)} = \text{BGCov}_{v_a}^{(3)}(h_{v_c}^{(2)}, h_{v_a}^{(2)}, h_{e_{(v_c,v_a)}})\\
&h_{v_c}^{(3)} = \text{BGCov}_{v_c}^{(3)}(h_{v_c}^{(2)}, h_{v_a}^{(3)}, h_{e_{(v_c,v_a)}})
\end{aligned}
\tag{21}
$$

We adopt a synchronous approach to compute vertex embeddings, which enables us to extract features of vertices more efficiently compared to the commonly used bipartite graph convolutional network.

4) Output Operator: Having obtained $h_{v_a}^{(3)}$ and $h_{v_c}^{(3)}$ through $\text{BGC}_{\text{actor}}$ and $\text{BGC}_{\text{critic}}$, we can now compute the output of the actor network and the critic network using equations (22) and (23), respectively. As discussed in Section III-B1, the state value can be evaluated by summing up the evaluations of the constraint vertices, and the probability distribution of selected actions can be obtained using the Softmax function. Our approach of synchronously computing the vertex embeddings enables faster feature extraction compared to the commonly used bipartite graph convolutional network.

$$
\text{Prob} = \text{Softmax}\left(\zeta_{\text{actor},1}^{\text{output}}\left(\zeta_{\text{actor},2}^{\text{output}}(h_{v_a}^{(3)})\right)\right) \tag{22}
$$

$$
V = \text{Sum}\left(\zeta_{\text{critic},1}^{\text{output}}\left(\zeta_{\text{critic},2}^{\text{output}}(h_{v_c}^{(3)})\right)\right) \tag{23}
$$

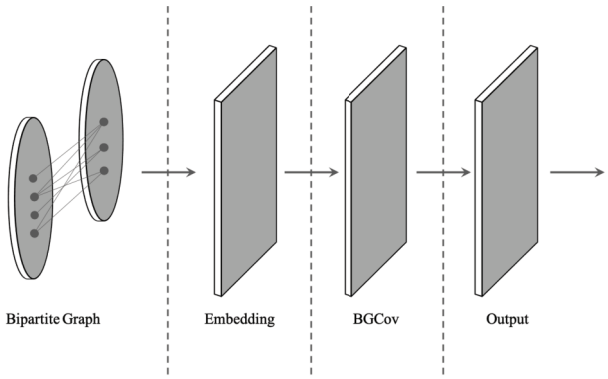

Fig. 5. The overall architecture of our BGCN model.

The actor and critic networks share the same architecture but differ in the details of the BGC and output operators. Fig. 5 illustrates the architecture of the bipartite graph convolutional network.

C. Proximal Policy Optimization and Value Function Approximation

Let value function $V(s_t|\theta_c)$ be the evaluation of state s_t, that is, the output of the critic network. It can be calculated using (23), where θ_c is the parameters of the critic network. Let policy $\pi(s_t|\theta_a)$ be the probability distribution of state s_t, that is, the output of the actor network. It can be calculated using (20). The policy is used to obtain a solution, which is a trajectory of the MDP, that is, $\{s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_\tau, a_\tau, r_\tau\}$. At stage t, we observe state s_t and action set $\mathcal{A}$, and the related bipartite graph $\mathcal{G}_t$. After that, the feature of $\mathcal{G}_t$ is obtained. Accordingly, the probability distribution of the actions, i.e., $\pi(s_t|\theta_a)$, can be calculated using the actor network. Finally, the action is selected using (24), where $\pi(a|s_t, \theta_a)$ denotes the probability of a being selected when state is s_t and actor parameters is θ_a.

$$
a_t \sim \pi(s_t|\theta_a) \tag{24}
$$

The relationship of $\pi(a|s_t, \theta_a)$ and $\pi(s_t|\theta_a)$ is given by $\pi(s_t|\theta_a) = \{\pi(a|s_t, \theta_a), a \in \mathcal{A}\}$ The policy is stochastic, i.e., (24) for training parameters of networks and deterministic, i.e., (25), for evaluating our approach.

$$
a_t = \arg\max_x \pi(x|s_t, \theta_a) \tag{25}
$$

Referring to the PPO [18], the actor and critic networks are updated as follows sections.

1) Update Critic Network: Actor and critic networks are updated according to the trajectory, which is obtained by stochastic policy. At stage t, action is selected using (24). In our approach, these two networks are trained on the stochastic scheme, and the deterministic scheme is adopted to obtain solutions according to the trained actor network. The trajectory and related returns are obtained using Algorithm 2.

Let advantage $\widetilde{A}_t$ is calculated using (26).

$$
\widetilde{A}_t(\theta_c) = G_t - V(s_t|\theta_c) \tag{26}
$$

The critic network are updated by minimizing the mean square of advantage, which is described in Algorithm 3.

Algorithm 2 The Procedure of Obtaining a Trajectory

Input: actor parameters θ_a, initial state s_1
Output: trajectory $\{s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_\tau, a_\tau, r_\tau\}$ and returns $\{G_1, G_2, \cdots, G_\tau\}$

for $t = 1$ to τ **do**
  observe state s_t
  obtain action set $\mathcal{A}$ using Algorithm 1
  construct bipartite graph $\mathcal{G}_t$
  obtain action a_t using (24)
  receive reward r_t
  update trajectory $\{s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_t, a_t, r_t\}$
end for
$G_\tau \leftarrow r_t$
for $t = \tau - 1$ to 1 **do**
  $G_t \leftarrow G_{t+1} + r_t$
end for
return $\{s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_t, a_t, r_t\}$, $\{G_1, G_2, \cdots, G_\tau\}$

Algorithm 3 The Critic Update Procedure

Input: trajectory $\{s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_\tau, a_\tau, r_\tau\}$, critic parameters θ_c, and returns $\{G_1, G_2, \cdots, G_\tau\}$
Output: updated critic parameters θ_c

for $i = 1$ to *epochs* **do**
  $\theta_c \leftarrow \arg\min_\theta \frac{1}{\tau} \sum_{t=1}^{\tau} \widetilde{A}_t^2(\theta)$
end for
return θ_c

2) Update Actor Network: Let $\beta(\theta, \theta')$ be the probability ratio, which is calculated using (27).

$$\beta(a_t|\theta, \theta') = \frac{\pi(a_t|\theta)}{\pi(a_t|\theta')} \tag{27}$$

The main objective of updating actor is $L_t(\theta|\theta')$, which can be calculated using (28)

$$L_t(\theta|\theta') = \min\left(\beta(a_t|\theta, \theta'), \text{clip}\left(\beta(a_t|\theta, \theta'), 1-\epsilon, 1+\epsilon\right)\right)\widetilde{A}_t(\theta_c) \tag{28}$$

Updating actor network is to maximize the mean square of $L_t(\theta|\theta')$, which is described in Algorithm 4.

Parameters of actor and critic networks are both optimized with the Adam optimization method [45].

D. Monotonicity Cut

In this section, we analyze the properties of the optimal solutions of our MDP, which has constructed a high-quality action space using an optimization method. We use monotone comparative statics to further improve the action set by considering the parameterized optimization problem and obtaining the increasing property of the optimal solution set. Theorem 2.4.3, Theorem 2.8.2, and Theorem 2.8.3 in [43] provide useful references for analyzing the increasing property of the optimal solution set. This property describes the mapping from the parameter to the optimal solution set, which establishes the optimization problem, namely, the parameterized optimization problem referred to above. If we treat states as parameters, monotone comparative statics is

Algorithm 4 The Actor Update Procedure

Input: trajectory $\{s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_\tau, a_\tau, r_\tau\}$, critic parameters θ_c, actor parameters θ_a, and returns $\{G_1, G_2, \cdots, G_\tau\}$
Output: updated actor parameters θ_a

$\theta_a' \leftarrow \theta_a$
for $i = 1$ to *epochs* **do**
  $\theta_a \leftarrow \arg\max_\theta \frac{1}{\tau} \sum_{t=1}^{\tau} L_t(\theta|\theta_a')$
end for
return θ_a

Algorithm 5 The Action Space Compact Procedure

Input: action history set $\mathcal{H}$, action space $\mathcal{A}$
Output: action history set $\mathcal{H}$, compacted action space $\bar{\mathcal{A}}$

$\bar{\mathcal{A}} = \{a | a \preceq a', \forall a \in \mathcal{A}, a' \in \mathcal{H}\}$
if $\bar{\mathcal{A}}$ is $\varnothing$ **then**
  $\bar{\mathcal{A}} := \mathcal{A}$
  $\mathcal{H} := \varnothing$
end if
return $\mathcal{H}$, $\bar{\mathcal{A}}$

suitable for MDPs, where state s is formed as a vector by the problem parameters, as described in Section III-A. The optimization of the MDP at each stage can be formulated as $V(s) = \max_a Q(s, a) : a \in \mathcal{A}$. If $Q(s, a)$ is supermodular, we can design a compact strategy for the action space according to the increasing property of the optimal solution set. The increasing optimal solution set implies that if $\mathcal{A}^*(s)$ is increasing, $a' \in \mathcal{A}^*(s')$ and $a'' \in \mathcal{A}^*(s'')$, then we have $a' \wedge a'' \in \mathcal{A}^*(s')$ and $a' \vee a'' \in \mathcal{A}^*(s'')$. We can design a compact strategy according to the increasing property of the optimal solution set, which we call the monotonicity cut. When selecting an action a_t at stage t and observing a new state s_{t+1}, we can obtain a new action space by compacting it using $\bar{\mathcal{A}}(s_{t+1}) = a_{t+1} | a_{t+1} \in \mathcal{A}(s_{t+1}), a^* t \preceq at + 1$. The monotonicity cut utilizes information from other stages and is represented as knowledge transfer in [46]. Algorithm 5 illustrates the monotonicity cut procedure. Note that the monotonicity cut may remove all actions. We select an action from the uncompacted action space to avoid this phenomenon, set $\mathcal{H}$ as $\varnothing$, and recollect history actions. The training process is described in Algorithm 6.

IV. Experimental Results

All experiments are performed on a computer with an Intel(R) Xeon(R) Silver 4110 CPU @ 2.10 GHz, 32 threads, 32 GB memory, NVIDIA GeForce GTX 1080 Ti GPU with 11 GB memory, and Ubuntu 18.04.2 LTS operation system.

A. Benchmark Datasets

The cutting stock problem is a classical grouping COP. The problem entails given rolls of fixed width and a set of orders for rolls of smaller widths, with the objective being to minimize the number of rolls by determining how to cut the rolls into smaller widths to satisfy the orders. To evaluate the effectiveness of our proposed agent and MDP

Algorithm 6 The Training Procedures of the BGCNMC Model

Input: training instance set $\mathcal{I}$, maximum training epoch E
Output: policy π

Initial: parameter of the policy network π and critic network $\hat{V}$ a
for $e = 1 : E$ **do**
  for $i \in \mathcal{I}$ **do**
    set action history set $h = \varnothing$
    while episode not stop **do**
      observe state s_t
      obtain action set $\mathcal{A}$ using Algorithm 1
      compact $\mathcal{A}$ using Algorithm 5
      construct bipartite graph $\mathcal{G}_t$
      obtain probability distribution prob using policy network
      select action a_t from $\mathcal{A}$ according to the probability distribution Prob
      $h := h + \{a_t\}$
      collect reward r_t by implementing action a_t, then obtain state s_{t+1} using transfer function $\mathcal{F}$
      if $t \neq 0$ **then**
      collect transition pair $(\mathcal{G}_t, a_t, r_t, \mathcal{G}_{t-1})$ using π
      end if
      set $t \leftarrow t + 1$
    end while
    update policy network and critic network using the transition pairs and PPO algorithm
  end for
end for
return policy network π

TABLE I
CLASSES OF INSTANCES PROPOSED IN THE LITERATURE

Class	Ins num [1]	Source	n	W
Falkenauer U	80	[47]	120-1000	150
Falkenauer T	80	[47]	60-501	1000
Scholl 1	720	[48]	50-500	100-150
Scholl 2	480	[48]	50-500	1000
Scholl 3	10	[48]	50-500	10000
Wäscher	17	[49]	57-239	10000
Schwerin 1	100	[50]	100	1000
Schwerin 2	100	[50]	120	1000
Hard28	28	[51]	160-200	1000
Irnich	240	[52]	125-500	500000-1500000
Random	250	[53]	50-1000	50-1000
AI	250	[53]	201-1002	2500-80000
ANI	250	[53]	201-1002	2500-80000

action space, we employ the cutting stock problem as an example. Specifically, we utilize 13 benchmark datasets of the cutting stock problem to evaluate our approach, which are Falkenauer U and Falkenauer T [47], Scholl 1, Scholl 2, and Scholl 3 [48], W"ascher [49], Schwerin 1 and Schwerin 2 [50], Hard28 [51], Irnich [52], Randomly Generated Instances [53], and Augmented Non IRUP (ANI) and Augmented IRUP (AI) Instances [53]. Further details regarding these instances are provided in Table I.

B. Baselines and Algorithms

In order to demonstrate the effectiveness of our proposed method, we compared it to the following baseline algorithms:

- Hierarchical Reinforcement Learning with Graph Pointer Networks (HRL-GPN): This algorithm, described in [37], utilizes the length of items as the item feature and does not consider packed items. The reward is defined as the number of bin numbers.
- Reinforcement Learning with Pointer Network (PTR): This algorithm, described in [34], has the same item feature and reward as the HRL-GPN.
- Ranked Reward Monte Carlo Tree Search (RRMCTS): This algorithm, described in [33], also has the same item feature and reward as the HRL-GPN.
- Reward Greedy (RG): This algorithm utilizes a greedy policy to select actions based on minimizing the trim loss.
- Fractional Greedy (FG): This algorithm also utilizes a greedy policy to select actions based on the fraction solution of the action, selecting the action with the fraction solution closest to the corresponding integer value.
- Residual Recombination Heuristic(RRH): Referring to [54] and [55], we collect the results via RRH to evaluate our approach. The RRH obtains the solution of CSP based on column generation, recombination technique, and greedy exhaustive repetition.

Our proposed algorithm is implemented in the following two versions.

- Bipartite Graph Convolutional Network (BGCN): This is basic version of our proposed method, which directly selects actions based on the probability distribution generated by the policy network.
- Bipartite Graph Convolutional Network with Monotonicity Cut (BGCNMC): This is a modification of our proposed method. Based on the policy trained by BGCN, it utilizes the monotonicity cut described above to compact the action space before evaluating the probability distribution of actions.

The code of algorithms above are avaliable.[1]

C. Computational Results

We have implemented the proposed BGCN and all the baseline algorithms mentioned above on the benchmark datasets described earlier. To evaluate the performance of these algorithms, we have used the gap metric concerning the optimal value, which is calculated using Equation (29).

$$\text{gap} = \frac{\text{result of the algorithm} - \text{optimal value}}{\text{optimal value}} \times 100\% \quad (29)$$

We have collected the results of the benchmarks mentioned above using both the baseline algorithms and our proposed algorithms. For all benchmark instances, we calculate the gaps obtained by each algorithm, and then the mean and standard deviation of gaps for each instance set. Then the mean gap and its standard deviation are rounded to 2 decimal places and reported in Table II. In addition, the mean computational

[1]https://github.com/ImfyShi/CGRL

TABLE II
(MEAN±STD) OF ALGORITHMS ON DATASETS

	RG	FG	FFD	BGCN	BGCNMC	PTR	HRL-GPN	RRMCTS	RRH
AI	0.7±0.45	**0.7±0.45**	**0.7±0.45**	**0.7±0.45**	**0.7±0.45**	**0.7±0.45**	26.28±1.59	26.39±1.56	27.09±1.86
ANI	0.37±0.25	0.37±0.25	**0.37±0.25**	0.37±0.25	**0.37±0.25**	0.37±0.25	25.89±1.38	26.02±1.35	26.71±1.48
Falkenauer_T	**11.97±2.98**	13.16±1.96	14.69±1.51	12.57±2.03	12.57±2.03	14.69±1.51	19.08±1.75	17.92±1.99	21.3±1.77
Falkenauer_U	1.37±0.59	**1.33±0.6**	1.37±0.59	1.34±0.61	1.34±0.61	1.68±0.51	39.53±2.3	39.12±2.22	35.94±13.32
Hard28_CSP	1.2±0.57	1.2±0.57	**1.2±0.57**	1.2±0.57	1.2±0.57	1.2±0.57	38.08±3.23	38.19±2.86	39.88±2.76
Irnich_CSP	1.11±0.71	1.11±0.71	1.11±0.71	1.11±0.71	**1.11±0.71**	3.86±1.02	40.89±2.72	40.97±2.7	41.21±2.7
Randomly_Generated_CSP	0.63±0.8	0.6±0.78	0.65±0.83	0.6±0.78	**0.6±0.78**	1.07±0.98	36.9±4.98	36.63±5.08	30.64±16.36
Scholl_1	0.43±0.98	**0.43±0.98**	0.47±1.04	0.43±0.99	0.43±0.99	0.85±1.18	27.73±9.47	27.43±9.43	21.39±16.53
Scholl_2	2.58±3.6	2.51±3.57	3.01±3.87	**2.36±3.52**	2.38±3.54	3.09±3.91	7.04±12.32	6.81±12.3	12.15±9.5
Scholl_3	**1.25±1.16**	1.61±1.26	6.06±0.94	1.43±1.07	1.43±1.07	6.06±0.94	13.88±1.55	12.82±0.75	15.66±1.11
Schwerin_1	5.28±1.21	4.89±1.81	5.56±0.0	4.67±2.04	4.67±2.04	5.56±0.0	0.39±1.42	**0.0±0.0**	5.67±0.78
Schwerin_2	4.76±1.36	3.34±2.44	4.9±1.27	2.7±2.35	2.7±2.35	4.9±1.27	0.38±1.29	**0.33±1.22**	4.94±1.35
Waescher_CSP	5.45±2.57	**5.45±2.57**	**5.45±2.57**	**5.45±2.57**	**5.45±2.57**	**5.45±2.57**	7.86±8.55	6.86±8.5	13.07±7.33

TABLE III
COMPUTATIONAL TIME (S) OF ALGORITHMS

	RG	FG	BGCN	BGCNMC	PTR	FFD	HRL-GPN	RRMCTS	RRH
AI	35.1501	34.4809	42.4309	59.5254	2.1487	0.0311	1.3268	32.4969	2.9284
ANI	31.4776	34.4773	42.8409	59.7085	2.1055	0.031	1.3457	39.6095	3.1057
Falkenauer_T	0.1237	0.1228	1.1603	1.3847	0.5782	0.0063	0.3997	13.786	0.258
Falkenauer_U	0.0694	0.0681	0.4548	0.4896	1.8181	0.0255	1.0899	27.1575	0.092
Hard28_CSP	0.714	0.9163	1.8903	1.9793	0.349	0.0032	0.266	9.3476	0.2698
Irnich_CSP	2.2135	2.2004	18.2052	42.0544	18.9526	0.8084	43.0978	303.5974	4.7123
Randomly_Generated_CSP	0.1691	0.1937	1.877	2.697	1.2071	0.0217	0.8773	26.3826	0.2343
Scholl_1	0.0208	0.0204	0.3006	0.3203	0.5906	0.0076	0.3396	12.171	0.0427
Scholl_2	3.2333	3.5949	3.8893	4.0098	0.4804	0.0037	0.3352	8.105	1.0805
Scholl_3	7.2521	8.2757	7.0057	7.1903	0.3737	0.003	0.2651	6.6472	4.0065
Schwerin_1	0.2307	0.3349	0.2378	0.238	0.1511	0.0009	0.1344	3.9798	0.1823
Schwerin_2	0.1777	0.2752	0.2391	0.2415	0.1829	0.001	0.1659	4.6645	0.2203
Waescher_CSP	0.4801	0.4674	0.4479	0.4564	0.2145	0.0011	0.1688	4.6151	0.229

time (second) is reported in Table III, where the mean computational time is rounded to 4 decimal places. We mark the best result for each instance set in Table II. For a given instance, the rule for comparing the results of two algorithms is as follows. If the mean gap of algorithm *a* is less than that of algorithm *b*, we consider algorithm *a* is better than algorithm *b*. If the mean gap of algorithm *a* is the same algorithm *b*, then the std gap is checked. If the std gap of algorithm *a* is less than that of algorithm *b*, we consider algorithm *a* is better than algorithm *b*.

Since the results in Table II are rounded, small differences between the results cannot be observed directly. To identify which algorithm is better, we compare the results of algorithms and report their comparison in Table IV. The effectiveness of our proposed algorithms and the baseline algorithms were compared, and the results are summarized in Table IV. The rows and columns of Table IV are indexed using the compared algorithms. For algorithm *a* in a row and algorithm *b* in a column, the value *f* in cell (algorithm *a*, algorithm *b*) indicates the number of instance sets where algorithm *a* outperforms algorithm *b* regarding the mean gap and std.

1) Performance of Our MDP: To evaluate our MDP, we compare three greedy algorithms based on MDPs with and without combination features. For the MDP with combination features, two greedy algorithms are performed. As introduced in Section IV-B, the first one is RG, which selects the action with the minimum trim loss (i.e., RG) at each stage. The second one is FG, which selects the action with minimum fractional value in RMP. FFD is a greedy algorithm based on the MDP without combination features. It selects the largest item that can be packed into the current stock. The comparison resluts of these algorithms are reported in Fig. 6. In the figure, each rectangle is split into two parts: the left part with the black diagonals represents the proportion of instances on which the left algorithm outperforms the right one, while the right part with the grey grid represents the proportion of instances on which the right algorithm outperforms the left one. The

TABLE IV
COMPARE OF ALGORITHMS RESULT ON DATASETS

alg 1 \ alg 2	RG	FG	FFD	BGCN	BGCNMC	PTR	HRL-GPN	RRMCTS	RRH
RG	-	4	**8**	4	2	**10**	**11**	**11**	**13**
FG	**7**	-	**8**	3	3	**9**	**11**	**11**	**13**
FFD	1	3	-	3	1	**7**	**11**	**11**	**13**
BGCN	**7**	**5**	**8**	-	3	**9**	**11**	**11**	**13**
BGCNMC	**6**	**8**	**9**	**5**	-	**10**	**11**	**11**	**13**
PTR	1	0	1	0	1	-	**11**	**11**	**13**
HRL-GPN	2	2	2	2	2	2	-	4	**10**
RRMCTS	2	2	2	2	2	2	**9**	-	**10**
RRH	0	0	0	0	0	0	3	3	-

[1] Full name of alg 1 is algorithm 1, which denotes the row index. Similarly, the full name of the alg 2 is algorithm 2, which denotes the column index.

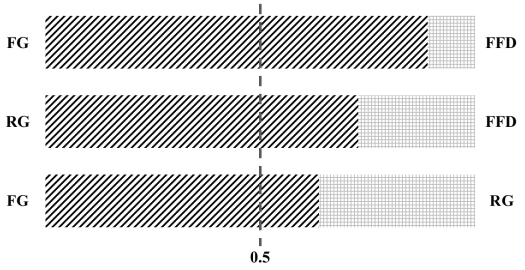

Fig. 6. Proportio of the preponderant instance number for greedy algorithms.

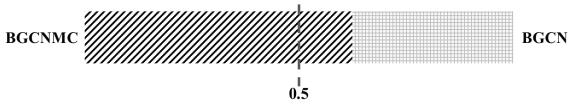

Fig. 7. Proportio of the preponderant instance number for our algorithms.

proportion is calculated using (30).

$$p(\text{alg}_a, \text{alg}_b) = \frac{f(\text{alg}_a, \text{alg}_b)}{f(\text{alg}_a, \text{alg}_b) + f(\text{alg}_b, \text{alg}_a)} \tag{30}$$

In (30), $f(\text{alg}_a, \text{alg}_b)$ is the number of instance sets where algorithm a outperforms algorithm b regarding the mean gap and std, which has been reported in Table IV. $p(\text{alg}_a, \text{alg}_b)$ denotes the proportion of instance sets where algorithm a outperforms algorithm b. Accordingly, we can obtain that algorithm a outperforms algorithm b if $p(\text{alg}_a, \text{alg}_b) > 0.5$. From Fig. 6, it is obvious that FG and RG both outperform FFD. In all three algorithms, the actions are selected based on the greedy policy. Their main difference is their MDPs. FG and RG employ our MDP, i.e., the MDP with combination feature, while FFD employs the MDP without combination feature. Hence, we can conclude that our MDP with combination feature is more effective than the MDP without combination feature. In addition, we can also find that FG outperforms RG. In FG, the greedy policy is performed according to the fraction value of variables, which are provided by the RMP of column generation. In RG, the greedy policy is performed according to the trim loss of the stock. Hence, we can obtain another conclusion that column generation can provide better features compared to that provided by the problem directly.

2) Performance of Monotonicity Cut: To show the effectiveness of the designed monotonicity cut, we compare the performnace of BGCNMC and BGCN. Similar with Fig. 6, Fig. 7 shows the proportion of BGCNMC and BGCN. From the figure, we can observe that BGCNMC outperforms BGCN, which implies that our monotonicity cut is effective in improving the objective value. The monotonicity cut is designed according to the monotonicity of the optimal action set, which can remove the redundant actions. Due to the removal of redundant actions, the remaining actions are of higher quality, resulting in a good solution.

3) Performance of Our Agent: At each stage, we generate new actions and add them to the action set, from which the RL agent then selects one action to be implemented. In the MDP, only the actions generated in previous stages are available for selection. This means that the agent cannot select actions that will be generated in subsequent stages. The solution is formed by the action sequence along with the trajectory of MDP. In this subsection, we collect all generated actions in the action set from the start stage to the terminal stage, and solve the MP with the collected actions. Then, the resulting solution is used to evaluate the ability of agent to select high-quality actions.

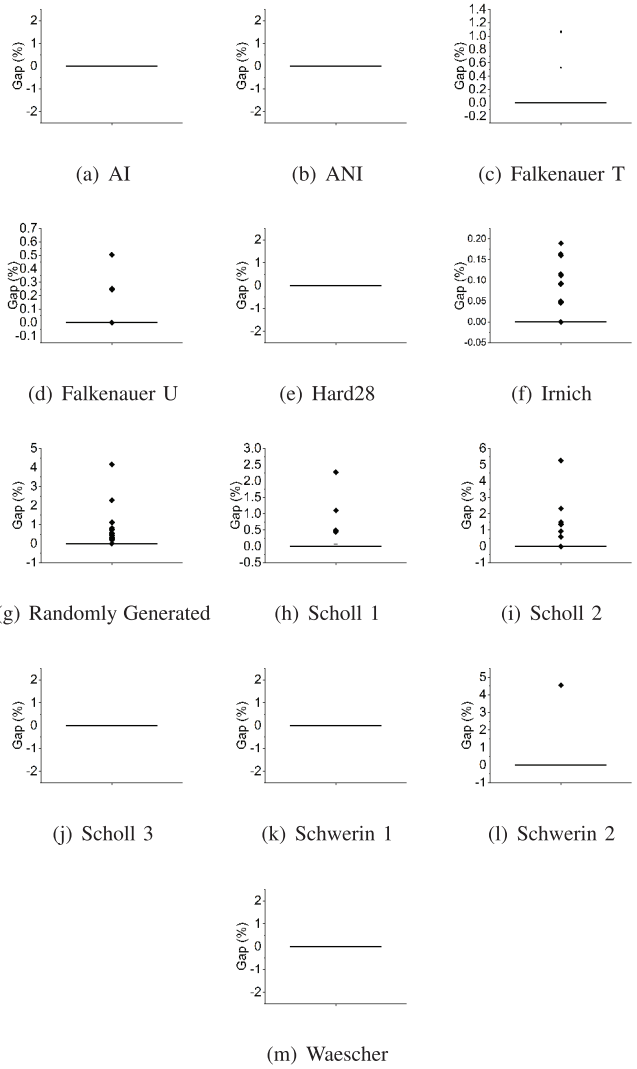

Fig. 8. Gaps of BGCNMC compared with the lower bound.

First, we update the action set and select an action to implement at each stage. Once the decision process is completed, we obtain a complete action set $\mathcal{A}\tau$. Then, the RMP with the actions set $\mathcal{A}_\tau$ is obtained and solved by an integer optimization solver. The resulting solution can be regarded as a lower bound for our approach and used to evaluate the performance of our method, i.e., whether the agent can select the best actions at each stage. We calculate the gaps between the solutions obtained by BGCNMC and the lower bounds using equation (31). Finally, we present the gaps using a box plot in Figure 8.

$$\text{gap} = \frac{\text{result of BGCNMC} - \text{lower bound}}{\text{lower bound}} \times 100\% \tag{31}$$

Fig. 8 shows the gaps between the BGCNMC and the lower bound for different datasets. For datasets such as AI, ANI,

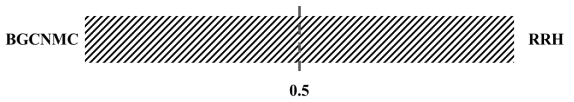

Fig. 9. Proportio of the preponderant instance number for our's and column generation based algorithms.

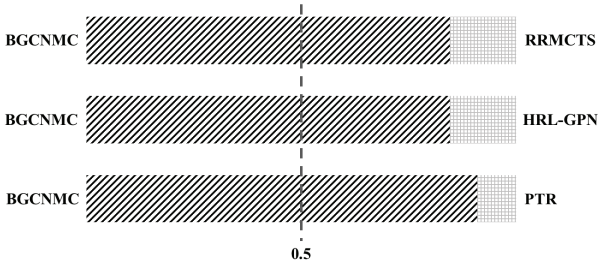

Fig. 10. Proportio of the preponderant instance number for RL algorithms.

Hard28, School 3, Schwerin 1, and Waescher, the median, first quartile, third quartile, minimum, and maximum gaps are all equal to 0, and there are no outliers. This suggests that the best actions are selected at each stage. For datasets such as Randomly Generated, Scholl 1, Scholl 2, and Schwerin 2, the median, first quartile, third quartile, minimum, and maximum gaps are all equal to 0, but there are outliers. On these datasets, the best actions are selected at almost all stages. For Falkenauer T, Falkenauer U, and Irnich datasets, the first quartile, third quartile, minimum, and maximum gaps equal 0, and the median gap is approximately 0, but there are outliers. On these three datasets, the best actions are selected at most stages. Overall, our method effectively selects the best actions from the given action set.

4) Comparison With the Approach Based on Column Generation: In our approach, the actions are obtained by column generation. The agent selects an action from the action set at each stage. In this subsection, we compare our BGCNMC with RRH, which is a heuristics based on column generation. The comparison result is shown in Fig. 9. It can be found that our approach outperforms RRH on all instance sets. In RRH, the solution is obtained by rounding the decision variable of RMP, removing columns with large trim loss, and repairing scheme. RRH employs a specific rule to obtain the solution based on the column generation. In our approach, the actions are selected by the agent trained using RL, and all selected actions form the solution to the problem. RL is an effective approach where agent learns to make the right decisions and refine its decision-making process by receiving feedback in the form of rewards. Hence, it is predictable that the learning approach is better than a heuristic rule.

5) Performance of Our Approach: Fig. 10 shows the comparison results between our approach BGCNMC and three state-of-the-art RL approaches, i.e., RRMCTS, HRL-GPN and PTR. The effectiveness of the MDP, monotonicity cut, and agent can be shown in the above subsections. Hence, it is reasonable that our approach BGCNMC significantly outperforms the state-of-the-art RL approaches.

V. Conclusion

This paper has presented an improved reinforcement learning approach using column generation. We constructed a high-quality action space by generating a subset of elements as actions. To further improve the quality of the action space, we designed a monotonicity cut based on monotone comparative statics as an optional strategy. To capture the feature structure of the proposed MDP, we introduced a bipartite graph and a related bipartite graph convolutional network architecture for the actor and critic networks. Our experimental results demonstrated that these components are effective, as our approach produces a better quality action space than existing approaches, the monotonicity cut improves the performance of BGCN, and our agent can select the best action at each stage.

The evaluation of our proposed approach is on the classical grouping problem of the cutting stock problem in this paper. However, the approach can be applied to both grouping and sorting problems. Although sorting problems require additional consideration of the sequence of elements in subsets, the approach has the potential to improve the agent's performance. This will be a focus of future work.

References

[1] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*. Boca Raton, FL, USA: CRC Press, 2017.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[3] W. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality: Second Edition*. Hoboken, NJ, USA: Wiley, 2011.

[4] D. Bertsekas, *Dynamic Programming and Optimal Control*, vol. 1. Belmont, MA, USA: Athena Scientific, 2012.

[5] D. P. Bertsekas, *Neuro-dynamic ProgrammingNeuro-Dynamic Programming*. Boston, MA, USA: Springer, 2001, pp. 1687–1692, doi: 10.1007/0-306-48332-7_333. https://doi.org/10.1007/0-306-48332-7333

[6] D. Zhang and D. Adelman, "An approximate dynamic programming approach to network revenue management with customer choice," *Transp. Sci.*, vol. 43, no. 3, pp. 381–394, Aug. 2009.

[7] D. J. Lizotte, M. Bowling, and S. A. Murphy, "Efficient reinforcement learning with multiple reward functions for randomized controlled trial analysis," in *Proc. ICML*, Jun. 2010, pp. 695–702. [Online]. Available: https://icml.cc/Conferences/2010/papers/464.pdf

[8] W. Powell, A. George, B. Bouzaiene-Ayari, and H. Simao, "Approximate dynamic programming for high dimensional resource allocation problems," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, vol. 5, 2005, pp. 2989–2994.

[9] J. A. Boyan, "Least-squares temporal difference learning," in *Proc. ICML*, Jun. 1999, pp. 49–56.

[10] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *J. Mach. Learn. Res.*, vol. 4, pp. 1107–1149, Dec. 2003.

[11] V. Mnih et al., "Playing Atari with deep reinforcement learning," in *Proc. NIPS Deep Learn. Workshop*, Jan. 2013, pp. 1–23.

[12] V. Mnih, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[13] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, New York, NY, USA, M. F. Balcan and K. Q. Weinberger, Eds., Jun. 2016, pp. 1995–2003. [Online]. Available: https://proceedings.mlr.press/v48/wangf16.html

[14] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. ICLR (Poster)*, Nov. 2015, pp. 1–23.

[15] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. SMC-13, no. 5, pp. 834–846, Sep./Oct. 1983.

[16] V. Mnih et al., "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[17] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. 32nd Int. Conf. Int. Conf. Mach. Learn.*, vol. 37, Jul. 2015, pp. 1889–1897.
[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
[19] T. Lillicrap et al., "Continuous control with deep reinforcement learning," in *Proc. ICLR (Poster)*, Jul. 2016, pp. 1–32.
[20] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," in *Proc. 34th Int. Conf. Mach. Learn.*, Aug. 2017, pp. 1352–1361.
[21] J. Ren, S. Guo, and F. Chen, "Orientation-preserving Rewards' balancing in reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 11, pp. 6458–6472, Nov. 2022.
[22] Z. Cao, K. Wong, and C.-T. Lin, "Weak human preference supervision for deep reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 12, pp. 5369–5378, Dec. 2021.
[23] Z. Huang, J. Wu, and C. Lv, "Efficient deep reinforcement learning with imitative expert priors for autonomous driving," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 10, pp. 7391–7403, Oct. 2023.
[24] E. S. Kokten and Ç. Sel, "A cutting stock problem in the wood products industry: A two-stage solution approach," *Int. Trans. Oper. Res.*, vol. 29, no. 2, pp. 879–907, Mar. 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/itor.12802
[25] İ. Ülkü, U. Tekeoğlu, M. Özler, N. Erdal, and Y. Tolonay, "Solving the cutting stock problem for a steel industry," in *Proc. Digitizing Prod. Syst.*, N. M. Durakbasa and M. G. Gençyılmaz, Eds., Cham, Switzerland: Springer, 2022, pp. 510–518.
[26] J. Kallrath, S. Rebennack, J. Kallrath, and R. Kusche, "Solving real-world cutting stock-problems in the paper industry: Mathematical approaches, experience and challenges," *Eur. J. Oper. Res.*, vol. 238, no. 1, pp. 374–389, Oct. 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221714002562
[27] F. Parreño, M. T. Alonso, and R. Alvarez-Valdes, "Solving a large cutting problem in the glass manufacturing industry," *Eur. J. Oper. Res.*, vol. 287, no. 1, pp. 378–388, Nov. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221720304446
[28] A. R. Pitombeira-Neto and A. H. F. Murta, "A reinforcement learning approach to the stochastic cutting stock problem," *EURO J. Comput. Optim.*, vol. 10, Jan. 2022, Art. no. 100027. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S219244062200003X
[29] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., Dec. 2015, pp. 2692–2700. [Online]. Available: https://proceedings.neurips.cc/paper/2015/file/29921001f2f04bd3baee84a12e98098f-Paper.pdf
[30] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," in *Workshop Proc. 5th Int. Conf. Learn. Represent.*, Jan. 2016, pp. 1–17.
[31] H. Hu, X. Zhang, X. Yan, L. Wang, and Y. Xu, "Solving a new 3D bin packing problem with deep reinforcement learning method," in *Proc. Workshop AI Appl. E-Commerce Co-Located 16th Int. Joint Conf. Artif. Intell.*, Jan. 2017, pp. 1–7.
[32] L. Duan et al., "A multi-task selected learning approach for solving 3D flexible bin packing problem," in *Proc. 18th Int. Conf. Auto. Agents MultiAgent Syst.*, Jan. 2018, pp. 1386–1394.
[33] A. Laterre et al., "Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization," in *Proc. Workshop Deep Reinforcement Learn. Co-Located 32nd Conf. Adv. Neural Inf. Process. Syst.*, Jan. 2018, pp. 1–11.
[34] M. Nazari, A. Oroojlooy, M. Takáč, and L. Snyder, "Reinforcement learning for solving the vehicle routing problem," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, vol. 31, Red Hook, NY, USA, Dec. 2018, pp. 9861–9871.
[35] Z. Wang, Y. Nakano, and K. Nishimatsu, "The vehicle routing problem with time windows and time costs," in *Proc. Int. Conf. Data Mining Workshops (ICDMW)*, Dec. 2021, pp. 278–287.
[36] Z. Zhang, H. Liu, M. Zhou, and J. Wang, "Solving dynamic traveling salesman problems with deep reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 4, pp. 2119–2132, Apr. 2023.
[37] Q. Ma, S. Ge, D. He, D. Thaker, and I. Drori, "Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning," in *Proc. AAAI Workshop Deep Learn. Graphs, Methodologies Appl.*, Jan. 2019, pp. 1–12.
[38] L. Duan et al., "Efficiently solving the practical vehicle routing problem: A novel joint learning approach," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*. New York, NY, USA, Aug. 2020, pp. 3054–3063, doi: 10.1145/3394486.3403356.
[39] L. Tang and Y. Meng, "Data analytics and optimization for smart industry," *Frontiers Eng. Manage.*, vol. 8, no. 2, pp. 157–171, Jun. 2021.
[40] L. Tang, Z. Li, and J.-K. Hao, "Solving the single-row facility layout problem by K-medoids memetic permutation group," *IEEE Trans. Evol. Comput.*, vol. 27, no. 2, pp. 251–265, Apr. 2023.
[41] H. Taha, *Operations Research an Introduction*. London, U.K.: Pearson, 2017. [Online]. Available: https://books.google.com/books?id=HbpKjwEACAAJ
[42] P. Milgrom and C. Shannon, "Monotone comparative statics," *Econometrica*, vol. 62, no. 1, p. 157, Jan. 1994. [Online]. Available: https://www.jstor.org/stable/2951479
[43] D. M. Topkis, *Supermodularity and Complementarity*. Princeton, NJ, USA: Princeton Univ. Press, 2011, doi: 10.1515/9781400822539.
[44] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., Dec. 2018, pp. 537–546. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/8d3bba7425e7c98c50f52ca1b52d3735-Paper.pdf
[45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, San Diega, CA, USA, Dec. 2014, pp. 1–23.
[46] X. Wang, Z. Dong, L. Tang, and Q. Zhang, "Multiobjective multitask optimization-neighborhood as a bridge for knowledge transfer," *IEEE Trans. Evol. Comput.*, vol. 27, no. 1, pp. 155–169, Feb. 2023.
[47] E. Falkenauer, "A hybrid grouping genetic algorithm for bin packing," *J. Heuristics*, vol. 2, no. 1, pp. 5–30, Jun. 1996, doi: 10.1007/bf00226291.
[48] A. Scholl, R. Klein, and C. Jürgens, "Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem," *Comput. Operations Res.*, vol. 24, no. 7, pp. 627–645, Jul. 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054896000822
[49] G. Wäscher and T. Gau, "Heuristics for the integer one-dimensional cutting stock problem: A computational study," *OR Spektrum*, vol. 18, no. 3, pp. 131–144, Sep. 1996, doi: 10.1007/bf01539705.
[50] P. Schwerin and G. Wäscher, "The bin-packing problem: A problem generator and some numerical experiments with FFD packing and MTP," *Int. Trans. Oper. Res.*, vol. 4, nos. 5–6, pp. 377–389, Sep. 1997. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1475-3995.1997.tb00093.x
[51] D. Beckedahl and N. Pillay, "Bi-space search: Optimizing the hybridization of search spaces in solving the one dimensional bin packing problem," in *Artificial Intelligence and Soft Computing: 21st International Conference, ICAISC 2022, Zakopane, Poland, June 19–23, 2022, Proceedings, Part II*. Berlin, Germany: Springer-Verlag, 2002, pp. 206–217, doi: 10.1007/978-3-031-23480-4_17.
[52] T. Gschwind and S. Irnich, "Dual inequalities for stabilized column generation revisited," *INFORMS J. Comput.*, vol. 28, no. 1, pp. 175–194, Feb. 2016.
[53] M. Delorme, M. Iori, and S. Martello, "Bin packing and cutting stock problems: Mathematical models and exact algorithms," *Eur. J. Oper. Res.*, vol. 255, no. 1, pp. 1–20, Nov. 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221716302491
[54] A. C. Cherri, M. N. Arenales, and H. H. Yanasse, "The one-dimensional cutting stock problem with usable leftover—A heuristic approach," *Eur. J. Oper. Res.*, vol. 196, no. 3, pp. 897–908, Aug. 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221708003998
[55] B. S. C. Campello, C. T. L. S. Ghidini, A. O. C. Ayres, and W. A. Oliveira, "A residual recombination heuristic for one-dimensional cutting stock problems," *Top*, vol. 30, no. 1, pp. 194–220, Apr. 2022, doi: 10.1007/s11750-021-00611-3.

Fengyuan Shi received the M.S. degree in system engineering from Northeastern University, Shenyang, China, in 2017, where he is currently pursuing the Ph.D. degree in system engineering.

His current research interests include reinforcement learning, machine learning, and combinatorial optimization.

Ying Meng received the Doctorate degree from Northeastern University, China, in 2011. She is currently an Associate Professor with the Centre for Artificial Intelligence and Data Science. She is also the Vice Director of the Key Laboratory of Data Analytics and Optimization for Smart Industry, Ministry of Education, Northeastern University. Her research interests cover evolutionary learning and reinforcement learning, integer optimization and combinatorial optimization, and computational intelligent optimization.

Jiyin Liu received the B.Eng. degree in industrial automation and the M.Eng. degree in systems engineering from Northeastern University, Shenyang, China, in 1982 and 1985, respectively, and the Ph.D. degree in manufacturing engineering and operations management from the University of Nottingham, Nottingham, U.K., in 1993. He is currently a Professor in Operations Management with the Business School, Loughborough University, Loughborough, U.K. Previously, he was with Northeastern University and The Hong Kong University of Science and Technology, Hong Kong. His research interests are in the areas of operations planning and scheduling in production and logistics, in modeling, analysis, and solution of practical operations problems. His research has been published in various academic journals, including *Operations Research*, *Manufacturing & Service Operations Management*, *IISE Transactions*, *Naval Research Logistics*, *European Journal of Operational Research*, IEEE TRANSACTIONS, *International Journal of Production Research*, and *Transportation Research*.

Lixin Tang (Fellow, IEEE) received the B.Eng. degree in industrial automation, the M.Eng. degree in systems engineering, and the Ph.D. degree in control science and engineering from Northeastern University, Shenyang, China, in 1988, 1991, and 1996, respectively.

He is a member of Chinese Academy of Engineering, the Director of Key Laboratory of Data Analytics and Optimization for Smart Industry, Ministry of Education, China, and the Director and a Chair Professor of the National Frontiers Science Center for Industrial Intelligence and Systems Optimization, Northeastern University. Meanwhile, he applies the above theories and technologies to engineering applications in steel manufacturing industry, equipment/chip manufacturing industry, energy industry, logistics industry, and information industry. He has published many articles in international journals, such as IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, IEEE TRANSACTIONS ON CYBERNETICS, IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY, IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, IEEE TRANSACTIONS ON POWER SYSTEMS, *Operations Research*, and *INFORMS Journal on Computing*. His research interests cover industrial intelligence and systems optimization theories and methods, covering data analytics and machine learning, deep learning and evolutionary learning, reinforcement learning and dynamic optimization, convex and sparse optimization, integer and combinatorial optimization, and computational intelligence-based optimization.

Dr. Tang article published on *IISE Transactions* received the Best Applications Paper Award of 2017. He currently serves as an Associate Editor for IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, IEEE TRANSACTIONS ON CYBERNETICS, *IISE Transactions*, *Journal of Scheduling*, and *International Journal of Production Research*. Meanwhile, he is on the editorial board of *Annals of Operations Research*, and serves as an Area Editor for the *Asia-Pacific Journal of Operational Research*.