Joint optimization of steel plate shuffling and truck loading sequencing based on deep reinforcement learning

(article starts on next page)

Contents lists available at ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei

Full length article

Joint optimization of steel plate shuffling and truck loading sequencing based on deep reinforcement learning

Zhezhuang Xu [a], Jinlong Wang [a], Meng Yuan [a,*], Yazhou Yuan [b], Boyu Chen [c], Qingdong Zhang [c], Cailian Chen [d], Xinping Guan [d]

[a] College of Electrical Engineering and Automation, Fuzhou University, Fuzhou, 350108, Fujian, China
[b] School of Electrical Engineering, Yanshan University, Qinhuangdao, 066004, Hebei, China
[c] Sansteel Minguang Co., Ltd., Sanming, 365000, Fujian, China
[d] Department of Automation, Shanghai Jiao Tong University, Shanghai, 200240, China

ARTICLE INFO

Keywords:
Steel plate shuffling
Truck loading sequencing
Optimization
Deep reinforcement learning
Industrial Internet of Things

ABSTRACT

Steel plate is one of the most valuable steel products which is highly customized in specification according to the demands of users. In this case, the outbound scheduling of steel plates is a challenging issue since its efficiency and complexity are impacted by both steel plate shuffling and truck loading sequencing. To overcome this challenge, we propose to jointly optimize steel plate shuffling and truck loading sequencing (SPS-TLS) by utilizing the data of steel plates and trucks collected by Industrial Internet of Things (IIoT). The SPS-TLS problem is firstly transformed as an orders scheduling problem which is formulated as a mixed-integer linear programming (MILP) model. Then an alternating iteration algorithm based on deep reinforcement learning (AltDRL) is proposed to solve the SPS-TLS problem. In AltDRL, the deep Q network (DQN) with prioritized experience replay (PER) and the heuristic algorithm are combined to iteratively obtain the near-optimal shuffling position of blocking plates and truck sequence. Experiments are executed based on data collected from a real steel logistics park. The results confirm that AltDRL can significantly reduce the number of plate shuffles and improve the outbound scheduling efficiency of steel plates.

1. Introduction

Steel plate is one of the most valuable steel products which is highly customized in specification according to the demands of users [1,2]. In this case, each target steel plate has to be precisely retrieved during outbound scheduling. Traditionally, the retrieval of target steel plates is manually scheduled, which is inefficient since it is too complex to be efficiently scheduled by humans. Therefore, it is important to develop an automatic algorithm to improve the outbound scheduling efficiency of steel plates.

Initially, all steel plates are distributed across multiple stacks and need to be retrieved within several days. Each steel plate is assigned an identifier representing its outbound date. During the retrieval process, if the plate to be retrieved next is not located at the top of a stack, all blocking plates above it must be shuffled to other stacks [3]. To achieve high scheduling efficiency, determining the proper destination stacks for blocking plates is essential. Otherwise, more shuffles will be generated directly or indirectly afterward [4]. Such shuffling operations significantly impact the efficiency of steel plate retrieval, consequently reducing the throughput of the storage yard. For this reason, some strategies have been developed to improve the retrieval efficiency of target steel plates [5–7]. The primary goal of these efforts is to complete the retrieval tasks with the least shuffling cost, given the specified retrieval sequence.

However, few existing studies have taken into account the impact of the truck loading sequence on the retrieval efficiency of steel plates. In fact, the truck loading sequence determines the retrieval order of target steel plates in stacks. Redundant shuffling will increase if the plate retrieved earlier is stored beneath the plate retrieved later. Such shuffles cause delays to the entire outbound scheduling process. Therefore, the truck loading sequence must be considered in the retrieval process of steel plates. Fig. 1 shows the scheduling of steel plate shuffling and truck loading sequencing in the retrieval process of steel plates.

To cope with these problems, in this paper, we propose to jointly optimize the steel plate shuffling and truck loading sequencing (SPS-TLS) based on the data of plates and trucks collected by Industrial Internet of Things (IIoT) [8,9]. The SPS-TLS problem is first transformed into an order scheduling problem with the goal of minimizing the number of plate shuffles during outbound scheduling for the day

[*] Corresponding author.
E-mail address: meng.yuan@fzu.edu.cn (M. Yuan).

https://doi.org/10.1016/j.aei.2024.102392
Received 8 September 2023; Received in revised form 25 December 2023; Accepted 26 January 2024
Available online 1 February 2024

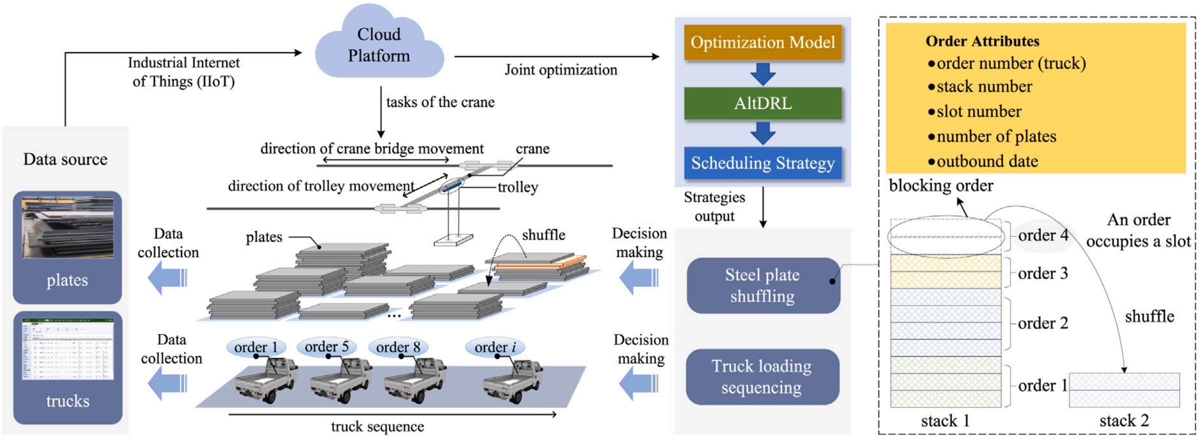


Fig. 1. Scheduling of steel plate shuffling and truck loading sequencing in the retrieval process of steel plates.

and the minimum number of plate shuffles during the forthcoming outbound process, where the second objective is defined by the number of blocking plates after retrieving all outbound orders of the day [10]. Then, an alternating iteration algorithm based on deep reinforcement learning (AltDRL) is designed to solve the SPS-TLS problem. In AltDRL, the deep Q network (DQN) with prioritized experience replay and a heuristic algorithm are combined to iteratively obtain the near-optimal shuffling position of blocking plates and truck sequence.

Specifically, this paper has the following contributions:

1. The steel plate retrieval problem is proposed by jointly optimizing the steel plate shuffling and truck loading sequencing (SPS-TLS) to minimize both the number of plate shuffles during the outbound scheduling for the day and the minimum number of plate shuffles during the forthcoming outbound process. By solving the problem, the optimal shuffling positions of blocking plates and truck sequence can be obtained.
2. The AltDRL algorithm is developed to iteratively solve the SPS-TLS problem, where the retrieval sequence of steel plates is determined by the heuristic algorithm, and the shuffling positions of blocking plates are determined by the DQN with prioritized experience replay (DQN-PER). Two types of features are extracted in AltDRL algorithm to describe the evolution of the state, and four heuristic rules are designed to act as the action space of the DQN agent.
3. Experiments are conducted to verify the effectiveness of the proposed scheme, utilizing data collected from a real steel logistics park. Based on the numerical experiments, the proposed method exhibits better convergence property compared to other benchmarking algorithms in scheduling. Furthermore, the joint optimization of steel plate shuffling and truck loading sequencing significantly reduces the number of plate shuffles and improves the retrieval efficiency of steel plates when compared to the optimization of only plate shuffling under fixed truck loading sequencing.

The rest of this paper is organized as follows. Section 2 provides an introduction about related works. Section 3 introduces the scheduling platform and utilizes the data collected by the IIoT to formulate the optimization problem of SPS-TLS. Section 4 presents the background of DQN and PER. The implementation details of the proposed AltDRL algorithm are shown in Section 5. Section 6 discusses the experimental results, and the conclusion is given in Section 7.

2. Related works

Issues related to item retrieval are widespread in steel plate yards and container terminals. Within these areas, items are piled up on the stacks to enhance space utilization. If the item to be retrieved is not positioned at the top of the stack, the shuffling process must be initiated [11,12]. Therefore, the retrieval problem becomes one of the most critical factors affecting the efficiency of yard operations.

2.1. Steel plate retrieval problem

Some research efforts have already considered the retrieval problem of steel plates. Tang et al. [5] studied the retrieval problem of steel plates, with the aim of selecting suitable plates in the storage yard under a given rolling plan to minimize the total shuffling cost. In [7], Tang et al. formulated the steel plate retrieval problem as a slab stack shuffling (SSS) problem and developed a modified genetic algorithm (GA) to solve it. In [13], Tang and Ren developed the dynamic programming approach and segmented dynamic programming-based heuristic to respectively solve small-scale and real-scale SSS problems. In [14], Shi and Liu developed a very large-scale neighborhood (VLSN) search algorithm to iteratively solve the steel hot rolling scheduling problem considering SSS. Rajabi et al. [15] considered two different variants of the SSS problem and proposed a new integer programming (IP) model for each variant. Wu et al. [3] introduced a stack-based retrieval method to minimize the unproductive relocation number according to a specific retrieval stack sequence. For other steel plate retrieval problems, Zhao and Tang [6] presented the heavy plate shuffling problem aiming at minimizing the total shuffling number and the total crane traveling distance. Zhao et al. [16] studied a multiple double-load crane scheduling problem. A two-phase model-based heuristic was developed to minimize the makespan in their work. Bruno et al. [17] proposed a bi-objective mathematical model aiming to minimize the number of shuffles and expired slabs simultaneously in a generic cutting/assembly center.

2.2. Container retrieval problem

Other studies on shuffling reduction have focused on container retrieval problem [18–22]. Petering and Hussein [19] introduced a new mathematical formulation of the block relocation problem with fewer decision variables, and developed a new look-ahead algorithm for addressing the problem. In [20], Tanaka and Mizuno proposed an exact algorithm to solve the unrestricted block relocation problem with distinct priorities. In [21], Zhu et al. developed an iterative deepening A* algorithms for larger instances of the container relocation problem. Zhang et al. [22] proposed machine learning-driven algorithms by integrating optimization methods and machine learning techniques, to solve the container relocation problem. Boysen and Emde [23] proposed a parallel stack loading problem (PSLP) aiming to reduce shuffling during the outbound process by minimizing the number of blocking items in the container storage phase. Furthermore, Boge and Knust [10] proposed a simulated annealing (SA) algorithm to address the same parallel stack loading problem. However, these studies are based on the premise that the truck loading sequence is known, i.e., each container has a unique priority.

Literature [24–27] investigated container retrieval problems in scenarios where the retrieval sequence is partially known. In [24], Azab and Morita introduced the block relocation problem with appointment scheduling, in which the truck loading sequence is partially adjustable. A similar problem was studied in [25], and 5 heuristic algorithms were proposed to solve the problem. Galle et al. [26] investigated the stochastic container relocation problem, and developed an optimal algorithm called Pruning-Best-First-Search (PBFS) to solve small-scale instances. Zweers et al. [27] proposed a new optimization model where the containers can be moved in both pre-processing phase and relocation phase. A branch-and-bound (B&B) algorithm was developed to address the problem.

2.3. Difference with existing works

Table 1 summarizes the main attributes of the retrieval problems in the literature. The research in this paper differs from the existing works in three aspects. Firstly, in this study, the steel plate shuffling and truck loading sequencing are jointly optimized. This is crucial because the outbound efficiency of steel plates is significantly influenced by both factors. Secondly, we simultaneously minimize the number of plate shuffles during the outbound scheduling for the day and the minimum number of plate shuffles during the forthcoming outbound process in the SPS-TLS problem, which has not been previously investigated. However, it is necessary because the remaining steel plates will be retrieved in the coming days, and the efficiency of retrieval in forthcoming outbound scheduling cannot be ignored [10,23]. Thirdly, the AltDRL algorithm is designed based on deep reinforcement learning (DRL) to solve the SPS-TLS problem iteratively.

Note that a large number of outbound orders are retrieved each day, and the number of combinations for truck loading sequences is factorial. This complexity renders the search space of exact algorithms extremely large, even leading to their ineffectiveness. Heuristic algorithms appear to be capable of providing scheduling results in a very short time. However, the design of an effective heuristic significantly relies on experts' experience. Faced with complex scheduling scenarios, relying solely on heuristic algorithms makes it challenging to ensure the quality of the solution. In contrast, the DRL excels in handling scheduling problems with complex environments and a large number of states [28]. It interacts with the environment in the offline phase, learns the best action at each decision point, and can obtain high-quality solutions without exploring the entire search space. For the SPS-TLS problem, when the plates to be retrieved at each scheduling point is determined, DRL utilizes global information to select the best target stack for the blocking plates, thereby ensuring a globally near-optimal solution. Therefore, the AltDRL algorithm is proposed in this paper based on DRL. The details of our works are provided in the following sections.

3. Problem description and model formulation

This section introduces the platform for steel plates retrieval scheduling. Then, the joint optimization problem of steel plate shuffling and truck loading sequencing (SPS-TLS) is investigated. Specifically, we formulate an integer programming model that aims to minimize both the number of plate shuffles during the daily outbound scheduling and the minimum number of plate shuffles during the forthcoming outbound process, considering the available data of plates and trucks. By solving the SPS-TLS problem, the optimal truck sequence and shuffling positions of blocking plates can be obtained.

3.1. Problem description

The plate yard is a crucial hub for the steel company to trade products with customers, which consists of multiple regular stacks for storing finished steel plates. Due to the large variety and quantity of steel plates and the limited number of stacks, plates of similar size are usually stacked on top of another at each stack. Meanwhile, in order to ensure retrieval efficiency and reduce truck waiting time during the outbound scheduling of steel plates, plates belonging to the same order are often placed together. Based on this real stacking distribution, in this paper, we treat an order as a scheduling unit, which means that if an order is moved, all the steel plates of the order must be moved to the same position.

It is important to note that the time for different trucks to arrive at the yard is uncertain in the real scheduling scenarios, and as mentioned before, this may cause a surge in shuffling. For example, in Fig. 1, if both orders 2 and 3 are the outbound orders, and the retrieval sequencing of order 2 precedes that of order 3, then orders 3 and 4 must be shuffled. On the other hand, if the position after the shuffling is not reasonable, these blocking orders will inevitably cause further reshuffling in the future. Repeated shuffling can greatly lower the scheduling efficiency and generate a huge waste of energy for the crane responsible for plate lifting.

For the above situation, in this paper, rather than passively waiting for the real-time arrival of trucks for solving the plate shuffling problem, we proactively provide the most suitable loading time windows for different trucks and positions for blocking orders in advance based on known truck loading information and plate distribution so as to avoid unnecessary truck waiting and plate shuffling.

3.2. Platform for steel plates retrieval scheduling

The cloud platform plays a crucial role in offering comprehensive solutions and services to companies and customers by effectively integrating resources in the industrial manufacturing domain [29]. This integration forms the basis for jointly optimizing the steel plate shuffling and truck loading sequencing. As shown in Fig. 1, the data of plates and trucks is first uploaded to the platform in advance through the IIoT. Leveraging the data, a joint scheduling strategy is devised to allocate loading time windows for different trucks and optimize the storage positions for blocking plates, which constitutes a key aspect of our work. Subsequently, the cloud platform deploys the optimized strategy to the steel plate yard. The outcome of this joint scheduling approach generates a task list for the cranes to efficiently move the steel plates within the yard, while minimizing energy consumption. Consequently, the joint optimization of plate shuffling and truck loading sequencing is vital in reducing waiting time for customers and operation cost for the cranes.

3.3. General assumptions

Before formulating the mathematical model, two general assumptions related to the SPS-TLS problem are made as follows.

(1) Blocking orders can be shuffled into any of the considered stacks. This is because all the stacks in the yard can be divided into different sets according to the specifications of the plates, and different sets are independent of each other. Our work is only for one set, but the methodology is applicable to other stack sets as well.

(2) Orders are shuffled if and only if any outbound order below them is to be retrieved. Adopting the assumption, the movement of non-blocking orders can be avoided, thereby decreasing the search range of the solution space. On the other hand, shuffling the non-blocking orders belongs to a pre-marshalling scope, which is beyond the intent of our work.

(3) The crane handles one steel plate at a time. In practical operations, magnetic overhead cranes are primarily used for handling steel plates, and the crane typically lifts only one steel plate at a time for safety.

Table 1
Main attributes for the retrieval problems in the literature.

Reference	Retrieve sequence			Retrieval in forthcoming outbound scheduling	Method
	Known	Partially known	Unknown		
Tang et al. [5]	✓				heuristic
Tang et al. [7]	✓				GA
Shi and Liu [14]	✓				VLSN
Rajabi et al. [15]	✓				IP
Petering and Hussein [19]	✓				LA-N
Boysen and Emde [23]	✓			✓	IP
Boge and Knust [10]	✓			✓	SA
Azab and Morita [24]		✓			IP
Galle et al. [26]		✓			PBFS
Zweers et al. [27]		✓			B&B
This paper			✓	✓	AltDRL

3.4. Notations definition for optimization model

To formulate the model, we extract five main attributes for each order, namely order number, stack number, slot number, number of plates and outbound date, respectively, as shown in Fig. 1. Orders are encoded as integers in this paper, and the slots where orders are stored are numbered incrementally from bottom to top in each stack. Some parameters and variables are defined as follows.

(1) Parameters:

$\mathcal{I}$: Set of all orders for steel plates stacked in the yard, where $\mathcal{I} = \{1, \ldots |\mathcal{I}|\}$, $|\mathcal{I}|$ is the total number of orders.

$\mathcal{I}_1$: Set of orders for outbound steel plates (outbound orders), where $\mathcal{I}_1 \subseteq \mathcal{I}$.

$\mathcal{I}_2$: Set of orders for steel plates that are not in the outbound plan (non-outbound orders), where $\mathcal{I}_2 \subseteq \mathcal{I}$.

$\mathcal{T}$: Set of stages, where $\mathcal{T} = \{1, \ldots, T\}$, $T = |\mathcal{I}_1| + 1$. For the first $|\mathcal{I}_1|$ stages, denoted as $\mathcal{T} \setminus \{T\}$, each stage is utilized to retrieve an outbound order, while the last stage, T, is employed to track the final stacking state.

$\mathcal{S}$: Set of stacks, where $\mathcal{S} = \{1, \ldots, S\}$, S is the total number of stacks.

$\mathcal{K}$: Set of slots in each stack with the index increasing from the bottom to the top, where $\mathcal{K} = \{1, \ldots, K\}$, K is the maximum height of the stack.

$\mathcal{V}$: Set of intervals, where $\mathcal{V} = \{1, \ldots \sigma\}$. One stage consists of σ intervals.

i, j : Index of the order, $i, j \in \mathcal{I}$.

k, l : Index of the slot in each stack, $k, l \in \mathcal{K}$.

ℓ : Index of the stack, $\ell \in \mathcal{S}$.

t : Index of the stage, $t \in \mathcal{T}$.

τ : Index of the interval, $\tau \in \mathcal{V}$.

m_i : Number of plates for order i.

d_i : Outbound date for order i, coded as an integer.

$\tilde{y}_{i\ell k}$: 1, if order i occupies slot k of stack ℓ in the initial stack layout; 0, otherwise.

d_{max} : Maximum outbound date for all orders in the yard.

(2) Decision Variables:

$x^t_{i\tau}$ 1, if order i is retrieved at interval τ of stage t; 0, otherwise.

$y^t_{i\ell k\tau}$ 1, if order i occupies the slot k of stack ℓ at interval τ of stage t; 0, otherwise.

$z^t_{i\ell k\tau}$ 1, if order i is moved from the slot k of stack ℓ at interval τ of stage t; 0, otherwise.

$u^t_{i\ell k\tau}$ 1, if order i is moved to the slot k of stack ℓ at interval τ of stage t; 0, otherwise.

$w_{\ell kl}$ 1, if slots k and l of stack ℓ are occupied by orders and the outbound date of the order in slot k is later than that of the order in slot l; 0, otherwise.

$\vartheta_{\ell k}$ 1, if the order in slot k of stack ℓ has to be shuffled during the process of future order retrieval; 0, otherwise.

$o_{i\ell k}$ 1, if the order i occupying slot k of the stack ℓ is a blocking order; 0, otherwise.

3.5. Mathematical model for SPS-TLS problem

The objective of the model considered in this paper contains two functions, as shown in expression (1).

$$\mathbf{P1} : \min f_1 + f_2 \tag{1}$$

where

$$f_1 = \sum_{t \in \mathcal{T} \setminus \{T\}} \sum_{i \in \mathcal{I}} \sum_{\ell \in \mathcal{S}} \sum_{k \in \mathcal{K}} \sum_{\tau \in \mathcal{V}} m_i u^t_{i\ell k\tau} \tag{2}$$

$$f_2 = \sum_{i \in \mathcal{I}} \sum_{\ell \in \mathcal{S}} \sum_{k \in \mathcal{K}} m_i o_{i\ell k} \tag{3}$$

The first function f_1 represents the number of plate shuffles during the orders outbound process, and the second function f_2 represents the total number of blocking plates after all outbound orders have been retrieved. The shuffles in f_1 will directly impact the outbound plates scheduling efficiency and the queue waiting time of trucks. Minimizing f_1 will effectively avoid yard congestion, thereby improving customer satisfaction. For the function f_2, literature [10,23] have shown that minimizing the total number of blocking items can fully improve the efficiency of outbound scheduling. These blocking items will be shuffled at least once, thus in this paper, we call f_2 the minimum number of plate shuffles during the forthcoming outbound process.

For the model constraints, we describe them in parts according to different functions.

(1) Order handling constraints:

$$\sum_{t \in \mathcal{T} \setminus \{T\}} \sum_{\tau \in \mathcal{V}} x^t_{i\tau} = 1, \forall i \in \mathcal{I}_1 \tag{4}$$

$$\sum_{i \in \mathcal{I}_1} \sum_{\tau \in \mathcal{V}} x^t_{i\tau} = 1, \forall t \in \mathcal{T} \setminus \{T\} \tag{5}$$

$$\sum_{t \in \mathcal{T} \setminus \{T\}} \sum_{\tau \in \mathcal{V}} x^t_{i\tau} \leq 0, \forall i \in \mathcal{I}_2 \tag{6}$$

$$\sum_{i \in \mathcal{I}_1} x^t_{i\tau} + \sum_{i \in \mathcal{I}} \sum_{\ell \in \mathcal{S}} \sum_{k \in \mathcal{K}} u^t_{i\ell k\tau} \leq 1, \forall t \in \mathcal{T} \setminus \{T\}, \forall \tau \in \mathcal{V} \tag{7}$$

$$\sum_{k \in \mathcal{K}} z^t_{i\ell k\tau} + \sum_{k \in \mathcal{K}} u^t_{i\ell k\tau} \leq 1, \forall t \in \mathcal{T} \setminus \{T\}, \forall i \in \mathcal{I}, \forall \ell \in \mathcal{S}, \forall \tau \in \mathcal{V} \tag{8}$$

$$\sum_{\ell \in \mathcal{S}} \sum_{k \in \mathcal{K}} z^t_{i\ell k\tau} = x^t_{i\tau} + \sum_{\ell \in \mathcal{S}} \sum_{k \in \mathcal{K}} u^t_{i\ell k\tau}, \forall t \in \mathcal{T} \setminus \{T\}, \forall i \in \mathcal{I}, \forall \tau \in \mathcal{V} \tag{9}$$

$$z^t_{i\ell k\tau} \leq y^t_{i\ell k\tau}, \forall t \in \mathcal{T} \setminus \{T\}, \forall i \in \mathcal{I}, \forall \ell \in \mathcal{S}, \forall k \in \mathcal{K}, \forall \tau \in \mathcal{V} \tag{10}$$

$$\sum_{i \in \mathcal{I}} z^t_{i\ell k\tau} \leq \sum_{i \in \mathcal{I}} \left(y^t_{i\ell k\tau} - y^t_{i\ell,k+1,\tau} \right), \forall t \in \mathcal{T} \setminus \{T\}, \forall \ell \in \mathcal{S}, \forall k \in \mathcal{K} \setminus \{\mathcal{K}\}, \forall \tau \in \mathcal{V} \tag{11}$$

$$\varphi^t_{\tau_1} \geq \varphi^t_{\tau_2} - S\left(2 - \theta^t_{\tau_1} - \theta^t_{\tau_2}\right), \forall t \in \mathcal{T} \setminus \{T\}, \forall \tau_1 \in \mathcal{V}, \forall \tau_2 \in \mathcal{V} \tag{12}$$

$$\varphi^t_{\tau_1} \leq \varphi^t_{\tau_2} + S\left(2 - \theta^t_{\tau_1} - \theta^t_{\tau_2}\right), \forall t \in \mathcal{T} \setminus \{T\}, \forall \tau_1 \in \mathcal{V}, \forall \tau_2 \in \mathcal{V} \tag{13}$$

$$\theta^t_{\tau_1} \leq 1 - \sum_{i \in \mathcal{I}_1} x^t_{i\tau}, \forall t \in \mathcal{T} \setminus \{T\}, \forall \tau \in \mathcal{V} \setminus \{\sigma\}, \tau_1 = \{\tau + 1, \ldots \sigma\} \tag{14}$$

Constraints (4)–(14) describe the critical properties of the retrieval and shuffle within each stage. Meanwhile, correlating each operation with the divided interval enhances the readability of the model and facilitates the tracking of solutions. In this group of constraints, constraints (4) and (5) respectively state that outbound orders must be retrieved within the considered stages and that only one order can be retrieved at one stage. Constraint (6) indicates that non-outbound orders will not be retrieved at any interval of any stage. Constraint (7) specifies that only one order can be shuffled or retrieved at any interval. Constraint (8) clarifies the relationship between decision variables $z^t_{i\ell k\tau}$ and $u^t_{i\ell k\tau}$, indicating that within the same time interval of a given stage, an order cannot be simultaneously moved into and moved out of the same slot. The options of order movement are illustrated in constraint (9), which means that once the movement occurs, the corresponding order is either retrieved or moved to a specific slot. Constraint (10) reveals the relationship between decision variables $z^t_{i\ell k\tau}$ and $y^t_{i\ell k\tau}$. Constraint (11) emphasizes the movement sequence of orders, that is, an order cannot be moved from its slot unless the slot above it is empty. Constraints (12)–(14) are a set of vital constraints in this paper, where $\varphi^t_\tau = \sum_{i\in\mathcal{I}}\sum_{\ell\in\mathcal{S}}\sum_{k\in\mathcal{K}}\ell z^t_{i\ell k\tau}$ and $\theta^t_\tau = \sum_{i\in\mathcal{I}}\sum_{\ell\in\mathcal{S}}\sum_{k\in\mathcal{K}} z^t_{i\ell k\tau}$. They emphasize that only one stack can be operated at a stage and that once the outbound order is retrieved, no operation will be performed at that stage.

(2) Updating constraints regarding stack configuration:

$$y^1_{i\ell k1} = \tilde{y}_{i\ell k}, \forall i \in \mathcal{I}, \forall \ell \in \mathcal{S}, \forall k \in \mathcal{K} \tag{15}$$

$$y^t_{i\ell k,\tau+1} = y^t_{i\ell k\tau} + u^t_{i\ell k\tau} - z^t_{i\ell k\tau}, \forall t \in \mathcal{T}\setminus\{T\}, \forall i \in \mathcal{I}, \forall \ell \in \mathcal{S}, \forall k \in \mathcal{K}, \forall \tau \in \mathcal{V}\setminus\{\sigma\} \tag{16}$$

$$y^{t+1}_{i\ell k1} = y^t_{i\ell k\sigma} + u^t_{i\ell k\sigma} - z^t_{i\ell k\sigma}, \forall t \in \mathcal{T}\setminus\{T\}, \forall i \in \mathcal{I}, \forall \ell \in \mathcal{S}, \forall k \in \mathcal{K} \tag{17}$$

$$\sum_{i\in\mathcal{I}} y^t_{i\ell k\tau} \le 1, \forall t \in \mathcal{T}, \forall \ell \in \mathcal{S}, \forall k \in \mathcal{K}, \forall \tau \in \mathcal{V} \tag{18}$$

$$\sum_{i\in\mathcal{I}} y^t_{i\ell k\tau} - \sum_{i\in\mathcal{I}} y^t_{i\ell,k+1,\tau} \ge 0, \forall t \in \mathcal{T}, \forall \ell \in \mathcal{S}, \forall k \in \mathcal{K}\setminus\{K\}, \forall \tau \in \mathcal{V} \tag{19}$$

$$\sum_{\ell\in\mathcal{S}}\sum_{k\in\mathcal{K}}\sum_{\tau_1=\{\tau+1,\ldots\sigma\}} y^t_{i\ell k\tau_1} + \sum_{\ell\in\mathcal{S}}\sum_{k\in\mathcal{K}}\sum_{\tau_1\in\mathcal{V}}\sum_{t_1=\{t+1,\ldots,T\}} y^{t_1}_{i\ell k\tau_1} \le \sigma T\left(1 - x^t_{i\tau}\right), \forall t \in \mathcal{T}\setminus\{T\}, \forall i \in \mathcal{I}, \forall \tau \in \mathcal{V} \tag{20}$$

$$\sum_{i\in\mathcal{I}}\sum_{k\in\mathcal{K}} m_i y^t_{i\ell k\tau} \le K, \forall t \in \mathcal{T}, \forall \ell \in \mathcal{S}, \forall \tau \in \mathcal{V} \tag{21}$$

The purpose of the proposed model is to determine the retrieval sequencing of the outbound orders and the shuffled positions of the blocking orders. However, the movement dynamically changes the stack configuration, and it is therefore essential to track these changes accurately. Following the model in [30], the transformations of the stack configuration can be expressed as constraints (15)–(21). In constraint (15), the stack configuration is initialized utilizing the parameter $\tilde{y}_{i\ell k}$, and the configuration status of adjacent intervals and adjacent stages can be updated by constraints (16) and (17), respectively. Constraint (18) indicates that a slot can only hold one order. Constraint (19) ensures that orders cannot be suspended. Constraint (20) ensures that the order cannot appear in the stack once it is retrieved. Constraint (21) forces the height of the stack not to exceed the maximum limit.

(3) Blocking degree constraints:

$$\sum_{i\in\mathcal{I}} d_i y^T_{i\ell l1} - \sum_{i\in\mathcal{I}} d_i y^T_{i\ell k1} \le -1 + \left(d_{max}+1\right)\left(1 - w_{\ell kl}\right), \forall \ell \in \mathcal{S}, \forall k \in \mathcal{K}\setminus\{1\}, l = \{1,\ldots,k-1\} \tag{22}$$

$$\sum_{i\in\mathcal{I}} d_i y^T_{i\ell l1} - \sum_{i\in\mathcal{I}} d_i y^T_{i\ell k1} \ge -w_{\ell kl}\left(d_{max}+1\right), \forall \ell \in \mathcal{S}, \forall k \in \mathcal{K}\setminus\{1\}, l = \{1,\ldots,k-1\} \tag{23}$$

$$\vartheta_{\ell k} \le \sum_{l\in\{1,\ldots,k-1\}} w_{\ell kl}, \forall \ell \in \mathcal{S}, \forall k \in \mathcal{K}\setminus\{1\} \tag{24}$$

$$\vartheta_{\ell k} \ge \sum_{l\in\{1,\ldots,k-1\}} w_{\ell kl}\Big/\sigma, \forall \ell \in \mathcal{S}, \forall k \in \mathcal{K}\setminus\{1\} \tag{25}$$

$$o_{i\ell k} \le y^T_{i\ell k1}, \forall i \in \mathcal{I}, \forall \ell \in \mathcal{S}, \forall k \in \mathcal{K}\setminus\{1\} \tag{26}$$

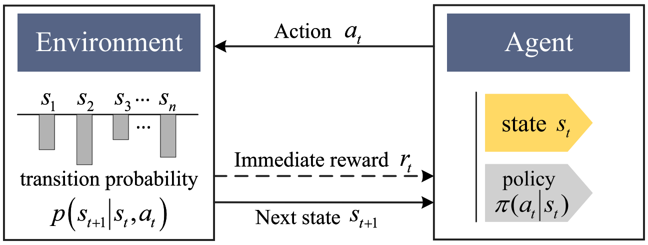


Fig. 2. Schematic of reinforcement learning.

$$o_{i\ell k} \le \vartheta_{\ell k}, \forall i \in \mathcal{I}, \forall \ell \in \mathcal{S}, \forall k \in \mathcal{K}\setminus\{1\} \tag{27}$$

$$o_{i\ell k} \ge y^T_{i\ell k1} + \vartheta_{\ell k} - 1, \forall i \in \mathcal{I}, \forall \ell \in \mathcal{S}, \forall k \in \mathcal{K}\setminus\{1\} \tag{28}$$

$$x^t_{i\tau}, y^t_{i\ell k\tau}, z^t_{i\ell k\tau}, u^t_{i\ell k\tau} \in \{0,1\}, \forall t \in \mathcal{T}, \forall i \in \mathcal{I}, \forall \ell \in \mathcal{S}, \forall k \in \mathcal{K}, \forall \tau \in \mathcal{V} \tag{29}$$

$$w_{\ell kl}, \vartheta_{\ell k}, o_{i\ell k} \in \{0,1\}, \forall i \in \mathcal{I}, \forall \ell \in \mathcal{S}, \forall k \in \mathcal{K}\setminus\{1\}, l = \{1,\ldots,k-1\} \tag{30}$$

To reduce the number of future shuffles, constraints (22)–(30) are used to formulate the degree of stacking blocking in the final stage. Constraints (22) and (23) use the decision variable $w_{\ell kl}$ to count how many orders are blocked by the order in slot k of stack s. Constraints (24) and (25) define the decision variable $\vartheta_{\ell k}$ using $w_{\ell kl}$. Constraints (26)–(28) are converted from $o_{i\ell k} = y^T_{i\ell k1}\vartheta_{\ell k}$, which reveals the relationship between orders and blocking slots, so that, in objective f_2, the minimum number of plate shuffles is minimized during the forthcoming outbound process. Constraints (29) and (30) determine the domain of decision variables.

The model **P1** is an NP-hard problem. Therefore, in order to improve the computation efficiency and obtain a high-quality steel plate outbound scheduling scheme, an alternating iteration algorithm based on deep reinforcement learning is proposed.

4. Background of deep Q network with prioritized experience replay

In this section, the background of deep Q network [31] with prioritized experience replay (DQN-PER) is described to provide a theoretical basis for solving the SPS-TLS problem.

4.1. Reinforcement learning and deep Q network

Reinforcement learning (RL) guides the behavior of the intelligent agent through the rewards gained by the agent interacting with the environment [28]. As shown in Fig. 2, at each decision point t, the agent observes the current state s_t, chooses and performs an action a_t according to a specific policy $\pi\left(a_t\,|s_t\right)$, obtains an immediate reward r_t, after which it gets into the next state s_{t+1} with transition probability $p\left(s_{t+1}\,|s_t, a_t\right)$ and so on until the terminal state.

The purpose of RL is to find the optimal strategy π^* which maximizes the expected sum of long-term rewards in Markov Decision Process, as defined in Eq. (31),

$$\begin{aligned}\pi^* &= \arg\max_\pi Q_\pi(s,a)\\ &= \arg\max_\pi \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots |s,a,\pi\right]\end{aligned} \tag{31}$$

where $Q_\pi(s,a)$ is action value function or Q-value, used to evaluate the merit of the agent executing action a in state s. $\gamma \in (0,1]$ is the discount factor, which reflects the relative importance of the long-term rewards. A larger γ indicates that the agent is more focused on the long-term rewards.

The Deep Q Network (DQN) [31], as an efficient RL algorithm, tackles complex Markov decision problems by employing deep neural networks to fit the Q-function, taking high-dimensional state features as inputs and Q values of state–action pairs as outputs. To reduce high variance of neural network parameter updating, the experience replay mechanism is introduced in DQN. At first, an experience pool D with capacity N is established to store the transition sequence $\left(s_t, a_t, r_t, s_{t+1}\right)$ at each time-step. As training progresses, once the experience pool is full, the earliest stored experience sequence is gradually replaced by the latest generated experience, and the capacity of D remains constant. Based on the minibatch experiences randomly selected from the experience pool, the parameters of the neural network are updated by calculating the loss function at each training step, which can be expressed as Eq. (32),

$$L(\theta) = \mathbb{E}\left[\left(r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'; \theta^-) - Q(s_t, a_t; \theta)\right)^2\right] \tag{32}$$

where Q is the online network, and $\hat{Q}$ is the target network. The Q value generated by $\hat{Q}$ plus the real observed reward r_t constitute the target value. θ^- and θ are respectively the parameters of the target network $\hat{Q}$ and the online network Q.

4.2. Deep Q network with prioritized experience replay

Although experience replay can eliminate the correlation of training samples, undifferentiated random sampling causes DQN to not prioritize high-value experience samples, thereby resulting in relatively slow algorithm convergence. Therefore, prioritized experience replay (PER) mechanism is developed in [32], and the value of temporal difference error ($TD-error$) is used as evaluation of sample importance, which can be defined as Eq. (33),

$$TD-error = \left|r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'; \theta^-) - Q(s_t, a_t; \theta)\right| \tag{33}$$

To prevent the neural network from overfitting, it is necessary to ensure that the probability of the lowest priority experience sequence being sampled is also non-zero. The sampling probability of experience sample i is defined as Eq. (34),

$$P(i) = p_i^{\alpha} \Big/ \sum_i p_i^{\alpha} \tag{34}$$

where p_i is the priority of experience sample; α represents how much prioritization is used. In Eq. (34), p_i is calculated according to the proportional prioritization method, which can be expressed as $p_i = TD-error + \epsilon$, where ϵ is the greedy rate, which avoids the experience sample not being replayed when $TD-error$ is 0. Meanwhile, to reduce the extremely high computational burden caused by traversing the experience pool, the SumTree structure is introduced to obtain the training samples.

The use of PER changes the data distribution, which is bound to make deviation. Therefore, the importance sampling weight ω_i is applied to correct for deviation, which is calculated as follows,

$$\omega_i = (N \cdot P(i))^{-\beta} \Big/ \max_i \omega_i \tag{35}$$

where β is the compensation coefficient.

After obtaining the importance sampling weight ω_i, and the loss function considering the sample priority is described as,

$$L(\theta) = \mathbb{E}\left[\omega_t\left(r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'; \theta^-) - Q(s_t, a_t; \theta)\right)^2\right] \tag{36}$$

5. AltDRL algorithm for SPS-TLS problem

In this section, the alternating iteration algorithm based on deep reinforcement learning (AltDRL) is proposed to solve the SPS-TLS problem, which is described as Fig. 3.

In AltDRL, the order to be handled at each scheduling point is determined by the heuristic algorithm, while the DQN-PER is responsible for determining the position of blocking orders in the shuffling process. Some effective state features are defined to describe the change of the environment configuration at different training steps. Then the design methods of the reinforcement learning actions and the reward are introduced successively. The details are as follows.

5.1. Alternating iteration algorithm based on deep reinforcement learning

The proposed alternating iteration algorithm based on deep reinforcement learning (AltDRL) is outlined in Algorithm 1. $\mathscr{C}_1$ represents the initial configuration of plate distribution. In line 2, the function `Search` is invoked on $\mathscr{C}_1$ to initiate the algorithm, which returns the first target outbound order and its corresponding stack. The function `Search` is implemented as a heuristic algorithm used to determine the retrieval sequence of outbound orders. Once the outbound order associated with the current scheduling point is identified, reinforcement learning algorithm is applied to relocate the blocking orders above it to other stacks until the target outbound order is retrieved. This is implemented through lines 6–9 of Algorithm 1, where action a_t is taken to perform the shuffling operation. Lines 9–11 are used to obtain the transition sequence $(\phi_t, a_t, r_t, \phi_{t+1})$ of DQN, where the reward r_t for scheduling point t is computed using the reward function `Reward`.

The online network and target network of the proposed AltDRL algorithm have the same network structure, where the number of nodes in the input layer and output layer is the same as the dimension of state features and the number of executable actions, respectively. Lines 12–18 utilize the prioritized experience replay (PER) mechanism to update the online network of the DQN. Furthermore, a soft target update strategy is introduced to update the parameters of the target network by slowly tracking the online network in line 20, where δ is a soft parameter forcing the target values to change smoothly. With this parameter update method, the stability of the algorithm can be effectively improved.

5.2. Heuristic algorithm to determine the target order

The target order in the proposed AltDRL algorithm is the outbound order which is retrieved at the current scheduling point. The target order with a potential contribution to reducing shuffles in current or future scheduling processes can be quickly identified by adopting the heuristic algorithm. This approach is already widely used in container relocation problems. Before illustrating the specific heuristic algorithm, several parameters are defined as follows:

j Index of the outbound order;

n_b The number of all plates blocking the chosen order;

g_1 The minimum outbound date among the orders in a stack;

g_2 The minimum outbound date among the orders in a stack that are not retrieved at the current date; we set the current date to 1;

g_3 The minimum outbound date among the orders stored in a stack above the highest-positioned outbound order.

g_4 The number of steel plates above the highest-positioned outbound order in a stack.

The procedure for determining the target order is shown in Algorithm 2, which is inspired by literature [25].

5.3. Definition of state features

After obtaining the target order to be retrieved and the stack in which it is located, the best shuffling position for the blocking orders should be determined by the AltDRL algorithm if the blocking orders in this step are not empty. To this end, we design two types of state features to describe the environment configuration in this part. Since the order to be retrieved at each scheduling point is determined according to Algorithm 2, the environment is completely observable. Based on this, two types of state features can be defined as follows,

(1) *Order blocking ratio:*

$$\omega_t(\ell) = \bar{\omega}_t(\ell) \Big/ K, \ell = 1, 2, \dots, S \tag{37}$$

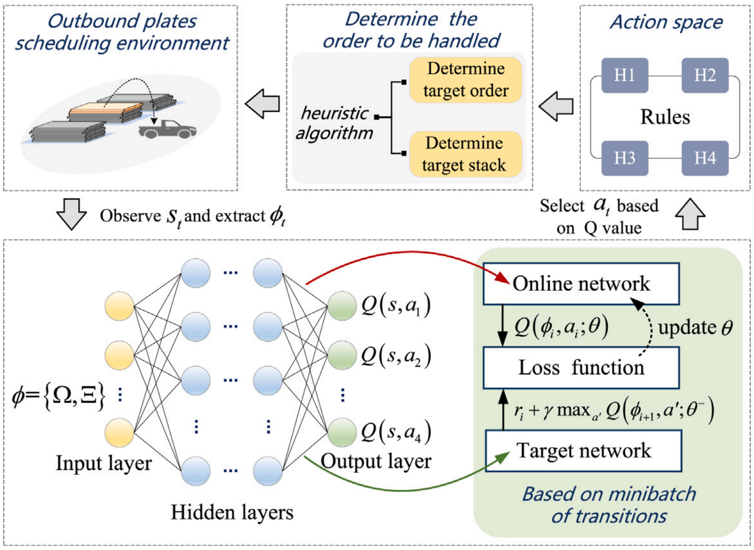


Fig. 3. Structure of the proposed AltDRL algorithm.

Algorithm 1 Alternating iteration algorithm based on deep reinforcement learning (AltDRL)

```
Initialize: initial configuration $\mathscr{C}_1$; memory pool $D$ with capacity $N$; online network $Q$ with parameters $\theta$; target network $\hat{Q}$ with parameters $\theta^- = \theta$; replay parameters $\alpha$ and $\beta$; number of training episodes $L$; maximum and minimum exploration probabilities $\epsilon_{\max}$, $\epsilon_{\min}$; $\epsilon = \epsilon_{\max}$
for episode = 1 : $L$ do
    Determine the first target order $j$ and its stack $\ell_j$ according to function Search($\mathscr{C}_1$);
    Generate the initial state $s_1$ with feature vector $\phi_1 = \{\Omega_1, \Xi_1\}$;
    $done$ = True; $t$ = 1;
    while $done$ do
        Set the topmost order $j'$ of stack $\ell_j$ as the order to be handled by the scheduling point $t$;
        With probability $\epsilon$ select a random action $a_t$, otherwise select $a_t = \arg\max Q(\phi_t, a; \theta)$, and execute action $a_t$;
        If $j' = j$, then determine the next target order $j$ and its stack $\ell_j$ according to function Search($\mathscr{C}_{t+1}$) at scheduling point $t+1$; Otherwise, set the topmost order $j'$ of stack $\ell_j$ as the order to be handled by the scheduling point $t+1$;
        Observe the next state $s_{t+1}$, and calculate the immediate reward $r_t$ according to function Reward($\Omega_t, \Omega_{t+1}, a_t$);
        Extract the feature vector $\phi_{t+1} = \{\Omega_{t+1}, \Xi_{t+1}\}$ of state $s_{t+1}$;
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$;
        Compute sampling probability $P(i)$ according to Eq. (34) and sample a minibatch of transitions $(\phi_i, a_i, r_i, \phi_{i+1})$ from the SumTree;
        Compute importance-sampling weight according to Eq. (35);
        Set $v_i = \begin{cases} r_i, & \text{if episode terminates at step } i+1 \\ r_i + \gamma \max_{a'} \hat{Q}(\phi_{i+1}, a'; \theta^-), & \text{otherwise} \end{cases}$
        Compute $TD\!-\!error$ according to Eq. (33);
        Update transition priority;
        Perform a gradient descent step on $w_i \cdot (v_i - Q(\phi_i, a_i; \theta))^2$ to update the online network;
        Set $done$ = False, if episode terminates at step $t$; otherwise, $t = t + 1$;
        The probability decrease gradually $\epsilon$ from $\epsilon_{\max}$ to $\epsilon_{\min}$;
        $\theta^- = \delta\theta + (1 - \delta)\theta^-$;
```

Algorithm 2 Procedure to determine the target order

```
Function Search($\mathscr{C}$)
S1: Calculate $n_b$ for each outbound order $j$;
S2: If there is only one order with $n_b = 0$, choose it as the target order at this scheduling point. If there are multiple orders with $n_b = 0$, then choose one randomly;
S3: If there is no order with $n_b = 0$, then calculate $g_2$ for each stack where the outbound orders are stored;
S4: If there is only one stack with largest $g_2$, choose the highest-positioned outbound order stored in this stack as the target order. If there are multiple stacks with largest $g_2$, calculate $g_3$ for these stacks;
S5: If there is only one stack with largest $g_3$, choose the highest-positioned outbound order stored in this stack as the target order. If there are multiple stacks with largest $g_3$, calculate $g_4$ for these stacks;
S6: If there is only one stack with smallest $g_4$, choose the highest-positioned outbound order stored in this stack as the target order. If there are multiple stacks with smallest $g_4$, then choose the stack with the lowest number, and the highest-positioned outbound order stored in it as the target order;
S7: Return the target order $j$ and the stack $\ell$ where it is located.
```

where $\bar{\omega}_t(\ell)$ represents the number of all steel plates in the stack ℓ at scheduling point t, which are blocking orders with the minimum delivery date. K is the maximum height of the stack. During the AltDRL training process, the $\omega_t(\ell)$ of each stack is used as the input to the online network Q, thus we encapsulate them with a feature vector Ω_t, i.e., $\Omega_t = [\omega_t(1), \omega_t(2), \ldots \omega_t(S)]$. By limiting the value range of the state feature $\omega_t(\ell)$ to [0,1], the gradient explosion of the neural network can be avoided.

(2) *Shuffle effect:*

The "shuffle effect" is defined as the outbound date of the order to be handled minus the minimum outbound date of the orders in each other stack, and then divided by the maximum outbound date of all orders. This feature reflects the effect of the order shuffle on retrieving other orders in each stack. In particular, two adjacent orders are used to calculate the feature at each scheduling point, so that the agent can perceive the differences in the evolution of the environment in advance under the two adjacent shuffling operations, thereby improving the learning efficiency. The calculating method is given in Algorithm 3.

The above features are extracted to serve as the input of online network Q in AltDRL, and the total dimension of the features is $3 \times S$.

5.4. Definition of actions

Action refers to retrieving the chosen target order or assigning proper shuffling stack to the blocking order. However, blindly shuffling the blocking orders to other stacks may cause the agent to repeatedly perform invalid actions, thereby resulting in the AltDRL being unable

Algorithm 3 Procedure to calculate the shuffle effect

```
Input: The retrieved order $j$ and the stack $\ell$ where it is located; outbound date $d_j$ of order $j$; maximum outbound date $D^{\max}$ of all orders.
Output: The feature vector $\Xi_t$ at scheduling point $t$ consisting of the shuffle effect.
Create an empty vector $\Xi_t$;
Extract order $i_1$ at the top of stack $\ell$;
if $i_1 = j$ then
    for $num = 1$ to $2S$ do
        Append $0$ to $\Xi_t$;
else
    for $h = 1$ to $S$ do
        if $h \neq \ell$ then
            if $h$ is not an empty stack then
                Calculate the shuffle effect $\xi = \left(d_{i_1} - D_h^{\min}\right)/D^{\max}$;  ▷ $D_h^{\min}$ is the minimum outbound date of orders in stack $h$.
                Append $\xi$ to $\Xi_t$;
            else
                Append $0$ to $\Xi_t$;
        else
            Append $0$ to $\Xi_t$;
    Extract order $i_2$ stored beneath the order $i_1$;
    if $i_2 = j$ then
        for $num = 1$ to $S$ do
            Append $0$ to $\Xi_t$;
    else
        for $h = 1$ to $S$ do
            if $h \neq \ell$ then
                if $h$ is not an empty stack then
                    Calculate the shuffle effect $\xi = \left(d_{i_2} - D_h^{\min}\right)/D^{\max}$;
                    Append $\xi$ to $\Xi_t$;
                else
                    Append $0$ to $\Xi_t$;
            else
                Append $0$ to $\Xi_t$;
```

to converge. On this account, we introduce four heuristic scheduling rules as the action space of the agent. The first three rules are inspired by container rehandling methods which were first proposed in [33], and the basic idea of them is to avoid future shuffling when choosing a new stack for the blocking orders. The fourth rule is to avoid the agent falling into local optimum when searching for the optimal combination of actions. Before describing the specific heuristic scheduling rules, two important parameters are defined as follows:

RI The total number of orders in a stack that are to be retrieved earlier than the blocking order;

BI The total number of steel plates that will block the highest-positioned order with the minimum outbound date in a stack if the blocking order is shuffled into that stack;

Based on the parameters *RI* and *BI*, the procedure of heuristic scheduling rule 1 is given in procedure H1. In steps S3, S4, and S6 of H1, one default condition for screening stacks is that the maximum height limit of stacks will not be exceeded if the blocking order is shuffled to those stacks. The *RI* is used in step S6 to minimize the blocking of retrieving orders in the future caused by shuffling. The procedure of heuristic scheduling rule 2 (H2) is the same as H1, except that *RI* is replaced with *BI* in S6. In the proposed H2, the *BI* is used to guarantee a minimum number of shuffles when the order is retrieved at subsequent scheduling points. Moreover, the main difference between the heuristic scheduling rule 3 (H3) and H1 is step S6, in which H3 assigns the stack with the g_1 value closest to d_j to the order j if there is only one stack, otherwise, the order j is shuffled to the stack with the smallest *BI*. The purpose of this design is to postpone the inevitable reshuffles as much as possible.

H1: Procedure of heuristic scheduling rule 1

S1: Extract the order j that needs to be handled and the outbound date d_j of the order j;
S2: If the order j is the retrieved order, return the *Empty* and end the procedure; Otherwise, go to S3;
S3: If there are stacks satisfying $g_1 = d_j$, then assign the stack with the fewest number of plates to the order j; Otherwise, go to S4;
S4: If there are stacks satisfying $g_1 > d_j$, then assign the stack with g_1 value closest to d_j to the order j; Otherwise, go to S5;
S5: If there are empty stacks, then assign any empty stack to the order j; Otherwise, go to S6;
S6: Assign the stack with the smallest *RI*. If more than one stack satisfies the condition, then assign the stack with the g_1 value closest to d_j to the order j.

Similarly, the procedure of heuristic scheduling rule 4 is shown in procedure H4.

H4: Procedure of heuristic scheduling rule 4

S1: Extract the order j that needs to be handled and the outbound date d_j of the order j;
S2: If the order j is the retrieved order, return the *Empty* and end the procedure; Otherwise, go to S3;
S3: If there are stacks satisfying $g_1 = d_j$, then assign the stack with the fewest number of plates to the order j; Otherwise, go to S4;
S4: If there are stacks satisfying $g_1 < d_j$, then assign the stack with g_1 value closest to d_j to the order j; Otherwise, go to S5;
S5: If there are stacks satisfying $g_1 > d_j$, then assign the stack with g_1 value closest to d_j to the order j; Otherwise, go to S6;
S6: Assign any empty stack to the order j;

Note that the scheduling rules H1 to H3 are greedy search algorithms, and the scheduling sequence formed by AltDRL combining them will fall into the local optimum. H4 can break the shackles of greedy search by performing the step S3 before S4, making AltDRL possible to obtain better scheduling results.

5.5. Definition of rewards

Since the objective of the SPS-TLS problem is to minimize both the number of plate shuffles during the orders outbound process and the minimum number of plate shuffles during the forthcoming outbound process, the reward r_t for the state–action pair (s_t, a_t) can be defined by two indicators, as shown in Algorithm 4. The first indicator r_t^1, obtained by executing action a_t, is called the number of plate shuffles at the scheduling point t. In lines 2 and 3 of Algorithm 4, if the return of action a_t is not *Empty*, then the shuffling must occur. In this case, the performance of action a_t under state s_t can be more intuitively reflected by introducing the number of steel plates into the reward evaluation system. The second indicator r_t^2, which is the minimum number of plate shuffles during the forthcoming outbound process, can be obtained indirectly by the state features "*Order blocking ratio*" at current state s_t and the next state s_{t+1}, calculated as in line 7 of algorithm 4. r_t^2 is designed to reflect the immediate gain of executing the action a_t for the second optimization objective. The final reward value is obtained by adding r_t^1 and r_t^2.

6. Numerical experiments

In this section, a series of numerical experiments based on data collected from a real steel logistics park are conducted to examine the effectiveness of the joint optimization of steel plate shuffling and truck loading sequencing during the orders outbound process and the superiority of the proposed AltDRL algorithm. The maximum quantity of steel

Table 2
Parameter settings of AltDRL.

Parameters	Value
Number of hidden layers	5
Number of nodes in each hidden layer	30
Activation function	ReLu
Learning rate	0.001
Memory pool size N	1000
Number of training episodes L	5000
Sample batches size	64
Maximum exploration probability $\epsilon_{\max}$	0.6
Minimum exploration probability $\epsilon_{\min}$	0.01
Discount factor λ	0.98
δ in the soft target update strategy	0.008
α in the prioritized experience replay mechanism	0.6
β in the prioritized experience replay mechanism	increase by 0.001 from 0.4 to 1

Algorithm 4 Procedure to calculate r_t

```
Input: The order blocking ratio at current state $s_t$ and the next state $s_{t+1}$, $\Omega_t$
    and $\Omega_{t+1}$; action $a_t$.
Output: Reward $r_t$.
Function Reward($\Omega_t, \Omega_{t+1}, a_t$)
if the return of $a_t$ is not Empty then
    $r_t^1 = -M_t$; // $M_t$ is the shuffling number of steel plates at scheduling point $t$.
else
    $r_t^1 = 1$;
Calculate $\bar{\Omega}_t = \sum_{s \in S} \omega_t(s)$ and $\bar{\Omega}_{t+1} = \sum_{s \in S} \omega_{t+1}(s)$;
$r_t^2 = (\bar{\Omega}_t - \bar{\Omega}_{t+1}) \times K$;
$r_t = r_t^1 + r_t^2$
Return the reward $r_t$
```

plates that each stack can accommodate is set to 60. All experiments are implemented in python 3.9 on a Windows 11 Professional 64-bit operating system with Intel Core i7-13700K @ 3.4 GHz CPU and 32 GB RAM.

The experiments contain four parts. To verify the convergence of AltDRL in different environment configurations, we compare it with a version that does not consider the prioritized experience replay, denoted by AltDRL(non-PER) in this paper. Except for the parameters unique to prioritized experience replay, both algorithms adopt the same parameter settings, as shown in Table 2. In addition, the convergence process of AltDRL for both the number of plate shuffles during the orders outbound process (f_1) and the minimum number of plate shuffles during the forthcoming outbound process (f_2) is analyzed in detail. Subsequently, the effectiveness of the AltDRL algorithm is further verified under multiple instances of small sizes and large sizes. In this part, the Gurobi 11.0 and the Branch-and-Bound (B&B) algorithm [27] are first used to compare with the AltDRL algorithm on the small-size instances. Then, the AltDRL algorithm is compared with each distinct heuristic scheduling rule from this paper, as well as the state-of-the-art heuristic algorithms, *Adjusted H4* (A_H4) and *Adjusted H5* (A_H5), introduced in [25], under the large-size instances. Last but not the least, we evaluate the superiority of the joint optimization by comparing it with optimizing only plates shuffling under the fixed trucks loading sequence. The specific experimental results are presented below.

6.1. Convergence of AltDRL algorithm

Fig. 4 presents the change of rewards with the increase of episodes in the learning process of AltDRL and AltDRL(non-PER). In this experiment, four instances are used to evaluate the performance of the solution algorithm, containing 8, 12, 16 and 20 stacks, respectively, with corresponding number of orders 69, 105, 146 and 199. In order to ensure the fairness of comparison, the same random seed is used in both algorithms for the same instance. The change of rewards under three instances are given in Fig. 4(a), Fig. 4(b), 4(c) and Fig. 4(d), respectively, and the results are output every 10 episodes.

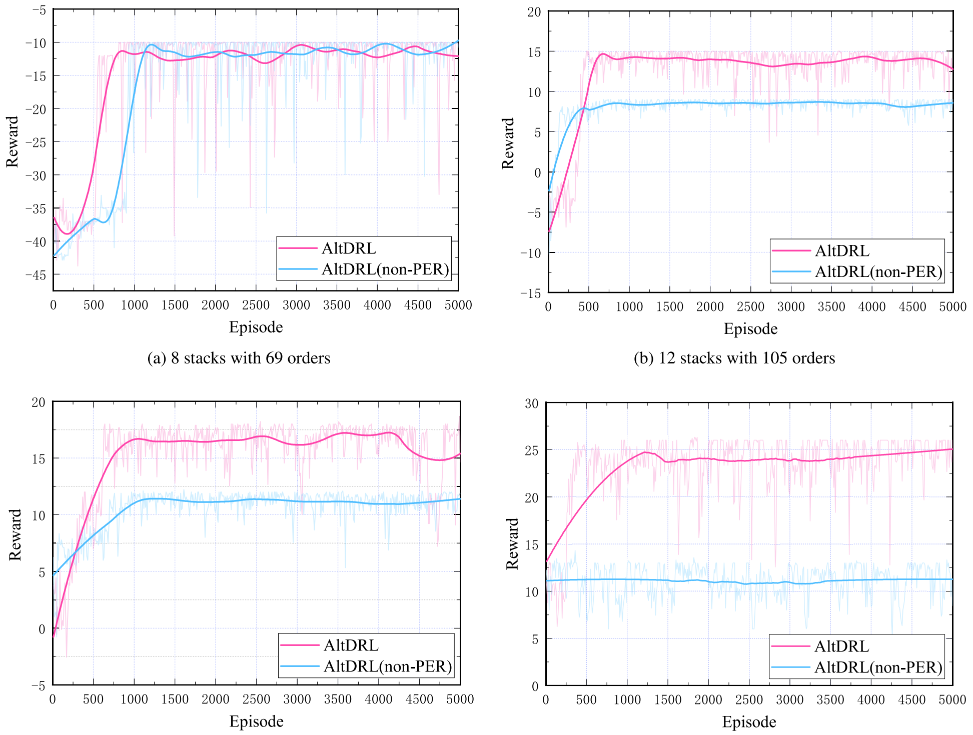
It can be seen from Fig. 4 that due to the high exploration probability in the early stage, AltDRL and AltDRL(non-PER) tend to randomly choose actions and the rewards fluctuate around the smaller value. However, with enough attempts, both algorithms have the opportunity to choose better actions in different states so as to obtain higher rewards. These experiences are stored in a memory pool, continuously improving the decision-making ability of the agent during the algorithm training process. At later stages, AltDRL and AltDRL(non-PER) tend to choose actions with higher Q values as the exploration probability gradually decreases with the number of iterations, and finally the reward stabilizes at a higher value.

It is worth noting that both AltDRL and AltDRL(non-PER) can converge to the same value as show in Fig. 4(a), since the valuable experience gained in a simple scheduling scenario can be well utilized. However, during the training process of AltDRL, higher sampling priority is assigned to high-value experience samples, and hence the convergence rate of AltDRL is faster than that of AltDRL(non-PER). Furthermore, the prioritized experience replay mechanism can significantly improve the convergence of the algorithm as shown in Fig. 4(b). By comparison, it is clear that the rewards of AltDRL after convergence are higher than those of AltDRL(non-PER). This advantage becomes more pronounced as the size of the instances increases, as shown in Fig. 4(c) and Fig. 4(d), which indicates that AltDRL can obtain better results.

6.2. Convergence process of the number of plate shuffles

In this part, the change trends of the number of shuffles are revealed with the increase of episodes in the learning process of AltDRL, including the number of plate shuffles during the orders outbound process (Fig. 5(a)), the minimum number of plate shuffles during the forthcoming outbound process (Fig. 5(b)), and the total number of plate shuffles (Fig. 5(c)). The experiment is performed based on an instance of 69 orders in 8 stacks, and the results are output every 10 episodes.

The number of plate shuffles during the orders outbound process (f_1) is a direct reflection of the current order outbound efficiency. A smaller f_1 indicates higher scheduling efficiency of outbound orders and less customer waiting time. In the early stages of AltDRL training, f_1 can obviously reach lower values, but this is at the expense of increasing the minimum number of plate shuffles during the forthcoming outbound process (f_2), as can be easily seen from Figs. 5(b) and 5(c). The f_2 is directly related to the energy consumption and the scheduling efficiency of the crane in the future order outbound process. As the training progresses, it can be observed that a slight increase in the value of f_1 can significantly decrease the value of f_2, as shown in Figs. 5(a) and 5(b). Until the later stages of AltDRL training, the best compromise of f_1 and f_2 can be achieved, as shown in Fig. 5(c), which proves the effectiveness of the proposed algorithm.



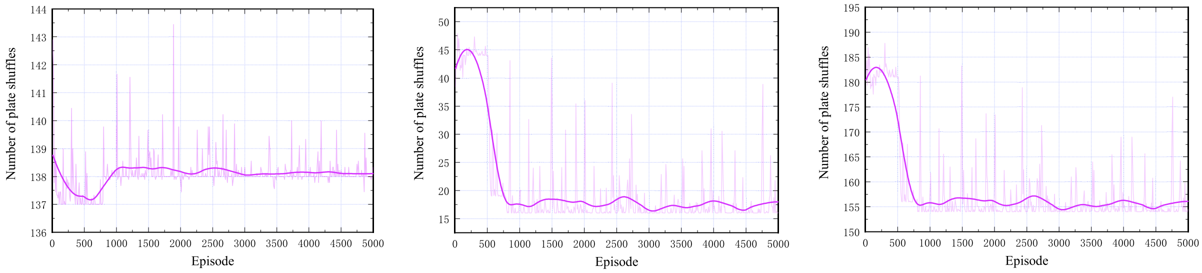
(a) 8 stacks with 69 orders

(b) 12 stacks with 105 orders

(c) 16 stacks with 146 orders

(d) 20 stacks with 199 orders

Fig. 4. Change of rewards in the learning process of AltDRL and AltDRL(non-PER) under different environment configurations.



(a) Number of plate shuffles during the orders outbound process (f_1) vs. episodes

(b) Minimum number of plate shuffles during the forthcoming outbound process (f_2) vs. episodes

(c) Total number of plate shuffles vs. episodes

Fig. 5. Change of the number of plate shuffles in the learning process of AltDRL.

Table 3
Instances with small sizes.

Index	S	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{I}_1\rvert$	Index	S	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{I}_1\rvert$
s1	3	12	4	s3	5	18	7
	3	14	4		5	24	8
	3	14	4		5	34	10
	3	13	5		5	36	12
s2	4	19	5	s4	6	29	10
	4	18	5		6	45	15
	4	17	6		6	38	12
	4	15	6		6	43	14

6.3. *Effectiveness of AltDRL algorithm*

In this subsection, we compare the AltDRL algorithm with both exact methods and heuristic algorithms to validate its effectiveness.

6.3.1. *Comparisons with the exact methods*

In this part, the Gurobi 11.0 and the B&B algorithm are utilized to compare with the proposed AltDRL algorithm, aiming to verify the effectiveness and applicability of the AltDRL algorithm. The Gurobi solver is a robust and efficient tool for solving MILP models, providing accurate solutions for complex optimization problems. The relative optimality gap for MILP in Gurobi is set to 0 in this paper. Additionally, the Branch-and-Bound algorithm introduced in [27], is one of the most state-of-the-art exact algorithms for container retrieval problems. The experiments are conducted on the small-size instances provided in Table 3. The Comparison results are presented in Table 4, where the term “CPU” denotes the runtime of the algorithms, with a maximum allowed runtime set at 14400 s. Specifically for the AltDRL algorithm, we provide both offline training time and online computation time for each instance, enabling an observation of the impact of instance size on the algorithm’s time complexity.

According to the results in Table 4, it is evident that Gurobi can only solve a limited number of instances within the specified time constraints. As the instance size increases, Gurobi may even fail to obtain a feasible solution. The B&B algorithm appears effective in quickly obtaining optimal solutions for small-size instances, as demonstrated in instance sets s1 and s2. However, the time complexity of the B&B algorithm is significantly influenced by the state of the steel plate stacking.

Table 4
Calculation results with small-size instances by AltDRL and exact methods.

Instances		Gurobi			B&B			AltDRL			CPU(s)		
		f_1	f_2	SUM	f_1	f_2	SUM	f_1	f_2	SUM	Gurobi	B&B	AltDRL
s1	3	12	3	15	12	3	15	12	3	15	145.61	0.93	143.48/≤0.01
	3	26	1	27	26	1	27	26	1	27	13 707.00	0.07	155.83/≤0.01
	3	9	2	11	8	3	11	9	2	11	7026.52	0.19	113.53/≤0.01
	3	20	26	46	20	26	46	20	26	46	1186.31	0.04	100.36/≤0.01
s2	4	–	–	–	25	7	32	25	7	32	–	460.95	239.46/≤0.01
	4	–	–	–	20	1	21	20	2	22	–	0.05	130.28/≤0.01
	4	22	25	47	14	25	39	14	26	40	14 400.00	0.54	194.49/≤0.01
	4	–	–	–	15	3	18	15	3	18	–	0.42	161.18/≤0.01
s3	5	–	–	–	8	0	8	8	1	9	–	0.90	164.41/≤0.01
	5	–	–	–	12	7	19	11	8	19	–	7.62	220.20/≤0.01
	5	–	–	–	32	14	46	33	11	44	–	14 400.00	395.37/0.01
	5	–	–	–	–	–	–	49	16	65	–	–	502.70/0.09
s4	6	–	–	–	81	2	83	86	4	90	–	1845.51	332.73/≤0.01
	6	–	–	–	–	–	–	42	9	51	–	–	586.67/0.02
	6	–	–	–	–	–	–	47	28	75	–	–	413.81/0.01
	6	–	–	–	–	–	–	62	4	66	–	–	584.17/0.02

*For the AltDRL column beneath the CPU column, the first value represents the algorithm's offline training time, and the second value represents the algorithm's online computation time.

Table 5
Instances with large sizes.

Index	S	$\vert\mathcal{I}\vert$	$\vert\mathcal{I}_1\vert$	Index	S	$\vert\mathcal{I}\vert$	$\vert\mathcal{I}_1\vert$
b1	8	69	22	b3	8	101	24
	12	105	37		12	128	29
	16	146	48		16	164	37
	20	199	48		20	175	42
b2	8	63	16	b4	8	48	15
	12	108	27		12	92	27
	16	108	31		16	127	38
	20	153	45		20	159	38

On the other hand, when the number of retrieval orders exceeds 10, the computational complexity of B&B becomes exceptionally high and may even render it ineffective. This is because the number of order retrieval combinations is factorial, in which case the B&B algorithm encounters a huge number of branches, resulting in exponential growth in both time complexity and memory requirements. With millions of potential branches, the algorithm may require an unacceptably long time to generate and process them, leading to timeouts. Additionally, storing the state and information for each branch requires a substantial amount of memory, exceeding available memory limits, rendering the algorithm unable to function properly.

On the contrary, the AltDRL algorithm can obtain optimal or near-optimal solutions for all instances after the offline training. Although the training time of the AltDRL algorithm gradually increases with the size of the instances, the AltDRL algorithm consistently completes offline training with 5000 episodes within 600 s for all instances. Additionally, the online computation is accomplished within 0.1 s. These results confirm that AltDRL consistently maintains excellent performance across various complex scheduling environments.

6.3.2. Comparisons with the heuristic algorithms

In this part, we conducted a performance comparison between AltDRL and the heuristic scheduling rules adopted in this paper. Specifically, in the comparison methods, the order that needs to be retrieved at each stage is determined by Algorithm 2, and the shuffling positions for the blocking orders are determined by each individual heuristic scheduling rule. In addition, algorithms A_HA and A_H5 are utilized as comparative benchmarks to further demonstrate the effectiveness. Four sets of instances extracted from the real scheduling scenario are considered in the experiment, each containing four instances, and the sizes of the instances are presented in Table 5. The results obtained by AltDRL and each heuristic scheduling rule are detailed in Table 6 under instances of different sizes. Meanwhile, the performance of AltDRL in comparison with A_H4 and A_H5 is displayed in Fig. 6.

It can be seen from Table 6 that the f_1 or f_2 obtained by AltDRL are better than the single heuristic scheduling rule in most cases, but in few instances, the performance of the single heuristic rule may exceed that of AltDRL. This is because heuristic scheduling rules are essentially greedy algorithms that execute operations that are most beneficial to the current environment based on an already designed policy. Different heuristic rules focus on different optimization objectives, such that only one of the values of f_1 and f_2 can be improved by performing a single heuristic scheduling rule. However, the sum of f_1 and f_2 generated by any kind of heuristic scheduling rules does not exceed that of AltDRL for all instances. This is because AltDRL can take full advantage of the learned experience after training, which enables it to form the optimal action sequences based on the state evolution, thereby achieving better scheduling results. These experimental results demonstrate the superiority of AltDRL over a single heuristic algorithm in reducing the number of plate shuffles during the orders outbound process, as well as the minimum number of plate shuffles during the forthcoming outbound process. The same phenomenon is also evident in Fig. 6, which not only illustrates the optimization performance of various algorithms for the SPS-TLS problem but also highlights the percentage of performance improvement achieved by the AltDRL algorithm in reaching the overall goal, compared to the best results obtained from the benchmark algorithms.

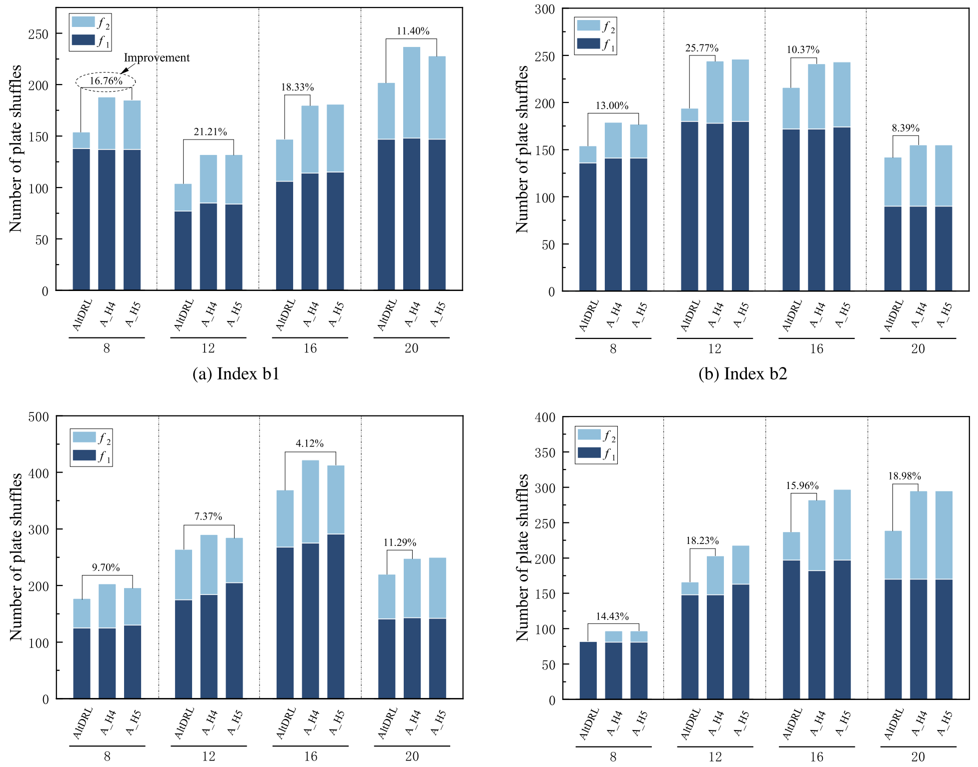
6.4. Performance of the joint optimization

In this part, the joint optimization of steel plate shuffling and truck loading sequencing (SPS-TLS) is compared with optimizing only plates shuffling under fixed truck loading sequencing (Fix-TLS) to verify the performance of the proposed scheme. All experiments are performed on the instances of set b1 in Table 5. For the sake of fairness, the experiments for Fix-TLS are executed by the DQN algorithm with prioritized experience replay, under the same settings of states, actions, and rewards as AltDRL. All the experimental results of Fix-TLS are averaged over five independent randomly generated trucks loading sequences realizations. The comparison results are shown in Fig. 7.

It can be seen from Fig. 7, the SPS-TLS can always achieve fewer plate shuffles than Fix-TLS, whether for objective f_1 or f_2. For the four instances shown in Fig. 7, the number of plate shuffles under SPS-TLS during the orders outbound process is reduced by 55.3%, 78.7%, 75.4% and 62.4% compared with Fix-TLS, respectively. The same advantages appear in the minimum number of plate shuffles during the

Table 6
Calculation results with large-size instances by AltDRL and each heuristic scheduling rule.

instances		AltDRL			H1			H2			H3			H4		
		f_1	f_2	SUM	f_1	f_2	SUM	f_1	f_2	SUM	f_1	f_2	SUM	f_1	f_2	SUM
b1	8	138	16	**154**	137	47	184	139	47	186	137	44	181	163	84	247
	12	77	27	**104**	97	35	132	94	26	120	77	39	116	77	54	131
	16	106	41	**147**	126	45	171	124	37	161	106	50	156	106	67	173
	20	147	55	**202**	159	64	223	147	69	216	147	69	216	152	84	236
b2	8	136	18	**154**	136	31	167	143	26	169	136	31	167	188	114	302
	12	180	14	**194**	178	21	199	185	14	199	178	21	199	213	149	362
	16	172	44	**216**	172	53	225	174	53	227	172	53	225	204	116	320
	20	90	52	**142**	90	52	142	90	52	142	90	52	142	90	78	168
b3	8	125	52	**177**	128	56	184	126	61	187	125	59	184	137	96	233
	12	175	89	**264**	176	95	271	199	73	272	175	93	268	175	141	316
	16	268	101	**369**	268	105	373	293	103	396	268	105	373	268	159	427
	20	141	79	**220**	142	79	221	142	79	221	140	80	220	140	105	245
b4	8	82	1	**83**	81	5	86	81	5	86	81	5	86	120	55	175
	12	148	18	**166**	163	14	177	148	29	177	148	29	177	174	72	246
	16	197	40	**237**	197	55	252	182	70	252	182	70	252	187	120	307
	20	170	69	**239**	170	79	249	170	79	249	170	79	249	170	132	302



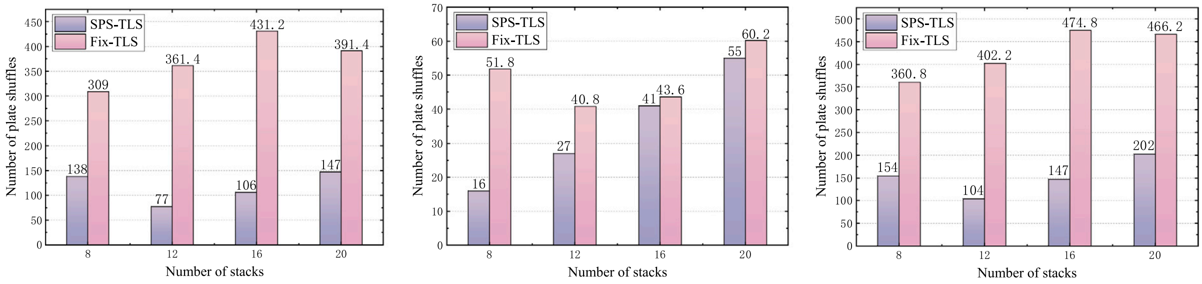
(a) Index b1

(b) Index b2

(c) Index b3

(d) Index b4

Fig. 6. Calculation results with large-size instances by AltDRL, A_HA and A_H5.



(a) Number of plate shuffles during the orders outbound process (f_1) under different instances

(b) Minimum number of plate shuffles during the forthcoming outbound process (f_2) under different instances

(c) Total number of plate shuffles under different instances

Fig. 7. Change of the number of plate shuffles based on the SPS-TLS and Fix-TLS under different environment configurations.

forthcoming outbound process, and the results of SPS-TLS are reduced by 69.1%, 33.9%, 6.0% and 8.6%compared to Fix-TLS, respectively. This is because for Fix-TLS, blindly giving the truck loading sequence forces the plate shuffling to be optimized only in a greedy way. The SPS-TLS can break the limitation of truck loading sequence and find the optimal plate shuffling scheme in a larger solution space. Therefore, the SPS-TLS is important in reducing steel plate shuffling and improving scheduling efficiency.

7. Conclusion

In this paper, we propose to utilize the data of plates and trucks collected from a real steel logistics park by the IIoT to jointly optimize the steel plate shuffling and truck loading sequencing, so as to improve the outbound efficiency of plates. The SPS-TLS problem is firstly transformed into an orders scheduling problem which is formulated as a mixed integer linear programming model. Moreover, the deep Q network with prioritized experience replay is introduced to develop the AltDRL algorithm to address the proposed problem. Meanwhile, two types of state features are designed in this paper to describe the environment configuration, and four heuristic scheduling rules are proposed to act as optional action sets for the intelligent agent.

Furthermore, the performance of the AltDRL algorithm is validated by comparing it with two advanced exact methods in small-size instances and popular heuristic algorithms in large-size instances. The experimental results indicate that, although the AltDRL algorithm may not achieve the same solution quality as exact methods in certain small-size instances, its stability in computational time surpasses that of exact methods. This stability of exact algorithms is notably influenced by the state of steel plate stacking and the number of orders to be retrieved. Additionally, despite having lower time complexity, heuristic algorithms exhibit noticeably weaker solution quality than the AltDRL algorithm. It is important to note that, for the SPS-TLS problem, the outbound steel plate orders are known one day in advance. This implies that there is sufficient time to train and improve models for specific scheduling environments, thereby achieving better scheduling results. For various practical and complex scheduling environments, the training of the AltDRL algorithm can typically be completed within one hour, presenting its outstanding applicability to real-world scheduling problems.

In future work, we aim to explore the utilization of reinforcement learning to develop a joint scheduling algorithm for plate storage and outbound processes, thereby providing better solutions for complex engineering problems.

CRediT authorship contribution statement

Zhezhuang Xu: Conceptualization, Investigation, Methodology, Project administration, Writing – original draft. **Jinlong Wang:** Methodology, Validation, Writing – original draft, Writing – review & editing. **Meng Yuan:** Conceptualization, Investigation, Methodology. **Yazhou Yuan:** Conceptualization. **Boyu Chen:** Data curation, Resources. **Qingdong Zhang:** Data curation, Resources. **Cailian Chen:** Formal analysis, Methodology. **Xinping Guan:** Formal analysis, Methodology.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work is supported by the Fujian Industry-University Cooperation Project under Grant 2022H6005, the National Natural Science Foundation of China under Grant 61973085, Grant 62273295 and the Overseas Qishan Scholars Program under Grant 511262.

References

[1] G.L. Putra, M. Kitamura, Study on optimal design of hatch cover via a three-stage optimization method involving material selection, size, and plate layout arrangement, Ocean Eng. 219 (2021) 108284.

[2] J. Wang, Z. Xu, W. Wen, R. Wang, Y. Lin, Y. Yuan, B. Chen, Q. Zhang, Optimization for storage scheduling of steel plates based on cloud manufacturing platform, IEEE Trans. Ind. Inform. 19 (12) (2023) 11653–11663.

[3] L. Wu, Z. Jiang, X. Li, A stack-based retrieval method for the steel plate yard retrieval problem in shipbuilding, Flex. Serv. Manuf. J. (2023) 1–35.

[4] B. Jin, S. Tanaka, An exact algorithm for the unrestricted container relocation problem with new lower bounds and dominance rules, European J. Oper. Res. 304 (2) (2023) 494–514.

[5] L. Tang, J. Liu, A. Rong, Z. Yang, An effective heuristic algorithm to minimise stack shuffles in selecting steel slabs from the slab yard for heating and rolling, J. Oper. Res. Soc. 52 (2001) 1091–1097.

[6] R. Zhao, L. Tang, Integer programming model and dynamic programming based heuristic algorithm for the heavy plate shuffling problem in the iron and steel industry, in: 2010 International Conference on Logistics Systems and Intelligent Management, ICLSIM, 3, IEEE, 2010, pp. 1381–1385.

[7] L. Tang, J. Liu, A. Rong, Z. Yang, Modelling and a genetic algorithm solution for the slab stack shuffling problem when implementing steel rolling schedules, Int. J. Prod. Res. 40 (7) (2002) 1583–1595.

[8] K. Keung, C. Lee, P. Ji, Industrial internet of things-driven storage location assignment and order picking in a resource synchronization and sharing-based robotic mobile fulfillment system, Adv. Eng. Inform. 52 (2022) 101540.

[9] Z. Xu, R. Wang, X. Yue, T. Liu, C. Chen, S.-H. Fang, FaceME: Face-to-machine proximity estimation based on RSSI difference for mobile industrial human–machine interaction, IEEE Trans. Ind. Inform. 14 (8) (2018) 3547–3558.

[10] S. Boge, S. Knust, The parallel stack loading problem minimizing the number of reshuffles in the retrieval stage, European J. Oper. Res. 280 (3) (2020) 940–952.

[11] G. Bruno, M. Cavola, A. Diglio, C. Piccolo, A unifying framework and a mathematical model for the Slab Stack Shuffling Problem, Int. J. Ind. Eng. Comput. 14 (1) (2023) 17–32.

[12] Z. Zhang, P. Wang, W. Wang, Optimization and operation scheduling for a steel plate yard based on greedy algorithm, J. Netw. 8 (7) (2013) 1654.

[13] L. Tang, H. Ren, Modelling and a segmented dynamic programming-based heuristic approach for the slab stack shuffling problem, Comput. Oper. Res. 37 (2) (2010) 368–375.

[14] Y. Shi, S. Liu, Very large-scale neighborhood search for steel hot rolling scheduling problem with slab stack shuffling considerations, IEEE Access 9 (2021) 47856–47863.

[15] P. Rajabi, G. Moslehi, M. Reisi-Nafchi, New integer programming models for slab stack shuffling problems, Appl. Math. Model. 109 (2022) 775–796.

[16] G. Zhao, J. Liu, L. Tang, R. Zhao, Y. Dong, Model and heuristic solutions for the multiple double-load crane scheduling problem in slab yards, IEEE Trans. Autom. Sci. Eng. 17 (3) (2019) 1307–1319.

[17] G. Bruno, M. Cavola, A. Diglio, C. Piccolo, A unifying framework and a mathematical model for the slab stack shuffling problem, Int. J. Ind. Eng. Comput. 14 (1) (2023) 17–32.

[18] M. Caserta, S. Schwarze, S. Voß, Container rehandling at maritime container terminals: A literature update, Handb. Termin. Plan. (2020) 343–382.

[19] M.E. Petering, M.I. Hussein, A new mixed integer program and extended look-ahead heuristic algorithm for the block relocation problem, European J. Oper. Res. 231 (1) (2013) 120–130.

[20] S. Tanaka, F. Mizuno, An exact algorithm for the unrestricted block relocation problem, Comput. Oper. Res. 95 (2018) 12–31.

[21] W. Zhu, H. Qin, A. Lim, H. Zhang, Iterative deepening A* algorithms for the container relocation problem, IEEE Trans. Autom. Sci. Eng. 9 (4) (2012) 710–722.

[22] C. Zhang, H. Guan, Y. Yuan, W. Chen, T. Wu, Machine learning-driven algorithms for the container relocation problem, Transp. Res. B 139 (2020) 102–131.

[23] N. Boysen, S. Emde, The parallel stack loading problem to minimize blockages, European J. Oper. Res. 249 (2) (2016) 618–627.

[24] A. Azab, H. Morita, The block relocation problem with appointment scheduling, European J. Oper. Res. 297 (2) (2022) 680–694.

[25] Q. Zeng, Y. Feng, Z. Yang, Integrated optimization of pickup sequence and container rehandling based on partial truck arrival information, Comput. Ind. Eng. 127 (2019) 366–382.

[26] V. Galle, V.H. Manshadi, S.B. Boroujeni, C. Barnhart, P. Jaillet, The stochastic container relocation problem, Transp. Sci. 52 (5) (2018) 1035–1058.

[27] B.G. Zweers, S. Bhulai, R.D. van der Mei, Optimizing pre-processing and relocation moves in the stochastic container relocation problem, European J. Oper. Res. 283 (3) (2020) 954–971.

[28] S. Luo, Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning, Appl. Soft Comput. 91 (2020) 106208.

[29] Z. Zhao, S. Liu, M. Zhou, A. Abusorrah, Dual-objective mixed integer linear program and memetic algorithm for an industrial group scheduling problem, IEEE/CAA J. Autom. Sin. 8 (6) (2021) 1199–1209.

[30] A. Azab, H. Morita, Coordinating truck appointments with container relocations and retrievals in container terminals under partial appointments information, Transp. Res. Part E: Logist. Transp. Rev. 160 (2022) 102673.

[31] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.

[32] T. Schaul, J. Quan, I. Antonoglou, D. Silver, Prioritized experience replay, 2015, arXiv preprint arXiv:1511.05952.

[33] L. Tang, W. Jiang, J. Liu, Y. Dong, Research into container reshuffling and stacking problems in container terminal yards, IIE Trans. 47 (7) (2015) 751–766.