Tree Search Reinforcement Learning for Two-Dimensional Cutting Stock Problem With Complex Constraints

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

***Abstract*— Reinforcement learning (RL) has been widely used in recent years to solve combinatorial optimization problems; however, it has some limitations when solving such problems with practical features. It is difficult for RL to ensure the feasibility of solutions when solving optimization problems with complex constraints, which limits its industrial application. Using a two-dimensional cutting stock problem derived from the practical plate design process in the steel industry as an example, we propose a learning and searching framework that enables RL to obtain a feasible solution that obeys complex constraints. We first formulate the two-dimensional cutting stock problem as a Markov decision process (MDP) and then design a tree-search-based reinforcement learning (TSRL) algorithm within the proposed learning and searching framework. In the learning process, we establish the approximate stage-independent Bellman equation (ASIB) of the MDP and obtain the agent decision model by solving the ASIB with approximate linear programming and column generation. In the search process, based on the obtained approximate value function, an efficient parallel two-stage tree search is developed as the agent decision generator of RL to obtain near-optimal plate design schemes. The experimental results show that the proposed TSRL algorithm outperforms baselines in terms of the capacity to obtain a feasible solution, computational time, and solution quality.**

***Note to Practitioners*—This article was motivated by the slab design problem in the steel industry but it also applied to other two-dimensional cutting stock problems with complex constraints. This work promotes making decisions in industrial production. Our key contribution is to provide a good-enough solution in a short time, which can improve the resource utilization and reduce the production cost of industrial enterprises. In terms of methodology, we provide new insight into reinforcement learning, which makes reinforcement learning more applicable to practice scenarios.**

Manuscript received 29 March 2024; accepted 24 July 2024. Date of publication 16 September 2024; date of current version 14 March 2025. This article was recommended for publication by Associate Editor F. Ju and Editor J. Li 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 is 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).

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

Digital Object Identifier 10.1109/TASE.2024.3456074

***Index Terms*— Reinforcement learning, complex constraints, two-dimensional cutting stock problem, parallel tree search, supermodularity.**

I. Introduction

THE cutting stock problem (CSP) is a well-known combinatorial optimization problem that was first described in reference [1]. It targets the filling of orders at minimum cost by cutting a specified amount of material from a given stock length. CSP widely exists in the manufacturing industries, such as the cutting of steel rolls, metal pipes, and cellophane rolls. CSP is a linear integer optimization problem. Several linear integer programming approaches, such as column generation as described in references [2] and [3] and the branch-and-bound approach with cutting planes as described in reference [4], are presented for the classical CSP.

In a practical scenario, some additional features must be considered, such as flexible stock size [5], minimum and maximum length constraints,[1] and irregular deformation of the stock [6]. Such practical features significantly increase the complexity of CSPs. This study investigates a two-dimensional CSP with complex constraints that is abstracted from a practical production scenario which is common in steel enterprises. This problem can be seen as a variation of the two-dimensional CSP. Some practical features have been considered in this problem, including flexible and multiple stock sizes, minimum and maximum length requirements, and irregular deformation. Such practical features distinguish our problem from the ones in literature and significantly increase the complexity of the problem. Although some effective metaheuristic methods have been proposed to solve the optimization problems with various complex constraints, the strategies for dealing with constraints in literature are customized case by case. Hence, the existing metaheuristic methods cannot be directly adopted to solve our problem.

Generally, reinforcement learning (RL) is an effective method to tackle the online optimization problem by learning from interactions with an environment [7], [8]. Due to its ability to learn knowledge from experience, RL can also be adopted as an effective solution method for solving

[1]Standard document of a steel enterprise, the slab has a maximum and minimum length: https://www.huaro-shanghai.com/document-download/

the offline combinatorial optimization problem. Many offline combinatorial optimization problems can be formulated as a Markov Decision Process (MDP) and then optimally solved by dynamic programming. When dealing with some complex problems of large-scale, dynamic programming usually suffers from the limitation of the "curse of dimensionality". In this case, to reduce the decision space, it is a good choice to replace the value or policy with a parameterized function, where the parameters can be obtained by RL. The iteration process of RL can be seen as the process of learning knowledge and utilizing knowledge. In the process of learning knowledge, an agent decision model is obtained to describe the functional relationship between reward, state, and action. In the process of utilizing knowledge, an agent decision generator is used to select actions based on the obtained agent decision model. In this context, RL can solve a variety of complex combinatorial optimization problems in a short time using agents; therefore, it has been rapidly developed in recent years [9], [10], [11], [12], [13], [14], [15].

Several algorithms have been developed to train an agent decision model to optimize the cumulative reward. One class of RL is the value-based approach, which is used to obtain an approximate value function. Some typical value-based approaches include Q-learning [16] and Dueling DQN [17], among others. Another class of RL is policy-based approaches, such as the policy gradient [18], which is used to obtain an approximate policy function. Several combinations of policy-based and value-based approaches have been developed, such as the actor-critic approach. Typical actor-critic approaches include A3C [19], DDPG [20], and PPO [21].

When solving a combinatorial optimization problem, the RL should be modified according to the structure of the problem. Regarding the bin packing problem and the knapsack problem, which are similar to the CSP, the commonly used methods are based on pointer networks. Reference [22] solves a CSP using a policy iteration approach. The agent decision model is an approximate value function, and the agent decision generator is the greedy method. Reference [23] solves a 3D-bin packing problem by deep RL, where the packing sequence of items is determined by a pointer network. A heuristic is designed as an action generator to determine the placement of items in the bin. In reference [24], a knapsack problem is solved by the pointer network-based RL, where the pointer network is trained as the agent decision model, and a designed decision generator called active search is used to obtain solutions. Reference [25] also adopts pointer network-based RL to solve the 3D-bin packing problem. The proposed algorithm adopts the pointer network as the agent decision model, and the agent decision generator combines a multitask method and the hill-climbing algorithm. The multitask method generates an initial solution, whereas the hill-climbing algorithm attempts to find a better solution. In addition to pointer network-based methods, reference [26] solves the bin packing problem via a Monte Carlo tree search (MCTS), using a neural network as the agent decision model and an MCTS as the agent decision generator. Reference [27] reviews papers that focused on RL for combinatorial optimization problems.

According to reference [28], finding feasible solutions to combinatorial optimization problems is not an easy task for learning methods. While solving combinatorial optimization problems by RL, the problems are usually formulated as a Markov decision process (MDP), where the decision space is split into several sub-spaces along with the stage. At each stage, an action is selected from the corresponding action space. Then, actions selected at all stages correspond to a solution for the original optimization problem. To obtain the feasible solutions, constraints should be tackled when selecting actions. According to reference [29], there are two kinds of constraints in the MDP, i.e., cumulative constraints and instantaneous constraints. The instantaneous constraint should be satisfied at each stage, while the cumulative constraint is used to restrict the whole action sequence.

The problem investigated in this study incorporates both cumulative and instantaneous constraints. In contrast to reference [29], the cumulative constraints in our problem are a set of "equality" constraints, that is, all order demands must be satisfied exactly. The instantaneous constraints include a set of "less than" constraints and a set of "greater than" constraints. Such instantaneous constraints apply the restriction that the total length of the items cut from the same stock should not be longer than the maximum length and not shorter than the minimum length. When making decisions, it is possible that the total length of all remaining items cannot satisfy the minimum length constraint; thus, the demands of these items cannot be satisfied. In this context, a conflict between the "equality" cumulative constraints and the "greater than" instantaneous constraints may occur, leading to the main challenge of solving the problem using RL.

Penalty and mask are the most commonly used methods for tackling constraints in RL. In mask-based methods, it is important to verify the feasibility of all actions for masking infeasible solutions. However, the satisfaction of cumulative constraints cannot be verified until the terminal state is reached. Therefore, the existing mask-based methods are inefficient when solving our problem. In penalty-based methods, the violation of cumulative constraints is avoided by adding a penalty to the reward. However, it is challenging to design a suitable penalty item. The existing penalty-based methods can only reduce violations but cannot completely avoid them.

To the best of our knowledge, no relevant studies have been reported on how to tackle the conflict between the cumulative and instantaneous constraints described above when solving CSP using RL. In view of the instantaneous constraints, the most related problems are the vehicle routing problem with time windows and the traveling salesman problem with time windows, where the time window constraints are similar to the instantaneous constraints, expressed in the forms of "greater than" and "less than". Reference [30] mentions that time windows can be enforced through a masking scheme. However, designing such a scheme might be a challenging task, possibly more difficult than solving the optimization problem itself. In reference [31], a masking scheme is proposed to address the

time-window constraints in the vehicle routing problem with time windows. In the problem, if a vehicle arrives after the time window closes, i.e., the "less than" constraint is violated, then the vehicle will be masked. However, a new route can be generated for a masked vehicle. On the new route, if the vehicle arrives before the time window opens, the time window constraints can be satisfied by allowing the vehicle to wait. However, such a masking strategy will not work for our problem, which is similar to the traveling salesman problem with time windows. According to reference [32], training an action mask may be helpful in addressing cumulative constraints. However, only an 80% success rate was achieved using the trained action mask. Reference [33] proposes hierarchical reinforcement learning with a graph pointer network and a penalty scheme (HRL-GPN) to address the time windows of the traveling salesman problem with time windows.

In this paper, we investigate a two-dimensional cutting stock problem with complex constraints arising from the actual plate design process in steel enterprises. The problem considers practical features such as flexible and multiple stock sizes, minimum length constraints, and irregular deformation. Specifically, the conflict between the cumulative and instantaneous constraints poses the main challenge in solving the problem. In this context, we propose a novel framework for RL, which can effectively handle the complex constraints involved in combinatorial optimization problems. Under this framework, we design an effective RL algorithm to obtain near-optimal solutions for our problem. Such a framework and the proposed tree search reinforcement learning algorithm can be generalized to other combinatorial optimization problems. The main contributions are as follows.

- A learning and searching framework is proposed for RL. In the learning process, an approximate value function is employed to serve as the agent decision model. In the searching process, a state-action tree is constructed based on the obtained approximate value function, and then a tree search approach is designed to search for the feasible near-optimal solutions on the state-action tree. Under the new framework, a tree-search-based reinforcement learning (TSRL) algorithm is proposed to solve the practical two-dimensional cutting stock problem with complex constraints.
- In the learning process, an approximate stage-independent Bellman equation, approximate linear programming, and column generation techniques are developed to obtain the agent decision model.
- In the searching process, a parallel two-stage tree search method is designed as a decision generator, which includes a split ratio to parallelize the tree search process and a monotonicity cut to reduce the search space.

The remainder of this paper is organized as follows. In Section II, we introduce the notation and the Markov design process (MDP) model for the plate design problem. In Section III, we detail the learning and searching framework and the proposed TSRL. In Section IV, we summarize the computational experiments. In Section V, we conclude the paper with a summary of our findings and contributions.

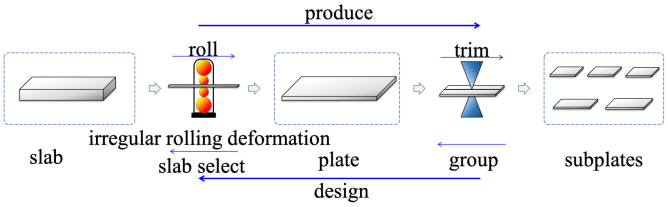


Fig. 1. In the steel plate production process, molten steel is refined in a converter and then poured into a continuous caster. The metal strand that emerges from the continuous caster is cut across its length to form slabs. These slabs are subsequently rolled into thinner mother plates through a series of horizontal and vertical rolling processes. Finally, the mother plates are cut across their widths and lengths to obtain subplates that satisfy customer requirements. The design process is the reverse of the production process.

II. Problem Statement and Markov Decision Process Formulation

In this section, we first introduce the practical scenario of the two-dimensional cutting stock problem, that is, the plate design problem mentioned in Section I. Next, a mathematical formulation is provided to describe the problem. Finally, we reformulate the problem as an MDP and provide the related stage-independent Bellman equation. In Appendix A, Table II summarizes the notations used in our problem statement and the solution method for ease of description.

A. Plate Design Problem Statement

Fig. 1 illustrates the steel plate production process. The design process of the steel plates can be regarded as the reverse of the production process. In the plate design process, we are given an order set N, a candidate plate set $T = \{0, 1, 2, \cdots, \tau - 1\}$, and a set of slab size M. The subplates required by orders, plates, and slabs are all rectangular. Each order $i \in N$ is specified by the length l_i, width w_i, thickness, and demand $s_{0,i} \in \mathbb{N}^+$ of the required subplate. Each slab size $m \in M$ is specified by its width W_m and thickness. There are two sets of decisions to be made. The first is to group the required subplates to form batches, each corresponding to a plate. The second is to choose one size $m \in M$ for each slab used to produce the plate. The objective is to minimize the total trim loss while satisfying all order requirements. When designing plates, only orders with the same thickness requirement need to be considered together since subplates cut from the same plate must have the same thickness.

Fig. 2 illustrates the trim loss of the plate. Owing to the deformation during the rolling process, the mother plate (referred to henceforth as the "plate") is usually not a regular rectangle. Therefore, the edges of the plate should be trimmed. Then, a target plate with a standard rectangular shape is obtained. Finally, the target plate is cut to satisfy the customer's requirements. Based on the widths and lengths of the plate and the corresponding target plate, we can calculate the trim loss of the plate. For a target plate, its width is equal to the largest width of the subplates placed on it, while its length is equal to the total length of the subplates placed on it. The width (or length) of a plate is equal to the sum of the width (or length) of the target plate and the trim width (or length) caused by deformation. In practice, the trim width (or

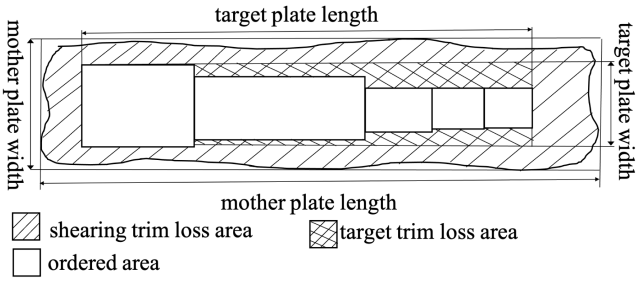


Fig. 2. Rolling leads to irregular deformation [6]. The inner rectangle is the target plate, whereas the outer rectangle is the mother plate, that is, the raw plate obtained by rolling. The width of the target plate is equal to the maximum width of the subplates placed on it, while its length is equal to the summation of the lengths of the subplates placed on it. The width (or length) of the mother plate is equal to the sum of the width (or length) of the corresponding target plate and the deformation width (or length). In this context, the trim loss of a plate is the sum of the trim area of the target plate and the trim area caused by deformation; it can also be calculated by subtracting the total area of the sub-plates placed on the mother plate from the area of the mother plate itself.

length) caused by deformation is related to the width extension ratio, which is calculated by dividing the plate width by the slab width.

We denote the decision variable $a_{t,m,i}$ to be the number of subplates required by order i that is cut from plate t with slab size m, and we use the auxiliary variable $\mathcal{I}_{t,m,i} \in \{0,1\}$ to indicate whether the subplate required by order i is the widest subplate placed on plate t with slab size m. Let $\beta_{m,i}$ denote the width extension ration, which is calculated by (1).

$$\beta_{m,i} = w_i / W_m \tag{1}$$

Then, the trim width $D^s(\mathcal{I}_t)$ and trim length $D^b(\mathcal{I}_t)$ caused by deformation can be obtained by (2) and (3), respectively.

$$D^s(\mathcal{I}_t) = \alpha^s \sum_{i\in N, m\in M} \beta_{m,i}\mathcal{I}_{t,m,i} + \delta^s \tag{2}$$

$$D^b(\mathcal{I}_t) = \alpha^b \sum_{i\in N, m\in M} \beta_{m,i}\mathcal{I}_{t,m,i} + \delta^b \tag{3}$$

where α^b and α^s are the given weights, δ^b and δ^s are the given biases. According to Fig.2, the length of a plate is equal to sum of the length of the target plate and the trim length. Then, the length of a plate can be calculated by (4).

$$\mathcal{L}(a_t, \mathcal{I}_t) = \sum_{i\in N} l_i \sum_{m\in M} a_{t,m,i} + D^b(\mathcal{I}_t) \tag{4}$$

where $\sum_{i\in N} l_i \sum_{m\in M} a_{t,m,i}$ is the length of target plate, i.e., total length of the subplates placed on plate t. Similarly, the width of plate t can be obtained by (5).

$$\mathcal{W}(\mathcal{I}_t) = \sum_{i\in N} w_i \sum_{m\in M} \mathcal{I}_{t,m,i} + D^s(\mathcal{I}_t) \tag{5}$$

where $\sum_{i\in N} w_i \sum_{m\in M} \mathcal{I}_{t,m,i}$ is the maximum width of the subplates placed on the plate. Hence, the trim loss of plate t with design scenario a_t can be calculated by Equation (6).

$$C(a_t) = \mathcal{L}(a_t, \mathcal{I}_t) * \mathcal{W}(\mathcal{I}_t) - \mathcal{Y}(a_t) \tag{6}$$

$$\mathcal{Y}(a_t) = \sum_{i\in N} l_i w_i \sum_{m\in M} a_{t,m,i} \tag{7}$$

where $\mathcal{Y}(a_t)$ is the total area of subplates placed on plate t, and the design scenario a_t is defined by (8)

$$a_t = [a_{t,m,i} | a_{t,m,i} \in \mathbb{N}^+ \cup \{0\}, i \in N, m \in M] \tag{8}$$

Three sets of hard constraints are involved in the plate design process: 1) the order demand constraints, 2) the maximum length constraints, and 3) the minimum length constraints. Moreover, a set of additional constraints is used to define the auxiliary variable $\mathcal{I}_t$.

The order demand constraints guarantee that all order demands must be satisfied and are expressed by Equation (9).

$$\sum_{t\in T} \sum_{m\in M} a_{t,m,i} - s_{0,i} = 0, \quad \forall i \in N \tag{9}$$

The maximum length constraints restrict the length of each plate t to not be larger than the given maximum length $\bar{l}_m$ by Equation (10).

$$\mathcal{L}(a_t, \mathcal{I}_t) - \sum_{m\in M} \bar{l}_m \sum_{i\in N} \mathcal{I}_{t,m,i} \le 0 \tag{10}$$

The minimum length constraints restrict the length of each plate t to not be smaller than the given minimum length $\underline{l}_m$ by Equation (11).

$$\mathcal{L}(a_t, \mathcal{I}_t) - \sum_{m\in M} \underline{l}_m \sum_{i\in N} \mathcal{I}_{t,m,i} \ge 0 \tag{11}$$

The additional constraints are used to define the relationship between a_t and $\mathcal{I}_t$. Constraint (12) requires that only one slab size can be selected for each plate. Constraint (13) means that if a slab size is selected to produce plate t, there must exist a widest subplate placed on plate t with that slab size. (14) and (15) ensure that the widest subplate on a plate can only be the one required by the order with the smallest index.

$$\sum_{i\in N} \sum_{m\in M} \mathcal{I}_{t,m,i} - 1 \le 0 \tag{12}$$

$$\min(1, \sum_{i\in N} a_{t,m,i}) - \sum_{i\in N} \mathcal{I}_{t,m,i} = 0, m \in M \tag{13}$$

$$(1 - \sum_{k=1}^{i-1} \mathcal{I}_{t,m,k}) \min(1, \sum_{k=1}^{i} a_{t,m,k}) - \mathcal{I}_{t,m,i} = 0, \quad i = N\backslash\{0\}, m \in M \tag{14}$$

$$\mathcal{I}_{t,m,0} - \min\left(1, a_{t,m,0}\right) = 0, m \in M \tag{15}$$

B. Markov Decision Process Formulation

An MDP is defined as a quadruple $\langle S, A, f, C\rangle$, where S denotes the state set, A denotes the action set, f denotes the transition function to describe the state transition rule, and C denotes the cost for the selected action. Accordingly, we define the MDP for the practical problem as follows:

- Stage: The entire decision process is divided into several stages according to the plate, that is, the design plan for a plate is determined at each stage t. At the terminal stage, it is observed that all order demands are exactly satisfied. The domain of the stage is T which is described in II-A.
- State: The state set is defined as $S = \{s | s = [s_i | s_i \in \mathbb{N}^+, s_i \le s_{0,i}, i \in N]\}$, where s_i is the potential state for order i without considering the stage, that is $S_t = S$ for each stage t.

- Action: Action space A is defined as the set of all possible design scenarios that satisfy instantaneous constraints (10)-(15), that is, $A = \{a|\text{s.t.(10)-(15)}\}$, where a is defined in (8). At each stage t, the action a_t is also limited by state s_t. Hence, the action space at stage t is $A_{s_t} = \{a_t|a_t \in A, s_{t,i} - \sum_{m\in M} a_{t,m,i} \geq 0, i \in N\}$, where each action a_t represents the design scenario.
- Transition function: The state transition function is defined as (16), where $\mathbf{1} = \{1\}^{|M|}$ is an m-dimensional identity matrix vector.

$$s_{t+1} = f(s_t, a_t) = s_t - \mathbf{1}a_t \tag{16}$$

  The state transition function implies that the unfulfilled demand for each order i is updated by $s_{t+1,i} = s_{t,i} - \sum_{m\in M} a_{t,m,i}$.
- Cost: Cost is defined as the trim loss of the plate designed at stage t, which is expressed as $C(a_t)$ in Equation (6).

To obtain the sequence of actions, we should calculate the values of states. The MDP in this paper has a finite horizon. Hence, a stage-dependent Bellman equation (17) is adopted.

$$V_t(s_t) = \min_{a_t \in A_{s_t}} \{C(a_t) + V_{t+1}(f(s_t, a_t))\},$$
$$s_t \in S_t, a_t \in A_{s_t}, t \in T \tag{17}$$

The stage-dependent Bellman equation must be solved to obtain the state value $V_t(s_t)$. If the subscript t is removed, we get the stage-independent equations shown in Equation (18), which can also be solved to obtain the state value, i.e, $V(s_t)$. Then, the scale of the Bellman equation is decreased. Stage-dependent Bellman equation is equivalent to stage-independent equations, as shown in Proposition 1 in Appendix B.

$$V(s) = \min_{a\in A_s} \{C(a) + V(f(s, a))\}, s \in S, a \in A_s \tag{18}$$

In our MDP, $a \in A_s$ is the instantaneous constraint, whereas $s_{\tau,i} = 0$ and (16) are the cumulative constraints.

III. Solution Method

In this section, we first introduce the proposed learning and searching framework, which can be generalized to solve the combinatorial optimization problem with different complex constraints. Then, under the learning and searching framework, a TSRL algorithm for solving the two-dimensional cutting stock problem is presented.

A. Learning and Searching Framework

Based on the formulated MDP, the original optimization problem can be solved by determining the action sequence in MDP according to the state value. The exact state value can be obtained using dynamic programming. To overcome the curse of dimensionality in dynamic programming, the approximate Bellman equations (19) are considered by replacing the exact state value with the parameterized function $\hat{V}(s)$, namely, the approximate value function.

$$\text{(ASIB)}\hat{V}(s) = \min_{a\in A_s} \{C(a) + \hat{V}(f(s, a))\}, s \in S, a \in A_s \tag{19}$$

In RL, the approximate value function can be designed in different structures, such as linear and neural network, according to the problem structure. As described in Section I, learning knowledge and utilizing knowledge are the key issues when adopting RL to solve the combinatorial optimization problem.

Learning knowledge is to obtain the agent decision model, i.e., the approximate value function $\hat{V}(s)$, by solving the ASIB. The common approaches used to obtain $\hat{V}(s)$ include Q-Learning, State-Action-Reward-State-Action, mathematical optimization approaches, and so on.

In the process of utilizing knowledge, the agent decision generator selects action at each stage based on the obtained agent decision model. The commonly used agent decision generators are given in (20) and (21).

$$a = \arg\min_{a\in A_s}(C(s, a) + \hat{V}(s')) \tag{20}$$

$$a = \pi(s_t) \tag{21}$$

As discussed in Section I, when adopting RL to solve the problem with constraints, penalties and masks are usually adopted to deal with the constraints. According to the description above, the constraints are considered in the agent decision model when using a penalty-based method, and in the agent decision generator when using a mask-based method. However, the existing penalty-based methods and mask-based methods cannot be used directly to deal with the constraints of our problem.

In this paper, we design a learning and searching framework for RL. Since the transition between states in our MDP is known, a masking scheme can be designed to consider cumulative constraints by checking their satisfaction when the terminal state is observed. Such a masking scheme can be seen as performing a search process. Accordingly, a search approach can be designed as the agent decision generator to obtain feasible solutions. Then the proposed framework is to learn knowledge and search for solutions according to the knowledge. Specifically, in the learning process, the approximate value function is learned as the agent decision model. In the searching process, a state-action tree (SAT) with monotonicity is constructed based on the obtained approximate value function. Then, a tree search approach is adopted as the agent decision generator to search for feasible solutions.

Under the proposed framework, a TSRL algorithm is designed to solve the practical two-dimensional cutting stock problem. The whole process of TSRL is illustrated in Fig. 3. In the learning process, a linear approximate value function $\hat{V}(s)$ is employed to serve as the agent decision model and describe the functional relationship between reward and state. The parameters in the linear approximate value function are obtained by solving a linear programming of the approximate Bellman equations, i.e., ASIB-LP. After that, we construct an RL-SAT with monotonicity and propose a monotonicity cut to remove the unpromising actions from the RL-SAT. Then, in the searching process, a parallel two-stage (i.e., bound search stage and optimal search stage) tree search approach is designed as the agent decision generator to obtain the near-optimal solutions.

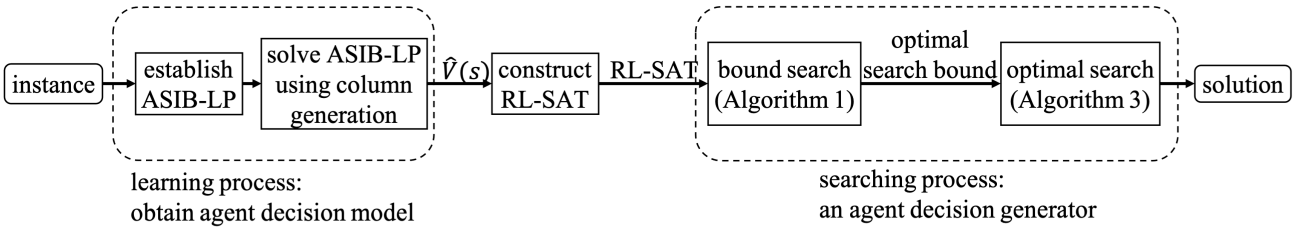


Fig. 3. Overall TSRL process.

The proposed Learning & Searching framework can be applied to solve general combinatorial optimization problems with complex constraints. When implementing the proposed framework, some customization modifications to the learning process may be required according to the specific problems. For example, the approximate value function can be replaced with a neural network whose parameters are obtained by DQN. However, constructing SAT trees with monotonicity and the tree search approach can be directly used to tackle the cumulative constraints and obtain near-optimal feasible solutions.

B. Learning Method for Obtaining the Agent Decision Model

In this paper, the problem can be seen as a resource allocation problem. According to references [34] and [35], a linear architecture is suitable to approximate the value function.

Hence, we select the linear function (22) as the approximate value function in ASIB.

$$\hat{V}(s) = \sum_{i \in N} \theta_i s_i, \theta \in \mathbb{R}_+^{|S|} \tag{22}$$

Many approaches, such as Q-Learning and State-Action-Reward-State-Action, can be used to obtain the parameters of the approximate value function by sampling states and actions. The trade-off between exploration and exploitation is important for these sampling-based approaches. However, finding a strategy for the best trade-off between exploitation and exploration is not an easy task. Inspired by reference [36], we adopted linear programming to obtain the parameters θ_i in approximate value function (22). Different from the sampling-based approaches, the optimal parameters can be obtained by directly solving the linear programming without considering the exploration-exploitation trade-off.

First, the optimal approximate value function at the initial state s_0 can be computed by the following linear programming (23) with decision variables $\hat{V}(s_t), \forall t$.

$$\begin{aligned} &\max_{\hat{V}(s_t)} \quad \hat{V}(s_0) \\ &s.t. \quad \hat{V}(s_t) \le C(a_t) + \hat{V}(s_{t+1}), t \in T \end{aligned} \tag{23}$$

Substituting Equation (22) into (23), we can obtain ASIB-LP with decision variable $\theta_i, \forall i$.

$$\begin{aligned} &\text{(ASIB-LP)} \\ &\max_{\theta \in \mathcal{R}^{|N|}} \quad \sum_{i \in N} \theta_i s_{0,i} \\ &s.t. \sum_{i \in N} \theta_i \sum_{m \in M} a_{m,i} - \mathcal{L}(a, \mathcal{I}) * \mathcal{W}(\mathcal{I}) + \mathcal{Y}(a_t) \le 0, a \in A \end{aligned} \tag{24}$$

By solving ASIB-LP, we can obtain the optimal value of parameter θ_i. Because the ASIB-LP has few variables and many constraints, enumerating all of the constraints is a time-consuming task. However, the dual problem of the ASIB-LP has few constraints and many variables and is therefore suitable for solving by column generation. According to duality theory, the variables θ in the ASIB-LP are equivalent to the dual variables of the dual problem of the ASIB-LP. Hence, θ can be obtained by solving the dual problem of ASIB-LP using column generation. Let λ_a denote the dual variables; then, according to reference [37], the dual problem $\mathrm{D_A}$ of the ASIB-LP is expressed in formulation (25).

$$\begin{aligned} &(\mathrm{D_A}) \quad \max_{\lambda_a > 0} \sum_{a \in A} \lambda_a \left(-\mathcal{L}(a, \mathcal{I}) * \mathcal{W}(\mathcal{I}) + \mathcal{Y}(a)\right) \\ &s.t. \quad \sum_{a \in U} \lambda_a \sum_{m \in M} a_{m,i} - s_{0,i} \ge 0, \quad i \in N \end{aligned} \tag{25}$$

Obviously, there is a large number of variables in (25), each corresponding to action $a_{m,i} \in A$ of the ASIB-LP. To obtain the optimal solution of D(A), we should know all actions $a \in A$. However, it is inefficient to enumerate all $a \in A$. Column generation is an effective approach to solve linear programming with a large number of variables by partially considering variables. Hence, column generation is adopted in this study to solve the dual problem of ASIB-LP.

The concept of the column generation algorithm is as follows. First, a restricted main problem, including only a subset of the columns of linear programming, is solved. Then, the values of the dual variables obtained by solving the restricted main problem are used to update the pricing subproblem, which is to find the columns with a negative reduced cost. If such columns exist, they are added to the restricted main problem. This procedure is repeated until no column with a negative reduced cost remains. The final solution obtained is the optimal solution for the primal linear programming.

In this study, θ_i is the dual variable that corresponds to the constraints of $\mathrm{D_A}$. The pricing subproblem, denoted as SP, can be formulated as the following integer programming:

$$\begin{aligned} &(\mathrm{SP}) \quad \min_{a \in A} \mathcal{L}(a, \mathcal{I}) * \mathcal{W}(\mathcal{I}) - \mathcal{Y}(a) - \sum_{i \in N} \theta_i \sum_{m \in M} a_{m,i} \\ &s.t. \quad (10) - (15) \end{aligned}$$

First, we only include a subset of A, that is, A', and the corresponding variables λ_a in the restricted main problem, denoted as $\mathrm{D_{A'}}$. In each iteration, the subproblem (SP) is solved to find the columns with a negative reduced cost. If the negative columns exist, we add them to $\mathrm{D_{A'}}$ and update the value of the dual variable θ_i. This procedure is repeated until

no column with a negative reduced cost can be generated; in this scenario, the value of dual variable θ_i is optimal. Using the optimal θ_i, we can obtain the approximate value function, that is, $\hat{V}(s) = \theta\phi(s)$, which serves as the agent decision model. The agent decision generator can then generate solutions based on the agent decision model.

C. Parallel Two-Stage Tree Search as Agent Decision Generator

As described in III-A, tree search is an appropriate approach in the process of utilizing knowledge. The MCTS, for example, AlphaZero [38], is a well-known, widely-adopted, and successful tree search approach in RL. A famous successful example is MCTS-based RL which has won world championship in Go. However, MCTS cannot be used directly to deal with the constraints in our study. This is because the goal of the MCTS is to win a game, but winning the game is not a strict constraint. The actions in the MCTS are determined by neural network evaluation, without considering the cumulative constraint. The upper confidence bound of the MCTS makes it easier to find a better solution, but the feasibility of the solution cannot be guaranteed. Therefore, we have designed a novel tree search approach to efficiently obtain feasible solutions for the problem with cumulative constraints.

First, we construct an SAT with monotonicity based on the agent decision model obtained during the learning process. The monotonicity of the SAT is beneficial for obtaining near-optimal solutions. We consider the action selection at each node as a parameterized optimization problem whose objective function is supermodular. We then propose a monotonicity cut to remove unpromising actions and thereby reduce the SAT. Finally, we design a parallel two-stage tree search to parallelize the searching process in the constructed SAT.

1) Construction of RL-SAT: The decision process starts at the root node s_0 and the state sequentially transfers from one node to another, along with the edges, until the state reaches a leaf node. There are two types of leaf nodes: feasible and infeasible. At an infeasible leaf node, there must be some unfulfilled demand but no feasible action can be implemented. For example, in the SAT of the PDPCD, the unfulfilled demand at the leaf node cannot be satisfied by forming any feasible plate. In another case, if all demands are satisfied at a node, then the node is a feasible leaf node. A trajectory is defined as the node sequence from the root node to the leaf node, from which we can obtain the state and action sequences. The action sequence forms a solution to the optimization problem. The solution is obviously infeasible if its relative trajectory ends at an infeasible leaf node in the SAT. For convenience, a feasible trajectory in the rest of the paper indicates that the solution corresponding to the trajectory is feasible.

We define RL-SAT as the state-action tree in which the child nodes s_t are sorted increasingly according to their RL evaluation function, that is, $C(a_t)+\hat{V}(s_{t+1})$. IC-SAT is defined as the state-action tree in which the child nodes are sorted increasingly according to the immediate cost, that is, $C(a_t)$. In each child node sequence, the evaluation of the left node is not greater than that of the right node. Based on the concepts of the left and right nodes, we can accordingly obtain the left and right trajectories. Given two nodes in the same child node sequence, the left trajectory is the trajectory from the root node crossing the left node, whereas the right trajectory is the trajectory from the root node across the right node. Then, we can obtain a monotonicity claim for the RL-SAT and IC-SAT.

Claim 1: In terms of the approximate value and immediate cost, the left trajectory is better than the right trajectory in RL-SAT and IC-SAT; in terms of the exact objective value, the left trajectory may be better than the right one.

The above claim implies the monotonicity of RL-SAT, and we can accordingly design an effective tree-search method to obtain near-optimal solutions.

2) Monotonicity Cut: At each node s, the selection of actions can be described as an optimization problem, that is $g(s) = \min_{a\in A_s}\{C(a) + \hat{V}(f(s,a))\}$, where s denotes the parameters and a denotes the variables. We can prove that the objective function of the parameterized optimization problem $g(s)$ is supermodular. Supermodularity is a cardinal property of a function defined on a lattice or a set, stating that a function has "increasing differences." In general, supermodularity is used to analyze the structural features of the problem with supermodular functions [39]. Moreover, supermodularity can derive the monotone comparative statics [40], which can be used to obtian the monotonicity of the optimal action set (described in detail in Appendix C). The monotonicity property can be described as follows:

In our RL-SAT, the state set S is a partial-order set. Hence, for a given node s' and its child node s, we can get $s \le s'$. Let A_s^* and $A_{s'}^*$ denote the corresponding optimal action sets for s and s', respectively. Then, according to the monotonicity property, we obtain relationship (26) for each optimal action $a \in A_s^*$ and $a' \in A_{s'}^*$.

$$a \wedge a' \in A_s^*, a \vee a' \in A_{s'}^* \tag{26}$$

where $a \wedge a' = \sup\{z : z \le a, z \le a', z \in A\}$ and $a \vee a' = \inf\{z : a \le z, a' \le z, z \in A\}$. According to the monotonicity of the optimal action set, we can determine that the reduced action set $\bar{A}_s = \{z | z \not\geq a \wedge a', z \in A_s\}$ must include at least one optimal action. Hence, the action set A_s can be replaced by $\bar{A}_s$ when generating child nodes for s. However, a is unknown. Thus, we try to relax the reduced action set $\bar{A}_s$. Based on the definition of $a \wedge a'$, we have $a' \ge a \wedge a'$. Subsequently, the action set $\bar{A}_s$ is a subset of $\mathcal{A} = \{z | z \not\geq a', z \in A_s\}$. It is evident that action set $\mathcal{A}$ must also include at least one optimal solution. Hence, we can remove actions that do not belong to $\mathcal{A}$, thereby reducing the action space at each stage.

3) Parallel Two-Stage Tree Search: To accelerate the search process, we propose a strategy to parallelize search tasks. The key to the parallelism strategy is the method by which the search tasks are divided. In this study, we use the left/right boundary trajectories to restrict the search space of the task. A trajectory belongs to a given task only if it is between the left and right boundary trajectories. Let $\alpha \in [0, 1]$ denote the split rate used to generate trajectories. Given a node s and its corresponding child set S'_s, we select the $\lceil \alpha | S'_s | \rceil$-th child from S'_s to generate the trajectory. We can then obtain a trajectory

from the root node to the current level with a split rate α. Using different split rates, we can generate all the trajectories in the RL-SAT. Let α_l and α_r denote the left and right split rates, respectively; then, the left boundary trajectory β_l and the right boundary trajectory β_r can be obtained using α_l and α_r, respectively. Thus, we can say that the trajectory generated using $\alpha \in [\alpha_l, \alpha_r]$ is between β_l and β_r. By introducing the concept of split rate, we can split the entire search space into different subspaces and then parallelize them.

Suppose that we wish to partition the search space into p subspaces. We can then split the range $[0, 1]$ into p subranges with equal length, that is, $[0, 1/p], [1/p, 2/p], \cdots, [(p-1)/p, 1]$. Each subrange is defined by the left and right bounds of the split rate. Using the above subranges, the RL-SAT can be split into p subspaces.

The relationship between the split rate, trajectory, and solution can be easily described as follows: the split rate is used to generate trajectories by selecting the actions and child nodes for each node, and the trajectory selected by the split rate represents the sequence of actions, which corresponds to a solution. In the SAT, the search space is split and restricted by the right and left boundary trajectories generated by the corresponding split rates.

Parallelizing the tree search can speed up the solution process; however, it is still impossible to traverse the entire RL-SAT and obtain a near-optimal solution in a reasonable amount of time. Therefore, we propose a two-stage search strategy that divides the tree-search process into two stages. In the first stage, we try to find a much smaller search space. In the second step, we enumerate all possible trajectories within the resulting smaller search space.

According to claim 1, it is known that near-optimal solutions tend to appear in the left space of the RL-SAT. Hence, we design a bound search (BS) to find a smaller search space on the left side of the RL-SAT, that is, the optimal search space. In the bound search, we attempt to find a feasible trajectory in the RL-SAT in parallel form. We can then obtain the optimal search space, in which the left bound is the leftmost trajectory of the RL-SAT and the right bound is the trajectory obtained by the bound search.

Algorithm 1 Bound Search (BS)

Input: root node s_0 of RL-SAT, number of bound search processors: p_1
Output: global rate upper bound $\overline{\alpha}_g$, global best solution H^*, global best objective value v^*

1: **Initial**: global rate upper bound $\overline{\alpha}_g = 1$, global best objective value $v^* = +\infty$, convergence gap $\delta = 10^{-3}$
2: **for** $z = 1 : p_1$ **do**
3: BSP$_z(z, \overline{\alpha}_g, v^*, \delta, s_0)$ ▷ Algorithm 2
4: **end for**
5: **return** $\overline{\alpha}_g, H^*, v^*$

Algorithm 1 illustrates the bound search process. We split the search space of the RL-SAT into p_1 subspaces using split rates. Each subspace is processed using a bound search processor (BSP). In Algorithm 1, $\overline{\alpha}_g$ is the upper bound of the global split rate used to generate the optimal search bound; H^* and v^* are the global best solution and the global best objective value, respectively. The values of $\overline{\alpha}_g$, H^*, and v^* are shared and maintained by each bound search processor. The workflow of the bound search processor is illustrated in Algorithm 2. First, the parameters, including the local upper bound of the split rate α_g, the left bound of the split rate $\underline{\alpha}$ used to generate the left boundary trajectory, and the right bound of the split rate $\overline{\alpha}$ used to generate the right boundary trajectory, are initialized, corresponding to line 3. We then calculate the split rate α using the formula in line 5. Subsequently, a trajectory is generated based on α, corresponding to lines 6×9. If the trajectory is feasible, the corresponding parameters, that is, $\overline{\alpha}_g$, H^*, and v^*, are updated. If the trajectory is infeasible, the parameter η is updated, and the split rate α is recalculated to generate another trajectory. In this process, η is used to control the step size to generate a new trajectory. In the bound search, once the upper bound of the global split rate $\overline{\alpha}_g$ is updated, the local left and right bounds of the split rate for all processors should be recalculated, that is, the subspaces for all processors are redivided. The processors then process the new search space until the gap in $\overline{\alpha}_g$ hits the tolerance. After obtaining the upper bound of the global split rate $\overline{\alpha}_g$, we can generate a feasible trajectory (i.e., the optimal search bound) to restrict the optimal search space. In the second phase, we design an optimal search (OS) to obtain a near-optimal solution in the obtained optimal search space.

Algorithm 2 Bound Search Processor: BSP_z

Input: processor ID z, global rate upper bound $\overline{\alpha}_g$, global best objective value v^*, convergence gap δ, root node s_0
Output: $\overline{\alpha}_g$, v^*, and global best trajectory H^*

1: **Initial**: $\eta = 0$, $\overline{\eta} = 3$, $\gamma = 0.5$
2: **while** $\eta < \overline{\eta}$ **do**
3: $\alpha_g = \overline{\alpha}_g$, $\overline{\alpha} = z\overline{\alpha}_g/p_2$, $\underline{\alpha} = (z-1)\overline{\alpha}_g/p_2$
4: H = emptyList(), $s = s_0$
5: $\alpha = \gamma^{\eta}(\overline{\alpha} - \underline{\alpha}) + \underline{\alpha}$
6: **while** notTerminalLeaf(s) **do**
7: A = getChildrenList(s)
8: $i = \lceil \alpha |A| \rceil$, $a = A_i$, append(H, a)
9: s =getChildNode(s, a)
10: **end while**
11: **if** isFeasibleLeaf(s) **then**
12: **if** $\overline{\alpha}_g > \alpha_c$ **then**
13: $\overline{\alpha}_g = \alpha_c$
14: **end if**
15: **if** objectiveValue(H)$< v^*$ **then**
16: $\alpha^* = \alpha$, $H^* = H$,v^* =objectiveValue(H)
17: **end if**
18: **end if**
19: **if** $\alpha_g - \overline{\alpha}_g < \delta$ **then**
20: $\eta \leftarrow \eta + 1$
21: **else**
22: $\eta = 0$
23: **end if**
24: **end while**
25: **return** $\overline{\alpha}_g$, H^*, v^*

Algorithm 3 Optimal Search (OS)

Input: right bound of optimal search space $\overline{\alpha}_g$, number of optimal search processors: p_2

Output: global best trajectory H^*, global best objective value v^*

Initial: bound for each processors: $0, \frac{\overline{\alpha}_g}{p_2}, \frac{2\overline{\alpha}_g}{p_2}, \cdots, \overline{\alpha}_g$, trajectory bound for each processor: $H_0, H_1, \cdots, H_{p_2}$
for $b = 1 : p_2$ **do**
  $\text{OSP}_b(H_{b-1}, H_b, s_0)$ ▷ Algorithm 4
end for
return H^*, v^*

Because the optimal search space is significantly reduced in the bound search, we can enumerate all trajectories to obtain a near-optimal trajectory in the OS. Similar to the bound search, we parallelize the search space for p_2 optimal search processors to accelerate the search process. Algorithm 3 shows the optimal search process. The optimal search processor

Algorithm 4 Optimal Search Processor: OSP_b

Input: left bound H_{b-1}, right bound H_b, root node s_0

Output: global best trajectory H^*, global best objective value v^*

Initial: $H = H_b$
while $H \neq H_{b-1}$ **do**
  $s = \text{getLastNode}(H)$
  if isFeasibleLeaf(s) **and** objectiveValue(H)<v^* **then**
    $v^* = \text{objectiveValue}(H)$, $H^* = H$
  end if
  if partialObjectiveValueFromRootTo(s) $\geq v^*$ **then**
    deleteLastNode(H)
  else if isInfeasibleLeaf(s) **then**
    deleteLastNode(H)
  else if noChildExist(s) **then**
    deleteLastNode(H)
  else
    $s = \text{getNextChild}(s)$
    append(H, s)
  end if
end while
return H^*, v^*

workflow is illustrated in Algorithm 4. The optimal search space is split into p_2 sub-spaces using $[0, \alpha^*/p_2], [\alpha^*/p_2, 2*\alpha^*/p_2], \cdots, [(p_2-1)*\alpha^*/p_2, \alpha^*]$. Each subspace is allocated to an optimal search processor. In an optimal search processor, all trajectories in the subspaces are enumerated. Although near-optimal solutions tend to appear in the left area of the RL-SAT, there are also many infeasible trajectories in this area. Therefore, we implement a depth-first tree search from right to left in each optimal search processor to avoid spending time on a large number of infeasible trajectories. In addition, a pruning strategy is adopted in the optimal search processor. In the searching process, if the objective value of a partial trajectory is larger than the global best objective value v^*, the corresponding branch should be pruned. If a feasible solution is found and its objective value is smaller than the global best objective value, we update the global best objective value and global best trajectory using the obtained solutions. An illustration of the parallel two-stage tree search is presented in Fig. 4.

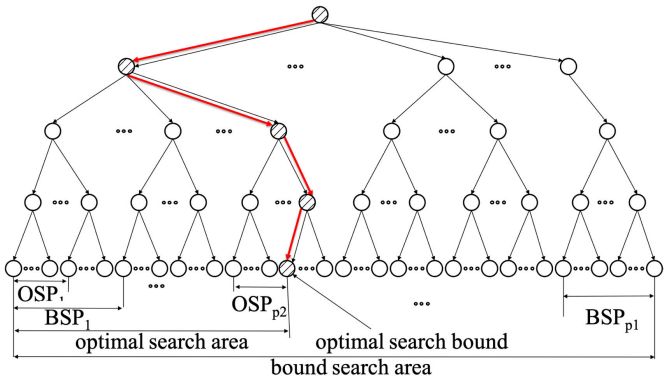


Fig. 4. Bound search and optimal search on RL-SAT.

IV. Experimental Results

All experiments were performed on a computer with an Intel(R) Xeon(R) Silver 4110 CPU @ 2.10 GHz with 32 threads, 32 GB of memory, and using the Ubuntu 18.04.2 LTS operating system. Instances were randomly generated based on real production data. In the steel enterprise under consideration, no more than four types of slab sizes and no more than seven orders at a time are considered when planners design steel plates. The demand for subplates of each order ranges from 1 to 3. As per this practical scenario, we generated a set of instances where the number of orders ranged from 1 to 10, the slab size ranged from 1 to 4, and the order demand ranged from 1 to 5. By combining the above parameters, we obtained instances of different scales. For each fixed instance scale, ten instances were randomly generated based on the given number of orders, slab size types, and order demands. In each instance, all information on orders and slab size is based on real-world data collected from the steel enterprise under consideration. In this way, we obtained a total of $10*4*5*10 = 2000$ instances, of which 1899 were feasible instances.

A. Baselines

To evaluate the performance of our algorithm, we compared it with multiple baselines in terms of the solution quality and computation time. GUROBI [41] is considered the most powerful mathematical optimization solver and can be directly employed to solve the integer programming model of the problem. Hence, we adopted GUROBI as a basis to evaluate the solution quality for the algorithms. For a given algorithm, we used the relative gap between the objective value obtained by the algorithm and that by GUROBI to indicate the solution quality. The calculation of the relative gap for the algorithm alg is given by Equation (27).

$$gap(alg) = \frac{Obj_{alg} - Obj_{GUROBI}}{Obj_{GUROBI}} \times 100\% \quad (27)$$

We set the time limit and nonconvex parameter for GUROBI to 3600 s and 2, respectively; the default values were used for the other parameters. Under these settings, GUROBI is employed to solve the integer programming model of our problem, that is, $\{\min_{a_t} \sum_{t\in T} C(a_t), s.t.(9)-(15)\}$ for all instances.

In our experiments, we evaluated the performance of TSRL by comparing it with different baselines. We evaluated the ability of TSRL to generate feasible solutions, the effectiveness of the proposed improvement strategies, and the effectiveness of RL-SAT. Several RL baselines are listed as follows.

- HRL-GPN: Referring to reference [33], we transfer CSP as a bin packing problem using a binary extension. The item features include the length and width of items as well as the width, maximum length, and minimum length of slabs. The packed items are not considered, and the reward is defined as the summation of the trim loss and the minimum constraints violation.
- Ptr-Net: Referring to reference [42], the definition of binary extension and item features were the same as that of the HRL-GPN. As per reference [43], we defined the reward as 10^4 if the solution is infeasible and trim loss otherwise.
- ranked reward MCTS (RRMCTS): Referring to reference [26], the definition of binary extension and item features were the same as that of the HRL-GPN. The reward was defined as the summation of the trim loss and minimum constraints violations.

To evaluate the performance of RL-SAT, we defined three myopic baselines: 1) greedy policy in RL-SAT (GRL), 2) greedy policy in IC-SAT (GIC), and 3) tree search in IC-SAT (TSIC). GRL and GIC perform the greedy policy on IC-SAT and RL-SAT, respectively, without considering the constraints, while TSIC searches for the first feasible trajectory using a depth-first tree search.

- GRL: Established an ASIB-LP for all instances. By using the ALP and column generation described in Section III-B, we could obtain the approximate parameters of the approximate value function. Accordingly, solutions were generated by applying a greedy policy at each stage.
- GIC: Generated solutions by applying a greedy policy at each stage using the immediate cost $C(a)$.
- TSIC: Referring to the best-first search in reference [44], the TSIC was selected as a baseline. In TSIC, IC-SAT is constructed using an immediate cost. Then, TSIC finds the first feasible trajectory using a depth-first tree search, which begins from the leftmost trajectory.

Furthermore, we verify the necessity and effectiveness of the monotonicity cut using the basic TSRL (b-TSRL) and TSRL.

- TSRL: the complete algorithm proposed in this study, which works as follows: 1) Establish ASIB-LP and obtain an approximate value function using ALP and column generation. 2) Construct RL-SAT according to the approximate function and reduce RL-SAT using monotonicity cut. Subsequently, a parallel two-stage tree search on the RL-SAT is conducted to obtain a near-optimal solution. According to the computational time requirement of steel enterprise, the optimal search time is limited by 120 s.
- b-TSRL: The process of b-TSRL is the same as that of TSRL, without using the monotonicity cut.

In addition, metaheuristic algorithm also shows good performance when solving complex combinatorial optimization problems. However, the published metaheuristic algorithms cannot be adopted to solve our problem directly. Therefore, with the minimal modification, we select two popular metaheuristic algorithms as the baseline to evaluate our proposed algorithm.

- SParEA: Referring to reference [45], set individual as a vector in which each unit range is $(0, |M|\times|N|\times s_0]$. Each unit labels a sub-plate, according to the unit's value, slab type, order type, and sub-plate are identified. The fitness value is the sum of the trim loss and the penalty of the violation of constraints, that is, the order demand and the maximum/minimum plate length constraints.
- ACO: Referring to reference [46], the ant colony optimization algorithm, which is used to solve 1D cutting stock problem, is adopted as a baseline. The individual is defined as the same as that in SParEA. The constraints are handled using the penalty function according to reference [47]. The fitness value is the same as that in SParEA.

To analyze the algorithm performance, we partitioned instances into 196 sets, grouped them by scales, and plotted the performance indices including the gap and computational time for each set. To highlight the trends in the lines in the figures and to make the performance of the algorithms easier to analyze, we smooth the lines using the slip average method with a slip window size of 5. Color, width, and transparency are used to distinguish between the data. The colors are used to distinguish the algorithm results, and the width and transparency lines are used to distinguish between the original and smoothed data. The non-transparent and width lines show the smoothed data, and the transparent and thin lines show the original data. In each figure, the instance axis shows the IDs of the instance sets, whereas the vertical axis shows the performance of the algorithms.

B. Performance in Generating Feasible Solutions

In this subsection, we demonstrate the ability of the proposed approach to generate a feasible solution using Fig. 5. Fig. 5 gives the number of instances in which each algorithm can obtain a feasible solution. From Fig. 5, we can make the following observations. First, in terms of generating feasible solutions, the tree-search-based algorithms (i.e., RRMCTS, TSIC, b-TSRL, and TSRL) outperform the traditional RL algorithms where the actions are selected based on maximum value and probability (i.e., Ptr-Net, HRL-GPN, GIC, and GRL). It implies that the tree search scheme is suitable to tackle the cumulative constraints and can provide more opportunities to get feasible solutions. Second, the penalty-based RL algorithms, i.e., Ptr-Net, and HRL-GPN, perform worst in terms of generating feasible solutions. It demonstrates that reducing constraint violations by penalty fails to tackle the complex constraints involved in our problem. Moreover,

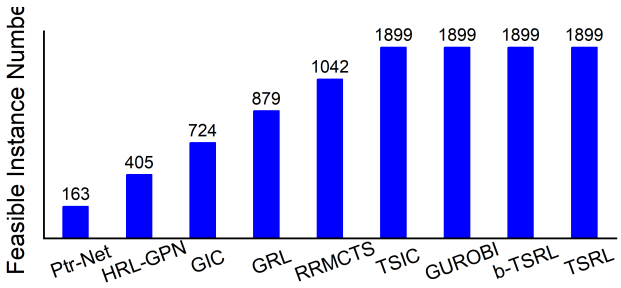
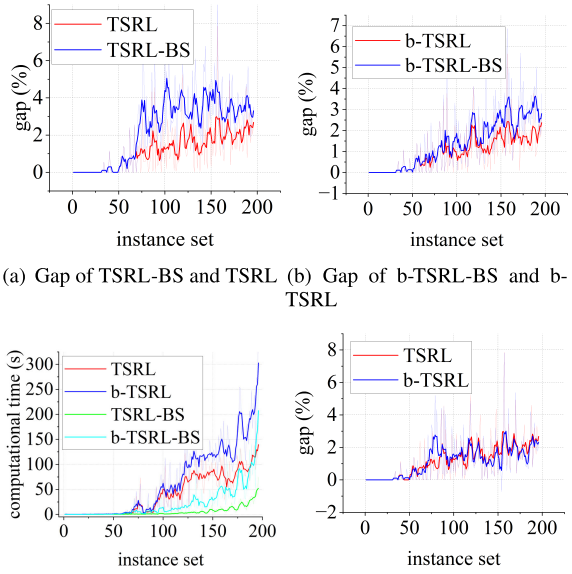


Fig. 5. Number of instances that obtain feasible solution.



(a) Gap of TSRL-BS and TSRL (b) Gap of b-TSRL-BS and b-TSRL

(c) Computational time of b-TSRL and TSRL (d) Gap of TSRL and b-TSRL

Fig. 6. Computational time and gap of b-TSRL and TSRL. TSRL-BS and b-TSRL-BS denote that TSRL and b-TSRL without optimal search process.

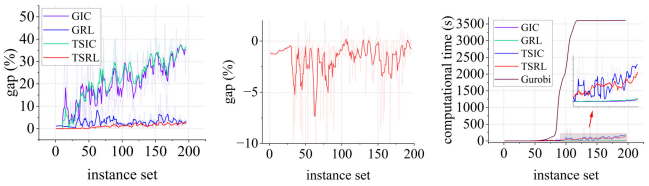
although RRMCTS is developed based on tree search, it does not obtain feasible solutions in all instances. It is because RRMCTS focuses on obtaining a better solution but not a feasible one. Finally, all algorithms under the learning and searching framework, i.e., TSRL, TSIC, and b-TSRL can obtain feasible solutions in all instances. We can conclude that the proposed framework can overcome the limitations of RL in addressing complex constraints.

C. Effectiveness of the Improvement Strategies

In this subsection, we evaluate the effectiveness of the proposed improvement strategies by comparing the complete TSRL with three different versions of TSRL, i.e., b-TSRL, TSRL-BS, and b-TSRL-BS. TSRL-BS is the TSRL without the optimal search process, and b-TSRL-BS is the TSRL without the monotonicity cut and optimal search process.

Figures 6(a) and 6(b) give the comparison of the relative gaps between the algorithms with and without optimal search process. From the comparison results, we can see that the optimal search process can significantly improve the feasible solutions obtained in the bound search process. It is worthwhile to spend computation time conducting an optimal search to obtain a near-optimal solution.

Fig. 6(c) and Fig. 6(d) show the computation time and relative gaps of the algorithms with and without monotonicity



(a) Gaps of greedy algorithms (b) Difference between GRL and TSRL gaps (c) Computational time of greedy algorithms

Fig. 7. Gap and computational time of greedy algorithms.

cut. We can see that the introduction of the monotonicity cut can significantly shorten the computation time and improve the solution quality (particularly for middle-scale instances). Since the monotonicity cut can remove the unpromising actions, TSRL is more stable than b-TSRL in terms of performance. Moreover, although TSRL is a tree search-based approach, it is hard to provide an exact time complexity analysis since the number of visited trajectories is unknown. However, by observing 6(c), we can find that the computation time of TSRL increases approximately linearly with increasing size.

D. Advantage of RL-SAT

To evaluate the advantages of RL-SAT over the greedy strategy and IC-SAT, we compared our TSRL with GIC, GRL, and TSIC. The comparison results are shown in Fig. 7.

GIC and TSIC both search for trajectories in IC-SAT and return different trajectories. GIC returns the leftmost trajectory, whereas TSIC returns the leftmost feasible trajectory. Hence, TSIC can guarantee that the obtained trajectory is feasible, whereas GIC cannot. For the instance in which the leftmost trajectory is feasible, TSIC and GIC obtain the same trajectory. Thus, we can see that TSIC and GIC have similar performance in terms of solution quality. TSRL and GRL search for trajectories in RL-SAT. From the results, we found that the objective values of the trajectories obtained in RL-SAT are better than those of the trajectories obtained in IC-SAT. It is clear that the gaps obtained by TSRL and GRL are much smaller than those obtained by GIC and TSIC. It means the solutions obtained by RL-SAT are better than those obtained by IC-SAT. This implies that the proposed RL-SAT outperforms IC-SAT and can effectively obtain a near-optimal solution.

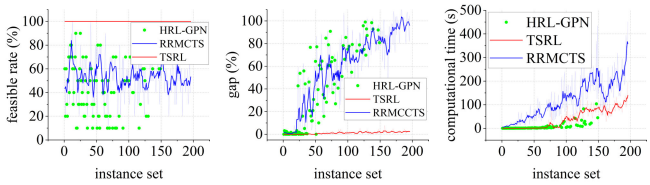
Fig. 7(b) shows the gap difference between the GRL and TSRL. From the results, we can see that TSRL outperforms GRL. As described in subsection IV-A, GRL searches for the leftmost trajectory of RL-SAT, whereas TSRL searches for the near-optimal trajectory guided by Claim 1. It is obvious that the TSRL obtains a trajectory better than the leftmost trajectory, which verifies the effectiveness of our TSRL. Fig. 7(c) shows the computation time of the algorithms. Although GRL and GIC are faster than TSRL, TSRL outperforms GRL in terms of solution quality and ability to obtain feasible solutions. Therefore, it is worth consuming longer computation time to guarantee the feasibility and quality of the solutions.

E. Comparison of TSRL With the State-of-the-Art

In this subsection, we compare the proposed TSRL algorithm with the state-of-the-art methods, including reinforcement learning methods and metaheuristic algorithms.

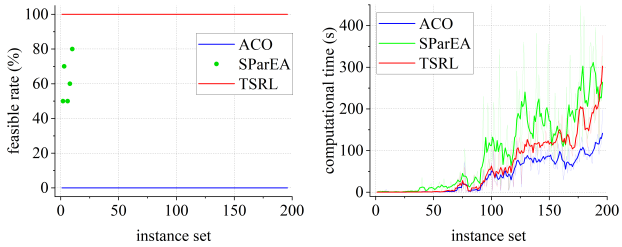
TABLE I
EVALUATE TSRL USING GAP AND TIME RATIO, WHERE EACH CELL IS REPORTED USING THE FORM OF GAP/TIME RATIO

$\|N\|$	$\|M\|$	$\|I\|$ 1	2	3	4	5	6	7	8	9	10
1	1	-/-	0/0.02	0/0.02	0/**2.78**	0.11/0.06	0.29/0.09	0/0.15	0.78/0.21	0.83/0.09	0.90/0.11
	2	-/-	0/0.01	0/0.02	0.11/0.06	0/0.18	0.61/0.25	0.75/0.20	1.25/0.52	1.40/**1.03**	1.06/0.59
	3	-/-	0/0.02	0/0.02	0/0.09	0.33/0.12	0.75/0.65	0.87/0.58	1.65/**7.50**	0.30/**5.87**	**-2.97/3.04**
	4	-/-	0/0.02	0/0.03	0.06/0.21	0.73/0.29	0.69/**1.61**	1.56/**11.17**	0.91/**90.74**	**-1.57/60.26**	**-7.04/29.20**
2	1	0/0.02	0/0.03	0.11/0.08	0.29/0.18	0.65/0.28	0.76/0.44	0.73/0.97	1.00/**7.30**	1.08/**70.59**	0.65/**63.36**
	2	0/0.01	0/0.06	0/0.24	0.70/0.19	1.43/**1.23**	1.07/**9.14**	0.84/**33.73**	1.01/**98.59**	**-0.48**/0.57	**-2.72/59.27**
	3	0/0.02	0/0.10	0.87/0.29	1.23/0.47	1.23/**12.42**	0.89/**12.51**	0.56/**127.38**	**-7.15/95.15**	**-2.13/18.62**	**-8.49/7.97**
	4	0/0.02	0.11/0.13	0.47/**1.15**	0.78/**6.63**	0.25/**13.14**	1.17/**122.05**	**-6.40/131.55**	**-2.96/74.85**	**-6.41/27.44**	**-5.67/18.69**
3	1	0/0.02	0/0.05	0.29/0.09	0.33/0.11	1.02/0.29	1.24/0.94	1.00/**6.93**	0.09/**63.52**	1.17/**63.06**	0.87/**43.00**
	2	0/0.02	0.11/0.21	0.75/0.29	1.27/0.64	1.20/**36.90**	0.47/**100.44**	0.77/**169.58**	**-3.16/31.95**	**-1.45/1.79**	**-1.82/23.13**
	3	0/0.02	0.29/0.21	0.90/0.66	1.64/**9.30**	0.23/**46.10**	**-1.44/121.50**	**-3.36/154.30**	**-13.09/43.04**	**-1.11/58.53**	1.99/**44.29**
	4	0/0.03	0/0.31	1.12/**1.15**	0.89/**37.76**	**-0.02/325.71**	**-0.96/101.55**	**-4.36/74.56**	**-6.70/57.74**	**-0.77/17.46**	2.73/**25.92**
4	1	0/0.02	0/0.06	0.15/0.13	0.20/0.15	0.96/0.60	1.29/**2.39**	1.38/**6.42**	1.13/**1.99**	1.29/**98.82**	**-1.42/53.38**
	2	0/0.02	0.23/0.14	0.75/0.51	0.63/**27.60**	1.04/**108.57**	**-0.50/73.45**	**-1.90/89.66**	**-4.46/68.47**	**-2.82/28.96**	**-2.08/23.79**
	3	0/0.03	0/0.25	1.41/**1.53**	0.86/**8.78**	**-0.25/228.38**	**-6.05/72.69**	**-2.55/25.57**	**-7.05/17.92**	1.90/**24.59**	2.27/**21.52**
	4	0/0.05	0.71/0.32	2.21/**14.16**	0.80/**336.60**	**-4.11/151.63**	**-4.49/194.27**	1.62/**21.31**	2.39/**26.45**	2.55/0.29	2.13/**15.14**
5	1	0/0.02	0/0.07	0.57/0.11	1.14/0.18	1.22/**1.14**	1.43/**3.19**	0.46/**65.07**	**-0.87/47.40**	**-1.64**/0.45	0.45/**61.18**
	2	0/0.04	0/0.18	0.70/0.85	1.37/**71.42**	1.08/**235.07**	**-3.32/79.78**	**-5.04/65.73**	**-4.50/33.98**	**-10.97/16.36**	**-10.53/3.06**
	3	0/0.04	0.78/0.35	1.84/**10.21**	0.87/**133.12**	**-1.58/170.01**	**-10.89/3.10**	**-14.57/3.01**	**-2.56/12.50**	2.41/0.41	2.31/**5.68**
	4	0/0.06	0.89/0.67	1.11/**21.47**	**-3.03/403.54**	**-4.85/112.26**	**-15.54/49.16**	1.53/**12.96**	2.30/**8.60**	2.20/0.57	2.40/**5.15**



(a) Feasible rate of RL algorithms (b) Gaps of RL algorithms (c) Computational time of RL algorithms

Fig. 8. Feasible rate, gap and computational time of RL algorithms.



(a) Feasible rate of metaheuristic algorithms (b) Computational time of metaheuristic algorithms

Fig. 9. Feasible rate and computational time of metaheuristic algorithms.

1) Reinforcement Learning Methods: Fig. 8 shows the performance indicators of TSRL, HRL-GPN, and RRMCTS, including the proportion of instances where feasible solutions are obtained, relative gaps, and computation time. HRL-GPN obtained feasible solutions in 21.33% of all instances and provided results in 93 instance sets. Therefore, the results of the HRL-GPN are presented using scatter plots.

From Fig. 8(a) and Fig. 8(c), we can see that HRL-GPN is faster than the tree-search-based RL algorithms, i.e., RRMCTS and TSRL. However, HRL-GPN performs the worst in terms of obtaining feasible solutions. The tree-search-based RL algorithms are time-consuming but have a strong ability to search for feasible solutions. Compare to RRMCTS and HRL-GPN, our TSRL can obtain feasible solutions for all instances within an acceptable computation time.

From Fig. 8(b), we can see that TSRL outperformed HRL-GPN and RRMCTS in terms of solution quality. In addition, the gap in TSRL did not worsen with increasing instance scale, whereas the gaps in HRL-GPN and RRMCTS increased significantly with increasing instance scale. The reasons that make TSRL perform well may include: 1) the designed approximate value function with linear structure is closer to the true value function than the approximate value functions with graph networks and multi-layer perception, 2) the optimal approximate parameters can be obtained by the approximate linear programming and column generation, and 3) an effective tree search algorithm is designed on an SAT with monotonicity.

2) Metaheuristic Methods: Ensuring solution feasibility is one of the main challenges for metaheuristic methods [47], [48]. As Fig. 9(a) shows, ACO does not obtain any feasible solution on all 1899 instances, and SParEA obtains 32 feasible solutions. It suggest that the complex constraints involved in our problem pose a significant bottleneck for metaheuristic methods, necessitating a customized strategy to ensure solution feasibility. Due to the absence of feasible solutions from ACO and SParEA, we only report the computational time of each algorithm in Fig. 9(b). The stop condition of ACO and SParEA is their computational time reaching that of TSRL. From the comparison, it can be seen that ACO and SParEA perform significantly worse than our proposed algorithm in obtaining feasible solutions within the similar computational time, thus verifying the effectiveness of our algorithm in handling optimization problems with complex constraints.

F. Comparison of TSRL With GUROBI

From a practical application perspective, a steel enterprise requires that near-optimal solutions can be obtained in a

relatively short time. Hence, in addition to solution quality, efficiency is also important.

Therefore, we compared TSRL to GUROBI within the same time limit and with the same objective value. In TABLE I, "v_1/v_2" is used to record the average comparison results for each instance set. "v_1" represents the average gap between TSRL and GUROBI with the same time limit, which is set as the computation time of TSRL; "v_2" represents the time ratio and is calculated by dividing the computation time of TSRL by that of GUROBI when GUROBI reach the objective value of TSRL. There were 20 instances in which GUROBI could not obtain a feasible solution within the time limit. For TSRL, the average gap is 0 for 39 sets, positive for 108, and negative for 49; the gaps decrease along the main diagonal. It is worth noting that the maximum gap between TSRL and GUROBI is only 2.7%, while the maximum improvement of TSRL over GUROBI is 15.5%. From the comparison results of time ratio, we can see that TSRL is more stable regarding the computational time to obtain the same objective value. For large-scale instances, GUROBI consumes more computation time than TSRL when adopting the same performance as the stopping condition. From the comparison results, we believe that TSRL is a competitive approach.

V. Conclusion

In this study, we propose a learning and searching framework to solve the combinational optimization problem with complex constraints. Under the new framework, a TSRL algorithm is proposed for the practical two-dimensional cutting stock problem derived from the plate design process in the steel industry. First, we design an approximate value function with a linear structure for MDP and then obtain the parameters of the approximate value function by ALP and column generation. Based on the obtained approximate value function, we construct an RL-SAT with monotonicity and accordingly propose a parallel two-stage tree search algorithm to search for near-optimal solutions. Finally, to further improve the solution quality, a monotonicity cut is proposed to remove the unpromising actions. By comparing with different baselines, the proposed framework, algorithms, and improvement strategies are proven to be well-designed and well-performed. The proposed framework can also be applied to other combinatorial optimization problems with complex constraints.

The proposed learning and searching framework makes several explorations and contributions in terms of handling complex constraints while using RL. It can also be used to solve other discrete-action MDPs. The learning process should be specifically designed according to the characteristics of the problem. In addition, when a different RL algorithm (e.g., action value-based RL, deterministic policy RL, or stochastic policy RL) is embedded in the learning and searching RL framework, the construction method of the state-action tree should be modified accordingly. For a given problem, designing an efficient learning and searching process is challenging. In the future, we will expand the proposed learning and searching framework to the other problems in more complex dynamic environments.

TABLE II
Summary Notations

N	orders $N = \{1, 2, \cdots, n\}$, which are sorted by width.
M	slab size set (identified by widths and thicknesses)
T	set of candidate plate/MDP stage, $T = \{0, 2, \cdots, \tau - 1\}$.
$a_{t,m,i}$	the number of subplates required by order i that is cut from plate t with slab size m, a_t denotes the design scenario
W_m	width of slab m
w_i	width of order i
$\beta_{m,i}$	the width extension ratio of the width of order i to the width of slab size m
l_i	length of order i
$\mathcal{I}_{t,m,i}$	widest label of order i in plate t with slab size m, which is determined by a_t
$D^s(\mathcal{I}_t)$	trim width caused by deformation
$D^b(\mathcal{I}_t)$	trim length caused by deformation
$\bar{l}_m$	maximum length of plate that is rolled using slab size m
$\underline{l}_m$	minimum length of plate that is rolled using slab size m
α^b, δ^b	weight and bias to calculate the deformation length.
α^s, δ^s	weight and bias to calculate the deformation width.
$C(a_t)$	cost of MDP, Trim loss of plate t which is formed using a_t.
$\mathcal{L}(a_t, \mathcal{I}_t)$	length of plate t which is formed using a_t.
$\mathcal{W}(\mathcal{I}_t)$	width of plate t which is formed using a_t.
$\mathcal{Y}(a_t)$	the total area of subplates placed on plate t
$C(a_t)$	total area of orders at plate t which is formed using a_t.
θ_i	weight of approximate value function for order i.
s_t	state of order at stage t where $s_t \in S_t$.
A	action space, A_{s_t} denotes action space for state s_t
a_t	action at stage t, design solution of plate t where $a_t \in A_{s_t}$.
$f(s_t, a_t)$	transition function.
$V(s_t)$	value of state s_t.
$\hat{V}(s_t)$	approximate value function to evaluate the value of state s_t.
λ_a	decision variable of master problem of column generation.
α	aplit rate which is used to generate a trajectory of the tree.
p	number of processors.
δ	stop situation of bound search.

Appendix A
Summary of Notations

A summary of the notations in this paper is listed as follows.

Appendix B
Proof of Proposition 1

Proposition 1: The stage-dependent Bellman equation is equivalent to the stage-independent Bellman equation.

Proof: The stage-dependent Bellman equation is equivalent to the stage-independent Bellman equation if the value of states is equal when observing the same state at a different stage. For an arbitrary stage t, the corresponding system state is s_t, representing the unfulfilled order demand. In the worst case, each plate only produces one subplate. Clearly, the unfulfilled order demands s_t can be satisfied by at most k plates, where $k = \sum_{i \in N} s_{t,i}$. The system can reach the terminal state by conducting at most k actions. The total cost (trim loss) of these k actions (plates) equals $sum_{j=t}^{t+k} C(a_j)$. For the unused plate j, we set $a_j = 0$ and the corresponding reward $C(a_j) = 0$.

According to the transition function, i.e., $s_{t+1} = s_t - \mathbf{1}a_t$, we have $s_{t+x} = s_t - \sum_{j=t}^{t+x-1} \mathbf{1}a_j$. In this context, we can get the values of the state s_x using (28), where $t \le x \le t + k$.

$$V_x(s_x) = \min_{a_x}\{C(a_x) + V_x(s_t - \sum_{j=t}^{x} \mathbf{1}a_j))\} \tag{28}$$

Since the order demand must be satisfied in the last stage, we have $a_{t+k} = s_{t+k}$. Then, the value of the last state can be calculated by (29).

$$V_{t+k}(s_{t+k}) = C(a_{t+k}) \tag{29}$$

By substituting (29) into (28) where $x = t+k-1$, we have (30).

$$V_{t+k-1}(s_{t+k-1}) = \min_{a_{t+k-1}} \{C(a_{t+k-1}) + C(a_{t+k})\} \tag{30}$$

When $x = t + k - 2$, $V(s_{t+k-2})$ can be calculated using (31).

$$\begin{aligned} &V_{t+k-2}(s_{t+k-2}) \\ &= \min_{a_{t+k-2}} \{C(a_{t+k-2}) + V_{t+k-1}(s_{t+k-1})\} \\ &= \min_{a_{t+k-2}} \{C(a_{t+k-2}) + \min_{a_{t+k-1}} \{C(a_{t+k-1}) + C(a_{t+k})\}\} \\ &= \min_{a_{t+k-2}, a_{t+k-1}, a_{t+k}} \{C(a_{t+k-2}) + C(a_{t+k-1}) + C(a_{t+k})\} \end{aligned} \tag{31}$$

In this way, we can obtain the formulation (32) to calculate $V_x(s_x)$ where $t \le x \le t + k$.

$$V_x(s_x) = \min_{a_x, a_{x+1}, \cdots, a_{t+k}} \sum_{j=x}^{t+k} C(a_j) \quad s.t. \sum_{j=x}^{t+k} \mathbf{1} a_j = s_x \tag{32}$$

For two arbitrary stages $t \neq t'$, and related states $s_t = s_{t'}$, the related optimal action sequences can be obtained according to (32). Because $s_t = s_{t'}$, the optimal action sequences $\{a_t^*, a_{t+1}^*, \cdots, a_{t+k}^*\}$ and $\{a_{t'}^*, a_{t'+1}^*, \cdots, a_{t'+k}^*\}$ are the same. Therefore, $V_t(s_t) = V_{t'}(s_{t'})$. Hence, we can conclude that the state's value is only relative to the state and does not depend on the stage. Then, we can drop subscript t, i.e., the stage-dependent Bellman equation is equivalent to the stage-independent Bellman equation. □

Appendix C
Monotonicity Property

According to Theorem 2.8.6 of book [40], the monotonicity property of optimal solution set in parameterized optimization problem has to satisfy conditions as follows:

1) S is a partially ordered set: it is obvious that S is a partially ordered set on the relationship $\le$.
2) A is a lattice: A is defined maximally equal to S, $\forall a_1, a_2 \in A$, $a_1 \vee a_2 \in A$ and $a_1 \wedge a_2 \in A$ hold.
3) $A(s)$ is a subset of A for each $s \in S$.
4) $A(s)$ is increasing in s on S: while S is increasing, A is increasing too.
5) $g(a, s)$ is quasi-supermodular in a on A for $s \in S$.
6) $g(a, s)$ has single crossing property in (a, s) on $A \times S$.

It is obvious that conditions (1) to (4) are satisfied. Following we will check the conditions (5) and (6).

A. Quasi-Supermodularity of Function g(a,x)

The optimization problem at each node is given as follows.

$$\min_{a \in A_s} \{C(a) + \hat{V}(f(s, a))\} \tag{33}$$

$$= \min_{a \in A_s} \{C(a) + \sum_{i \in N} \theta_i (s_i - \sum_{m \in M} a_{m,i})\} \tag{34}$$

It is obvious that (33) is equivalent to (35).

$$\max_{a \in A_s} -\{C(a) + \hat{V}(f(s, a))\} \tag{35}$$

$$= \max_{a \in A_s} -\{C(a) + \sum_{i \in N} \theta_i (s_i - \sum_{m \in M} a_{m,i})\} \tag{36}$$

We define function $g(a, s) = -\{C(a) + \sum_{i \in N} \theta_i (s_i - \sum_{m \in M} a_{m,i})\}$. According to reference [49], $g(a, s)$ is quasi-supermodular if conditions (37) and (38) hold for $a', a'' \in A_s$.

$$g(a', s) \ge g(a' \wedge a'', s) \Rightarrow g(a' \vee a'', s) \ge g(a'', s) \tag{37}$$

$$g(a', s) > g(a' \wedge a'', s) \Rightarrow g(a' \vee a'', s) > g(a'', s) \tag{38}$$

By substituting the function of $g(\cdot, s)$ into $g(a', s) \ge g(a' \wedge a'', s)$, we can obtain (40). From (40), we can find that $a' \wedge a'' \le a'$ where $a' \wedge a''$ could obtain all of the elements in the A. Therefore, (40) could be derived as: $g(\boldsymbol{a}, s) \ge g(\boldsymbol{b}, s)$ holds for $\forall \boldsymbol{a}, \boldsymbol{b} \in A$ and $\boldsymbol{a} \ge \boldsymbol{b}$. In right side of (37), $a' \vee a'' \ge a'$ and $a' \vee a'' \in A$, so we can obtain $g(a' \vee a'', s) \ge g(a'', s)$. Therefore, condition (37) is satisfied, in the same way, we can obtain that (38) is satisfied too. Hence, $g(a, s)$ satisfies the quasi-supermodular property in (a, s), i.e., condition (5) is satisfied.

$$\begin{aligned} &g(a', s) \ge g(a' \wedge a'', s) \\ &\Rightarrow -\left[C(a') + \sum_{i \in N} \theta_i (s_i - \sum_{m \in M} a'_{m,i})\right] \ge \\ &-\left[C(a' \wedge a'') + \sum_{i \in N} \theta_i (s_i - \sum_{m \in M} a'_{m,i} \wedge a''_{m,i})\right] \end{aligned} \tag{39}$$

$$\Rightarrow C(a' \wedge a'') - C(a') \ge \sum_{i \in N} \theta_i \left[\sum_{m \in M} (a'_{m,i} \wedge a''_{m,i} - a'_{m,i})\right] \tag{40}$$

B. Single Crossing Property of Function g(u,x)

According to [50], $g(a, s)$ is quasi-supermodularity if for $a' \in A_{s'}, a'' \in A_{s''}$ where $a' > a''$ and $s' > s''$, conditions (41) and (42) should be satisfied.

$$g(a', s'') > g(a'', s'') \Rightarrow g(a', s') > g(a'', s') \tag{41}$$

$$g(a', s'') \ge g(a'', s'') \Rightarrow g(a', s') \ge g(a'', s') \tag{42}$$

By substituting the function of $g(\cdot, s)$ into $g(a', s'') > g(a'', s'')$, we can obtain (43).

$$\begin{aligned} &g(a', s'') > g(a'', s'') \\ &\Rightarrow -\left[C(a') + \sum_{i \in N} \theta_i (s''_i - \sum_{m \in M} a'_{m,i})\right] > \end{aligned} \tag{43}$$

$$-\left[C(a'') + \sum_{i \in N} \theta_i (s''_i - \sum_{m \in M} a''_{m,i})\right] \tag{44}$$

By adding $-(s' - s'')$ on both sides of (43), the following relationships are obtained

$$-\left[C(a') + \sum_{i \in N} \theta_i (s''_i - \sum_{m \in M} a'_{m,i})\right] - (s' - s'') > \tag{45}$$

$$-\left[C(a'')+\sum_{i\in N}\theta_i(s_i''-\sum_{m\in M}a_{m,i}'')\right]-(s'-s'')$$

$$\Rightarrow -\left[C(a')+\sum_{i\in N}\theta_i(s_i'-\sum_{m\in M}a_{m,i}')\right]> \quad (46)$$

$$-\left[C(a'')+\sum_{i\in N}\theta_i(s_i'-\sum_{m\in M}a_{m,i}'')\right]$$

$$\Rightarrow g(a',s')>g(a'',s') \quad (47)$$

It is obvious that (41) holds, in the same way, (42) holds too. Therefore, $g(a,s)$ has single crossing property in a on A for each s in S, i.e., condition (6) is satisfied.

As discussed above, all conditions are satisfied, hence we can use the monotonicity property of optimal solution set to cut some unpromising child nodes in the RL-SAT.

REFERENCES

[1] L. V. Kantorovich, "Mathematical methods of organizing and planning production," *Manag. Sci.*, vol. 6, no. 4, pp. 366–422, 1960, doi: 10.1287/mnsc.6.4.366.

[2] R. C. E. Gilmore and R. Gomory, "A linear programming approach to the cutting-stock problem," *Oper. Res.*, vol. 9, no. 6, pp. 849–859, Dec. 1961, doi: 10.1287/opre.9.6.849.

[3] P. C. Gilmore and R. E. Gomory, "A linear programming approach to the cutting stock problem—Part II," *Oper. Res.*, vol. 11, no. 6, pp. 863–888, Dec. 1963, doi: 10.1287/opre.11.6.863.

[4] C. Goulimis, "Optimal solutions for the cutting stock problem," *Eur. J. Oper. Res.*, vol. 44, no. 2, pp. 197–208, Jan. 1990. [Online]. Available: https://www.sciencedirect.com/science/article/pii/037722179090355F

[5] J. Wy and B.-I. Kim, "Two-staged guillotine cut, two-dimensional bin packing optimisation with flexible bin size for steel mother plate design," *Int. J. Prod. Res.*, vol. 48, no. 22, pp. 6799–6820, Nov. 2010.

[6] M.-S. Chun, I.-T. Ahn, and Y.-H. Moon, "Deformation behavior of a slab with width reduction in a hot mill," *J. Mater. Eng. Perform.*, vol. 14, no. 3, pp. 408–412, Jun. 2005.

[7] Z. Wang, C. Chen, H. Li, D. Dong, and T. Tarn, "Incremental reinforcement learning with prioritized sweeping for dynamic environments," *IEEE/ASME Trans. Mechatronics*, vol. 24, no. 2, pp. 621–632, Apr. 2019.

[8] Z. Wang, C. Chen, and D. Dong, "Lifelong incremental reinforcement learning with online Bayesian inference," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 8, pp. 4003–4016, Aug. 2022.

[9] K. Li, T. Zhang, R. Wang, Y. Wang, Y. Han, and L. Wang, "Deep reinforcement learning for combinatorial optimization: Covering salesman problems," *IEEE Trans. Cybern.*, vol. 52, no. 12, pp. 13142–13155, Dec. 2022.

[10] B. Lin, B. Ghaddar, and J. Nathwani, "Deep reinforcement learning for the electric vehicle routing problem with time windows," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 8, pp. 11528–11538, Aug. 2022.

[11] L. Tang and Y. Meng, "Data analytics and optimization for smart industry," *Frontiers Eng. Manage.*, vol. 8, no. 2, pp. 157–171, Jun. 2021.

[12] R. Zhang et al., "Learning to solve multiple-TSP with time window and rejections via deep reinforcement learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 1, pp. 1325–1336, Jan. 2023.

[13] R. Chen, W. Li, and H. Yang, "A deep reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for the job-shop scheduling problem," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1322–1331, Feb. 2023.

[14] Z. Pan, L. Wang, C. Dong, and J.-F. Chen, "A knowledge-guided end-to-end optimization framework based on reinforcement learning for flow shop scheduling," *IEEE Trans. Ind. Informat.*, vol. 20, no. 2, pp. 1853–1861, Feb. 2024.

[15] H. Yu, T. Taleb, and J. Zhang, "Deep reinforcement learning based deterministic routing and scheduling for mixed-criticality flows," *IEEE Trans. Ind. Informat.*, vol. 19, no. 8, pp. 8806–8816, Aug. 2023.

[16] V. Mnih, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[17] Z. Wang, N. D. Freitas, and M. Lanctot, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Int. Conf. Mach. Learn.*, vol. 48, M. F. Balcan and K. Q. Weinberger, Eds., New York, NY, USA, Jun. 2016, pp. 1995–2003. [Online]. Available: https://proceedings.mlr.press/v48/wangf16.html

[18] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. 12th Int. Conf. Neural Inf. Process. Syst.* Cambridge, MA, USA: MIT Press, 1999, pp. 1057–1063.

[19] V. Mnih et al., "Asynchronous methods for deep reinforcement learning," in *Proc. 33rd Int. Conf. Int. Conf. Mach. Learn.*, vol. 48, 2016, pp. 1928–1937.

[20] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," in *Proc. ICLR*, 2016, pp. 1–14. [Online]. Available: https://iclr.cc/archive/www/doku.php%3Fid=iclr2016:main.html

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[22] 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

[23] 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.*, 2017, pp. 1–7.

[24] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," in *Proc. 5th Int. Conf. Learn. Represent.*, 2017, pp. 1–15.

[25] 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.* Richland, SC, USA: International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 1386–1394.

[26] 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.*, 2018, pp. 1–11.

[27] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Comput. Oper. Res.*, vol. 134, Jan. 2021, Art. no. 105400.

[28] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *Eur. J. Oper. Res.*, vol. 290, no. 2, pp. 405–421, 2021.

[29] Y. Liu, A. Halev, and X. Liu, "Policy learning with constraints in model-free reinforcement learning: A survey," in *Proc. Thirtieth Int. Joint Conf. Artif. Intell.*, Z.-H. Zhou, Ed., Aug. 2021, pp. 4508–4515, doi: 10.24963/ijcai.2021/614.

[30] M. Nazari, A. Oroojlooy, M. Takáč, and L. V. Snyder, "Reinforcement learning for solving the vehicle routing problem," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates, 2018, pp. 9861–9871.

[31] 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.

[32] Y.-C. Wu, B.-H. Tseng, and C. E. Rasmussen, "Improving sample-efficiency in reinforcement learning for dialogue systems by using trainable-action-mask," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2020, pp. 8024–8028.

[33] Q. Ma, S. Ge, D. He, D. Thaker, and I. Drori, "Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning," in *Proc. AAAI*, 2020, pp. 1–8.

[34] A. Forootani, R. Iervolino, M. Tipaldi, and J. Neilson, "Approximate dynamic programming for stochastic resource allocation problems," *IEEE/CAA J. Autom. Sinica*, vol. 7, no. 4, pp. 975–990, Jul. 2020.

[35] 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, Jul. 2005, pp. 2989–2994.

[36] D. P. de Farias and B. Van Roy, "The linear programming approach to approximate dynamic programming," *Operations Res.*, vol. 51, no. 6, pp. 850–865, Dec. 2003.

[37] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[38] D. Silver et al., "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018, doi: 10.1126/science.aar6404.

[39] A. Prorok, "Robust assignment using redundant robots on transport networks with uncertain travel time," *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 4, pp. 2025–2037, Oct. 2020.
[40] D. M. Topkis, *Supermodularity and Complementarity*. Princeton, NJ, USA: Princeton Univ. Press, 1998.
[41] Gurobi Optimization LLC. (2022). *Gurobi Optimizer Reference Manual*. [Online]. Available: https://www.gurobi.com
[42] H. Zhao, Q. She, C. Zhu, Y. Yang, and K. Xu, "Online 3D bin packing with constrained deep reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, May 2021, vol. 35, no. 1, pp. 741–749.
[43] R. Zhang, A. Prokhorchuk, and J. Dauwels, "Deep reinforcement learning for traveling salesman problem with time windows and rejections," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2020, pp. 1–8.
[44] G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*. Berlin, Germany: Springer, 2018. [Online]. Available: http://gameaibook.org
[45] K. H. Rahi, H. K. Singh, and T. Ray, "Partial evaluation strategies for expensive evolutionary constrained optimization," *IEEE Trans. Evol. Comput.*, vol. 25, no. 6, pp. 1103–1117, Dec. 2021.
[46] G. Evtimov and S. Fidanova, *Ant Colony Optimization Algorithm for 1D Cutting Stock Problem*. Cham, Switzerland: Springer, 2018, pp. 25–31, doi: 10.1007/978-3-319-65530-7_3.
[47] J. Liang et al., "A survey on evolutionary constrained multiobjective optimization," *IEEE Trans. Evol. Comput.*, vol. 27, no. 2, pp. 201–221, Apr. 2023.
[48] N. Dupin, R. Parize, and E.-G. Talbi, "Matheuristics and column generation for a basic technician routing problem," *Algorithms*, vol. 14, no. 11, p. 313, Oct. 2021.
[49] P. Milgrom and C. Shannon, "Monotone comparative statics," *Econometrica*, vol. 62, no. 1, pp. 157–180, Jan. 1994.
[50] K. Kultti and H. Salonen, "Iterated dominance in quasisupermodular games with strict single crossing property," *Int. J. Game Theory*, vol. 27, no. 2, pp. 305–309, Aug. 1998.

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 Ph.D. degree from Northeastern University in 2011. She is currently a Professor with the National Frontiers Science Center for Industrial Intelligence and Systems Optimization. She serves as the Vice Director for the Key Laboratory of Data Analytics and Optimization for Smart Industry (Northeastern University), Ministry of Education, China. Her research interests include evolutionary learning and reinforcement learning, integer optimization and combinatorial optimization, and computational intelligent optimization.

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, 1996, respectively.

He is a member of Chinese Academy of Engineering; the Director of the Key Laboratory of Data Analytics and Optimization for Smart Industry, Ministry of Education, China; and the Director and the Chair Professor of National Frontiers Science Center for Industrial Intelligence and Systems Optimization, Northeastern University. 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 include 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. 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.

Dr. Tang's paper published on *IISE Transactions* received the Best Applications Paper Award in 2017. He serves as an Associate Editor for 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*. He serves as an Area Editor for *Asia-Pacific Journal of Operational Research*.