Engineering Applications of Artificial Intelligence 133 (2024) 108487

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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

A Double Deep Q-Network framework for a flexible job shop scheduling problem with dynamic job arrivals and urgent job insertions

Shaojun Lu [a,b,c], Yongqi Wang [a], Min Kong [a,d,*], Weizhong Wang [d], Weimin Tan [d], Yingxin Song [d]

[a] School of Management, Hefei University of Technology, Hefei, 230009, PR China
[b] Center for Applied Optimization, Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL, USA
[c] Key Laboratory of Process Optimization and Intelligent Decision-making of the Ministry of Education, Hefei, 230009, PR China
[d] School of Economics and Management, Anhui Normal University, Wuhu, 241000, PR China

ARTICLE INFO

Keywords:
Semiconductor manufacture
Dynamic flexible job shop scheduling
Double deep Q-Network
Dynamic job arrivals
Urgent job insertions

ABSTRACT

In the semiconductor manufacturing industry, the Dynamic Flexible Job Shop Scheduling Problem is regarded as one of the most complex and significant scheduling problems. Existing studies consider the dynamic arrival of jobs, however, the insertion of urgent jobs such as testing chips poses a challenge to the production model, and there is an urgent need for new scheduling methods to improve the dynamic response and self-adjustment of the shop floor. In this work, deep reinforcement learning is utilized to address the dynamic flexible job shop scheduling problem and facilitate near-real-time shop floor decision-making. We extracted eight state features, including machine utilization, operation completion rate, etc., to reflect real-time shop floor production data. After examining machine availability time, the machine's earliest available time is redefined and incorporated into the design of compound scheduling rules. Eight compound scheduling rules have been developed for job selection and machine allocation. By using the state features as inputs to the Double Deep Q-Network, it is possible to acquire the state action values (Q-values) of each compound scheduling rule, and the intelligent agent can learn a reasonable optimization strategy through training. Simulation studies show that the proposed Double Deep Q-Network algorithm outperforms other heuristics and well-known scheduling rules by generating excellent solutions quickly. In most scenarios, the Double Deep Q-Network algorithm outperforms the Deep Q-Network, Q-Learning, and State-Action-Reward-State-Action (SARSA) frameworks. Moreover, the intelligent agent has good generalization ability in terms of optimization for similar objectives.

1. Introduction

Memory chips serve as vital components in computers and communication devices, playing a crucial role in data memory and processing across various platforms including personal computers, smartphones, data centers, and cloud computing infrastructure (Kim et al., 2023). With their fast, reliable, and high-capacity characteristics, memory chips enable the memory and transmission of vast amounts of data, forming the cornerstone for the growth of industries such as the Internet, social media, e-commerce, and online entertainment. The manufacturing of memory chips requires precision equipment and extensive technical expertise to ensure that the chips meet the required standards in terms of quality and performance. By optimizing wafer processing in memory chip manufacturing, companies can achieve numerous benefits such as improved productivity, cost reduction, enhanced product quality and consistency, and increased flexibility and responsiveness. Optimizing wafer processing in memory chip manufacturing helps manufacturers compete by satisfying customer expectations and providing high-quality products rapidly.

Fig. 1 illustrates the main processes of a wafer. Wafer processing is critical, and it involves several important operational steps, i.e., lithography, etching, and deposition. Each phase has a distinct process. Given the highly intricate nature of chip circuit construction, the aforementioned operations are repeated throughout wafer processing till completion (Perraudat et al., 2022). The wafer processing problem in memory chip production can be abstracted as a flexible job shop scheduling problem (FJSP) (Lin, 2019), where each step permits the allocation of different operations to any available machine. This flexibility in machine allocation makes the scheduling process more complicated, thereby aligning it with the characteristics of an FJSP. The

* Corresponding author. School of Management, Hefei University of Technology, Hefei, 230009, PR China.
E-mail address: minkong@ahnu.edu.cn (M. Kong).

https://doi.org/10.1016/j.engappai.2024.108487
Received 25 October 2023; Received in revised form 8 April 2024; Accepted 19 April 2024
Available online 26 April 2024

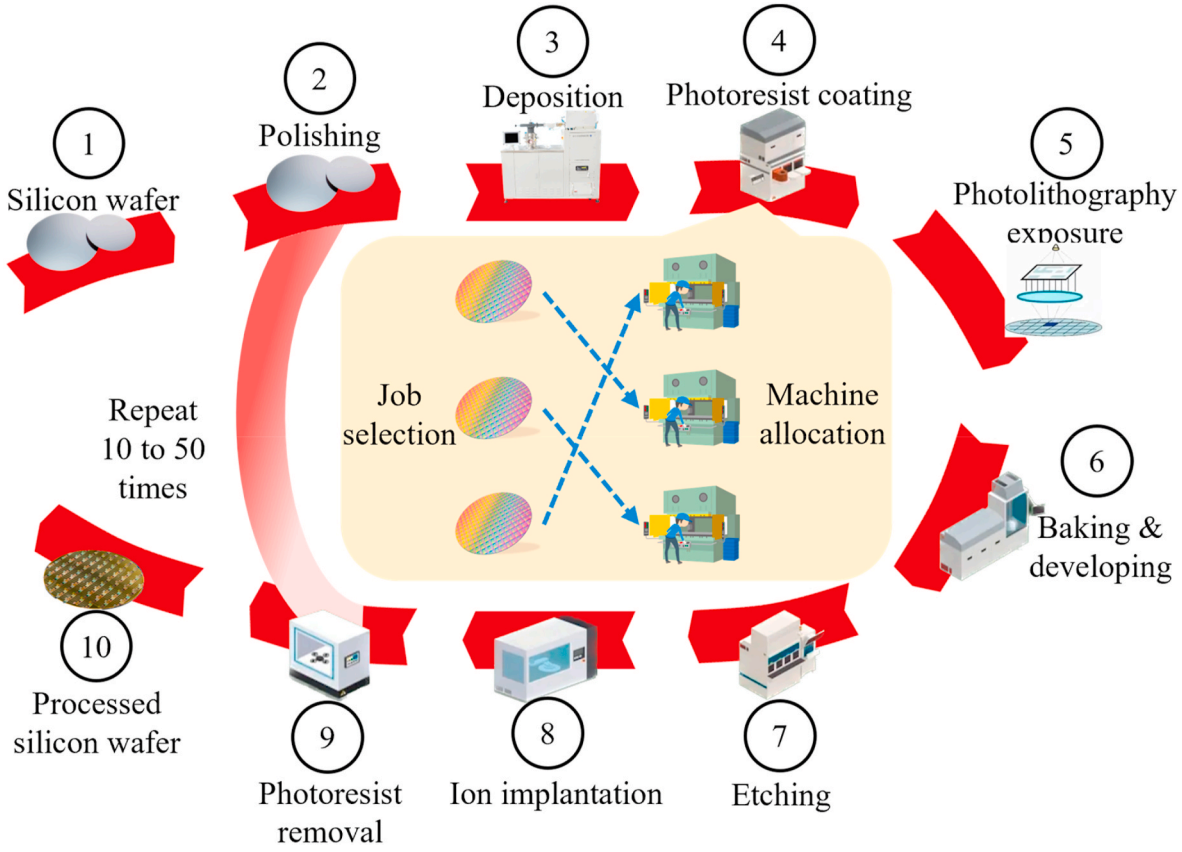

Fig. 1. The dynamic flexible job shop scheduling problem in the silicon wafer processing.

goal of the scheduling problem is to assign each job (wafer) to a machine to reduce the makespan or maximize equipment utilization under limitations such as machine availability, processing time, and operation precedence (Fathollahi-Fard et al., 2021; Lu et al., 2017).

Real-world production scheduling is often dynamic and unpredictable due to various random events such as machine failure, unexpected processing times, order cancellation, and stochastic arrival of urgent orders (Bazargan-Lari and Taghipour, 2022; Fathollahi-Fard et al., 2024; Ferreira et al., 2022). Consider the insertion of urgent jobs as an example. The insertion of urgent jobs has far-reaching consequences for existing production schedules and resource allocations in a chip manufacturing plant. They cause disruptions in the original production scheduling and planning, requiring scheduled jobs to be postponed or rescheduled. Resources such as machines, personnel, and raw materials must be diverted from other tasks in order to handle urgent jobs, which may result in a shortage of resources on other production lines, affecting the schedule and quality of non-urgent jobs. Prioritizing urgent jobs may cause congestion on the production line, especially if resources are limited, affecting not only the completion of urgent jobs but also other jobs in the queue. The consequences include missed delivery deadlines, higher production costs, and lower productivity. To address this challenge, researchers created dynamic flexible job shop scheduling problem (DFJSP) models for real-time job shop scheduling. DFJSP models consider the dynamic nature of the production environment and allow for the adjustment of scheduling schemes in response to changes in the shop floor information (Wang et al., 2020). The application of dynamic flexible job shop scheduling in memory chip manufacturing can help manufacturing companies meet the challenges of diverse production demands, fast lead times, equipment and process flexibility, and optimal resource utilization. This experience can be applied to other manufacturing environments as well. For example, in automotive manufacturing, electronic equipment manufacturing, pharmaceutical manufacturing, and food processing, reasonable dynamic flexible job shop scheduling can likewise optimize production planning, improve production efficiency, reduce production costs, and thus improve the flexibility and efficiency of the entire production chain (Sun et al., 2021).

In uncertain production environments, scheduling systems need to become more intelligent and adaptive to cope with various temporary changes and uncertainties that occur during the production process. In recent years, research in the field of uncertain production scheduling has made significant progress. Fathollahi-Fard et al. (2024) introduced a comprehensive optimization model that addresses sustainability and uncertainty concerns within the context of a distributed permutation flow-shop scheduling problem. Wei et al. (2023) dealt with a dynamic flexible job shop scheduling problem considering machine breakdowns. An integer programming model with production efficiency and stability as objectives was established. The game theory method was used to balance the two objectives. Zhu et al. (2023) proposed a new model of the distributed flexible job-shop scheduling problem (DFJSPI) – considering operation inspection in the distributed flexible job-shop scheduling problem. A modified memetic algorithm was designed to solve the DFJSPI. Rohaninejad et al. (2023) developed a scheduling and rescheduling methodology for the simultaneous lot-sizing and job-shop scheduling problem, taking into account sequence-dependent setup time and capacity constraints. A new adjustable model based on satisfiability modulo theories was developed to address the uncertainty in demand and processing time. Li et al. (2023a) investigated the order assignment and scheduling problem with direct distribution under uncertainty. Proposed a novel MILP model with a predetermined order processing sequence.

The literature on DFJSP is extensive, with numerous techniques and algorithms developed. One common approach is to use heuristics and metaheuristics to find good solutions quickly, without relying on exact

Table 1
Existing RL methods for DFJSP.

Work	Dynamic Events	Objective	Algorithm
Luo (2020)	New job insertions	Total tardiness	DQN
Luo et al. (2021)	New job insertions	Total weighted tardiness rate Average machine utilization	THDQN
Li et al. (2022)	Insufficient transportation resources	Completion time Total energy consumption	HDQN
Liu et al. (2022)	Constant job arrivals	Cumulative tardiness	DDQN
Zhang et al. (2022b)	Order insertions Machine failure	Time, cost, and resource utilization	PPO
Johnson et al. (2022)	Dynamic job arrivals	Makespan	DDQN
Chang et al. (2022)	Random job arrivals	Penalties for earliness and tardiness	DDQN
Oh et al. (2022)	Machine failure Setup change	Makespan	IQN
Zhang et al. (2022a)	Machine breakdowns	Completion time Robustness	CNN
Gui et al. (2023)	Due date change Dynamic job arrivals	mean tardiness	DDPG
This work	Dynamic job arrivals Urgent job insertions	Completion time Average tardiness	DDQN

mathematical models. Various metaheuristics have been proposed for DFJSP, including genetic algorithms, simulated annealing, and ant colony optimization. Zhou et al. (2020) proposed three co-evolutionary hyper-heuristics that focus on machine assignment rules and job sequencing rules to address the DFJSP. Baykasoğlu et al. (2020) developed a constructive algorithm utilizing the Greedy Randomized Adaptive Search Procedure to solve both FJSP and DFJSP. Li et al. (2021) considered various factors such as new job arrivals, machine failures, job cancellations, and variations in job processing time. They proposed a rescheduling method based on the Monte Carlo Tree Search algorithm to minimize job time and effectively handle the dynamic nature of the scheduling environment.

Some studies use mathematical programming techniques to formulate the problem as an optimization problem and find exact solutions. This approach involves modeling the problem using linear or integer programming, which can be computationally intensive for large-scale instances (Abreu et al., 2008). Some studies have also proposed hybrid approaches that combine metaheuristics with mathematical programming to balance solution quality and computational efficiency. Mokhtari and Dadgar (2015) investigated the DFJSP with maintenance concerns and built a MILP model for the solution, intending to delay the number of jobs and reduce the overall availability restriction. Ozturk et al. (2019) presented two unique ways for developing tailored priority rules for dynamic scheduling problems using simulation and gene expression programming, highlighting the importance of function and terminal set design on results. Shahgholi Zadeh et al. (2018) devised a heuristic model for addressing the DFJSP, utilizing the artificial bee colony algorithm. Their objective was to achieve near-optimal scheduling and minimize the makespan, considering task variations.

In recent years, there has been a growing interest in employing machine learning and data-driven approaches to solve DFJSP. These approaches involve using historical data and machine learning techniques to predict job arrival times and enhance productivity. Some studies have also proposed employing reinforcement learning and deep learning algorithms to optimize DFJSP solutions. Luo (2020) devised a Deep Q-Network (DQN) algorithm to tackle the DFJSP problem, incorporating new job insertion to reduce tardiness. Luo et al. (2021) introduced the Two-Hierarchy Deep Q-Network (THDQN) algorithm, optimizing both the total weighted tardiness and average machine utilization. Li et al. (2022) proposed the Hybrid Deep Q-Network (HDQN) algorithm to address the DFJSP with insufficient transportation resources, effectively handling its multi-objective nature. To achieve real-time control and solve the DFJSP, Liu et al. (2022) developed a hierarchical distributed architecture Double Deep Q-Network (DDQN) algorithm. Zhang et al. (2022b) developed a Deep reinforcement learning (DRL)-based multi-intelligent body manufacturing system, which considers dynamic job arrivals, integrated self-organizing mechanisms, and self-learning strategies. Johnson et al. (2022) devised a Multi-Agent Reinforcement Learning (MARL) system to manage robotic dynamic job arrivals within an assembly cell. Chang et al. (2022) proposed a DDQN architecture to address DFJSP with random job arrivals, updating the Q-network using an ε-greedy behavioral policy. Oh et al. (2022) solved the DFJSP problem by integrating an independent learner and an implicit quantile network in a multi-agent framework that manages high variability factors like machine failure and setup change efficiently. Considering machine failures, Zhang et al. (2022a) created a bi-objective model and proposed a two-stage algorithm based on convolutional neural networks to address the DFJSP. Gui et al. (2023) developed a novel strategy of composite scheduling actions consisting of single dispatching rules and continuous weight variables which overcame action space discreteness constraints and achieved mean tardiness optimization. Table 1 summarizes the differences between the aforementioned work and this work.

While performing production operations on the shop floor, new operations may be inserted into the current production schedule. In this case, the new job and the unstarted operations in the existing job must be rescheduled. The start times may vary from job to job and from machine to machine. Thus, the initial scheduling phase is a standard FJSP, while the scheduling problem containing the insertion of new jobs is a DFJSP. Different machines have different start times, depending on how long it takes them to complete the operation they are performing. To ensure that there is no interruption in production processing on the shop floor, the time for scheduling newly arrived jobs should be very short. In the field of memory chip manufacturing, there is a demand for the production of test chips in addition to regular memory chip production orders. These test chips have a very high urgency to the R&D process because they directly affect the design verification and quality control of new products. In the DFJSP, test chips are regarded as urgent jobs and thus need to be prioritized and scheduled to meet their stringent time requirements. Research on the DFJSP has primarily focused on one dynamic event. Zhang et al. (2022a) studied how machine breakdowns affect scheduling, while Johnson et al. (2022) considered dynamic job arrivals. However, the flexible flow job shop is a complex system where multiple random events can occur simultaneously (Yenisey and Yagmahan, 2014). The problem of minimizing DFJSP completion time and average tardiness while accounting for dynamic job arrivals and urgent job insertions has not been solved. The proposed approach aims to minimize both completion time and average tardiness, enabling real-time optimization and decision-making in DFJSP. The studied problem is significant for optimizing memory chip manufacture and other similar production problems. Solving this issue could boost memory chip manufacturing productivity and lower prices. In addition, through this prioritization strategy, manufacturers are able to ensure that there are no delays in R&D progress and accelerate the process of bringing products to market. This practice is essential for improving market responsiveness and maintaining competitiveness, reflecting the flexibility and efficiency of enterprises in the face of complex and changing market demands. The main contributions of this research are as follows.

(1) We conducted extensive research on a memory chip manufacturer and discovered that this company frequently receives orders for test chips, causing changes in the production schedule. Based on this market demand, we define the dynamic flexible job shop scheduling problem, which includes dynamic job arrivals and urgent job insertions. This problem not only reflects the

Table 2
Notations for the mathematical model.

Parameters	Description
n	Total number of jobs
m	Total number of machines
J_i	The i th job, $i = 1,2,\cdots,n$
n_i	Total number of operations belonging to the job J_i
M_k	The k th machine, $k = 1,2,\cdots,m$
$M_{i,j}$	The compatible machine set for operation $O_{i,j}$, $i = 1,2,\cdots,n, j = 1,2,\cdots,n_i$
$O_{i,j}$	The j th operation of job J_i, $i = 1,2,\cdots,n$, $j = 1,2,\cdots,n_i$
i,h	Index of jobs, $i,h = 1,2,\ldots,n$
j,g	Index of operations belonging to the job J_i and J_h, $j = 1,2,\ldots,n_i$, $g = 1,2,\ldots,n_h$
k	Index of machines, $k = 1,2,\ldots,m$
M	An infinitely large positive real number
$t_{i,j,k}$	The processing time of operation $O_{i,j}$ on machine M_k, $i = 1,2,\cdots,n, j = 1,2,\cdots,n_i, k = 1,2,\cdots,m$
A_i	The arrival time of the job J_i, $i = 1,2,\cdots,n$
E_i	The expiration time of the job J_i, $i = 1,2,\cdots,n$
Decision variables	
$S_{i,j}$	The starting time of $O_{i,j}$, $i = 1,2,\cdots,n, j = 1,2,\cdots,n_i$
$C_{i,j}$	Completion time for operation $O_{i,j}$, $i = 1,2,\cdots,n, j = 1,2,\cdots,n_i$
C_i	Completion time for the job J_i, $i = 1,2,\cdots,n$
C_{max}	Maximum completion time
T_i	The tardiness time of the job J_i, $i = 1,2,\cdots,n$
$X_{i,j,k}$	If $O_{i,j}$ is assigned to the machine M_k, then $X_{i,j,k} = 1$; Otherwise, $X_{i,j,k} = 0$
$Y_{i,j,h,g,k}$	If $O_{i,j}$ is a predecessor operation of $O_{h,g}$ in M_k, then $Y_{i,j,h,g,k} = 1$; If $O_{i,j}$ is a successor operation of $O_{h,g}$ in M_k, then $Y_{i,j,h,g,k} = 0$

industry's actual needs but also provides a new direction for theoretical research.

(2) We created a mixed integer programming model with an objective function that is a linear combination of maximum completion time and average tardiness. This model considers not only scheduling efficiency but also the effect of tardiness, and thus more accurately reflects the needs of actual production. The model includes a number of complex constraints, such as the processing order of operations, job completion time, machine processing capacity, and tardiness limits. These not only make the model more practical but also demonstrate our innovation and depth in model building.

(3) We proposed a new definition of the earliest processable time, applied it to job selection and machine assignment rules, and created new scheduling rules. More refined time management improves scheduling efficiency and practicality.

(4) We proposed a flexible dynamic scheduling algorithm model based on the double-deep Q-network. The state space we created comprehensively reflects the workshop's real-time information from multiple dimensions; the action space provides multiple decision-making strategies for the intelligent agent; and the reward function can effectively guide the intelligent agent to improve in the direction of the strategies. This not only demonstrates our research innovation in the field of intelligent scheduling algorithms, but it also offers an efficient and adaptable solution for real-world applications.

The remainder of this study is organized as follows. Section 2 establishes the mathematical model of DFJSP that accounts for dynamic job arrival and urgent job insertions. DRL background and the DDQN algorithm implementation details are in Section 3. In Section 4, we report numerical experiment findings. Finally, Section 5 provides the concluding remarks.

2. Problem formulation

In this section, the complexities of scheduling in semiconductor manufacturing environments are first explored, specifically the DFJSP. Then a mathematical model is developed that utilizes various notations to represent various aspects of the problem, including job and machine scheduling decisions. The problem and its formal representation are comprehensively presented through two subsections such as description and mathematical model.

2.1. Problem description

There are n successively arriving jobs, i.e., silicon wafers $J = \{J_1, J_2, \ldots, J_n\}$ to be processed on m machines $M = \{M_1, M_2, \ldots, M_m\}$, including photolithography machines, etching machines, grinding and polishing machines, etc. Each silicon wafer J_i consists of n_i operations, such as heat treatment quenching, photoresist deposition, and lithography, denoted by $\{O_{i,1}, O_{i,2}, \ldots, O_{i,n_i}\}$. Each operation $O_{i,j}$ requires a compatible machine from a set of machines denoted by $M_{i,j}(M_{i,j} \subset M)$.The processing time of the $O_{i,j}$ on machine M_k is represented as $t_{i,j,k}$. The completion time of the operation $O_{i,j}$ and the entire silicon wafer J_i are denoted by $C_{i,j}$ and C_i, respectively. The arrival time of the wafer J_i is A_i and the expiration time is E_i. The dynamic arrival of jobs obeys a Poisson distribution, where the number of jobs arriving in any given period has a well-defined average rate E_{ave}, and these arrival events are independent in time, reflecting the randomness between job arrival times. This configuration enhances the model's realism and enables changes to simulate various production densities and pressures. Typically, urgent jobs constitute a small proportion of the overall number of jobs in typical production conditions. Jobs labeled as urgent are given top priority in the manufacturing process. The scheduling system ensures these jobs are prioritized to meet their deadlines. The goal is to minimize both the completion time and average tardiness of all silicon wafers. In this study, we represent the objective function of the DFJSP using the equation (1). The following assumptions are made for the problem under consideration.

(1) A machine can only process one operation at a time.
(2) Operations from different jobs can be processed concurrently, but each operation within a job must be processed in accordance with its sequence.
(3) Once an operation starts, it cannot be interrupted.
(4) Jobs are classified as either urgent or normal based on their priority level, with urgent jobs taking precedence over normal jobs in the processing queue.
(5) Ignore setup and transit times between operations.

2.2. Mathematical model

Before presenting the mathematical model for the DFJSP, we provide a detailed description of the notations that will be used. These notations, which are summarized in Table 2, include variables related to jobs, machine resources, processing times, and scheduling decisions.

In real-world production, urgent jobs can increase completion time and cause tardiness, ultimately leading to financial losses. The objective of the studied problem is to minimize both the completion time and average tardiness of jobs. Previous research by Lu et al. (2017) aimed to decrease the maximum completion time while also minimizing additional resource consumption. In this study, we propose a model that integrates the objectives of minimizing completion time and average tardiness into a single optimization goal. In addition, we modified the machine's processing capacity limit constraints and added constraints related to tardiness. Based on the notations and Lu et al. (2017), the objective and restrictions of DFJSP can be stated as follows.

$$Minimize\ C_{max} + \frac{\sum_{i=1}^{n} T_i}{n} \tag{1}$$

$$\sum_{k\in M_{i,j}} X_{i,j,k} = 1, \forall i,j \tag{2}$$

$$S_{i,j} \geq A_i, \forall i,j \tag{3}$$

$$S_{i,j} + \sum_{k\in M_{i,j}} X_{i,j,k} \cdot t_{i,j,k} = C_{i,j}, \forall i,j \tag{4}$$

$$S_{i,j+1} \geq C_{i,j}, \forall i,j \tag{5}$$

$$C_{i,j+1} \geq C_{i,j} + \sum_{k\in M_{i,j+1}} X_{i,j+1,k} \cdot t_{i,j+1,k}, \forall i,j \tag{6}$$

$$C_{i,j} \geq C_{g,h} + t_{i,j,k} - M\left(2 + Y_{i,j,g,h,k} - X_{i,j,k} - X_{g,h,k}\right), \forall i,j,g,h,k \in M_{i,j} \cap M_{g,h} \tag{7}$$

$$C_{g,h} \geq C_{i,j} + t_{g,h,k} - M\left(3 - Y_{i,j,g,h,k} - X_{i,j,k} - X_{g,h,k}\right), \forall i,j,g,h,k \in M_{i,j} \cap M_{g,h} \tag{8}$$

$$C_i \geq C_{i,j}, \forall i,j \tag{9}$$

$$C_{max} \geq C_i, \forall i \tag{10}$$

$$T_i \geq C_i - E_i, \forall i \tag{11}$$

$$X_{ijk} = \begin{cases} 1, & If\ O_{ij}\ \text{is}\ assigned\ to\ M_k \\ 0, & else \end{cases} \tag{12}$$

$$Y_{i,j,h,g,k} = \begin{cases} 1, & If\ O_{ij}\ \text{is}\ processed\ in\ M_k\ \text{before}\ O_{h,g} \\ 0, & else \end{cases} \tag{13}$$

$$S_{i,j}, C_{i,j}, C_i, C_{max}, T_i \geq 0 \tag{14}$$

Objective (1) is a linear combination of the maximum completion time and the average tardiness. Constraint (2) ensures that at any time, a single operation can only be assigned to one of the available machine sets. Constraint (3) indicates that processing cannot begin until the job arrives. Constraint (4) means that the start processing time of the operation plus the processing time is equal to the completion time. Constraints (5) and (6) impose limits on the operation processing sequence for the job J_i. Constraints (7) and (8) ensure that a machine can process only one operation at a time. Constraint (9) computes the job J_i completion time, while constraint (10) computes the maximum completion time. The job's tardiness is calculated using constraint (11). Constraints (12) and (13) define the values of X_{ijk} and $Y_{i,j,h,g,k}$, respectively. These variables are used to represent assignment of operations and precedence between operations. Constraint (14) ensures that all variables are non-negative, ensuring the physical meaning and solvability of the model.

3. DRL algorithm for DFJSP problem

The definition of state features is given initially in this section. Next, for every rescheduling point, the compound scheduling rule (action) and the reward specification are provided in subsection 3.2. Lastly, the DDQN training methodology and network structure are presented in subsection 3.3.

3.1. Deep reinforcement learning

Reinforcement learning (RL) is often described as a Markov Decision Process (MDP), which is defined by the five-tuple (S, A, P, γ, R) (Ogryczak et al., 2013). S is the set of possible states that the agent can be in. A denotes the set of possible actions that the agent can take in each state. P represents the transition probability function that determines the agent's state transition probability when it acts. γ is the discount factor that determines the relative importance of immediate rewards versus future rewards. R denotes the agent's immediate reward for a given action in a state (Zhang et al., 2023b). The RL agent interacts with an environment, taking actions from an initial state s and transitioning to a new state s'. In this process, the environment provides the agent with an immediate reward r. The agent aims to identify the optimal policy $\pi^*(a|s)$ that maximizes cumulative reward over time. The optimal policy determines the best state action to maximize the expected reward (Kaelbling et al., 1996).

Q-learning is a common RL approach (Wang et al., 2023). The Q-Learning technique, an off-policy temporal difference (TD) algorithm, defined by Equation (15) where $\alpha\epsilon[0,1]$ is a learning rate and $max_a Q(s',a')$ is the ideal future value estimate (Karimi-Mamaghan et al., 2023). By contrast, State-Action-Reward-State-Action (SARSA) is an on-policy TD control mechanism like Q-Learning. The SARSA method is characterized by using the next action.

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma\ max_{a'} Q(s',a') - Q(s,a)] \tag{15}$$

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)] \tag{16}$$

SARSA and Q-Learning diverge in terms of their update equations (Equations (15) and (16)). SARSA utilizes a tuple (s,a,r,s',a') as training data, which is derived sequentially during the update process. In contrast, Q-learning uses a' is solely for estimation and does not consider it during updates.

Deep Q-Networks (DQN) train neural networks using reinforcement learning and nonlinear value functions to efficiently manage information complexity. A convolutional neural network approximates the Q-value of all actions in each state in DQN (Mnih et al., 2015). The network uses gradient descent to update weights θ_t to minimize the loss function. The loss function is the mean-squared error between the predicted and target Q-values, calculated using the Bellman equation in Equation (17). R_t represents the reward at time t, γ is the discount factor, s_{t+1} is the next state, and θ_t^- represents the weights of a separate target network used to calculate the target Q-value. The target network is updated every τ step from the online network, stabilizing the learning process.

$$Y_t^{DQN} = R_t + \gamma\ max_a Q\left(s_{t+1}, a; \theta_t^-\right) \tag{17}$$

One of the main issues with DQN is their tendency to overestimate Q-values, which can lead to suboptimal or unstable policies. DDQN addresses this by decoupling the action selection and evaluation processes, using separate Q-networks for each. DDQN uses one network to select the optimum action based on its estimated Q-value and another to evaluate it (Zhang et al., 2022c). This improves Q-value estimates and decreases overestimation. The update rule for DDQNs can be expressed using two Q-functions with different weights, as shown in Equation (18).

$$Y_t^{DDQN} = R_t + \gamma Q\left(s_{t+1}, argmax_a Q(s_{t+1}, a; \theta_t); \theta_t^-\right) \tag{18}$$

3.2. Definition of DRL problem

To solve the proposed DFJSP problem using DDQN, we must represent the problem in a format compatible with reinforcement learning. At each rescheduling point t, the DRL format requires state $s(t)$, action $a(t)$, and reward $r(t)$ (Mnih et al., 2015). There are several difficulties in solving the DFJSP using the DDQN algorithm. First, the complexity of the state representation is a key feature. The state $s(t)$ represents the current state information of the job shop, including the current job assignments, machine statuses, and any other relevant information. The DDQN algorithm is able to efficiently deal with this high-dimensional, complex state space, extracting key features to guide the decision-making. Second, the algorithm needs to dynamically decide the action $a(t)$ at each rescheduling point to cope with the dynamic arrival of jobs and the insertion of urgent jobs, which requires high flexibility and real-time responsiveness. The design of the reward function $r(t)$,

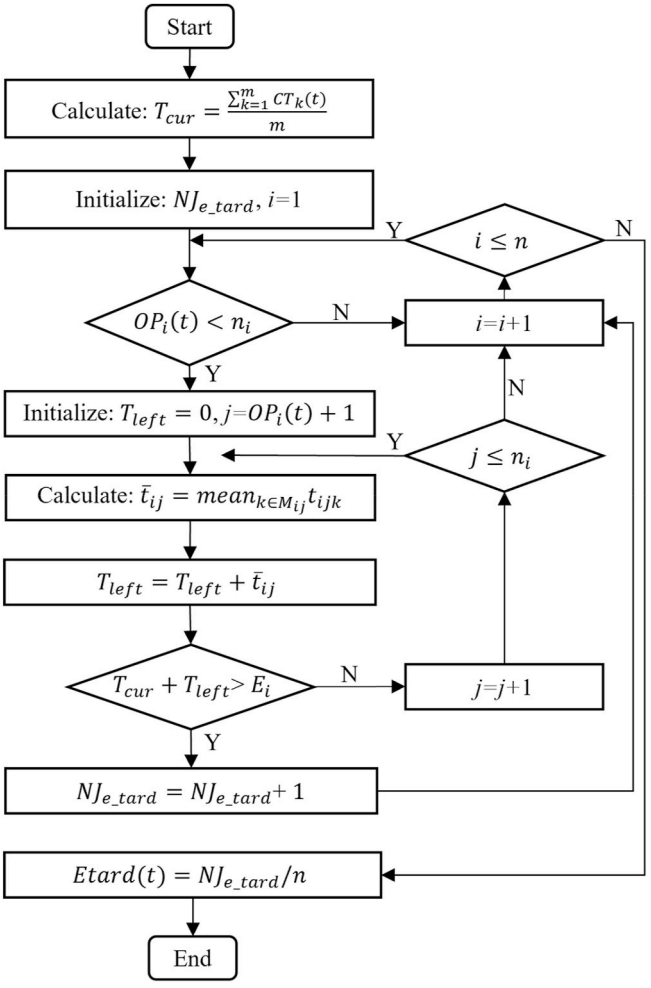

Fig. 2. Calculation process of the estimated tardiness rate $Etard(t)$.

Table 3
Job selection rules.

Number	Job selection rule
(1)	$J_i \leftarrow argmin_{i \in UCJ_U(t)/UCJ_N(t)} \left\{ E_i - \frac{\sum_{k=1}^{m} CT_k(t)}{m} \right\}$
(2)	$J_i \leftarrow argmax_{i \in UCJ_U(t)/UCJ_N(t)} \{ \sum_{j=OP_i(t)+1}^{n_i} mean_{k \in M_{ij}} t_{ij,k} \}$
(3)	$J_i \leftarrow argmin_{i \in UCJ_U(t)/UCJ_N(t)} \{ S_{i,OP_i(t)+1,k} \; for \; k = 1 : m \}$

which evaluates the quality of the decision through predefined performance metrics, is also a major challenge. Requiring the algorithm to not only optimize a single metric but also consider multiple performance metrics in a comprehensive manner. DDQN introduces two independent Q-networks for action selection and action evaluation, respectively. This helps to mitigate the problem of overestimation in DQN, reduces target volatility, and improves learning stability (Van Hasselt et al., 2016). DDQN employs an experience replay mechanism (Zhang et al., 2023a), in which previous experience is stored in an experience replay buffer and randomized samples are drawn from it for training. This helps to break the temporal correlation between data and improves the algorithm's sample efficiency. The DDQN algorithm combines deep learning and reinforcement learning, and its ability to learn and adapt in parallel enables it to perform well in dynamic environments while focusing on long-term performance optimization to continuously improve overall performance through long-term learning.

3.2.1. Definition of state features

At the rescheduling point t, DRL-based algorithms extract many state features to represent the current environment. In the study of DRL-based production scheduling problems, state features often include the number of jobs in the shop floor, mean processing time of current operations (MOPT) (Shahrabi et al., 2017), percentage of completion of jobs, total effective processing time of jobs, machine efficiency, and others. Including all job information as state features can cause problems. Large numerical values may overwhelm smaller ones for certain features (Shiue et al., 2018), which may bias decision-making. The state space might rapidly expand as new jobs arrive, making it difficult to find an optimal solution. For these issues, the proposed DDQN algorithm utilizes 8 ratio parameters as state features, with values in the [0,1] interval.

At time t, we define $CT_k(t)$ as the completion time of the previous operation performed on the machine M_k, $CT_i(t)$ as the completion time of the last operation of the job J_i, and $OP_i(t)$ as the current number of operations completed for the job J_i.

(1) The average utilization of machines, $\overline{U}_m(t)$, is defined by Equation (19):

To determine the machines' utilization rate, we first add up the cumulative processing time for each machine and divide it by $CT_k(t)$. Next, we sum up the utilization rates for all machines and take their average to obtain $\overline{U}_m(t)$.

$$\overline{U}_m(t) = \frac{\sum_{k=1}^{m} \left(\frac{\sum_{i=1}^{n} \sum_{j=1}^{OP_i(t)} t_{i,j,k} X_{i,j,k}}{CT_k(t)} \right)}{m} \tag{19}$$

(2) The standard deviation of the average machine usage, $U_{std}(t)$, is defined by Equation (20):

Calculate the utilization rate of each machine in the same way as in (1). Next, calculate the standard deviation of this dataset to obtain $U_{std}(t)$.

$$U_{std}(t) = \sqrt{\frac{\sum_{k=1}^{m} \left(\frac{\sum_{i=1}^{n} \sum_{j=1}^{OP_i(t)} t_{i,j,k} X_{i,j,k}}{CT_k(t)} - \overline{U}_m(t) \right)^2}{m}} \tag{20}$$

(3) Operation completion rate, $CRO(t)$, is defined by Equation (21):

The total number of completed operations is obtained by summing the number of completed operations for each job J_i, and then dividing by the sum of the number of operations for all jobs to obtain $CRO(t)$.

$$CRO(t) = \frac{\sum_{i=1}^{n} OP_i(t)}{\sum_{i=1}^{n} n_i} \tag{21}$$

(4) Estimated tardiness rate, $Etard(t)$.

T_{cur} represents the average completion time of the most recent operations performed on all machines at time t. T_{left} represents the expected processing time remaining for the job J_i. Additionally, NJ_{e_tard} represents the estimated number of tardy jobs. If the sum of T_{cur} and T_{left} exceeds the expiration time E_i, it is anticipated that the job j_i will be late. The estimated tardiness rate $Etard(t)$ is calculated by dividing the estimated

number of late jobs by the total number of jobs. The computation method for this is described in Fig. 2.

(5) Actual tardiness rate, $Atard(t)$.

NJ_{a_tard} represents the total number of tardy jobs. If the completion time of the latest operation currently performed for the job J_i exceeds the expiration time E_i, then job J_i is considered late. The actual tardiness rate $Atard(t)$ is determined by dividing the number of late jobs by the total number of jobs. The computation method for this is described in Algorithm 2.

Algorithm 2. Calculation process of the actual tardiness rate $Atard(t)$.

Input: $OP_i(t)$, E_i
Output: $Atard(t)$
$NJ_{a\_tard} \leftarrow 0$
for $i = 1{:}n$ **do**
  if $OP_i(t) < n_i$ **then**
    if $C_{i,OP_i(t)} > E_i$ **then**
      $NJ_{a\_tard} \leftarrow NJ_{a\_tard} + 1$
    end if
  end if
end for
$Atard(t) \leftarrow NJ_{a\_tard}/n$
Return $Atard(t)$

(6) Completion rate of urgent jobs, $CUJ(t)$.

The variable $UJ_N(t)$ represents the initial number of urgent jobs plus the number of urgent jobs inserted before time t, while $UJ_C(t)$ represents the number of urgent jobs that have been processed by time t. The variable u_i takes on the value of either 0 or 1, where 0 indicates that job J_i is a normal job and 1 indicates that it is an urgent job. The completion rate of urgent jobs, denoted by $CUJ(t)$, is calculated by dividing $UJ_C(t)$ by $UJ_N(t)$. The computation method for this is described in Algorithm 3.

Algorithm 3. Calculation process of the completion rate of urgent jobs $CUJ(t)$.

Input: $OP_i(t)$, U_i
Output: $CUJ(t)$
$UJ_N(t) \leftarrow 0$
$UJ_C(t) \leftarrow 0$
for $i = 1{:}n$ **do**
  if $u_i = 0$ **then**
    continue
  else
    $UJ_N(t) \leftarrow UJ_N(t) + 1$
    if $OP_i(t) = n_i$ **then**
      $UJ_C(t) \leftarrow UJ_C(t) + 1$
    end if
  end if
end for
$CUJ(t) \leftarrow \frac{UJ_C(t)}{UJ_N(t)}$
Return $CUJ(t)$

(7) Completion rate of jobs, $CRJ(t)$.

The variable $UJ_C(t)$ represents the number of jobs that have been processed by time t. The completion rate of jobs, denoted by $CRJ(t)$, is calculated by dividing $UJ_C(t)$ by n. The computation method for this is described in Algorithm 4.

Algorithm 4. Calculation process of the completion rate of jobs $CRJ(t)$

Input: $OP_i(t)$
Output: $CRJ(t)$
$UJ_C(t) \leftarrow 0$
for $i = 1:n$ **do**
  if $OP_i(t) = n_i$ **then**
    $UJ_C(t) \leftarrow UJ_C(t) + 1$
  end if
end for
$CRJ(t) \leftarrow \frac{UJ_C(t)}{n}$
Return $CRJ(t)$

(8) Completion rate of urgent operations, $CUO(t)$.

The variable $UO_N(t)$ represents the total number of operations for all urgent jobs, while $UJ_C(t)$ represents the number of operations from urgent jobs that have been processed by time t. The completion rate of urgent operations, denoted by $CUO(t)$, is calculated by dividing $UO_C(t)$ by $UO_N(t)$. The computation method for this is described in Algorithm 5.

Algorithm 5. Calculation process of the completion rate of urgent operations $CUO(t)$

Input: $OP_i(t)$, U_i, n_i
Output: $CUO(t)$
$UO_N(t) \leftarrow 0$
$UO_C(t) \leftarrow 0$
for $i = 1:n$ **do**
  if $u_i = 0$ **then**
    continue
  else
    $UO_N(t) \leftarrow UO_N(t) + n_i$
    $UO_C(t) \leftarrow UO_C(t) + OP_i(t)$
  end if
end for
$CUO(t) \leftarrow \frac{UO_C(t)}{UO_N(t)}$
Return $CUO(t)$

3.2.2. Definition of action features

To execute an operation, the agent selects a waiting job and assigns its next operation to an available machine based on the current state. We propose some scheduling rules to solve the subproblems of operation selection and machine allocation. In operation selection, we consider factors such as the job's expiration time, remaining processing time, and the earliest time that processing can begin. For machine allocation, we consider the operation completion time and workload.

At time t, the maximum number of idle times of the machine M_k is equal to the number of scheduled operations minus one, which is represented as L_k. The length of any idle time on the machine M_k, denoted as $idle_k^l (l = 0,1,2,\cdots,L_k)$, is equal to the start time of the latter of the two adjacent operations on the machine M_k minus the end time of the previous operation. The start time of the idle time of the machine M_k is denoted as $idle_s^l k$.

Lemma 1. *At the rescheduling point t, for machine M_k which can process operation $O_{i,j}$, if $idle_k^l < t_{i,j,k}$, then the earliest processable time for operation $O_{i,j}$ on that machine is $S_{i,j,k} = max\{A_i, CT_i(t), CT_k(k)\}$.*

Proof: The initiation of processing for operation $O_{i,j}$ on machine M_k necessitates the arrival of the job before its processing can commence, which is indicated by the constraint $S_{i,j,k} \geq A_i$. As per the process constraint, the operation $O_{i,j}$ can only begin processing upon the completion of its preceding operation $O_{i,j-1}$, i.e., $S_{i,j,k} \geq CT_i(t)$. The initiation of a new operation on machine M_k is constrained by the

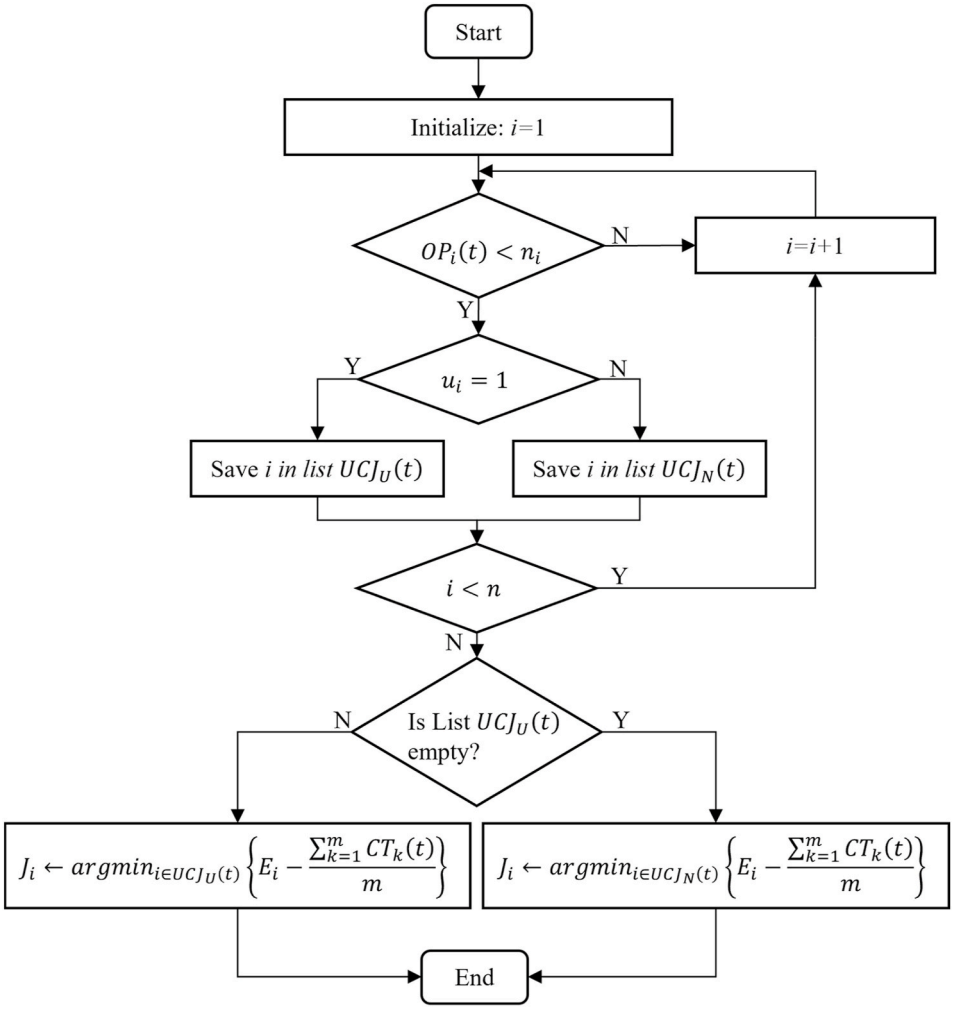

Fig. 3. Machine selection rule (1).

completion of the operation currently being processed on that machine, implying $S_{i,j,k} \geq CT_k(k)$. Consequently, the earliest feasible start time for the operation $O_{i,j}$ on machine M_k is $S_{i,j,k} = max\{A_i, CT_i(t), CT_k(k)\}$.

Lemma 2. *At the rescheduling point t, if the earliest idle time* $idle_k^l$ *exists for any machine* M_k *capable of processing operation* $O_{i,j}$*, which satisfies the conditions* $idle_s^l k \geq (A_i \vee CT_i(t))$ *and* $idle_k^l \geq t_{i,j,k}$*, then the earliest processable time for operation* $O_{i,j}$ *is* $S_{i,j,k} = idle_s^l k$.

Proof: The start time of the l th idle time, $idle_s^l k$, is greater than either A_i or $CT_i(t)$. Therefore, the earliest processable time for operation $O_{i,j}$ can be established as $S_{i,j,k} \geq idle_s^l$. Since $idle_k^l$ exceeds the real processing time of the operation $O_{i,j}$ on machine M_k, $O_{i,j}$ can be assigned to process at this idle time, making the earliest processing time for $O_{i,j}$ $S_{i,j,k} = idle_s^l k$.

Lemma 3. *At the rescheduling point t, if the earliest idle time* $idle_k^l$ *exists for any machine* M_k *capable of processing operation* $O_{i,j}$*, which satisfies the conditions* $idle_s^l k < (A_i \vee CT_i(t))$ *and* $(idle_k^l - (idle_s^l k - (A_i \vee CT_i(t))) \geq t_{i,j,k}$*, then the earliest processable time for operation* $O_{i,j}$ *is* $S_{i,j,k} = A_i \vee CT_i(t)$.

Proof: The start time of the l th idle time, $idle_s^l k$, is less than either A_i or $CT_i(t)$. Therefore, the earliest processable time for operation $O_{i,j}$ can be established as $S_{i,j,k} \geq (A_i \vee CT_i(t)) > idle_s^l k$. Since $idle_k^l$ exceeds the real processing time of the operation $O_{i,j}$ on machine M_k add the difference between $idle_s^l k$ and A_i or $idle_s^l k$ and $CT_i(t)$, $O_{i,j}$ can be assigned to process at this idle time, making the earliest processing time for $O_{i,j}$ $S_{i,j,k} = idle_s^l k$.

We developed four distinct job selection rules. $UCJ_U(t)$ and $UCJ_N(t)$ refer to incomplete urgent and normal jobs, respectively. Regardless of the job selection rule being used, an urgent job always takes precedence over a normal job when there is an unfinished urgent job. The three job

Table 4
Machine selection rules.

Number	Machine selection rule
(1)	$M_k \leftarrow argmin_{k \in M_{i,j}} \{S_{i,OP_i(t)+1,k} + t_{i,j,k}\}$
(2)	$M_k \leftarrow argmin_{k \in M_{i,j}} \{\sum_{i=1}^{n} \sum_{j=1}^{OP_i(t)} t_{i,j,k} X_{i,j,k}\}$

Table 5
Compound scheduling rules.

Number	Job selection rule	Machine selection rule
(1)	Job selection rule (1)	Machine selection rule (1)
(2)	Job selection rule (1)	Machine selection rule (2)
(3)	Job selection rule (2)	Machine selection rule (1)
(4)	Job selection rule (2)	Machine selection rule (2)
(5)	Job selection rule (3)	Machine selection rule (1)
(6)	Job selection rule (3)	Machine selection rule (2)
(7)	Job selection rule (4)	Machine selection rule (1)
(8)	Job selection rule (4)	Machine selection rule (2)
(9)	Randomly choose an uncompleted job J_i	Randomly choose a machine M_k

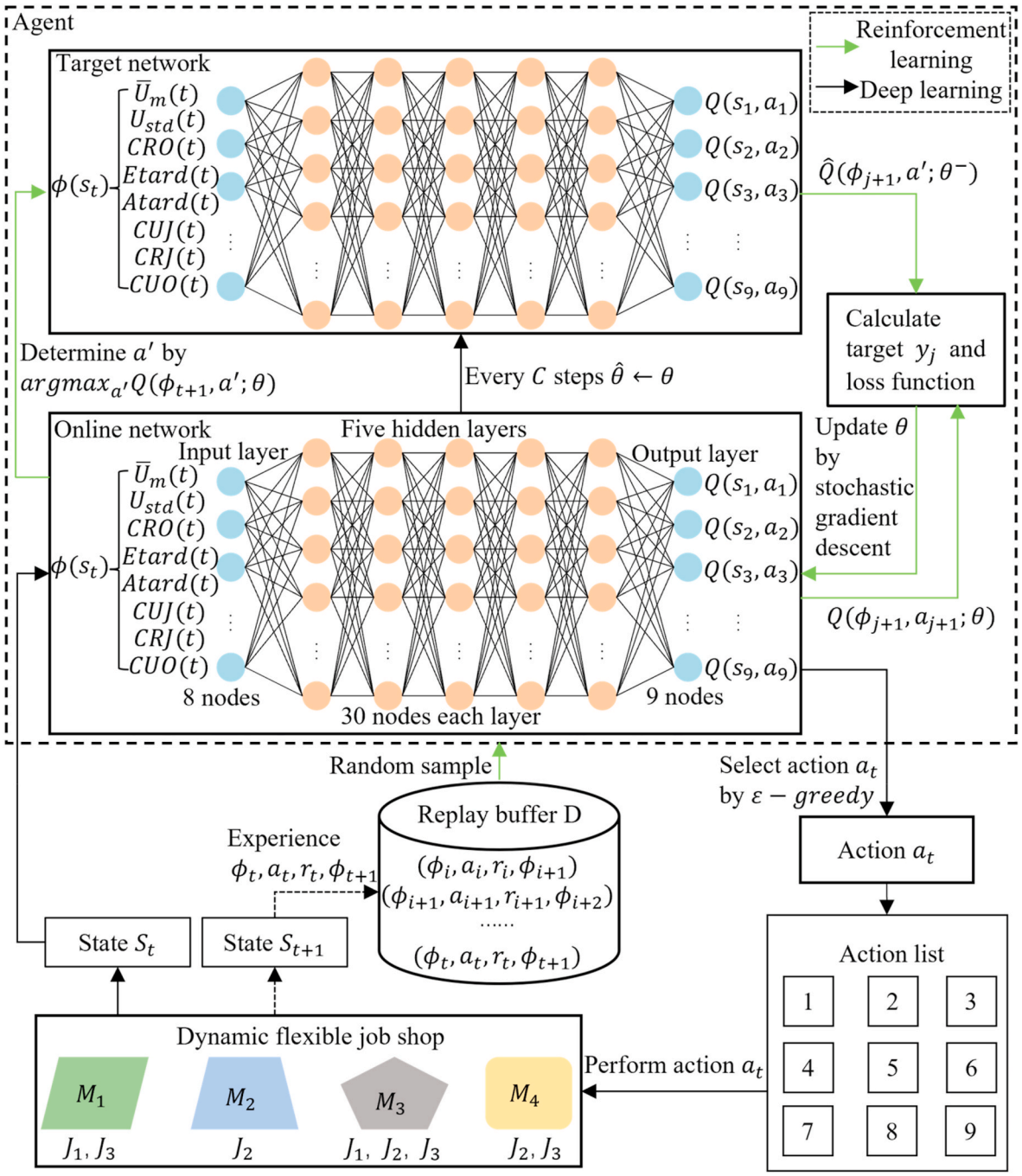

Fig. 4. The model structure of solving the DFJSP using the DDQN.

selection rules are shown in Table 3. Specifically, the job selection rule (3) builds upon the principles established in **Lemma 1**, **Lemma 2**, and **Lemma 3**. We have introduced job selection rule 4, which is comprehensively outlined in Algorithm 6.

Rule (1): We use $CT_k(t)$ to denote the completion time of the last operation on the machine M_k at time t. Job selection rule (1) prioritizes the next operation of the job $O_{i,OP_i(t)+1}$ that is closest to its expiration time in either $UCJ_U(t)$ or $UCJ_N(t)$. Fig. 3 provides a detailed overview of the job selection process.

Rule (2): $M_{i,j}$ is the compatible machine set for operation $O_{i,j}$. In job selection rule (2), the next operation $O_{i,OP_i(t)+1}$ for the job with the highest remaining estimated processing time in either $UCJ_U(t)$ or $UCJ_N(t)$ is chosen.

Rule (3): The earliest possible start time for the processing of the next operation, $O_{i,OP_i(t)+1}$, on all machines is denoted by $S_{i,OP_i(t)+1,k}$. Under job selection rule (3), the job with the earliest startable processing time in either $UCJ_U(t)$ (t) or $UCJ_N(t)$ is selected.

Rule (4): The set $UCJ_{tard}(t)$ represents the estimated jobs that will be tardy, while $UCJ_{ntard}(t)$ includes the estimated jobs that will not be tardy. If $UCJ_U(t)$ contains any jobs, a random job is selected from this set. If there are no urgent jobs and $UCJ_{tard}(t)$ is not empty, select the job with the highest tardiness time from $UCJ_{tard}(t)$. Otherwise, choose the job with the earliest expiration time from $UCJ_{ntard}(t)$.

Algorithm 6. Procedure of job selection rule 4

$T_{cur} = \frac{\sum_{k=1}^{m} CT_k(t)}{m}$
for $i = 1{:}n$ **do**
  if $OP_i(t) < n_i$ **then**
    $T_{left_i} \leftarrow 0$
    for $j = OP_i(t) + 1{:}n_i$ **do**
      $\bar{t}_{i,j} \leftarrow mean_{k \in M_{i,j}} t_{i,j,k}$
      $T_{left_i} \leftarrow T_{left_i} + \bar{t}_{i,j}$
    end for
  end if
end for
$UCJ_U(t) \leftarrow \{i | OP_i(t) < n_i \ and \ U_i = 1\}$
$UCJ_{tard}(t) \leftarrow \{OP_i(t) < n_i \ and \ T_{cur} + T_{left_i} > E_i\}$
$UCJ_{ntard}(t) \leftarrow \{OP_i(t) < n_i \ and \ T_{cur} + T_{left_i} \leq E_i\}$
if $UCJ_U(t)$ not empty **then**
  Randomly choose an uncompleted job J_i from $UCJ_U(t)$
else if $UCJ_{tard}(t)$ not empty **then**
  $J_i \leftarrow argmax_{i \in UCJ_{tard}(t)} \{T_{cur} + T_{left_i} - E_i\}$
else
  $J_i \leftarrow argmin_{i \in UCJ_{ntard}(t)} \{E_i - T_{cur} + T_{left_i}\}$
end if

Table 6
Parameters of various production arrangements.

Parameter	Value
Number of machines (m)	{10, 30, 50}
Number of initial jobs (n_{ini})	15
Number of newly added jobs (n_{add})	{25, 50, 75, 100}
The average value of exponential distribution between two successive job arrivals (E_{ave})	{30, 50, 100}
Number of operations in a job (n_i)	U [1, 20]
Job type (u_i)	B (1, 0)
Expiration time urgency factor ($ETUF$)	{1.5(u_i = 1), 1 (u_i = 0)}
Processing time of an operation on a machine ($t_{i,j}$)	U [10, 50]

We designed two machine selection rules for the job scheduling process. Here, i denotes the sequence number of the job, as determined by the rules presented in Table 3, and j represents the sequence number of the next operation of the job J_i. These two machine selection rules are listed in Table 4.

Rule (1): $S_{i,OP_i(t)+1,k}$ represents the earliest startable processing time for the job J_i on Machine M_k. In machine selection rule (1), we take into account both the availability of the machine and the processing time of the subsequent operation when selecting the machine for processing. Ultimately, the machine with the minimum completion time for the operation $O_{i,OP_i(t)+1}$ is chosen.

Rule (2): The workload of machine k is calculated as the sum of the processing times for all operations that have been processed on the machine before time t. The job J_i is dispatched to the machine with the lowest workload.

Eight compound scheduling rules were created by combining job and machine selection rules. We also introduced rules for randomly selecting jobs and machines to boost scheduling flexibility. These nine compound scheduling rules are in Table 5.

Table 7
Hyperparameters and values.

Hyperparameter	Value
Replay buffer size (N)	2000
Minibatch size	64
Replay start size	1000
Epsilon (ε)	Linear drop from 1 to 0.01
Discount rate (γ)	0.95
Learning rate (η)	0.001
Target network update frequency (C)	200

3.2.3. Definition of reward

Because the goal of production scheduling is to decrease completion time and average tardiness while the objective of the DDQN algorithm is to maximize the cumulative reward, the direction of cumulative reward

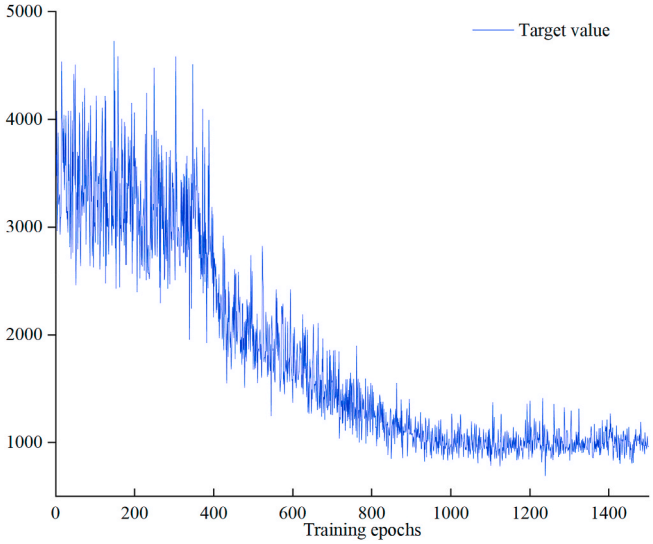

Fig. 5. Target value at each training epoch.

increase must align with the direction of production scheduling objective optimization. To facilitate efficient learning by the agent, we have developed a heuristic immediate reward function based on Equation (22).

$$r_t = r_{1_t} + r_{2_t}$$

$$r_{1_t} = \begin{cases} maxCT_k(t) - C_{i,OP_i(t)+1}, maxCT_k(t) \geq C_{i,OP_i(t)+1}, \\ 0, else \end{cases} \tag{22}$$

$$r_{2_t} = \begin{cases} D_i - CT_i(t) - \sum_{j=OP_i(t)+1}^{n_i} \overline{t_{i,j,k}}, D_i \geq CT_i(t) + \sum_{j=OP_i(t)+1}^{n_i} \overline{t_{i,j,k}}, \\ 0, else \end{cases}$$

This function aims to improve agent learning efficiency. If the completion time of the operation $O_{i,OP_i(t)+1}$ is less than the current maximum completion time $max\ CT_k(t)$, the immediate reward r_{1t} will be positive. Furthermore, if the action a_t optimizes tardiness by reducing expected completion time, the immediate reward r_{2t} will be positive, leading to an increase in the cumulative reward r_t. When the schedule optimizing objective declines, each DDQN algorithm iteration yields a more immediate reward. If the operation $O_{i,OP_i(t)+1}$ completes later than the current maximum completion time and the tardiness becomes larger, the immediate reward is zero. The cumulative reward is negatively correlated with the scheduling optimizing objective. The immediate reward function evaluates the action a_t selected at each rescheduling point t and transforms the scheduling objective problem into a cumulative reward objective problem, improving the agent's understanding of complex policies.

3.3. Framework of DDQN

The final Q-network obtained from the DDQN algorithm training is used for making predictions in a production environment. The proposed DDQN algorithm flowchart is shown in Fig. 4.

The training approach is based on the DDQN architecture. During training, the decision point 't' is defined as each occurrence of a new job arrival or completion of an operation. Following the aforementioned procedures, the general framework of the DDQN algorithm can be outlined as follows:

Algorithm 7. The DDQN algorithm training method

Initialize replay buffer D with the size of N
Initialize online action-value function network Q with random weights (θ)
Initialize target action-value function network $\hat{Q}$ with random weights ($\theta^- = \theta$)
for episode = 1: L **do**;
  for t = 1: T **do**;
    Select a_t with ε-greedy method
    Otherwise, select $a_t = argmax_a Q(\emptyset_t, a; \theta)$
    Take action a_t, observe the next state s_{t+1} and r_t, $\emptyset_{t+1} = \phi(s_{t+1})$
    Store data $(\emptyset_t, a_t, r_t, \emptyset_{t+1})$ in the replay buffer D
    Sample random minibatch of data $(\emptyset_j, a_j, r_j, \emptyset_{j+1})$ from the replay buffer
    Calculate $y_j = \begin{cases} r_j, & terminate\ at\ s_{j+1} \\ r_j + \gamma \hat{Q}(\emptyset_{j+1}, argmax_{a'} Q(\emptyset_{j+1}, a'; \theta); \theta^-) & otherwise \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\emptyset_j, a_j; \theta))^2$ concerning θ
    Every C step reset $\theta^- = \theta$
  end for
end for

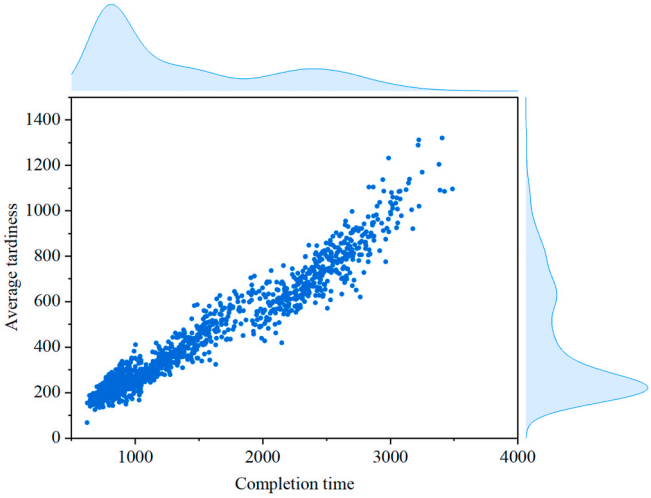

Fig. 6. Tardiness and completion time for all the instances.

Table 8
Target value by DDQN and Gurobi.

E_{ave}	Gurobi	Runtime/s	DDQN	Runtime/s
30	339.80	45.30	385.65	0.27
	349.80	67.51	417.36	0.40
	316.40	102.34	340.60	0.50
	376.00	266.50	404.40	1.20
50	376.00	52.04	417.11	0.30
	404.60	72.26	454.94	0.42
	342.60	105.23	367.00	0.51
	494.60	281.15	515.80	1.23
100	590.40	53.60	608.03	0.33
	968.80	165.05	980.20	0.92
	491.40	93.22	498.00	0.45
	646.40	345.37	652.00	1.57

4. Numerical experiment

The problem instance was generated based on a study of a local memory chip manufacturing company. In the real-world production,

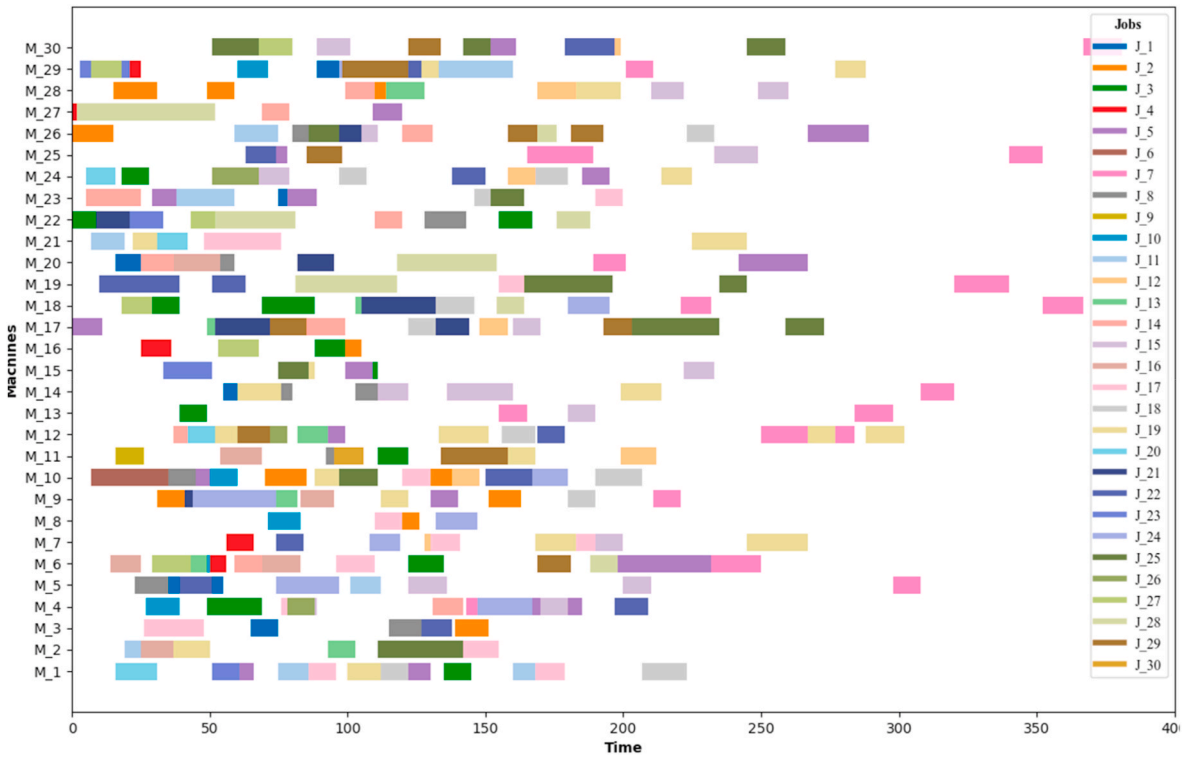

Fig. 7. The Gantt Chart of an instance with 30 jobs and 30 machines.

Table 9
An example of the DFJSP.

Job	Operation	Alternative machines with processing time		
		M_1	M_2	M_3
J_1	O_{11}	4	3	5
	O_{12}	–	6	8
J_2	O_{21}	10	–	8
	O_{22}	5	6	4
J_3	O_{31}	7	10	–
	O_{32}	3	4	5

Operation	O_{11}	O_{12}	O_{21}	O_{22}	O_{31}	O_{32}
$OS(O_{ij})$	1	2	3	6	4	5
$MS(O_{ij})$	M_1	M_2	M_3	M_2	M_1	M_2

Fig. 8. Operator coding method.

dynamic job arrivals and urgent job insertions have a significant impact on productivity, with the arrival of new jobs or the completion of operations triggering rescheduling. We consider that there are initially 15 jobs present on the shop floor. Job arrivals follow a Poisson distribution (Li et al., 2021; Ramasesh, 1990), with an arrival interval that has a negative exponential distribution with an average rate of E_{ave}. The operation number (n_i) and process time ($t_{i,j}$) for the j th operation ($O_{i,j}$) of J_i are determined by a uniform distribution, and the job type (u_i) is determined by a 0–1 distribution. The value 1 indicates that the job J_i is considered urgent, while 0 represents a normal job. The expiration time urgency factor (*ETUF*) is 1 for the expiration time of an urgent job and 1.5 for the expiration time of a normal job. The ratio of urgent jobs to normal jobs is 1:9. For a job J_i with n_i operations arriving at the time A_i, its expiration time E_i can be calculated as $E_i = A_i + (\sum_{j=1}^{n_i} \bar{t}_{i,j}) \cdot ETUF$. The parameter settings are shown in Table 6.

The experiments were conducted on a computer equipped with an Intel Core i7-11700F 2.50 GHz processor, and 16 GB RAM. The computer was running the Windows 10 operating system and had software environments configured for Python 3.9.13 and Pytorch 2.0.1.

4.1. Training specifics

The DDQN algorithm was trained using a simulation of a dynamic flexible job shop consisting of 30 machines. The simulation included 75 dynamic job arrivals, with an average value of the exponential distribution between two successive job arrivals (E_{ave}) set at 50. Table 7 lists hyperparameters and values. Fig. 5 displays the target value obtained from the initial 1500 epochs as computed by the DDQN algorithm developed in this study. The observed curves illustrate a gradual decline in the target value, finally reaching a plateau, accompanied by a reduction in volatility as the number of training steps grows. The rate of learning experiences minimal fluctuations following the completion of the 1000th period. This demonstrates that the scheduling agent acquires knowledge of the suitable dispatching rules based on variations in the production states, hence enhancing its adaptability in resolving the DFJSP.

The scatter plots in Fig. 6 illustrate the two optimization objective circumstances of the DDQN algorithm for each instance, depicting completion time and average tardiness. It is evident from the plots that there exists a linear correlation between completion time and average tardiness, suggesting the feasibility of simultaneous optimization. The distribution curves on the upper and right sides of the scatter plot reveal a concentration of points in the lower left corner, indicating that the DDQN algorithm effectively learns suitable compound scheduling rules to optimize both completion time and average tardiness, ultimately yielding improved scheduling solutions.

4.2. Comparisons with gurobi

The proposed MIP model is solved using the Gurobi solver under some small-scale static scenarios before comparing the DDQN algorithm with other methods. Unlike the general scenarios that contain urgent job

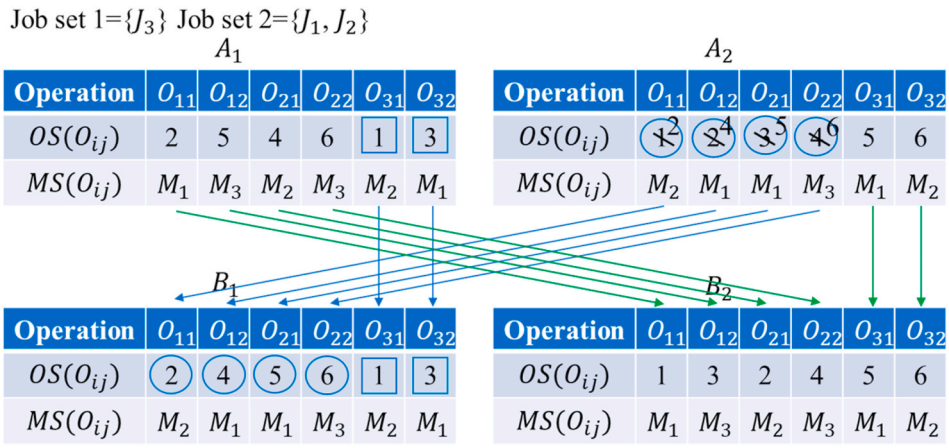

Fig. 9. The first operator transformation.

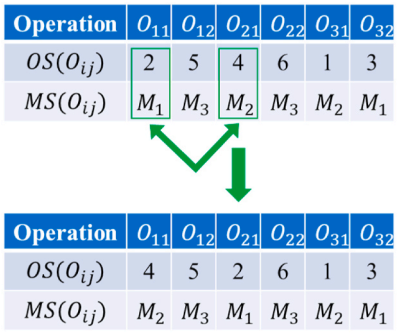

Fig. 10. The second operator transformation.

insertions, the job types in the static scenarios are all normal jobs. The MIP model can only be solved by Gurobi when there is no urgent job insertion. The target value and solution time are averaged separately. The number of machines is randomly generated between [10, 50] and n_{add} is taken between [25, 50]. Table 8 displays the experimental outcomes. Fig. 7 shows one of Gurobi's solutions. Gurobi, a well-known commercial solver, can find solutions closer to the global optimum for standard problems in static scenarios. In contrast, DDQN, as a reinforcement learning-based method, has some balance issues between exploration and exploitation, failing to find the optimal solution. Although DDQN has a slightly lower objective value, it outperforms Gurobi in terms of solution time. Gurobi's solution times range from tens of seconds to several minutes, while DDQN's solution times are mostly under 1 s. This is critical for application scenarios that require a rapid response. In real industrial applications, particularly in environments where a large number of tasks must be processed, the solution's speed is frequently more important than minor differences in objective values.

4.3. Comparisons with other metaheuristic algorithms

To test the performance of the proposed DDQN algorithm, we conducted a comparative analysis with five other metaheuristic algorithms: Genetic Algorithm (GA) (Homayouni et al., 2014; Zhang et al., 2020), Variable Neighborhood Search Algorithm (VNS) (Lu et al., 2022), Adaptive Large Neighborhood Search Algorithm (ALNS) (Cota et al., 2019), Simulated annealing algorithm (SA) (Dehghan-Sanej et al., 2021), and Artificial Bee Colony Algorithm (ABC) (Li et al., 2023b). Metaheuristic algorithms, including GA, VNS, ALNS, SA, and ABC, belong to a class of efficient algorithms designed to address combinatorial optimization problems (Fathollahi-Fard et al., 2022; Pasha et al., 2022), specifically in the domain of job shop scheduling problems.

An example is used below to illustrate the encoding method used for the operators in this work as well as the operator transformation method. Table 9 depicts a DFJSP problem with three jobs, each of which has two operations. There are three processing machines available. Two decisions must be made in order to create a production schedule for the DFJSP. First, assign each operation to one of the available machines. Second, the operations must be ordered. The DFJSP operator is typically composed of two parts, *OS* and *MS*, which represent operation ordering and machine assignment, respectively.

Fig. 8 shows an example of the DFJSP operator. Let $MS(O_{ij})$ denote the machine used to process O_{ij}. Clearly, $MS(O_{ij})$ must be an element of the set of alternative machines for O_{ij}. The assignment orders are indicated by a sequence of operations. Let $OS(O_{ij})$ denote the order of O_{ij} in a given sequence of operations. It should be noted that the *OS* section's coding scheme is a positional list representation, with each value specifying the order of the associated items. Thus, the *OS* section in Fig. 8 can be converted into a specific sequence of operations, namely $< O_{11}, O_{12}, O_{21}, O_{31}, O_{32}, O_{21} >$. If $OS(O_{ab}) < OS(O_{cd})$, an operation O_{ab} is scheduled before O_{cd}.

The first operator transformation operation is illustrated in Fig. 2. For the OS and MS parts in Fig. 2, precedence operation crossover (POX) is used. POX is a well-known crossover operator used for sorting problems. The job set 1 represents the operation O_{ij} has the same position $OS(O_{ij})$ in A_x and B_x ($x = 1, 2$) if it belongs to J_3, and where B_x denotes the offspring solution. On the contrary, the job set 2 indicates that B_1 (B_2) inherits from A_1 (A_2) the priority relation between J_1 and J_2 operations. Fig. 3 depicts another operator transformation operation in which two mutation points on the original operator are selected to exchange their positions and generate a new operator.

The first operator transformation operation is illustrated in Fig. 9. For the OS and MS parts in Fig. 9, precedence operation crossover (POX) is used. POX is a well-known crossover operator used for sorting problems. The job set 1 represents the operation O_{ij} has the same position $OS(O_{ij})$ in A_x and B_x ($x = 1, 2$) if it belongs to J_3, and where B_x denotes the offspring solution. On the contrary, the job set 2 indicates that B_1 (B_2) inherits from A_1 (A_2) the priority relation between J_1 and J_2 operations. Fig. 10 depicts another operator transformation operation in which two mutation points on the original operator are selected to exchange their positions and generate a new operator.

The genetic algorithm has a population size of $N = 10$, uses the first operator transformation approach for crossover, and has a crossover rate of $p_c = 0.8$. The mutation technique is the second operator transformation method with a mutation rate of $p_m = 0.2$. The variable neighborhood search algorithm's neighborhood transformation strategy chooses the first operator transformation method. The second operator transformation method for the shaking operation. The maximum number of neighborhood modifications is set at $k_{max} = 10$. The adaptive

Table 10
Target value by DDQN and other metaheuristic algorithms.

E_{ave}	m	n_{add}	DDQN	GA	VNS	ALNS	SA	ABC
30	10	25	1740.95	1846.63	**1694.29**	1909.73	1913.44	1880.34
		50	**2717.23**	2938.85	2731.3	3056.46	3019.85	2987.23
		75	**3819**	4146.28	3917.29	4216.08	4216.63	4211.56
		100	**4771.08**	5144.13	4874.51	5264.99	5259.18	5259.43
	30	25	**554.01**	848.32	776.14	883.56	881.52	872.12
		50	**744.31**	1296.95	1228.91	1359.84	1340.21	1345.63
		75	**949.76**	1693.22	1565.55	1766.31	1717.46	1706.37
		100	**1236.16**	2193.76	2053.5	2266.75	2210.19	2230.02
	50	25	**395.85**	661.95	613.28	687.05	694.1	682.72
		50	**475.37**	927.59	852.56	953.41	953.55	932.15
		75	**580.62**	1199.95	1131.13	1254.84	1253.65	1237.19
		100	**676.41**	1506.82	1426.44	1558.46	1556.04	1542.78
50	10	25	**1750.61**	1930.52	1771.79	1991.01	1988.22	1977.62
		50	**2750.91**	3104.01	2867.77	3164.02	3184.53	3131.33
		75	**3807.33**	4245.22	3921.8	4305.08	4266.3	4255.99
		100	**4877.76**	5292.44	4970.44	5380.65	5342.67	5312.81
	30	25	**579.06**	877.62	804.35	903.64	915.86	898.88
		50	**784.96**	1401.02	1304.66	1436.39	1455.31	1406.64
		75	**992.87**	1765.07	1671.92	1834.85	1825.13	1811.81
		100	**1271.94**	2288.98	2154.25	2329.75	2334.46	2329.51
	50	25	**435.79**	693.06	631.68	712.07	716.49	711.45
		50	**520.66**	994.05	935.68	1032.78	1014.5	1016.07
		75	**636.61**	1326.35	1206.74	1345.09	1331.79	1339.11
		100	**714.79**	1566.18	1474.16	1633.32	1626.08	1609.93
100	10	25	**1829.66**	2156.35	1940.98	2161.02	2151.27	2133.27
		50	**2871.95**	3324.22	3031.38	3335.83	3312.3	3308.07
		75	**3942.18**	4436.9	4120.77	4443.96	4445.27	4407.85
		100	**5071.74**	5591.28	5230.56	5575.13	5595.03	5615.18
	30	25	**771.46**	1127.61	1027.84	1151.22	1167.54	1149.8
		50	**925.33**	1667.13	1485.65	1649.38	1636.69	1629.21
		75	**1153.06**	2103.31	1918.8	2099.4	2098.91	2093.32
		100	**1428.47**	2564.5	2386.11	2569.44	2595.09	2577.47
	50	25	**627.62**	937.88	863.35	935.88	943.77	939.76
		50	**712.66**	1256.9	1179.91	1251.41	1279.84	1263.52
		75	**822.55**	1586.46	1467.49	1616.26	1606.84	1583.26
		100	**932.52**	1943.36	1778.75	1956.12	1881.28	1914.25

large neighborhood search algorithm starts with an initial temperature of 1000 and a cooling coefficient of 0.95. The loop-breaking combination techniques consist of the first and second operator transformation methods, with the evaluation parameter φ having the following value.

$$\varphi = max\begin{cases}\omega_1 = 0.8, & \textit{if the new solution is a new global best},\\ \omega_2 = 0.6, & \textit{if the new solution is better than the current one},\\ \omega_3 = 0.4, & \textit{if the new solution is accepted},\\ \omega_4 = 0.2, & \textit{if the new solution is reject}.\end{cases}$$

The formula for updating the probability of selecting each method is $p_x = \lambda p_x + (1-\lambda)\varphi\ (x = 1,2)$, with λ being a parameter set at 0.8. The simulated annealing algorithm starts with an initial temperature of 1000, a cooling coefficient of 0.95, and randomly selects one operator transformation at a time for neighborhood modification. The artificial bee colony algorithm consists of a population size of $N = 10$, with the first operator transformation selected for employed bees, the second operator transformation for observer bees, and the random transformation for scout bees.

We conducted 100 iterations of the proposed DDQN algorithm and five other metaheuristics under 36 different scenarios. Each scenario consists of 30 instances, and the results of these instances were averaged to obtain the overall results for each scenario. Table 10 displays the outcomes achieved by each method across various scenarios, with the best results emphasized in bold.

Compared to the metaheuristic algorithm, the DDQN algorithm did not show the best performance only in the minimum size scenario. This advantage becomes more pronounced as the number of machines increases. For instance, when $m = 10$, the DDQN algorithm surpasses the GA algorithm by an average of 9.46%, the VNS algorithm by 2.69%, the ALNS algorithm by 10.81%, the SA algorithm by 10.59%, and the ABC algorithm by 10.61%. When the number of machines increases $m = 30$, the DDQN algorithm outperforms the GA algorithm by 42.54% on average, the VNS algorithm by 38.05%, the ALNS algorithm by 43.81%, the SA algorithm by 43.57%, and the ABC algorithm by 43.22%. Finally, in the case of $m = 50$, the DDQN algorithm exhibits an average superiority of 48.67% over the GA algorithm, 44.75% over the VNS algorithm, 49.87% over the ALNS algorithm, 49.61% over the SA algorithm, and 49.31% over the ABC algorithm. Fig. 11 is a comparison plot of the algorithm iterations for four different scenarios. The figure shows that the DDQN algorithm converges rapidly from the beginning, while the other metaheuristics converge gradually. These experimental findings lead us to conclude that both the proposed DDQN algorithm and the compound scheduling rules are effective and robust in terms of solution quality and convergence speed.

To assess the time efficiency of the DDQN algorithm, we conducted experiments by limiting the running time to 3 s. These experiments involved comparing the performance of the DDQN algorithm with five metaheuristics, and the results are presented in Table 11. The results demonstrate that the DDQN algorithm consistently outperforms the metaheuristics in the 33 scenarios. However, in some of the scenarios with $n_{add} = 100$, the solution of the DDQN algorithm is surpassed by the comparison algorithms, yielding a poor solution. These scenarios have the largest number of jobs and machines and therefore require more computation time. The DDQN algorithm did not complete all the

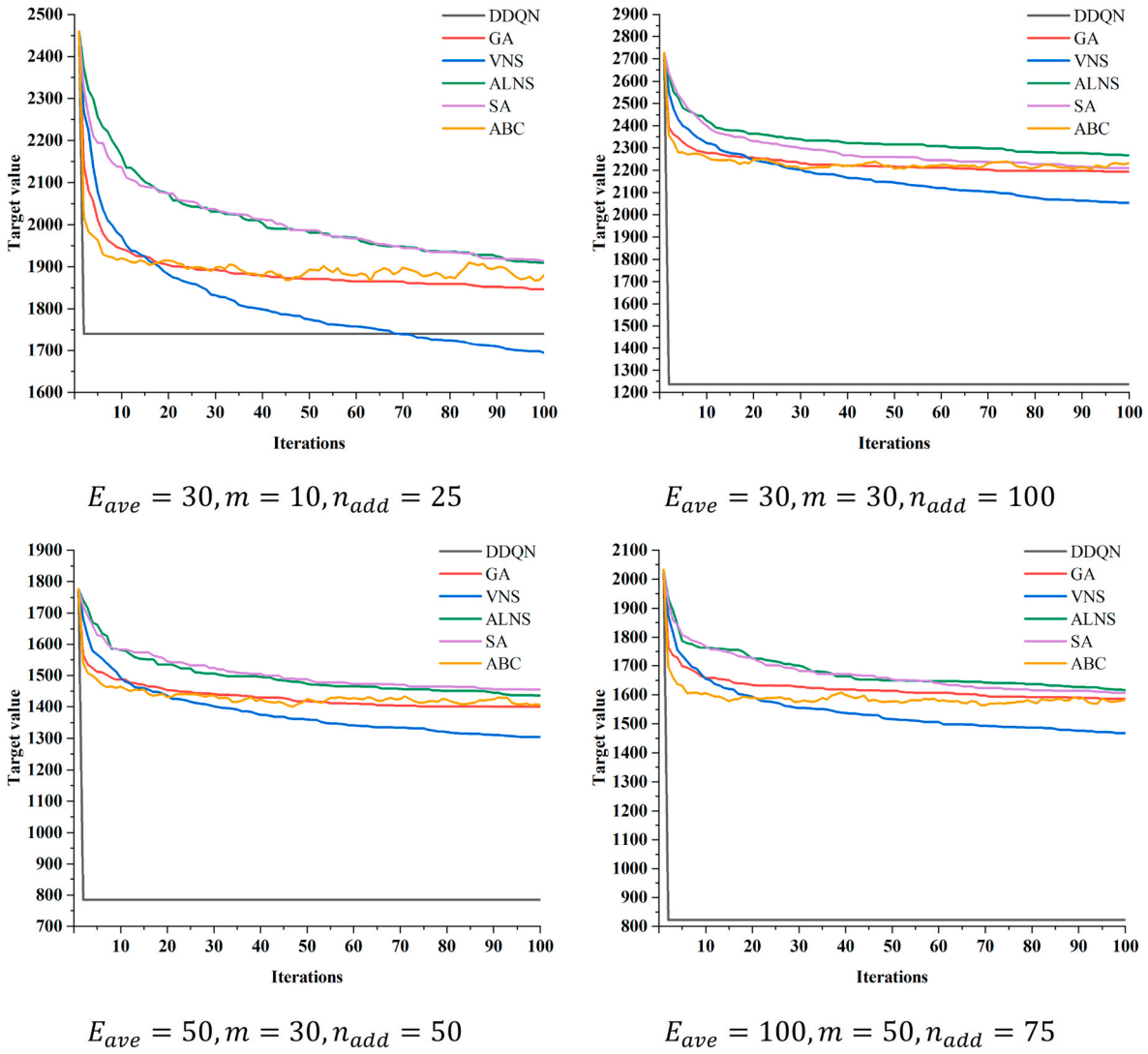

Fig. 11. Some results of comparison between DDQN and other metaheuristic algorithms (100 iterations).

computations within the 3-s time limit, resulting in a poor output. To address this, we extended the running time to 5 s for these scenarios. The resulting values for the DDQN, GA, VNS, ALNS, SA, and ABC algorithms in the scenario $E_{ave} = 30, m = 50, n_{add} = 100$ are 676.41, 1586.5, 1587.26, 1570.96, 1565.62, and 1632.64, respectively. The resulting values for the DDQN, GA, VNS, ALNS, SA, and ABC algorithms in the scenario $E_{ave} = 50, m = 50, n_{add} = 100$ are 714.79, 1678.76, 1611.43, 1640.18, 1632.37, and 1709.61, respectively. The resulting values for the DDQN, GA, VNS, ALNS, SA, and ABC algorithms in the scenario $E_{ave} = 100, m = 50, n_{add} = 100$ are 982.72, 2003.08, 1954.64, 1945.51, 1949.05 and 2055.18, respectively. Once again, the DDQN algorithm emerged as the optimal solution, thus confirming its time efficiency.

4.4. Comparisons with dispatching rules

We tested the DDQN algorithm to five additional dispatching rules to further confirm its superiority: Longest Processing Time (LPT), Shortest Processing Time (SPT), Longest Remaining Time First (LRTF), Earliest Due Date (EDD), and First In First Out (FIFO). Each dispatching rule is explained briefly below.

1. LPT: Prioritize jobs with the longest processing time for processing allocation.
2. SPT: Priority is given to jobs with the shortest processing time.
3. LRTF: Select the job with the longest remaining processing time.
4. EDD: Priority processing for jobs with the earliest delivery date.
5. FIFO: First-arriving jobs are processed first.

The processing time of an operation is approximated for LPT, SPT, and LRTF by determining the average value of its processing times over all available machines. Note that these dispatching rules do not explicitly determine the processing machine, which makes them inadequate for handling the problem addressed in this work. To solve this limitation, we modified the five dispatching rules by assigning the chosen operation to the first machine that became available. This modification is intended to lower completion time and average tardiness, allowing for fair comparisons with the DDQN algorithm.

We tested the DDQN algorithm along with additional comparison rules on 30 separate instances in different scenarios. The results for each method can be found in Table 12.

In all 36 scenarios, the DDQN algorithm outperforms the compared scheduling rules. In 22 scenarios, the DDQN algorithm outperforms the other scheduling rules by more than 60% on average. This indicates that our proposed compound scheduling rule is effective and outperforms the five comparative scheduling rules. It also proves the effectiveness of our proposed compound scheduling rule in reducing completion time and average tardiness. Fig. 12 shows the comparison of the algorithms in four different scenarios.

Table 11
The results of comparison between DDQN and other metaheuristic algorithms (3 s).

E_{ave}	m	n_{add}	DDQN	GA	VNS	ALNS	SA	ABC
30	10	25	**1740.95**	1870.72	1748.68	1811.18	1817.02	2031.56
		50	**2717.23**	3055.95	2947.59	3006.78	2989.98	3147.26
		75	**3819**	4338.97	4266.7	4334.58	4298.99	4428.82
		100	**4771.08**	5425.9	5356.97	5385.91	5416.32	5499.25
	30	25	**554.01**	867.37	826.24	829.44	838.61	941.84
		50	**744.31**	1389.3	1346.56	1340.26	1355.8	1435.17
		75	**949.76**	1776.46	1760.34	1767.49	1778.34	1818.47
		100	**1236.16**	2320.8	2311.4	2302.15	2322.18	2340.53
	50	25	**395.85**	682.22	650.7	664.29	653.67	737.06
		50	**475.37**	943.69	939.28	949.57	948.21	991.05
		75	**580.62**	1288.37	1269.81	1270.62	1265.07	1318.39
		100	1809.09	1589.45	1602.33	**1582.28**	1606.75	1640.09
50	10	25	**1750.61**	1987.34	1856.11	1898.57	1901.7	2100.11
		50	**2750.91**	3209.58	3120.84	3144.8	3169.41	3296.78
		75	**3807.33**	4379.87	4361.86	4341.23	4353.33	4486.72
		100	**4877.76**	5520.95	5497.32	5519.7	5471.3	5532.39
	30	25	**579.06**	894.79	831.75	879.11	866.54	965.13
		50	**784.96**	1470.37	1427.84	1440.52	1428.42	1553.8
		75	**992.87**	1854.69	1853.72	1868.68	1880.71	1916.08
		100	**1271.94**	2418.47	2422.98	2402.85	2398.65	2428.97
	50	25	**435.79**	714	664.73	682.54	686.14	755.02
		50	**520.66**	1032.52	1018.01	1031.38	1031.05	1106.47
		75	**636.61**	1395.59	1392.74	1365.48	1371.21	1429.38
		100	1779.59	1698.03	**1677.52**	1686.22	1691.43	1712.5
100	10	25	**1829.66**	2211.33	2033.88	2103.76	2057.06	2371.73
		50	**2871.95**	3409.6	3304.33	3321.82	3332.77	3545.58
		75	**3942.18**	4555.53	4579.49	4508.67	4553.78	4673.08
		100	**5071.74**	5856.49	5713.29	5734.08	5716.87	5869.43
	30	25	**771.46**	1174.3	1098.45	1120.52	1107.07	1272.82
		50	**925.33**	1729.77	1597.91	1620.58	1631.66	1778.25
		75	**1153.06**	2193.76	2125.67	2160.34	2127.83	2230.55
		100	**1632.71**	2672.03	2687.6	2645.96	2685.71	2718.89
	50	25	**627.62**	958.68	896.86	919.34	906.4	1031.59
		50	**712.66**	1310.96	1243.32	1258.52	1276.79	1375.61
		75	**852.68**	1668.84	1628.26	1637.2	1613.05	1705.12
		100	2254.74	2052.33	2001.4	**1982.94**	1990.81	2036.73

4.5. Comparisons with other RL-based scheduling methods

In addition, to verify the superiority of the DDQN algorithm, we conducted a comparison with three other reinforcement algorithms: DQN (Li et al., 2022), Q-learning (Wang et al., 2020), and SARSA (Chen et al., 2020). For each type of scenario, 30 independent instances are generated. Within each instance, DDQN and other reinforcement learning algorithms are independently recomputed 20 times. Finally, the twenty target values obtained from each instance are averaged. To ensure a fair evaluation, each of these methods employed the same action space consisting of 9 compound scheduling rules, matching the DDQN algorithm. The state space of the DQN algorithm remained consistent with that of the DDQN algorithm, and its loss function was calculated using Equation (16) in Section 3.1. Regarding Q-learning and SARSA, their state sets were divided into ten states, denoted as $S = [S(1), S(2), \ldots, S(10)]$ (Shahrabi et al., 2017). At time t, the state $S(t)^*$ for Q-learning and SARSA was determined by taking the average of the eight state features from the DDQN algorithm. The interval value of $S(t)^*$ was set to 0.1, meaning that $S(t) = S(1)$ when $S(t)^* \in [0, 0.1]$, $S(t) = S(2)$ when $S(t)^* \in [0.1, 0.2]$, and so on. The Q-learning and SARSA Q-tables are updated in a way that refers to Equations (15) and (16), respectively. The comparative results across different production environments can be found in Table 13, with the most optimal outcomes highlighted in bold.

The DDQN algorithm outperforms other reinforcement learning algorithms in 25 out of 36 scenarios. Even in scenarios where DDQN is not optimal, its solution is very close to the optimal solution. The DDQN algorithm's performance is weakest in the test scenarios with $E_{ave} = 100$, $m = 30$ and $n_{add} = 75$. DDQN obtains a solution of 1178.25, however, it is only 0.299% worse than the optimal solution of 1174.74. In contrast, the DQN algorithm fails to reach the optimal solution in any of the test scenarios, likely due to its large estimation bias and lag. The Q-learning algorithm and the SARSA algorithm can only achieve the optimal solution in a few cases. Refer to Fig. 13 for a comparison of the number of wins between DDQN and other reinforcement learning methods under different E_{ave}.

To further validate the performance of the intelligent agent of the trained DDQN for other optimization objectives, we compare it with the DQN proposed by Luo (2020) and Chang et al. (2022) respectively. The optimization objective of Luo (2020) is to minimize total tardiness, see Equation (23). The optimization objective of Chang et al. (2022) is to minimize penalties for earliness and tardiness, see Equation (24). where w_i^e is the unit of earliness cost obeying a uniform distribution of [1, 1.5] and w_i^t is the unit of tardiness cost obeying a uniform distribution of [1, 2]. We modified the optimization objectives of the intelligent agent to ensure consistency with theirs. We set up different scenarios according to their parameters respectively, and each scenario contains 10 instances. The experimental results are shown in Table 14.

$$Minimize \sum_{i=1}^{n} \max\left\{C_{i,n_i} - D_i, 0\right\} \tag{23}$$

$$Minimize \left\{ \sum_{i=1}^{n} \left(w_i^e \times \max(D_i - C_i, 0) + \left(w_i^t \times \max(C_i - D_i, 0)\right) \right) \right\} \tag{24}$$

Table 12
Target value by DDQN and other dispatching rules.

E_{ave}	m	n_{add}	DDQN	LPT	SPT	LRTF	EDD	FIFO
30	10	25	**1737.02**	4421.59	4359.27	2207.70	5361.41	6140.09
		50	**2745.62**	6753.48	6628.27	3669.32	8027.68	9239.82
		75	**3817.70**	9643.09	9356.81	5228.23	11410.83	13204.61
		100	**4821.60**	12265.36	11947.98	6716.82	14701.46	17024.55
	30	25	**541.91**	1484.09	1458.20	863.42	1544.46	1735.32
		50	**734.42**	2069.70	2050.40	1145.81	2306.66	2504.04
		75	**976.08**	2875.69	2978.86	1593.29	3256.14	3578.31
		100	**1190.83**	3693.48	3484.33	2045.41	3927.88	4497.11
	50	25	**380.78**	982.71	948.66	728.97	1067.64	1078.62
		50	**485.39**	1215.55	1286.40	843.15	1374.44	1403.82
		75	**579.45**	1647.65	1669.77	975.05	1774.87	1888.13
		100	**675.20**	2127.63	2131.07	1193.61	2221.77	2439.97
50	10	25	**1773.64**	4266.11	4075.02	2180.21	4842.07	5691.32
		50	**2766.30**	6883.63	6862.36	3726.93	8179.28	9343.36
		75	**3855.77**	9706.99	9635.34	5316.73	11739.20	13516.71
		100	**4848.34**	12692.38	12388.56	6789.08	15011.61	17201.62
	30	25	**594.82**	1432.33	1445.05	893.39	1609.58	1699.19
		50	**755.32**	2094.47	2023.41	1160.91	2277.20	2411.35
		75	**987.68**	2906.87	2890.06	1627.02	3118.25	3478.96
		100	**1219.65**	3593.47	3551.41	2073.40	3864.80	4387.74
	50	25	**455.79**	990.15	1014.62	773.99	1028.28	1046.92
		50	**520.19**	1361.45	1315.88	905.70	1396.13	1462.18
		75	**620.51**	1701.52	1724.30	1050.30	1794.77	1901.16
		100	**717.56**	2008.17	2073.11	1232.36	2129.50	2346.11
100	10	25	**1857.38**	4485.52	4377.84	2372.95	5148.78	5776.53
		50	**2838.18**	6922.85	6752.13	3819.36	8018.85	9261.77
		75	**3899.48**	9402.63	9324.174	5329.90	11254.84	12962.20
		100	**5109.88**	13142.78	12456.31	7104.74	15526.43	17560.28
	30	25	**690.77**	1457.14	1476.79	1006.68	1528.26	1596.37
		50	**948.84**	2283.16	2256.92	1315.59	2382.10	2583.19
		75	**1201.72**	2952.52	2992.93	1769.65	3106.30	3376.79
		100	**1409.53**	3762.55	3720.81	2246.94	3991.91	4414.06
	50	25	**598.34**	1122.08	1106.48	877.08	1074.07	1072.65
		50	**721.42**	1350.61	1358.05	1073.93	1382.50	1378.64
		75	**835.02**	1676.66	1695.40	1230.70	1700.26	1814.74
		100	**956.27**	5124.93	2245.19	1416.55	2189.23	2354.06

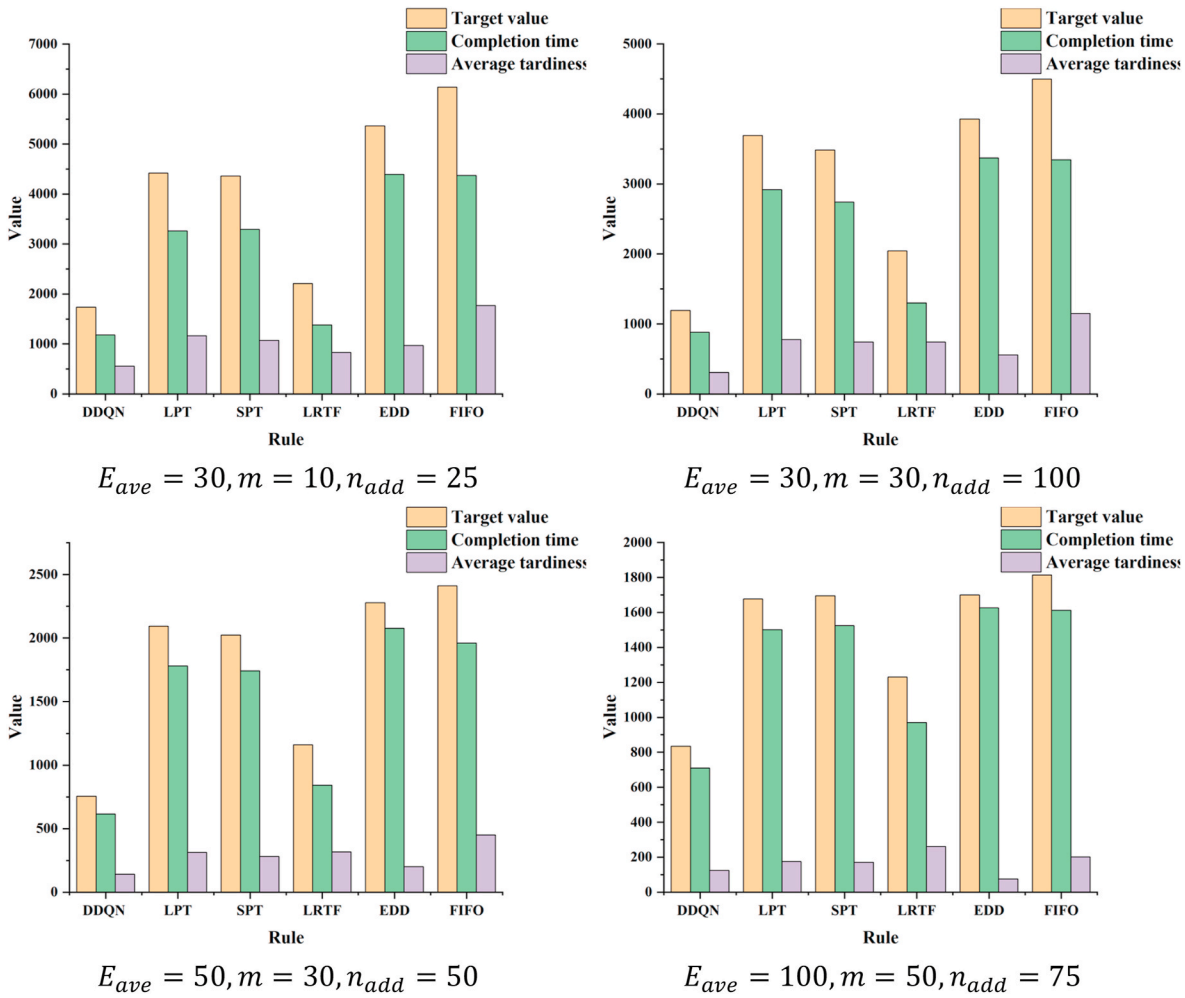

Fig. 12. Some results of the comparison between DDQN and other dispatching rules.

Table 13
Target value by DDQN and other RL methods.

E_{ave}	m	n_{add}	DDQN	DQN	Q-learning	SARSA
30	10	25	**1702.70**	1827.02	1894.26	1853.15
		50	**2815.22**	2932.20	3246.04	3224.07
		75	**3816.87**	3940.71	4402.81	4384.79
		100	**4840.10**	5012.09	5667.10	5616.25
	30	25	542.49	581.80	**541.52**	543.42
		50	**709.63**	764.44	723.66	710.78
		75	**951.50**	1047.61	969.09	951.67
		100	**1195.87**	1339.97	1238.99	1208.85
	50	25	383.17	402.46	**382.82**	387.90
		50	**490.32**	516.74	495.87	490.98
		75	564.06	595.32	571.21	**563.38**
		100	676.36	724.01	687.06	**675.03**
50	10	25	**1732.38**	1874.22	1875.76	1856.97
		50	**2749.24**	2901.75	3121.13	3091.33
		75	**3753.34**	3877.64	4346.34	4314.89
		100	**4824.68**	4982.82	5557.93	5537.33
	30	25	562.31	601.61	573.55	**561.94**
		50	**782.05**	851.28	786.89	785.98
		75	**1010.04**	1104.58	1025.81	1013.23
		100	**1256.65**	1424.95	1275.36	1267.86
	50	25	**434.23**	460.58	436.37	434.38
		50	527.43	551.51	**526.79**	528.05
		75	**627.25**	705.10	632.35	628.49
		100	721.15	775.83	724.93	**720.41**
100	10	25	**1769.69**	1920.82	1857.60	1849.26
		50	**2889.28**	2986.53	3225.15	3205.45
		75	**3936.09**	4040.68	4474.71	4478.74
		100	**5005.77**	5132.91	5685.05	5690.88
	30	25	706.10	775.80	**705.69**	708.17
		50	909.20	987.42	914.02	**908.97**
		75	1178.25	1295.63	**1174.74**	1179.32
		100	**1421.44**	1604.07	1439.17	1446.13
	50	25	596.74	643.88	**596.13**	597.67
		50	**739.39**	801.08	741.49	741.61
		75	**800.22**	898.01	801.12	802.85
		100	**910.81**	1009.29	916.89	915.04

It can be found that compared to Luo's (Luo, 2020) method our intelligent agent obtains better solutions in 6 scenarios, but in comparison with Chang et al.'s (Chang et al., 2022) method is that our intelligent agent performs poorly and has only 1 better solution. This may be due to the fact that the optimization objective of Luo (2020) only considers tardiness and our intelligent agent was also trained with the objective of reducing tardiness. In contrast, Chang et al.'s (Chang et al., 2022) objective not only considers tardiness but also an earliness objective that conflicts with tardiness. Our intelligent agent was not trained under that objective and the earliness was not considered in our proposed compound rules.

Table 14
Results of comparison with other methods.

E_{ave}	m	n_{add}	This work	Luo (2020)	This work	Chang et al. (2022)
50	10	50	**1775**	10100	22866	**14000**
		100	**5650**	16800	**32255**	58888
	30	50	2315	**1400**	24464	**12700**
		100	7398	**2730**	33403	**18500**
100	10	50	**222**	5900	40160	**11900**
		100	**881**	6100	62645	**60300**
	30	50	**421**	790	38557	**13400**
		100	**1649**	1720	62829	**22400**

5. Conclusion

This work presents a novel Double Deep Q-Network (DDQN) algorithm with generalization for addressing the DFJSP involving dynamic job arrivals and urgent job insertions. The algorithm incorporates 8 carefully designed features that respond to the state of the shop floor. Furthermore, eight compound scheduling rules are devised, complemented by random rules for selecting unfinished jobs and assigning them to suitable machines. Reward functions were designed to unify the scheduling objectives and the optimization direction of the cumulative rewards obtained by the agents. During the training process, a greedy strategy is employed to explore a wider range of options in the early stages, while in later stages, the optimal action is selected with a higher probability. Extensive simulation experiments are conducted across various production scenarios to assess the effectiveness and generalizability of the proposed DDQN algorithm. In comparison experiments with metaheuristic rules, the advantage of the proposed algorithm increases as the number of machines increases. Compared to other well-known scheduling rules, the proposed algorithm performs effectively in all experimental scenarios, outperforming the other rules by an average of 59.64%. Compared to other RL algorithms, the proposed algorithm performs optimally in 69.44% of scenarios, indicating that its performance is more consistent. When compared to the results of other scholars' research, we discovered that the proposed algorithm has a good generalization ability to optimize objectives similar to the goals of this work (completion time, tardiness). The deep reinforcement learning algorithm optimizes the production plan and enhances wafer production efficiency by analyzing various factors such as equipment status and order priority. The method is adaptive, allowing the task allocation and workflow to be adjusted in real-time to ensure the stability and consistency of the production plan in the face of changing production demands and equipment state. In conclusion, the use of a deep reinforcement

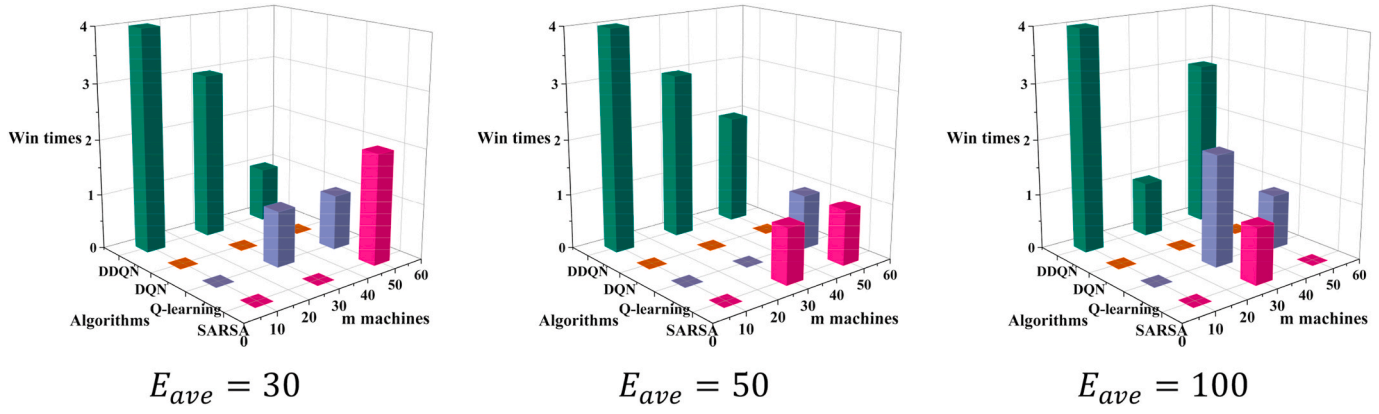

Fig. 13. The win times between DDQN and other RL methods.

learning algorithm improves the efficiency, flexibility, and competitiveness of the wafer production shop operation. The method suggested in this work is relevant not only to wafer processing but also to the fabrication of complicated equipment, such as automotive assembly and electronics manufacturing. Deep reinforcement learning can improve the processes and operations of a production line. By interacting with smart manufacturing devices and systems, deep reinforcement learning algorithms can find the best control techniques for maximizing productivity and resource utilization.

For future work, the current state and action settings of deep reinforcement learning may be improved. Optimization objectives in various dimensions will be considered, and the model will be trained with the goal of producing intelligent agents with improved generalization capabilities. Future explorations will also encompass other metaheuristics and deep reinforcement learning algorithms. Quantitative research on the application of training problem instances will aid in the advancement of dynamic scheduling techniques based on deep reinforcement learning. Furthermore, we aim to incorporate additional uncertainty factors into the problem-modeling process. To achieve this, we will continue to enhance the problem modeling through robust optimization techniques.

CRediT authorship contribution statement

Shaojun Lu: Writing – original draft, Methodology, Conceptualization. **Yongqi Wang:** Writing – original draft, Visualization, Methodology, Conceptualization. **Min Kong:** Writing – review & editing, Writing – original draft, Methodology. **Weizhong Wang:** Writing – review & editing, Conceptualization. **Weimin Tan:** Writing – review & editing, Conceptualization. **Yingxin Song:** Writing – review & editing, Conceptualization.

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.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (Nos. 72101071, 72071056, 72371095, 72101077, 72201089, 72301004 and 72301005), Key Research and Development Project of Anhui Province (2022a05020023), the Fundamental Research Funds for the Central Universities of China (JZ2023HGTB0278), Natural Science Foundation of Anhui Province(2308085MG225), the Ministry of Education of Humanities and Social Science Project (22YJC630050), the China Postdoctoral Science Foundation (2022M710996), the Educational Commission of Anhui Province (KJ2020A0069, KJ2021A0095), Base of Introducing Talents of Discipline to Universities for Optimization and Decision-making in the Manufacturing Process of Complex Product (111 project: B17014).

References

Abreu, C.F., May, J.H., Spangler, W.E., Vargas, L.G., 2008. Conflict identification and reconciliation in a collaborative manufacturing scheduling task. Int. J. Inf. Technol. Decis. Making 7 (1), 147–174.

Baykasoğlu, A., Madenoğlu, F.S., Hamzadayı, A., 2020. Greedy randomized adaptive search for dynamic flexible job-shop scheduling. J. Manuf. Syst. 56, 425–451.

Bazargan-Lari, M.R., Taghipour, S., 2022. A hybrid data-driven approach for forecasting the characteristics of production disruptions and interruptions. Int. J. Inf. Technol. Decis. Making 21 (4), 1127–1154.

Chang, J.R., Yu, D., Hu, Y., He, W.W., Yu, H.Y., 2022. Deep reinforcement learning for dynamic flexible job shop scheduling with random job arrival. Processes 10 (4), 760.

Chen, R.H., Yang, B., Li, S., Wang, S.L., 2020. A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem. Comput. Ind. Eng. 149, 106778.

Cota, L.P., Guimaraes, F.G., Ribeiro, R.G., Meneghini, I.R., de Oliveira, F.B., Souza, M.J. F., Siarry, P., 2019. An adaptive multi-objective algorithm based on decomposition and large neighborhood search for a green machine scheduling problem. Swarm Evol. Comput. 51, 100601.

Dehghan-Sanej, K., Eghbali-Zarch, M., Tavakkoli-Moghaddam, R., Sajadi, S.M., Sadjadi, S.J., 2021. Solving a new robust reverse job shop scheduling problem by meta-heuristic algorithms. Eng. Appl. Artif. Intell. 101, 104207.

Fathollahi-Fard, A.M., Hajiaghaei-Keshteli, M., Tavakkoli-Moghaddam, R., Smith, N.R., 2022. Bi-level programming for home health care supply chain considering outsourcing. Journal of Industrial Information Integration 25, 100246.

Fathollahi-Fard, A.M., Woodward, L., Akhrif, O., 2021. Sustainable distributed permutation flow-shop scheduling model based on a triple bottom line concept. Journal of Industrial Information Integration 24, 100233.

Fathollahi-Fard, A.M., Woodward, L., Akhrif, O., 2024. A distributed permutation flow-shop considering sustainability criteria and real-time scheduling. Journal of Industrial Information Integration 39, 100598.

Ferreira, C., Figueira, G., Amorim, P., 2022. Effective and interpretable dispatching rules for dynamic job shops via guided empirical learning. Omega 111, 102643.

Gui, Y., Tang, D.B., Zhu, H.H., Zhang, Y., Zhang, Z.Q., 2023. Dynamic scheduling for flexible job shop using a deep reinforcement learning approach. Comput. Ind. Eng. 180, 109255.

Homayouni, S.M., Tang, S.H., Motlagh, O., 2014. A genetic algorithm for optimization of integrated scheduling of cranes, vehicles, and storage platforms at automated container terminals. J. Comput. Appl. Math. 270, 545–556.

Johnson, D., Chen, G., Lu, Y.Q., 2022. Multi-agent reinforcement learning for real-time dynamic production scheduling in a robot assembly cell. IEEE Rob. Autom. Lett. 7 (3), 7684–7691.

Kaelbling, L.P., Littman, M.L., Moore, A.W., 1996. Reinforcement learning: a survey. J. Artif. Intell. Res. 4, 237–285.

Karimi-Mamaghan, M., Mohammadi, M., Pasdeloup, B., Meyer, P., 2023. Learning to select operators in meta-heuristics: an integration of Q-learning into the iterated greedy algorithm for the permutation flowshop scheduling problem. Eur. J. Oper. Res. 304 (3), 1296–1330.

Kim, S.J., Jung, W.B., Jung, H.S., Lee, M.H., Heo, J., Horgan, A., Godron, X., Ham, D., 2023. The bottom of the memory hierarchy: semiconductor and DNA data storage. MRS Bull. 48 (5), 547–559.

Li, K.X., Deng, Q.W., Zhang, L.K., Fan, Q., Gong, G.L., Ding, S., 2021. An effective MCTS-based algorithm for minimizing makespan in dynamic flexible job shop scheduling problem. Comput. Ind. Eng. 155, 107211.

Li, Y., Côté, J.-F., Coelho, L.C., Zhang, C., Zhang, S., 2023a. Order assignment and scheduling under processing and distribution time uncertainty. Eur. J. Oper. Res. 305 (1), 148–163.

Li, Y., Gu, W., Yuan, M., Tang, Y., 2022. Real-time data-driven dynamic scheduling for flexible job shop with insufficient transportation resources using hybrid deep Q network. Robot. Comput. Integrated Manuf. 74, 102283.

Li, Y.B., Liao, C., Wang, L., Xiao, Y., Cao, Y., Guo, S.S., 2023b. A reinforcement learning-artificial bee colony algorithm for flexible job-shop scheduling problem with lot streaming. Appl. Soft Comput. 146, 110658.

Lin, J., 2019. Backtracking search based hyper-heuristic for the flexible job-shop scheduling problem with fuzzy processing time. Eng. Appl. Artif. Intell. 77, 186–196.

Liu, R.K., Piplani, R., Toro, C., 2022. Deep reinforcement learning for dynamic scheduling of a flexible job shop. Int. J. Prod. Res. 60 (13), 4049–4069.

Lu, C., Li, X.Y., Gao, L., Liao, W., Yi, J., 2017. An effective multi-objective discrete virus optimization algorithm for flexible job-shop scheduling problem with controllable processing times. Comput. Ind. Eng. 104, 156–174.

Lu, S.J., Ma, C.Y., Kong, M., Zhou, Z.P., Liu, X.B., 2022. Solving a stochastic hierarchical scheduling problem by VNS-based metaheuristic with locally assisted algorithms. Appl. Soft Comput. 130, 109719.

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

Luo, S., Zhang, L.X., Fan, Y.S., 2021. Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning. Comput. Ind. Eng. 159, 107489.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529–533.

Mokhtari, H., Dadgar, M., 2015. Scheduling optimization of a stochastic flexible job-shop system with time-varying machine failure rate. Comput. Oper. Res. 61, 31–45.

Ogryczak, W., Perny, P., Weng, P., 2013. A compromise programming approach to multiobjective Markov decision processes. Int. J. Inf. Technol. Decis. Making 12 (5), 1021–1053.

Oh, S.H., Cho, Y.I., Woo, J.H., 2022. Distributional reinforcement learning with the independent learners for flexible job shop scheduling problem with high variability. Journal of Computational Design and Engineering 9 (4), 1157–1174.

Ozturk, G., Bahadir, O., Teymourifar, A., 2019. Extracting priority rules for dynamic multi-objective flexible job shop scheduling problems using gene expression programming. Int. J. Prod. Res. 57 (10), 3121–3137.

Pasha, J., Nwodu, A.L., Fathollahi-Fard, A.M., Tian, G.D., Li, Z.W., Wang, H., Dulebenets, M.A., 2022. Exact and metaheuristic algorithms for the vehicle routing

problem with a factory-in-a-box in multi-objective settings. Adv. Eng. Inf. 52, 101623.
Perraudat, A., Dauzère-Pérès, S., Vialletelle, P., 2022. Robust tactical qualification decisions in flexible manufacturing systems. Omega 106, 102537.
Ramasesh, R., 1990. Dynamic job shop scheduling: a survey of simulation research. Omega 18 (1), 43–57.
Rohaninejad, M., Janota, M., Hanzálek, Z., 2023. Integrated lot-sizing and scheduling: mitigation of uncertainty in demand and processing time by machine learning. Eng. Appl. Artif. Intell. 118, 105676.
Shahgholi Zadeh, M., Katebi, Y., Doniavi, A., 2018. A heuristic model for dynamic flexible job shop scheduling problem considering variable processing times. Int. J. Prod. Res. 57 (10), 3020–3035.
Shahrabi, J., Adibi, M.A., Mahootchi, M., 2017. A reinforcement learning approach to parameter estimation in dynamic job shop scheduling. Comput. Ind. Eng. 110, 75–82.
Shiue, Y.R., Lee, K.C., Su, C.T., 2018. Real-time scheduling for a smart factory using a reinforcement learning approach. Comput. Ind. Eng. 125, 604–614.
Sun, J.H., Zhang, G.H., Lu, J., Zhang, W.Q., 2021. A hybrid many-objective evolutionary algorithm for flexible job-shop scheduling problem with transportation and setup times. Comput. Oper. Res. 132, 105263.
Van Hasselt, H., Guez, A., Silver, D., 2016. Deep reinforcement learning with double Q-learning. Proc. AAAI Conf. Artif. Intell. 30 (1).
Wang, H., Sarker, B.R., Li, J., Li, J., 2020. Adaptive scheduling for assembly job shop with uncertain assembly times based on dual Q-learning. Int. J. Prod. Res. 59 (19), 5867–5883.
Wang, J., Tang, H.T., Lei, D.M., 2023. A Q-learning artificial bee colony for distributed assembly flow shop scheduling with factory eligibility, transportation capacity and setup time. Eng. Appl. Artif. Intell. 123, 106230.
Wei, L., He, J., Guo, Z., Hu, Z., 2023. A multi-objective migrating birds optimization algorithm based on game theory for dynamic flexible job shop scheduling problem. Expert Syst. Appl. 227, 120268.
Yenisey, M.M., Yagmahan, B., 2014. Multi-objective permutation flow shop scheduling problem: literature review, classification and current trends. Omega 45, 119–135.
Zhang, C., Meng, Y., Prasanna, V., 2023a. A framework for mapping DRL algorithms with prioritized replay buffer onto heterogeneous platforms. IEEE Trans. Parallel Distr. Syst. 34 (6), 1816–1829.
Zhang, G.H., Hu, Y.F., Sun, J.H., Zhang, W.Q., 2020. An improved genetic algorithm for the flexible job shop scheduling problem with multiple time constraints. Swarm Evol. Comput. 54, 100664.
Zhang, G.H., Lu, X.X., Liu, X., Zhang, L.T., Wei, S.W., Zhang, W.Q., 2022a. An effective two-stage algorithm based on convolutional neural network for the bi-objective flexible job shop scheduling problem with machine breakdown. Expert Syst. Appl. 203, 117460.
Zhang, Y., Rao, X., Liu, C., Zhang, X., Zhou, Y., 2023b. A cooperative EV charging scheduling strategy based on double deep Q-network and Prioritized experience replay. Eng. Appl. Artif. Intell. 118, 105642.
Zhang, Y., Zhu, H.H., Tang, D.B., Zhou, T., Gui, Y., 2022b. Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems. Robot. Comput. Integrated Manuf. 78, 102412.
Zhang, Y.C., Bai, R.B., Qu, R., Tu, C.F., Jin, J.H., 2022c. A deep reinforcement learning based hyper-heuristic for combinatorial optimisation with uncertainties. Eur. J. Oper. Res. 300 (2), 418–427.
Zhou, Y., Yang, J.J., Huang, Z., 2020. Automatic design of scheduling policies for dynamic flexible job shop scheduling via surrogate-assisted cooperative co-evolution genetic programming. Int. J. Prod. Res. 58 (9), 2561–2580.
Zhu, K., Gong, G., Peng, N., Zhang, L., Huang, D., Luo, Q., Li, X., 2023. Dynamic distributed flexible job-shop scheduling problem considering operation inspection. Expert Syst. Appl. 224, 119840.