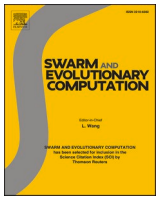
Swarm and Evolutionary Computation 98 (2025) 102122

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

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



Dynamic cooperative scheduling toward distributed and reconfigurable manufacturing via multi-agent deep reinforcement learning

Shengluo Yang [a], Zhigang Xu [b,*], Fangfang Zhang [c], Yi Mei [c], Quanke Pan [d,e], Mengjie Zhang [c]

[a] School of Mechanical Engineering, University of Shanghai for Science and Technology, Shanghai, 200093, China
[b] Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 110016, China
[c] Centre for Data Science and Artificial Intelligence & School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand
[d] School of Mechatronic Engineering and Automation, Shanghai University, Shanghai, 200444, China
[e] School of Computer Science, Liaocheng University, Liaocheng 252000, China

ARTICLE INFO

Keywords:
Dynamic scheduling
Deep reinforcement learning
Distributed workshop scheduling
Multi-agent reinforcement learning
Reconfigurable manufacturing system

ABSTRACT

Distributed production and reconfigurable manufacturing have become increasingly prevalent in globalized and customized manufacturing environments. In distributed production systems with dynamic events, deep reinforcement learning (DRL) has shown promise in achieving real-time and near-optimal scheduling due to its learning and generalization capabilities. However, two key challenges arise in solving the distributed reconfiguration scheduling problem using DRL: coordinating scheduling and reconfiguration processes, and managing reconfiguration decisions across heterogeneous workshops. To address these challenges, this study integrates multi-agent reinforcement learning (MARL) and DRL, forming a multi-agent DRL (MADRL) framework to enable real-time scheduling and dynamic coordination in distributed reconfigurable flowshops with dynamic job arrivals. The system architecture of intelligent scheduling and reconfiguration is proposed by designing training and execution processes for both DRL-based scheduling and MARL-based reconfiguration agents. In addition, intelligent reconfiguration and scheduling systems are modeled by designing novel rewards, action spaces, and state representations. A reconfiguration judgment mechanism is introduced to reduce unnecessary reconfigurations, ensuring effective coordination between scheduling and reconfiguration processes. Furthermore, the cooperative MARL paradigm is employed to train reconfiguration agents across heterogeneous workshops, enabling collaborative decision-making guided by a global joint reward. Extensive training and comparison experiments on 140 test instances demonstrate that the proposed MADRL algorithms significantly outperform four widely used DRL algorithms and three efficient meta-heuristics in terms of learning efficiency and solution quality. This study contributes to real-time and cooperative scheduling in distributed and reconfigurable manufacturing systems.

1. Introduction

Distributed manufacturing is frequently adopted by large-scale companies to produce products in different regions within the context of economic globalization [1]. Through distributed manufacturing, companies can expand the market scale, use local resources, and reduce the delivery cost. For example, as shown in Fig. 1, the Tesla company established manufacturing bases in different regions, such as the United States, China, Germany, etc. Considering the workshop resources in different regions, the global orders should be properly scheduled.

To provide effective scheduling schemes under distributed manufacturing, researchers investigated the distributed workshop scheduling problem. This problem involves efficiently assigning jobs to workshops and sequencing jobs within each workshop. Given the ubiquitous nature of distributed manufacturing across various industries, this scheduling problem has been investigated by considering various workshop types, including the distributed flowshop [2–4], distributed hybrid flowshop [5–7], distributed job shop [8–10], and distributed assembly flowshop [11]. In addition, some production characteristics have also been considered, such as preventive maintenance [12,13], energy efficiency [14], lot-streaming [15], and setup times [16].

* Corresponding author.
E-mail addresses: yangshengluo@usst.edu.cn (S. Yang), zgxu@sia.cn (Z. Xu), fangfang.zhang@vuw.ac.nz (F. Zhang), yi.mei@ecs.vuw.ac.nz (Y. Mei), panquanke@shu.edu.cn (Q. Pan), mengjie.zhang@ecs.vuw.ac.nz (M. Zhang).

https://doi.org/10.1016/j.swevo.2025.102122
Received 12 April 2025; Received in revised form 23 June 2025; Accepted 2 August 2025

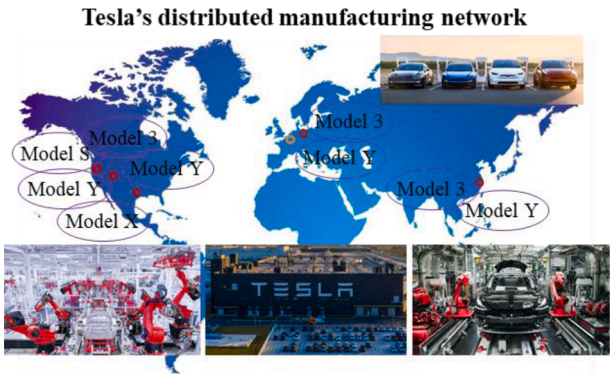


Fig. 1. Tesla establishes distributed manufacturing workshops in different regions.

In real production environments, orders arrive dynamically. Moreover, in customized manufacturing, distinct orders frequently necessitate the use of various production line types. For instance, different Tesla models require specific production lines. This underscores the need for distributed workshops to possess dynamic reconfiguration capabilities. However, real-time scheduling in the context of dynamic job arrivals and workshop reconfigurations under distributed manufacturing has not been fully addressed.

In considering dynamic job arrivals, scheduling plans should undergo frequent optimization to appropriately schedule jobs and determine the types of production lines. However, traditional meta-heuristics encounter challenges in solving dynamic scheduling problems due to their time-consuming nature [17–19]. Recent studies indicate deep reinforcement learning (DRL) holds promise for achieving real-time scheduling [20–22]. DRL can learn scheduling heuristics using limited training data and provide stable scheduling results [23,24].

DRL has been applied to solve various scheduling problems of a single workshop and distributed workshops. For a single workshop, DRL has been utilized to address the flowshop scheduling problem (FSP) [25,26], job shop scheduling problem (JSP) [27–30], and assembly workshop scheduling problem [31]. For distributed workshops, DRL has been applied to the distributed flowshop scheduling problem [32,33] and the distributed job shop scheduling problem [34]. More recently, DRL was also applied to schedule reconfigurable workshops [35,36]. Many Markov decision models and DRL algorithms have been proposed or adapted to solve workshop scheduling problems. However, very few studies have explored the dynamic reconfiguration of distributed workshops using DRL. For distributed reconfigurable workshops, production scheduling and workshop reconfiguration among multiple workshops are interdependent, making the problem more challenging to solve.

Two challenges arise when using DRL to solve the dynamic distributed reconfigurable workshop scheduling problem. They are how to coordinate the dynamic scheduling and reconfiguration for each workshop and among all workshops. Thus, further studies are required to solve the dynamic cooperative scheduling of distributed reconfigurable workshops.

Multi-agent reinforcement learning (MARL) has demonstrated good performance in solving cooperative scheduling problems involving multiple machines or resources, such as the JSP with multiple equipment [20,37], JSP with limited robots [38], and the maintenance and scheduling problem [39]. Cooperative MARL can train and coordinate agents to achieve collective goals [40]. Under the MARL architecture, a group of agents can work cooperatively and coordinate their actions to optimize system performance, making it a suitable approach to solving concurrent scheduling and reconfiguration problems. Inspired by the advantage of MARL, we applied MARL to train agents of distributed workshops to make cooperative scheduling decisions.

MARL algorithms can be categorized into three types based on their training and execution models: centralized training and execution (CTE), decentralized training and execution (DTE), and centralized training and decentralized execution (CTDE) [41,42]. CTE methods assume that both training and execution have access to a central shared state, enabling better performance compared to DTE. However, they suffer from poor scalability due to exponentially growing action and observation spaces with the number of agents [42] and increased communication overhead for maintaining a central node [43]. DTE methods rely solely on local agent information, making them easier to implement. However, they struggle with non-stationarity caused by concurrent agent training, leading to unstable learning, poor convergence [41], and less accurate models [43] due to limited observability. CTDE methods use centralized information during training but execute decentralized by using each agent's local information. By combining the benefits of centralized training and decentralized execution, CTDE methods are more scalable than CTE methods, do not require communication during execution, and can perform well. CTDE methods are the most commonly used method [42]. Thus, the CTDE paradigm is adopted in this paper.

The schematic of MARL is illustrated in Fig. 2, where multiple agents select individual actions, forming a joint action [41]. This joint action updates the environment state, and agents receive local observations and a joint reward for the decision. Because a joint action is required, all agents should make decisions together.

Since a joint action is required, scheduling distributed workshops may not be suitable for MARL. Fig. 3 presents Gantt charts in distributed manufacturing with three factories. Each factory is assumed to have a reconfigurable permutation flowshop, meaning the job sequence remains identical across all machines. Therefore, only the first machine, M1, requires an agent to make scheduling or reconfiguration decisions. In Fig. 3 (A), for the scheduling decisions, under the independent decision mode, factory f3 receives a reward after its action fully executes, accurately reflecting the action's impact. However, MARL requires a joint action, forcing f3 to make a new decision when f2 begins processing the next job, even if f3's previous action is still executing. This misalignment leads to inaccurate reward feedback for f3, making MARL less suitable for frequent scheduling decisions with short processing times. Considering that literature [33] effectively used DRL to solve the distributed permutation flowshop scheduling problem, we also use DRL to make scheduling decisions.

For reconfiguration decisions, each factory reconfigures after processing several jobs. Fig. 3 (B) compares execution durations of a reconfiguration action for factory f3 under independent and joint decision modes. Since reconfiguration actions take longer to execute, forcing factory f3 to decide earlier than its expected decision point has a relatively smaller impact, and the reward can still reasonably reflect action effectiveness. Given the cooperative nature of distributed workshops, MARL can be explored for the collaborative reconfiguration of multiple workshops, and this study investigates its potential performance improvements. Thus, this paper uses DRL and MARL to train scheduling and reconfiguration systems, respectively.

This paper aims to address the dynamic distributed reconfigurable flowshop scheduling problem (DRFSP) by considering dynamic cooperation between scheduling and reconfiguration among distributed workshops, through integrating the MARL and DRL approaches, referred to as MADRL. MARL is employed to train workshop agents to make cooperative reconfiguration decisions to select proper production line types, while DRL is used to train the scheduling agent to select suitable jobs for each workshop. Markov decision models were designed for the reconfiguration system and the scheduling system, respectively. Furthermore, a coordination mechanism was designed to coordinate the scheduling and reconfiguration processes. Extensive experiments showed the superior learning efficiency and solution quality of the proposed MADRL algorithms over DRL and metaheuristic methods. The main contributions are as follows.

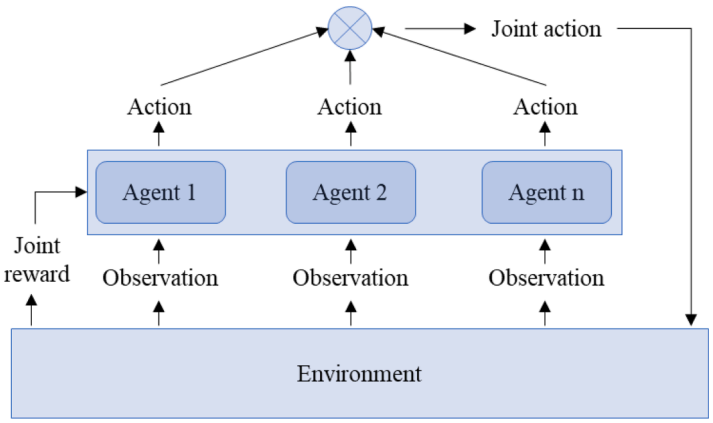


Fig. 2. Schematic of multi-agent reinforcement learning [41].

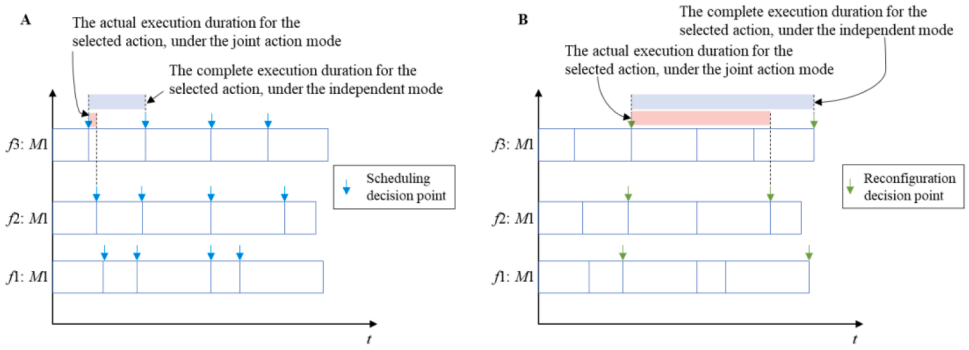


Fig. 3. Illustration of action execution duration under independent and joint decision-making modes in multiple factories/workshops: A) scheduling decisions, B) reconfiguration decisions.

1) We proposed a novel architectural framework for real-time scheduling and reconfiguration in dynamic DRFSP by integrating MARL and DRL. The training and decision-making processes were designed to accommodate the characteristics of distributed manufacturing and reconfigurable production. Specifically, the DRL-based scheduling agent leveraged data from multiple workshops to enhance generalization, while MARL-based reconfiguration agents enabled collaborative reconfiguration based on global production information. Extensive experimental results demonstrated that our MADRL approach significantly outperformed popular single-agent DRL methods and efficient metaheuristics across diverse problem instances.
2) We formulated the distributed scheduling and reconfiguration problems as Markov decision processes, with DRL governing scheduling and MARL handling reconfiguration. For both scheduling and reconfiguration systems, problem-specific reward functions, action spaces, and state representations were designed. The reward functions were directly aligned with optimization objectives, capturing the impact of action execution on the overall production process. The state representations effectively incorporated both static and dynamic system information while remaining independent of the number of jobs and machines, ensuring scalability to different problem sizes. The action design, based on well-known and problem-specific dispatching rules, enabled the system to efficiently prioritize job sequencing and production line selection under various scenarios. The experimental results demonstrated a strong correlation between reward optimization and objective value improvement, validating the effectiveness and learning efficiency of our approach.
3) We introduced a novel coordination mechanism to synchronize scheduling and reconfiguration processes in distributed and reconfigurable manufacturing systems. Considering the scalability and credit assignment problems [44], we adopted the popular centralized training and decentralized execution paradigm, enabling reconfiguration agents across heterogeneous workshops to make cooperative decisions in a partially observable environment. Additionally, we proposed four MADRL algorithms for jointly optimizing scheduling and reconfiguration policies. Comparative experiments confirmed that our MADRL algorithms significantly outperformed widely used single-agent DRL approaches without coordination, achieving superior solution quality and decision-making stability.

The rest of this paper is organized as follows: Section 2 reviews some related literature on the DRL-based workshop scheduling problem. Section 3 provides the system architecture and mathematical formulation. Sections 4 and 5 establish the Markov decision models for the reconfiguration and scheduling systems. Section 6 provides MARL algorithms. Section 7 illustrates training and comparison experiments. Section 8 concludes this paper.

2. Related work

With DRL advancing swiftly and demonstrating high performance in real-time scheduling tasks, it has been increasingly applied to tackle several workshop scheduling problems, considering diverse production characteristics. One of the extensively researched workshop scheduling problems employing DRL is the JSP. Li et al. [30] studied the dynamic flexible JSP considering AGV transportation using a hybrid deep Q network (DQN) algorithm. Liu and Huang [27] addressed dynamic JSP with random job arrivals and machine breakdowns, utilizing graph neural network (GNN) and proximal policy optimization (PPO) algorithms. Wang et al. [38] employed the MARL algorithm to tackle JSP with resource pre-emption, training job agents to make cooperation

decisions concerning available robots. Zhang et al. [45] investigated flexible JSP using DRL with multi-agent graphs. Oh et al. [46] investigated the flexible JSP using a distributional DRL algorithm. Chen et al. [47] used disjunctive graph embedding and DRL to solve the JSP. Xu et al. [48] combined genetic programming and DRL to solve the dynamic JSP, where DRL actions were evolved through Niching genetic programming.

Within DRL-based FSP research, Li et al. [49] explored large-scale flexible FSP using the double DQN algorithm. Yang and Xu [21] addressed reconfigurable FSP by concurrently considering dynamic scheduling and reconfiguration, leveraging an actor-critic algorithm. Zhang et al. [39] employed MARL to address the multi-stage hybrid FSP with machine deterioration, demonstrating its effectiveness in solving the joint decision problem.

For the distributed workshop scheduling problem, Ren et al. [50] investigated the DPFSP using the NASH Q-learning algorithm. Huang et al. [34] solved the distributed JSP using GNN and PPO algorithms to make hierarchical decisions. Yang et al. [51] used a variant of the DQN algorithm to solve the distributed workshop scheduling problems with reconfiguration, but the cooperation among heterogeneous workshops is overlooked. Most studies adopted the single-agent reinforcement learning (SARL) approach for joint decisions. However, for complex manufacturing systems with multiple coupled decisions, it is challenging for SARL to offer system-level optimization decisions [39].

For the MARL-based workshop scheduling problem, Li et al. [52] used a multi-agent PPO algorithm to solve the JSP considering limited automated guided vehicles (AGVs). Three kinds of agents were modelled for job filtering, job selection, and AGV selection. Li et al. [35] studied the reconfigurable shop scheduling problem with batch processing and worker cooperation using MARL. They modelled two kinds of agents. The high-level job sorting agent assigns jobs to virtual manufacturing cells, and the low-level agent selects a job to use the equipment and workers. Gui et al. [53] studied collaborative scheduling by considering warehouse, buffer, machines, and AGV in a manufacturing system using MARL algorithms. Zhang et al. [39] used the counterfactual MARL algorithm to solve the production scheduling and maintenance problem. However, no research has used MARL to solve the dynamic DRFSP.

The literature review illustrates extensive DRL and MARL application in addressing diverse workshop scheduling challenges marked by varying production attributes. Commonly used DRL methodologies include the DQN algorithm, the PPO algorithm, and their variations. DRL-based and MARL-based scheduling literature set the foundation for modeling and algorithmic frameworks in our dynamic DRFSP investigation. However, solving dynamic DRFSP presents heightened complexities due to simultaneous scheduling and reconfiguration cooperation across multiple workshops. Since MARL aids cooperative decision-making training, this study integrates DRL and MARL to enable real-time scheduling solutions for the dynamic DRFSP.

3. System framework and mathematical formulation

3.1. Architecture framework for RL-based DRFSP

The studied problem is the dynamic DRFSP considering new job arrivals. There are F reconfigurable workshops distributed in different regions. Each workshop contains a reconfigurable flow line, which can be reconfigured to X production line types by rearranging and replacing different machines and resources. Each workshop calls the scheduling or reconfiguration agent to select the next candidate job for production or the next production line type to be reconfigured. When reconfigured, a workshop requires a setup time t_{setup} to adjust its production line. Each production line features permutation characteristics, ensuring consistent job sequences across all machines. Thus, only the first machine within a workshop calls the scheduling/reconfiguration agent to make decisions.

Fig. 4 illustrates the framework for intelligent decision-making in distributed reconfigurable workshops, leveraging MARL and DRL. When a job is finished in the first machine of a workshop, a scheduling or reconfiguration decision is required. Based on the current production status, the scheduling and reconfiguration coordination mechanism determines whether to make scheduling or reconfiguration decisions. When making decisions, the scheduling or reconfiguration agent receives the current states, outputs an action, selects a candidate job or production line type, and receives a reward for this step. The states, actions, and rewards are stored as transitions for agent training. The scheduling and reconfiguration agents are trained by DRL and MARL,

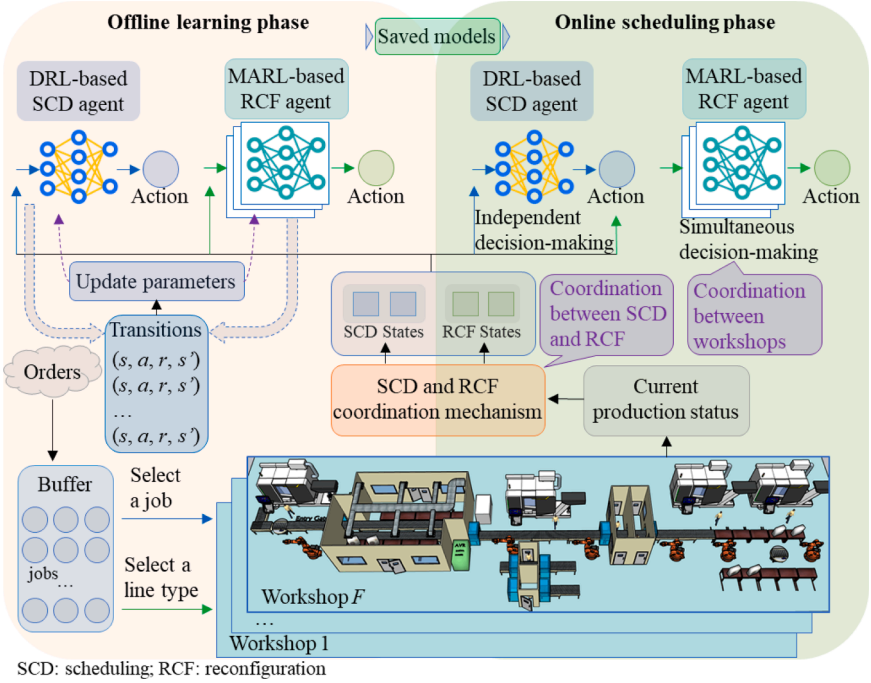


Fig. 4. A comprehensive framework for the dynamic DRFSP using MARL and DRL.

respectively.

Fig. 5 depicts the scheduling and reconfiguration processes for distributed workshops. In workshop f, the scheduling agent selects a job for processing, while the reconfiguration agent chooses a production line for reconfiguration. Utilizing a MARL approach, the reconfiguration system combines the F reconfiguration actions into a joint action $\boldsymbol{A}$. Following the execution of the joint action, a joint reward $\boldsymbol{R}$ is generated for the concurrent decision, as shown in Fig. 2. The scheduling and reconfiguration processes finish when all jobs in the system are scheduled and processed.

3.2. Mathematical formulation

Dynamic job arrivals are considered in the DRFSP. Randomly arriving jobs have varying due dates. If a job exceeds its deadline, a tardiness penalty is incurred. The optimization goal is to minimize the overall tardiness cost in the system, see Eq. (1). Based on the notations in Table 1, the problem formulation is as follows.

$$\textbf{Minimize}: \sum_{j=1}^{n} \{\alpha_j \times \max(0, C_j - d_j)\} \tag{1}$$

Subject to:

$$C_j = C_{mj}, \forall j = 1, 2, \ldots, n \tag{2}$$

$$d_j = [CP(1 - \mathrm{TF} - \frac{\mathrm{RDD}}{2}), CP(1 - \mathrm{TF} + \frac{\mathrm{RDD}}{2})], j = 1, 2, \ldots, n \tag{3}$$

$$CP = \frac{1}{m} \times \sum_{j=1}^{n} \sum_{i=1}^{m} t_{ij} \tag{4}$$

$$C_{1j} - t_{1j} \geq AT_j, \forall j = 1, 2, \ldots, n \tag{5}$$

$$C_{ij} - t_{ij} \geq \max\{C_{(i-1)j}, C_{i(j-1)}\}, i = 2, 3, \ldots, m, j = 1, 2, \ldots, n \tag{6}$$

$$C_{0j} = 0, C_{i0} = 0 \tag{7}$$

Eqs. (2) and (3) define the completion time and due date of a job. Based on literature [54], the due date d_j follows a uniform distribution and is determined by Eq. (4), with TF and RDD set at 0.5 [54]. Eq. (5) ensures that a job can only be processed (on the first machine) after its arrival. Eq. (6) ensures a job can be processed on a machine after the job is finished on the previous machine and the receiving machine is ready. Eq. (7) defines the initial completion time of a job and a machine.

Table 1
The definition of notations.

	Notations	Definitions
Indexes	j, i, k, f	Job, machine, production line, workshop
	n, m, X, F	The quantity of jobs, machines, production lines, and workshops
	f', k'	The current workshop, production line
Variables	BF_k, $BF_{k'}$	The buffer of production lines k, and k'
	t_{ij}, C_{ij}	The processing and completion times of job j on machine i
	t_c	Current production simulation time
	d_j	Due date of job j
	α_j, ψ_j	Unit and current unit tardiness cost of job j
	CP	Assessing the time taken to complete all jobs
Constant	TF	Tardiness factor
	RDD	Relative due date

4. Reconfiguration system modeling

This section constructs the MARL-based model for the reconfiguration system. A coordination mechanism for scheduling and reconfiguration is developed. Additionally, detailed designs for the joint reward, reconfiguration actions, local observations, and global states are provided.

4.1. Coordination mechanism between scheduling and reconfiguration

The coordination mechanism determines when to trigger the scheduling or reconfiguration agent to take action. The reconfiguration agent will be activated under one of the following three conditions:

1) If no jobs are left in the current buffer $BF_{k'}$, this implies the current production line k' must reconfigure to other types to continue processing jobs.

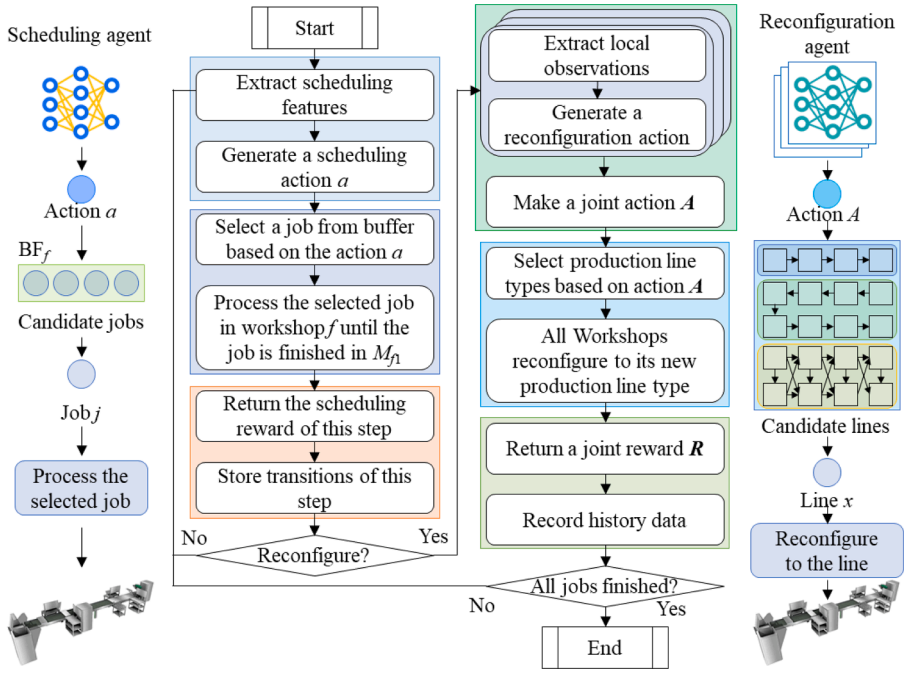


Fig. 5. Reconfigurable production processes utilizing DRL and MARL agents for distributed workshops.

2) The jobs in $BF_{k'}$ are within their deadlines, and the current unit tardiness cost $\psi_{k'}$ is below the Pth percentile of the whole set Ψ, as shown in Eqs. (8) and (9). This condition implies that the current production line's urgency is lower compared to other lines.

$$\psi_{k'} = \frac{1}{n_{k'}} \sum_{j=1}^{n_{k'}} \psi_j, k' = 1, 2, ..., X \tag{8}$$

$$\Psi = \{\psi_1, \psi_2, ..., \psi_X\} \tag{9}$$

3) Upon processing another job (n_{jud}), if $\psi_{k'}$ is below the Pth percentile of Ψ, where n_{jud} is computed as n_{jud}=max{n_{prc}, $n/(X^*n_{\text{rcf\_}X})$}. Here, $n_{\text{rcf\_}X}$ represents the minimum reconfiguration times. After tuning, the values of n_{prc}, $n_{\text{rcf\_}X}$, and P were set to 3, 3, and 70, respectively. This criterion lowers the frequency of reconfiguration decisions, allowing a reconfiguration decision to be made only after processing n_{prc} jobs.

4.2. Joint reward of reconfiguration agents

Reconfiguration agents aim to minimize the cumulative tardiness cost incrementally at each decision step. As depicted in Fig. 6, the current decision step's tardiness cost originates from the buffer (BF), work-in-progress (WIP), and finished jobs (FNS). Therefore, the joint reconfiguration reward $\boldsymbol{R}$ during [t_S, $t_{S'}$] is formulated as follows.

$$R = -\frac{1}{t_{S'} - t_S}(TP_{BF} + TP_{WIP} + TP_{FNS}) \tag{10}$$

$$TP_{BF} = \sum_{k=1}^{X} \sum_{j=1}^{n_k} \alpha_j z_{j_{S'}} \left[t_{S'} - \max(t_S, d_j)\right] \tag{11}$$

$$TP_{WIP} = \sum_{f=1}^{F} \sum_{j \in WIP_f} \alpha_j z_{j_{S'}} \left[t_{S''} - \max(t_S, d_j)\right] \tag{12}$$

$$z_{j_{S'}} = \begin{cases} 1, d_j < t_{S'} \\ 0, \quad \text{else} \end{cases} \tag{13}$$

$$t_{S''} = \begin{cases} C_j, \text{if job} j \text{is finished at} t_{S'} \\ t_{S'}, \qquad\qquad \text{else} \end{cases} \tag{14}$$

$$TP_{FNS} = \sum_{f=1}^{F} \sum_{j \in FNS_f} \alpha_j z_{jC} \left[C_j - \max(t_S, d_j)\right] \tag{15}$$

$$z_{jC} = \begin{cases} 1, d_j < C_j \\ 0, \quad \text{else} \end{cases} \tag{16}$$

Where TP_{BF}, TP_{WIP}, and TP_{FNS} represent the additional tardiness penalty from BF, WIP, and FNS, respectively; n_k indicates the number of jobs in BF_k; z_{jC} and $z_{jS'}$ indicate whether job j reached the due date times C_j and $t_{S'}$, respectively; and $t_{S''}$ denotes the actual end time of job j during [t_S, $t_{S'}$].

4.3. Reconfiguration action representation

Reconfiguration actions determine the subsequent production line type to be reconfigured by a workshop. By analyzing decision objectives and optimization actions, as depicted in Fig. 7, eleven actions (A_1-A_{11}) are designed from various perspectives to minimize the overall increase in tardiness cost.

1) Choose the line having the highest total ψ_j. This criterion prioritizes the line generating the most significant tardiness penalty per unit time.

$$A_1 = \text{argmax}_k \left(\sum_{j=1}^{n_k} \psi_j \right), \ \forall k$$

2) Choose the line with the highest average ψ_j. This action complements A_1 by favoring lines with fewer jobs but higher unit tardiness penalties.

$$A_2 = \text{argmax}_k \left(\frac{1}{n_k} \sum_{j=1}^{n_k} \psi_j \right), \ \forall k$$

3) Choose the line with the most jobs beyond their deadlines.

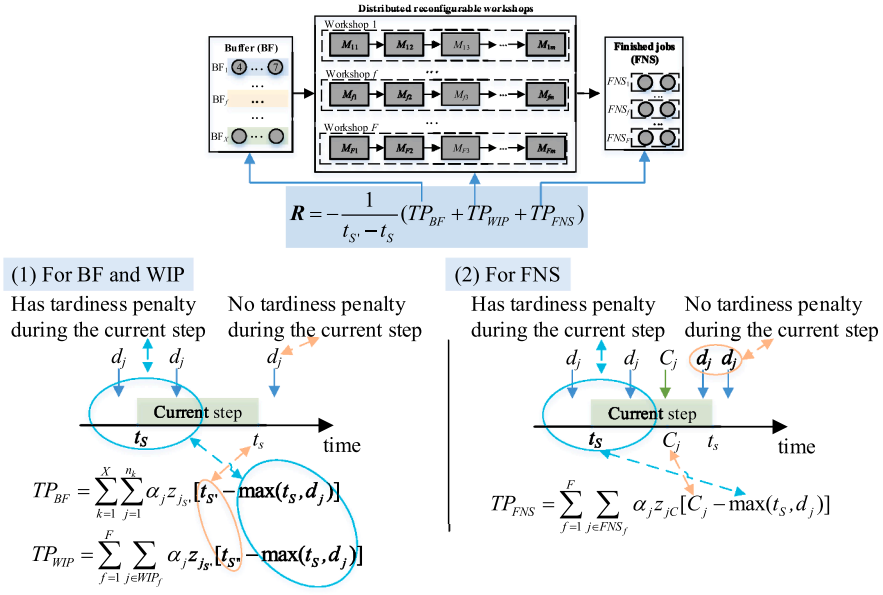


Fig. 6. Schematic diagram of the joint reward design for reconfiguration agents.

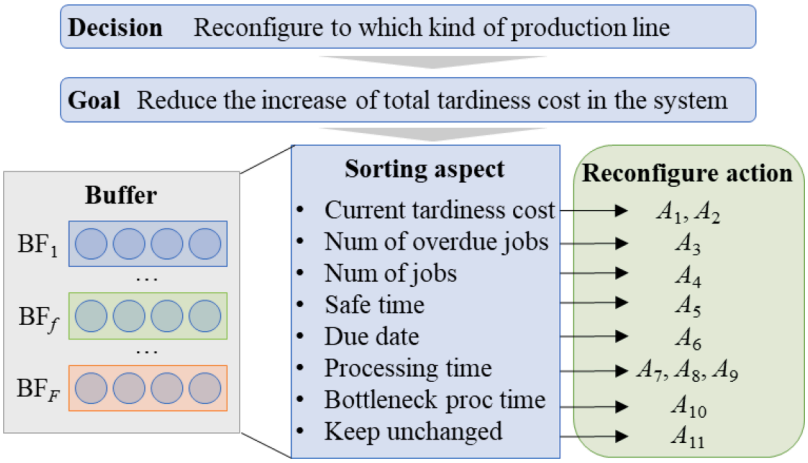


Fig. 7. Schematic diagram of reconfiguration action design.

$$A_3 = \text{argmax}_k\left(\sum_{j\in BF_k} z_{jS'}\right),\ \forall k$$

4) Choose the line with the most jobs in its buffer. This action is particularly beneficial when no jobs are overdue.

$$A_4 = \text{argmax}_k(n_k),\ \forall k$$

5) Choose the line having the lowest average safe time.

$$A_5 = \text{argmin}_k\left(\frac{1}{n_k}\sum_{j=1}^{n_k} ST_j\right),\ \forall k$$

The safe time (ST_j) for job j is computed as follows.

$$ST_j = d_j - \sum_{i=1}^{m} t_{ij} - t_c \tag{17}$$

6) Choose the line with the earliest average due date.

$$A_6 = \text{argmin}_k\left(\frac{1}{n_k}\sum_{j=1}^{n_k} d_j\right),\ \forall k$$

7) Choose the line having the shortest average processing time.

$$A_7 = \text{argmin}_k\left(\frac{1}{n_k}\sum_{j=1}^{n_k}\sum_{i=1}^{m} t_{ij}\right),\ \forall k$$

8) Choose the line having the highest average processing time.

$$A_8 = \text{argmax}_k\left(\frac{1}{n_k}\sum_{j=1}^{n_k}\sum_{i=1}^{m} t_{ij}\right),\ \forall k$$

9) Choose the line having the longest processing time.

$$A_9 = \text{argmax}_k\left(\sum_{j=1}^{n_k}\sum_{i=1}^{m} t_{ij}\right),\ \forall k$$

10) Choose the line having the lowest average bottleneck process time. This action is advantageous in specific scenarios, as bottleneck processing time directly impacts machine utilization and completion time.

$$A_{10} = \text{argmin}_k\left[\frac{1}{n_k}\sum_{j=1}^{n_k} \underset{i}{max}(t_{ij})\right],\ \forall i,k$$

11) Choose not to reconfigure, potentially reducing unnecessary reconfigurations.

4.4. Local observation and global state

4.4.1. Local observation

Under the MARL paradigm of centralized training and decentralized execution, each reconfiguration agent only receives its local observation when making decisions. In selecting the next production line type, the key consideration revolves around the buffer information. Thus, local observation is designed by extracting critical features and characteristics of the current buffer $BF_{k'}$.

1) $ft_1 = n_{k'}$, job count of $BF_{k'}$.
2) $ft_2 = \{\psi_j\}, \forall j \in BF_{k'}$, the current unit tardiness cost of $BF_{k'}$.
3) $ft_3 = \{\alpha_j\}, \forall j \in BF_{k'}$, the unit tardiness cost of $BF_{k'}$.
4) $ft_4 = \{ST_j\}, \forall j \in BF_{k'}$, the safe time ($ST_j$) of $BF_{k'}$.
5) $ft_5 = \{d_j\}, \forall j \in BF_{k'}$, the due date of $BF_{k'}$.
6) $ft_5 = \{AT_j\}, \forall j \in BF_{k'}$, the arrival time ($AT_j$) of $BF_{k'}$.
7) $ft_7 = \{\sum_{i=1}^{m} t_{ij}\}, \forall j \in BF_{k'}$, the total processing time of $BF_{k'}$.
8) $ft_8 = \{u_{jf'}\}, \forall j \in BF_{k'}$, job utilization rate estimation in workshop f'.

The $u_{jf'}$ is calculated as follows, with $WT_{ijf'}$ representing the total waiting time of machine i for job j in workshop f'.

$$u'_{jf'} = 1 - \left(\sum_{i=2}^{m} WT'_{ijf'} \Big/ \sum_{i=1}^{m} t_{ij}\right), j \in BF_{k'} \tag{18}$$

As ft_2 through ft_8 are arrays, each feature is represented by four statistical measures (max, min, mean, and dev). For ft_2, both zero and nonzero value counts are included. Consequently, the dimension of the reconfiguration agent's local observation is 31 (1+7*4+2).

4.4.2. Global state representation

The global state can be used for the credit assignment of the joint reward. By considering the information of all buffers (BF_1, BF_2, …, BF_k, …, BF_X), nine global state features (Ft_1, Ft_2, …, Ft_9) are designed.

1) $Ft_1 = \{n_1, n_2, ..., n_k, ..., n_X\}$, buffer job counts.
2) $Ft_2 = \left\{\sum_{j=1}^{n_1}\psi_j, \sum_{j=1}^{n_2}\psi_j, ..., \sum_{j=1}^{n_k}\psi_j, ..., \sum_{j=1}^{n_X}\psi_j\right\}$, the total ψ_j of each buffer.
3) $Ft_3 = \left\{\frac{1}{n_1}\sum_{j=1}^{n_1}\psi_j, \frac{1}{n_2}\sum_{j=1}^{n_2}\psi_j, ..., \frac{1}{n_k}\sum_{j=1}^{n_k}\psi_j, ..., \frac{1}{n_X}\sum_{j=1}^{n_X}\psi_j\right\}$, the average ψ_j of each buffer.
4) $Ft_4 = \left\{\frac{1}{n_1}\sum_{j=1}^{n_1} z_{js'}, \frac{1}{n_2}\sum_{j=1}^{n_2} z_{js'}, ..., \frac{1}{n_k}\sum_{j=1}^{n_k} z_{js'}, ..., \frac{1}{n_X}\sum_{j=1}^{n_X} z_{js'}\right\}$, the number of overdue jobs in each buffer.
5) $Ft_5 = \left\{\frac{1}{n_1}\sum_{j=1}^{n_1} ST_j, \frac{1}{n_2}\sum_{j=1}^{n_2} ST_j, ..., \frac{1}{n_k}\sum_{j=1}^{n_k} ST_j, ..., \frac{1}{n_X}\sum_{j=1}^{n_X} ST_j\right\}$, the average safe time of jobs in each buffer.
6) $Ft_6 = \left\{\frac{1}{n_1}\sum_{j=1}^{n_1} d_j, \frac{1}{n_2}\sum_{j=1}^{n_2} d_j, ..., \frac{1}{n_k}\sum_{j=1}^{n_k} d_j, ..., \frac{1}{n_X}\sum_{j=1}^{n_X} d_j\right\}$, the average due date of jobs in each buffer.
7) $Ft_7 = \left\{\frac{1}{n_1}\sum_{j=1}^{n_1}\sum_{i=1}^{m} t_{ij}, \frac{1}{n_2}\sum_{j=1}^{n_2}\sum_{i=1}^{m} t_{ij}, ..., \frac{1}{n_k}\sum_{j=1}^{n_k}\sum_{i=1}^{m} t_{ij}, ..., \frac{1}{n_X}\sum_{j=1}^{n_X}\sum_{i=1}^{m} t_{ij}\right\}$, the average total process time of each buffer.
8) $Ft_8 = \left\{\sum_{j=1}^{n_1}\sum_{i=1}^{m} t_{ij}, \sum_{j=1}^{n_2}\sum_{i=1}^{m} t_{ij}, ..., \sum_{j=1}^{n_k}\sum_{i=1}^{m} t_{ij}, ..., \sum_{j=1}^{n_X}\sum_{i=1}^{m} t_{ij}\right\}$, the total processing time of jobs in each buffer.
9) $Ft_9 = \left\{\frac{1}{n_1}\sum_{j=1}^{n_1} max_i(t_{ij}), \frac{1}{n_2}\sum_{j=1}^{n_2} max_i(t_{ij}), ..., \frac{1}{n_k}\sum_{j=1}^{n_k} max_i(t_{ij}), ..., \frac{1}{n_X}\sum_{j=1}^{n_X} max_i(t_{ij})\right\}$, the average bottleneck process time of each buffer.

Since the nine global states are arrays, four statistical features are also used to reflect the array features. Thus, the total dimension for the global state is 36.

5. Scheduling system modelling

In this section, we outline the rewards, actions, and state representations for the scheduling system. The scheduling agent generates an action based on the current state. Upon executing the action, a reward is returned to evaluate its performance.

5.1. Reward of the scheduling agent

For workshop f', the scheduling agent chooses a job from its current buffer $BF_{k'}$. To minimize the overall tardiness cost, each scheduling action should minimize the tardiness cost increase resulting from its current production line type. As shown in Fig. 8, the scheduling reward at the current decision step originates from both the buffer and WIP. Hence, the scheduling reward r during the time step $[t_s, t_{s'}]$ is formulated as follows.

$$r = -\frac{1}{t_{s'} - t_s}\left(tp_{BF_{k'}} + tp_{WIP_{F_{k'}}}\right) \tag{19}$$

$$tp_{BF_{k'}} = \sum_{j \in BF_{k'}} z_{js'}\alpha_j\left[t_{s'} - \max(t_s, d_j)\right] \tag{20}$$

$$tp_{WIP_{F_{k'}}} = \sum_{f \in F_{k'}}\sum_{j \in WIP_f} z_{js'}\alpha_j\left[t_{s''} - \max(t_s, d_j)\right] \tag{21}$$

$$z_{j_{s'}} = \begin{cases} 1, d_j < t_{s'} \\ 0, \quad \text{else} \end{cases} \tag{22}$$

$$t_{s''} = \begin{cases} C_j, \text{job} j \text{is finished at} t_{s'} \\ t_{s'}, \qquad\qquad \text{else} \end{cases} \tag{23}$$

Where $tp_{BF_{k'}}$ and $tp_{WIP_{F_{k'}}}$ denote the recently generated tardiness cost of $BF_{k'}$ and all WIP belonging to k' during $[t_s, t_{s'}]$. The $z_{js'}$ denotes if job j is overdue at time $t_{s'}$, and $t_{s''}$ denotes the actual end time of a job for calculating the increased tardiness cost compared with the end time $t_{s'}$.

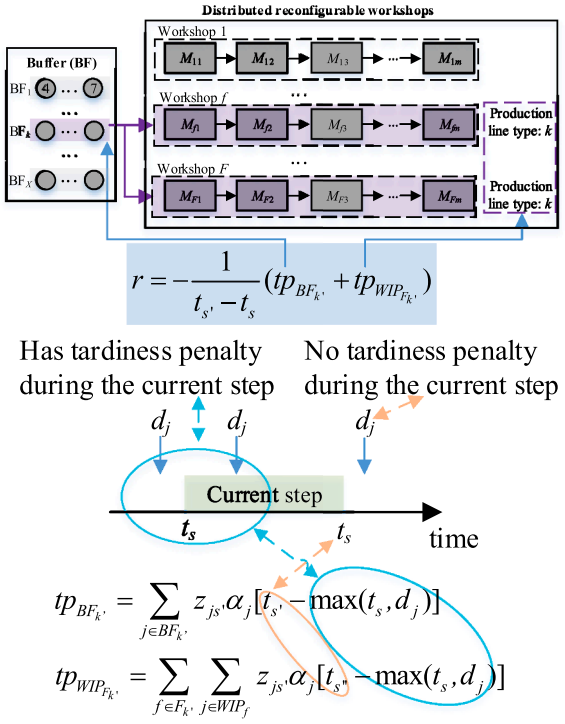


Fig. 8. Schematic diagram of scheduling reward design.

5.2. Scheduling action representation

For workshop f', the scheduling action chooses a candidate job from the corresponding buffer $BF_{k'}$. To reduce the total tardiness cost of this buffer, we designed eight scheduling actions (a_1-a_8) by considering some problem-specific and famous priority dispatching guidelines, as shown in Fig. 9.

1) $a_1 = \text{argmax}_j(\psi_j),\ j \in BF_{k'}$
2) $a_2 = \text{argmax}_j(\alpha_j),\ j \in BF_{k'}$
3) $a_3 = \text{argmin}_j(ST_j),\ j \in BF_{k'}$

The a_1 and a_2 select a job from the perspective of the tardiness cost. The a_3 selects the job that is tight by considering the due date and total processing time.

1) $a_4 = \text{argmin}_j(d_j),\ j \in BF_{k'}$, corresponding to the earliest due date dispatching rule.
2) $a_5 = \text{argmin}_j(AT_j),\ j \in BF_{k'}$, corresponding to the first-come-first-out dispatching rule.
3) $a_6 = \text{argmin}_j\left(\sum_{i=1}^{m} t_{ij}\right),\ j \in BF_{k'}$, corresponding to the shortest total processing time dispatching rule.
4) $a_7 = \text{argmax}_j\left(\sum_{i=1}^{m} t_{ij}\right),\ j \in BF_{k'}$, corresponding to the longest total processing time dispatching rule.
5) $a_8 = \text{argmax}_j(u_{jf'}),\ j \in BF_{k'}$, select the job that most closely aligns with the workshop's current production level by considering the machine utilization.

5.3. Scheduling state representation

Recall that the scheduling agent picks a candidate job from the corresponding buffer $BF_{k'}$. The scheduling state features need to mirror the data of $BF_{k'}$. Consequently, the state feature design for the

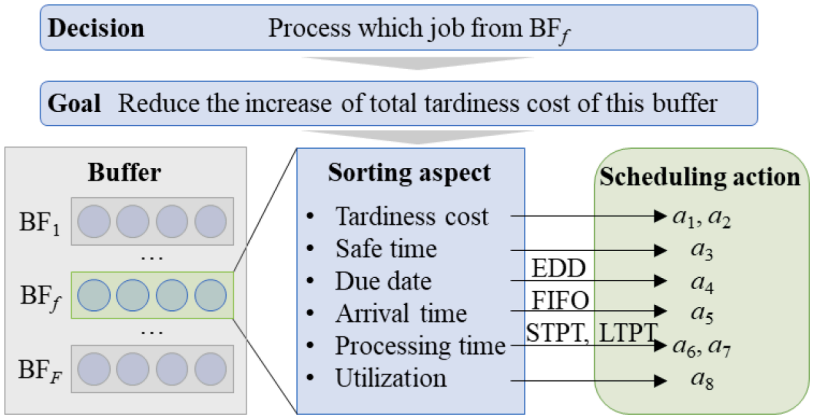


Fig. 9. Schematic diagram of scheduling action design.

scheduling agent is the same as the design of local observations for reconfiguration agents. While the state representation remains consistent, state values differ due to varying trigger times.

6. Multi-agent reinforcement learning algorithms

Recall that the reconfiguration and scheduling systems were solved by MARL and DRL approaches, respectively. For the DRL approach, the DQN algorithm [55] and PPO algorithm [56] were adopted to train the scheduling agents, since the two DRL algorithms have shown good performance and achieved state-of-the-art (SOTA) on some scheduling problems [28,57]. For the MARL approach, the value-decomposition networks (VDN) algorithm [58] and the counterfactual multi-agent reinforcement learning (COMA) algorithm [59], were adapted to train the reconfiguration agents. The two MARL algorithms are value-based and policy-based, respectively. By integrating two MARL algorithms with two DRL algorithms, we provided four MADRL algorithms, i.e., VDN_DQN, COMA_DQN, VDN_PPO, and COMA_PPO, to solve the DRFSP.

6.1. Value-decomposition networks

The VDN algorithm employs a paradigm of centralized training and decentralized execution to facilitate cooperative decision-making. As a value-based MARL approach, its core idea, as outlined in [58], is to decompose the total state-action value $Q_{tot}(\boldsymbol{\tau}, \boldsymbol{A})$ into individual state-action values for each agent. Fig. 10 illustrates the architecture of the VDN algorithm.

As depicted in Fig. 10, the total state-action value $Q_{tot}(\boldsymbol{\tau}, \boldsymbol{A})$ for the system is the sum of the individual state-action values $Q_f(\tau_f, A_f)$ for all agents, as shown below.

$$Q_{tot}(\tau, A) = \sum_f Q_f(\tau_f, A_f) \tag{24}$$

Where A_f and τ_f are the action and action-observation history of agent f, respectively. The $\boldsymbol{A}$ and $\boldsymbol{\tau}$ are the joint action and joint action-

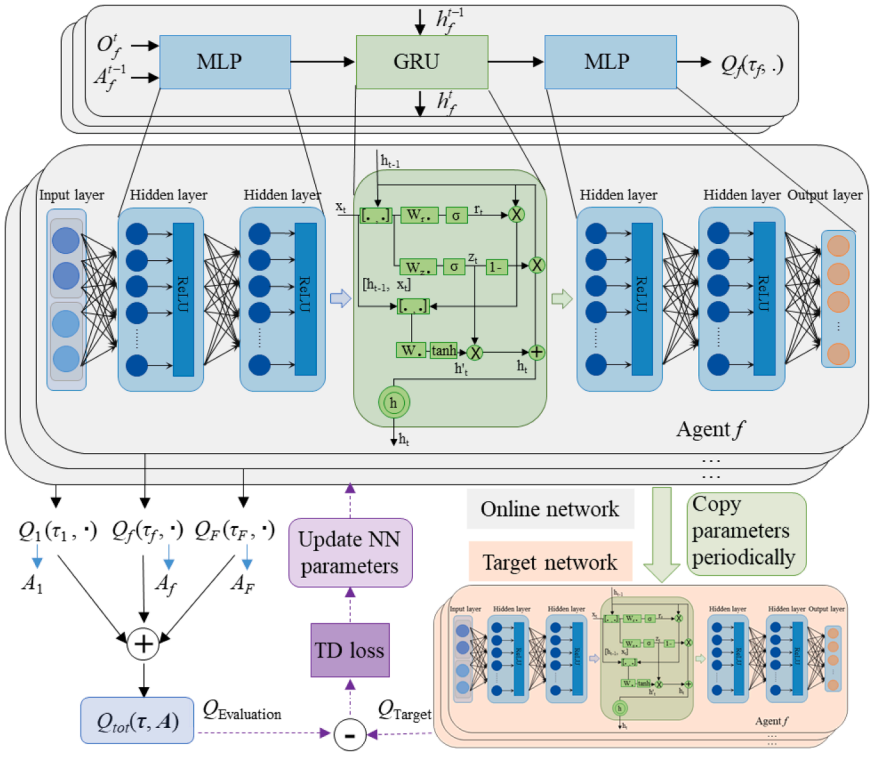


Fig. 10. Schematic diagram of the VDN algorithm.

observation history of all agents.

For agent f, the local observation O_f^t passes through a multi-layer perceptron (MLP), gated recurrent unit (GRU), and MLP to obtain the state-action value $Q_f(\tau_f, .)$. Based on $Q_f(\tau_f, .)$, an action A_f is chosen for agent f. All selected actions make a joint action $\boldsymbol{A}=\{A_1, A_2, \ldots, A_F\}$, which will be executed in the production system.

Throughout centralized training, a joint reward $\boldsymbol{R}$ is provided for the executed joint action. Given the VDN algorithm's two network sets, namely the online network $Q(\cdot; \zeta)$ and the target network $Q(\cdot; \zeta^-)$, the temporal difference (TD) error between these networks serves to adjust the parameters of the online network. The loss function $L(\zeta)$ is computed as follows.

$$L(\zeta) = \mathrm{E}_{(\tau,A,R,\tau')}\left[(R + \gamma V(\tau'; \zeta^-) - Q(\tau; A; \zeta))^2\right] \tag{25}$$

$$V(\tau'; \zeta^-) = \max_{A'} Q(\tau', A'; \zeta^-) \tag{26}$$

Where $\boldsymbol{A}'$ and $\boldsymbol{\tau}'$ are the joint action and joint action-observation history of the next state.

6.2. Counterfactual multi-agent reinforcement learning

As a policy gradient-based MARL algorithm, the COMA algorithm [59] employs a centralized critic and decentralized actor paradigm, where the critic is used during learning, and the actor makes cooperative decisions. Fig. 11 illustrates the execution phase, where each actor receives a local observation O_f and decides U_f based on this observation and hidden information h_f. The collective actions of all agents constitute a joint action $\boldsymbol{u}$. Following its execution, the environment yields a joint reward $\boldsymbol{R}$. In the learning phase, the critic network utilizes the local observations of all agents, the global state, joint action, and joint reward to compute gradients.

To address the credit assignment of the total reward, the COMA algorithm uses a counterfactual baseline, i.e., replacing the action of an agent with a default action to obtain the advantage of the chosen action. The gradient g for an actor is calculated as follows.

$$g = \mathrm{E}_{\pi}\left[\sum_f \nabla_\theta \log \pi^f\left(u^f|\tau^f\right) A^f(s,u)\right] \tag{27}$$

Where $A^f(s, \boldsymbol{u})$ denotes the advantage of agent f's action under joint action $\boldsymbol{u}$ and global state s, $\pi^f(u^f|\tau^f)$ denotes the probability of selecting action u under the local observation τ^f for agent f. The advantage value of agent f is calculated by using the total state-action value subtracting a baseline $b(s,\boldsymbol{u}^{-f})$, as shown below.

$$A^f(s,u) = Q(s,u) - b\left(s, u^{-f}\right) \tag{28}$$

When computing the baseline $b(s,\boldsymbol{u}^{-f})$, the action of agent f is substituted with alternative actions to represent the advantage of the selected action. Consequently, the advantage function is expressed as follows.

$$A^f(s,u) = Q(s,u) - \sum_{u'^f} \pi^f\left(u'^f|\tau^f\right) Q\left(s, \left(u^{-f}, u'^f\right)\right) \tag{29}$$

Where u'^f denotes a possible action of agent f, $\pi^f(u'^f|\tau^f)$ denotes the probability of selecting the action u'^f under action-observation history τ^f, $\boldsymbol{u}^{-f}$ denotes the joint action marginalizes out the action of agent f, i.e., u^f.

The critic uses the temporal difference error to update network parameters. Similar to the VDN algorithm, two sets of networks, i.e., the online and target networks, are used for the critic. The optimization function is as follows.

$$L_t(\theta^c) = \left(y^{(\lambda)} - f^c(\cdot_t, \theta^c)\right)^2 \tag{30}$$

Where $y^{(\lambda)}$ denotes the target state value, λ is a constant, and $f^c(\cdot_{t+n}, \theta^c)$ denotes the state value estimated by the online network with parameter θ^c.

The $y^{(\lambda)}$ can be calculated by the l-step return $G_t^{(l)}$ using bootstrapped values estimated by the target network $f^c(\cdot, \widehat{\theta}^c)$.

$$y^{(\lambda)} = (1-\lambda)\sum_{l=1}^{\infty} \lambda^{l-1} G_t^{(l)} \tag{31}$$

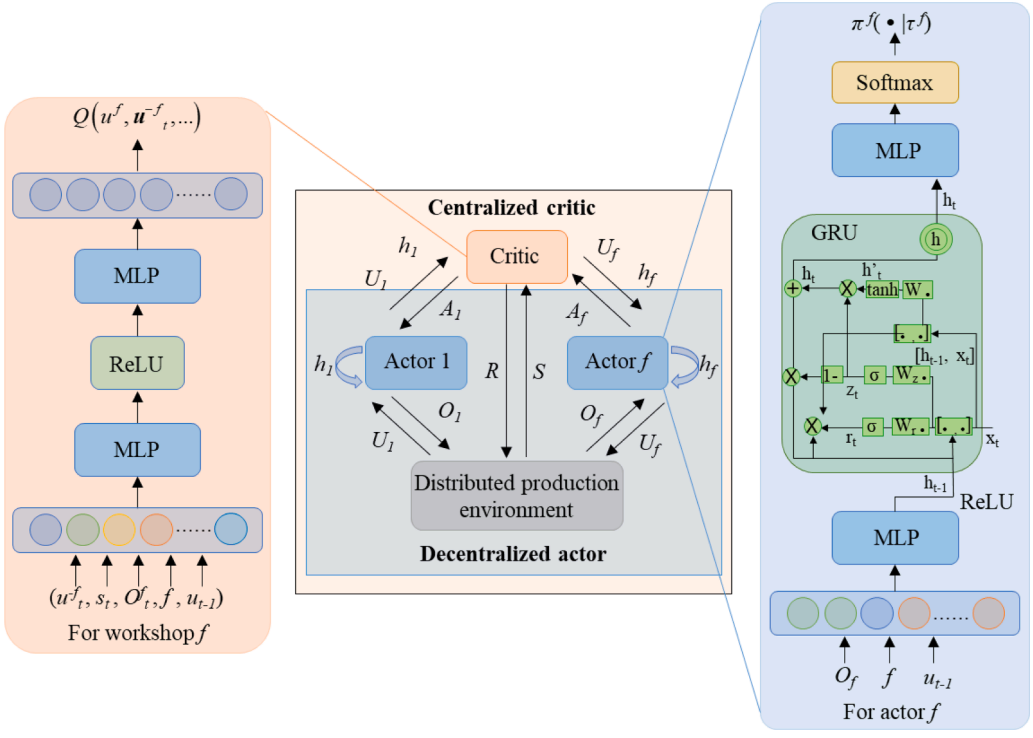


Fig. 11. The architecture of the COMA algorithm under the paradigm of centralized critic and decentralized actor.

$$G_t^{(n)} = \sum_{x=1}^{l} \gamma^{x-1} r_{t+x} + \gamma^l f^c(\cdot_{t+l}, \widehat{\theta}^c) \quad (32)$$

7. Training and comparison experiments

In this section, the scheduling and reconfiguration decision systems are trained utilizing four MADRL algorithms. Subsequently, a comparison is drawn between the adapted MADRL algorithms and four widely-used DRL algorithms, along with three classical meta-heuristics. The four baseline DRL algorithms—DQN [60], double DQN (DbDQN) [61], dueling DQN (DlDQN) [62], and PPO [63]—were widely used to solve workshop scheduling problems [29,34,64]. The three meta-heuristics comprise the iterated greedy algorithm (IG) [21], genetic algorithm (GA) [21], and artificial bee colony algorithm (ABC). Those DRL and meta-heuristic algorithms have been used as baselines for various scheduling problems recently [29,34,65]. All baseline algorithms were carefully adapted to solve the studied DRFSP, considering its specific problem characteristics.

7.1. Experimental design and parameter tuning

To ensure the agents are trained and the learning performance is fairly tested, two distinct sets of datasets, namely the training and test datasets, were created. As depicted in Fig. 12, the training dataset comprising 80 instances was utilized for model training and to assess learning efficiency across training epochs. Subsequently, upon completion of the training phase, the trained models were applied to address the test dataset consisting of 140 instances unseen in the training stage.

The training and test datasets were created by integrating various production configurations, referencing benchmark generations of distributed workshop scheduling problems [11]. The determination of production configuration and parameter settings for instances, as illustrated in Tables 2 and 3, was guided by relevant literature on distributed workshop scheduling [11,66] and DRL-based scheduling problems [18]. Given the consistent number of distributed factories/workshops over an extended period, the quantity of distributed workshops remained fixed, with three workshops involved in this study.

Table 4 provides the parameter settings for algorithms, drawing insights from pertinent DRL-based scheduling literature [17,18,67]. For the scheduling agent trained with the DQN or PPO algorithm, learning occurred before a reconfiguration operation. In contrast, for the reconfiguration agent trained with a MARL algorithm, the learning process took place at the episode's conclusion. The simulation of an instance is regarded as an episode. During each learning iteration, the off-policy algorithm VDN randomly selected a mini-batch of transitions from memory, while the on-policy MARL algorithm COMA utilized the transitions from the current episode to update network parameters.

Based on the tuning outcomes from various DRL-based scheduling literature [33,67], the learning rate notably impacts learning performance. In this study, we examined the learning rate at three different levels for each algorithm, detailed in Table 5. The tuning experiment was carried out for 1000 epochs. Fig. 13 shows that the tuning curves vary notably across the three learning rate levels. Among these levels,

Table 2
Production configurations for the training and test datasets.

	Number of jobs (n)	Number of machines (m)	Number of production line types (X)	Total number of instances
Training dataset	{50, 80, 120, 150, 200}	{5, 8, 10, 15}	{4, 6, 8, 10}	5*4*4=80
Test dataset	{50, 80, 100, 120, 150, 180, 200}	{5, 8, 10, 12, 15}	{4, 6, 8, 10}	7*5*4=140

Table 3
Parameter settings for a training or test instance.

Parameter	Value
The proportion of initial jobs (p_{init})	5 %
Arriving duration	E(1/10)
Processing time t_{ij}	U[100, 200]
Setup time of reconfiguration t_{setup}	U[200, 400]
Unit tardiness cost α_j	U[0, 2]

Table 4
Parameter settings for algorithms.

	Value-based		Policy-based	
	DQN	VDN	PPO	COMA
Hidden layers	2	2	2	2
Neurons within each hidden layer	100	64	100	64
Discount factor (γ)	0.98	0.98	0.98	/
Memory size	1000	300	/	/
Batch size	32	32	/	/
Interval for parameter updates	200	200	/	/
Repeated updating times (B)	/	/	5	/

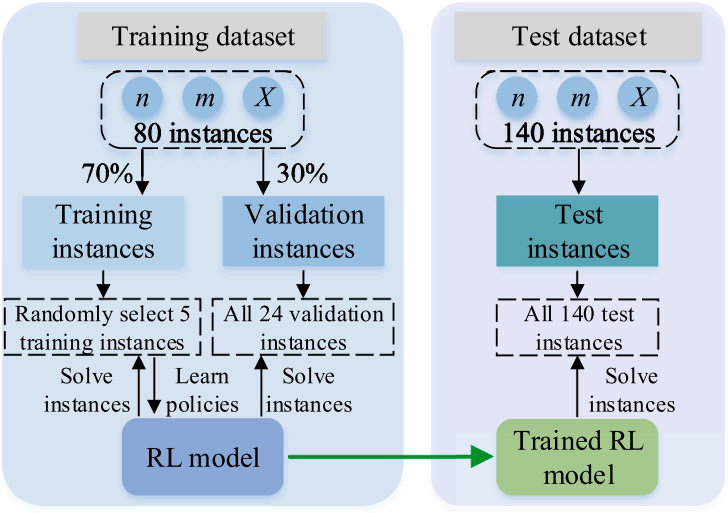


Fig. 12. Usage of the training and test datasets.

Table 5
Learning rate settings for the tuning experiment.

			Factor level (lv)		
			1	2	3
MARL	VDN		0.00003	0.0001	0.0003
	COMA	Actor	0.00003	0.0001	0.0003
		Critic	0.0002	0.0005	0.0015
DRL	DQN		0.00003	0.0001	0.0003
	PPO	Actor	0.00003	0.0001	0.0003
		Critic	0.0001	0.0003	0.001

lv2 exhibited superior learning performance and was consequently selected for all four MADRL algorithms.

7.2. Training processes

The distributed scheduling and reconfiguration system underwent training with the four MADRL algorithms for 10000 epochs, utilizing the optimal learning rates and other appropriate parameter configurations. Throughout each epoch, 5 training instances were chosen at random to train scheduling and reconfiguration agents. Upon completion of each epoch, the trained agents were evaluated across all validation instances. Fig. 14 illustrates the training curves for the four MADRL algorithms.

As depicted in Fig. 14, the objective value exhibited a decreasing trend across all algorithms during training, suggesting the efficacy of all MADRL algorithms in facilitating effective learning. The robust learning performance of these algorithms further validates the feasibility of employing MARL and DRL in addressing concurrent scheduling and reconfiguration challenges within distributed workshops. Additionally, it verifies the soundness of the system design for cooperative scheduling in distributed reconfigurable workshops.

To better illustrate the learning outcomes of the scheduling and reconfiguration agents, we present their average episode rewards in Figs. 15 and 16, respectively. The subplots in both figures exhibit a consistent upward trend during training, suggesting that the agents have successfully acquired improved policies. It's worth noting that the reward curves display some fluctuations, likely attributed to the exploration characteristics inherent in the RL approach and the complexities of the studied problem, in contrast to FSP and JSP.

7.3. Comparison experiment

To evaluate the performance of MADRL approaches in addressing the dynamic DRFSP, we compared the four MADRL algorithms with four popular DRL algorithms and three classical meta-heuristics. The seven baseline algorithms were frequently used to solve [20,68] or as benchmarks [65] in many workshop scheduling problems. For each algorithm, the test dataset was repeatedly calculated three times.

7.3.1. System adaptation and parameter settings for baseline algorithms

For the four DRL baselines, switching from multi-agent to single-

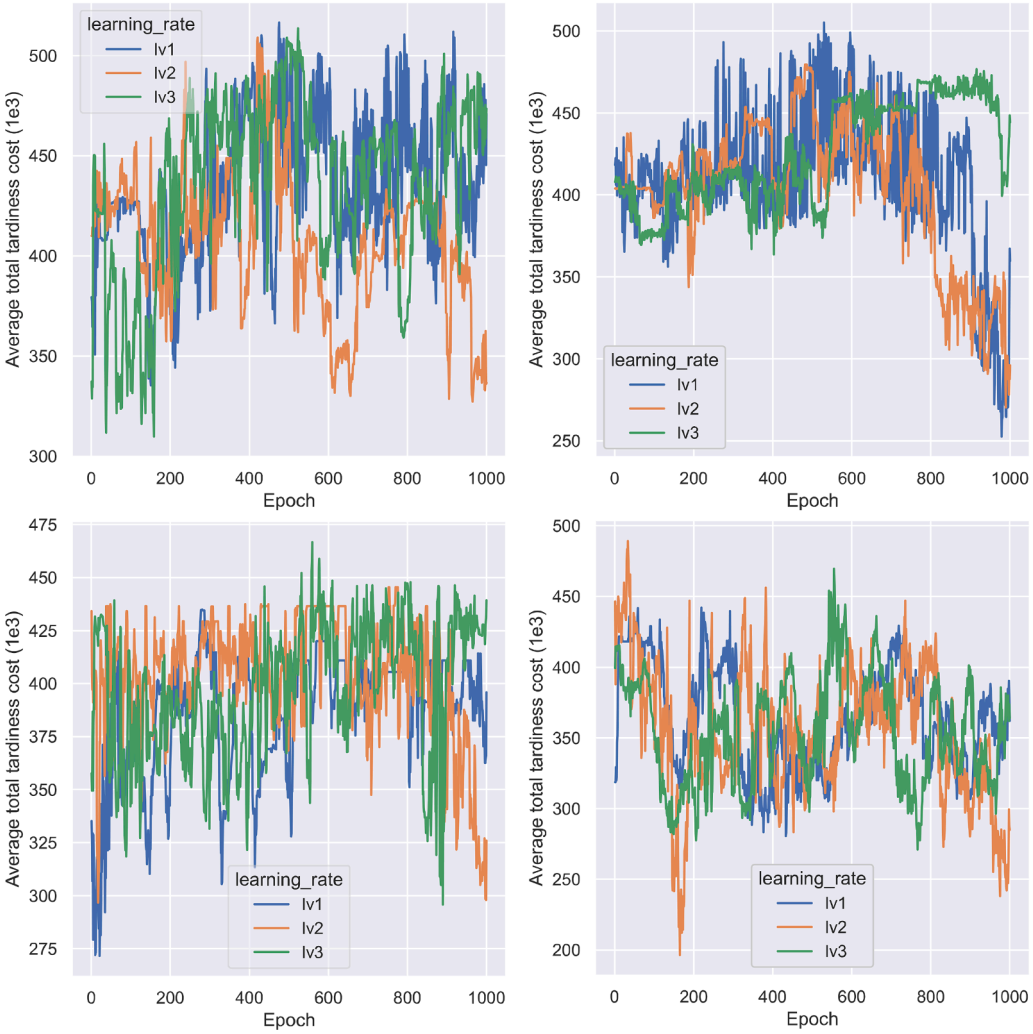


Fig. 13. The tuning curves of learning rates tested at three levels for the four MADRL algorithms.

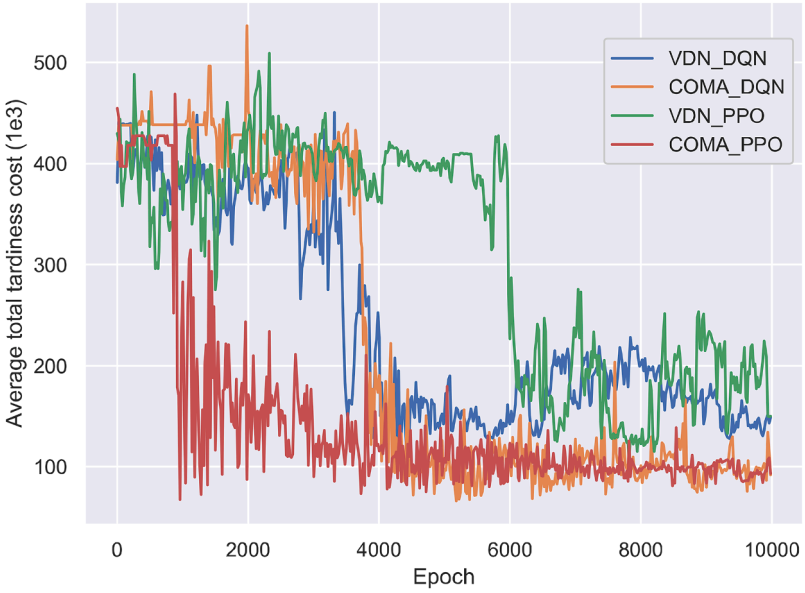


Fig. 14. Training curves showing average total tardiness cost for the four MADRL algorithms.

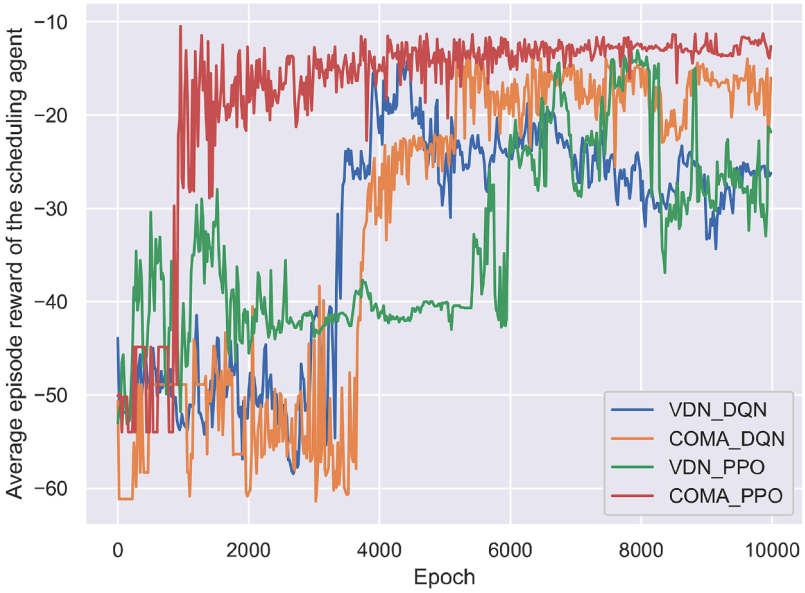


Fig. 15. The average episode rewards of scheduling agents for the four MADRL algorithms.

agent, the system design of the reconfiguration system should be adapted accordingly. For the reconfiguration system, all distributed workshops use the same reconfiguration agent to make decisions independently. Therefore, instead of utilizing the joint reward, we design the individual reconfiguration reward R as follows.

$$R = -\frac{1}{t_{S'} - t_S}\left(TP_{BF} + TP_{WIP_{f'}} + TP_{FNS_{f'}}\right) \tag{33}$$

Where $TP_{WIP_{f'}}$ and $TP_{FNS_{f'}}$ denote the added tardiness costs of WIP and finished jobs in the current workshop f', within the current decision step $[t_S, t_{S'}]$.

For the four DRL baselines, we trained and validated the scheduling and reconfiguration agents using the same training and validation datasets. The learning rates for DQN, DbDQN, and DlDQN algorithms are set to 0.0005. For the PPO algorithm, the learning rates for the critic and actor are 0.0005 and 0.0002, respectively. The training curves depicted in Fig. 17 demonstrate effective learning across all four DRL baselines. Notably, the PPO algorithm exhibited the most optimal learning performance, as indicated by its smoother learning curve with fewer fluctuations.

The parameter settings for meta-heuristics are displayed in Table 6. The comparison experiment was conducted on a computer equipped with an Intel(R) Core(TM) i7-12700F processor.

7.3.2. Performance comparisons

All algorithms are compared on the same set of test instances. The performance metric, relative percentage deviation (RPD), is calculated as follows.

$$RPD = \frac{obj_{alg} - obj_{best}}{obj_{best}} \times 100\% \tag{34}$$

Here, obj_{alg} represents the objective value obtained by an algorithm, while obj_{best} stands for the best objective value across all compared algorithms.

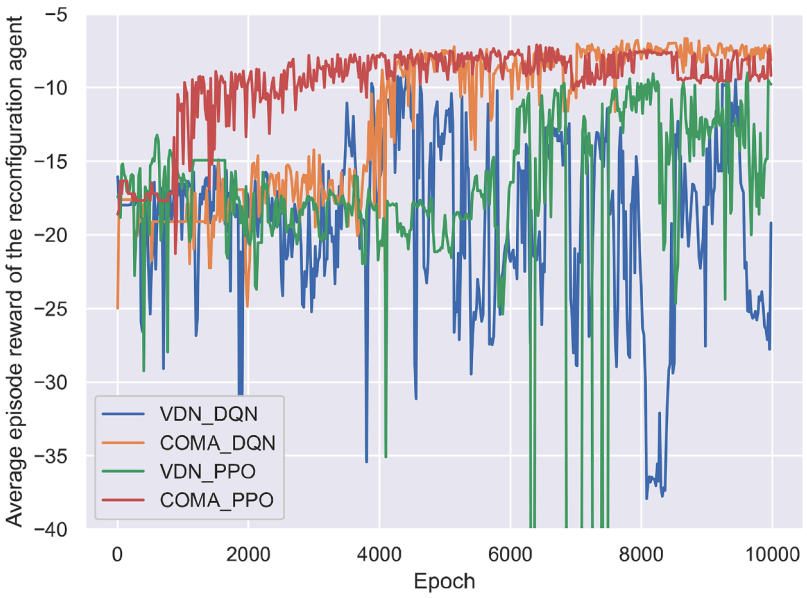


Fig. 16. The average episode rewards of reconfiguration agents for the four MADRL algorithms.

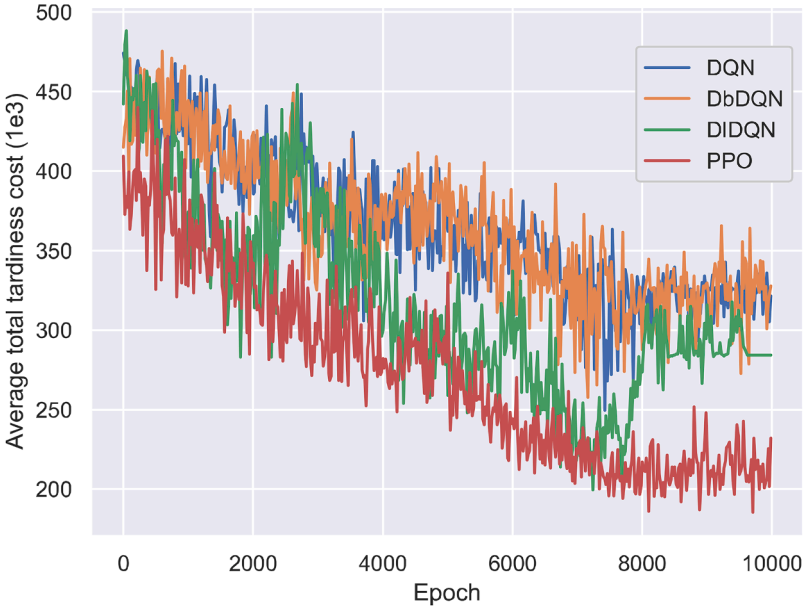


Fig. 17. Training curves of the four DRL baselines.

Table 6
Parameter settings for meta-heuristics.

For all	Rescheduling points	5
	Evolutionary generations	100
IG	Number of destruction jobs	5
GA	Population size	30
	Crossover rate	0.8
	Mutation rate	0.1
ABC	Population size	30
	Number of food sources	5
	Maximum limit	10

Fig. 18 presents the average RPDs obtained by all 11 algorithms. The results displayed in Fig. 18 indicate a significant performance advantage of MADRL algorithms over both DRL algorithms and meta-heuristics. This underscores the efficacy of integrating MARL and DRL in addressing the distributed scheduling and reconfiguration problem. By properly designing the coordination mechanism between distributed workshops and the joint reward for reconfiguration agents, the MARL algorithms can further generate better solutions than those of DRL algorithms.

Among the four MADRL algorithms, the COMA_DQN algorithm achieves the highest performance, with the COMA_PPO algorithm following closely. This indicates the effectiveness of the COMA algorithm in addressing distributed reconfiguration problems. As COMA is an on-policy algorithm, this finding suggests that, for the distributed reconfiguration problem under study, learning from data generated in the current episode is more effective than using replayed data sampled from historical memory. Moreover, the four DRL algorithms outperform the three meta-heuristics clearly, which is in line with some current scheduling studies [29,34,65].

To further evaluate the performance of all compared algorithms

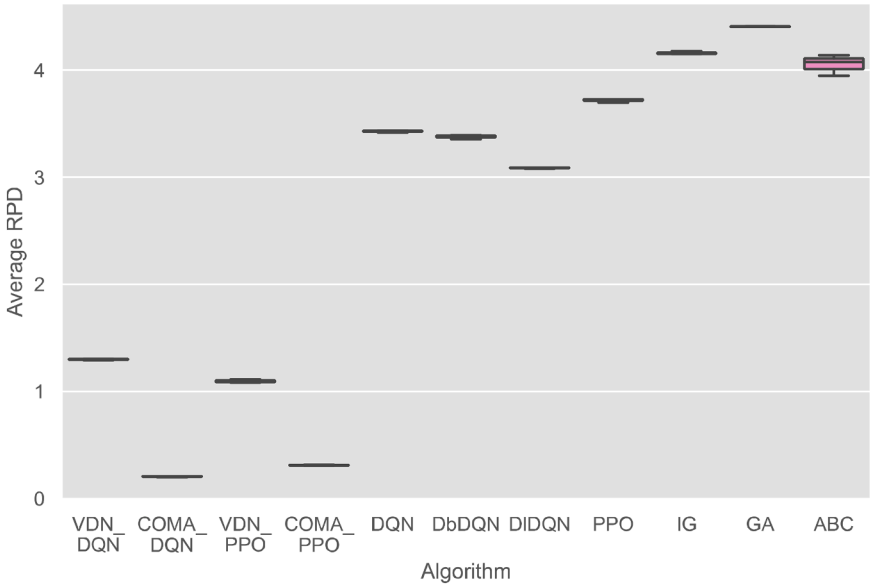


Fig. 18. Comparisons between the four MADRL algorithms and seven baseline algorithms, including four popular DRL algorithms and three effective meta-heuristics.

across various problem scales, we depicted the average RPDs on nine representative test instances in Fig. 19. Additionally, interaction plots between algorithms and problem scales are presented in Fig. 20. From the two figures, we can know that although the algorithm performances differ on different production scales, the MADRL algorithms generally outperform both DRL and meta-heuristics.

Table 7 presents the specific objective values of all compared algorithms. As presented in Table 7, the COMA_DQN algorithm obtains the best results under most production configurations. On average, the COMA_DQN algorithm outperforms the three DQN baselines, i.e., DQN, DbDQN, DlDQN, and PPO algorithms, for 94.2 %, 94.1 %, 93.5 %, and 94.6 %, respectively. In addition, the COMA_DQN algorithm significantly outperforms the three meta-heuristics, validating the effectiveness of integrating the MARL and DRL approaches to solve the distributed and reconfigurable scheduling problem.

7.3.3. *Computation efficiency of RL approaches*

To verify whether the MADRL and DRL approaches can provide real-time scheduling after training, the CPU time for computing a test instance was recorded. Fig. 21 illustrates the computing times of the four MADRL algorithms and four DRL algorithms under different production configurations. The maximum computing time for an instance with 200 jobs is less than 0.30 s. For a single job, the average simulation and decision time is less than 1.5 ms, facilitating real-time scheduling.

Fig. 21 shows that as the number of jobs and machines increases, the computing time also rises. This may be attributed to the sequential

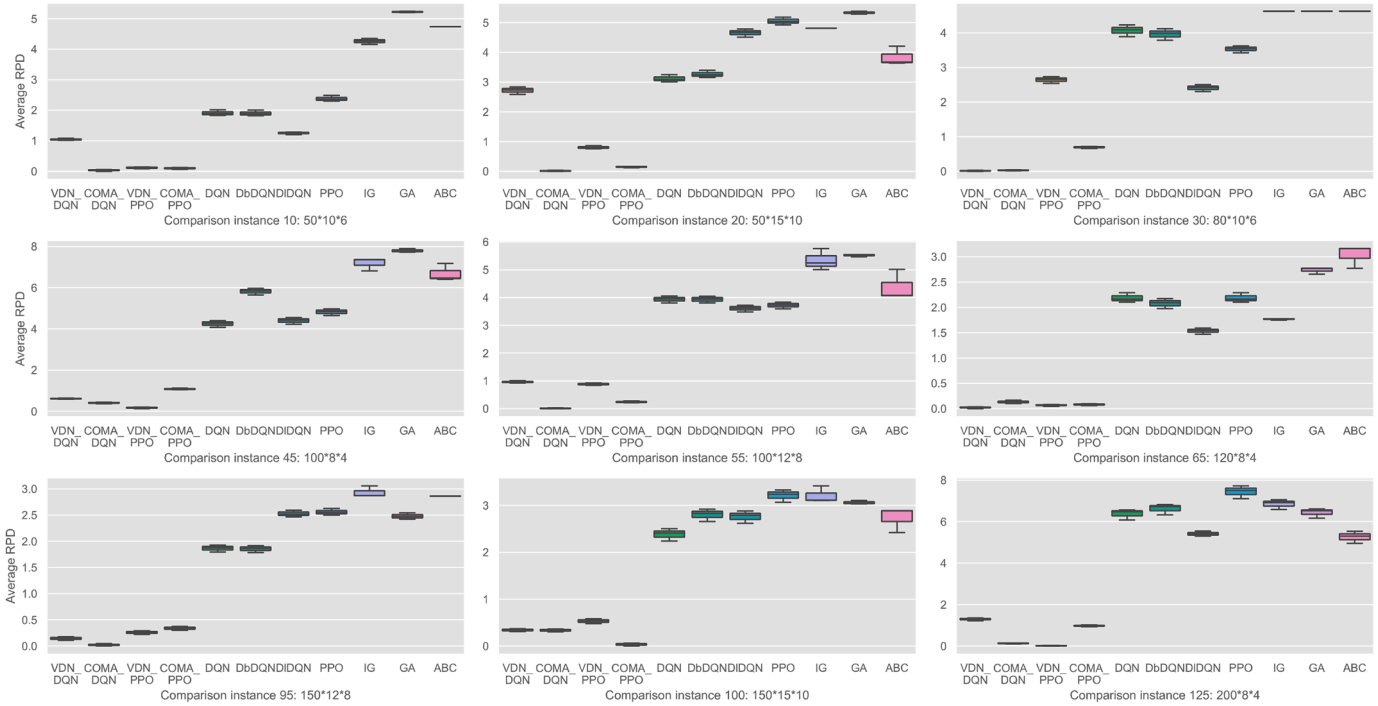


Fig. 19. Average RPDs of the compared algorithms on nine selected instances.

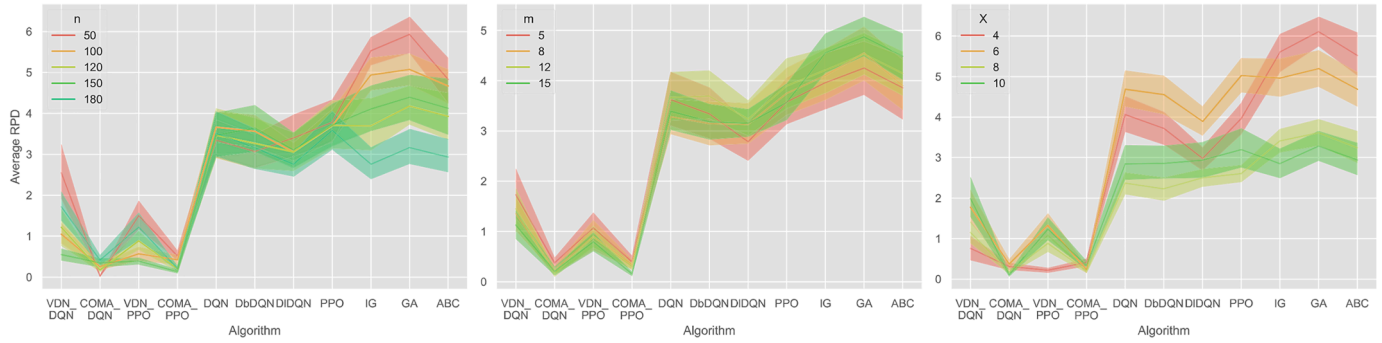


Fig. 20. Interaction plots between algorithms and production configurations on the test instances.

Table 7
The specific average RPDs of all algorithms on the test instances. The best results are bolded.

		MADRL				DRL				Meta-heuristics		
		VDN_DQN	COMA_DQN	VDN_PPO	COMA_PPO	DQN	DbDQN	DlDQN	PPO	IG	GA	ABC
n	50	2.39	**0.03**	1.44	0.48	3.25	3.07	3.38	3.72	5.51	5.99	4.81
	80	0.86	**0.09**	1.67	0.33	3.11	3.15	3.13	3.53	4.6	4.76	4.64
	100	0.99	**0.25**	0.59	0.40	3.66	3.53	2.96	3.66	4.84	4.92	4.65
	120	1.25	**0.17**	0.92	0.22	3.37	3.20	3.02	3.61	3.63	4.03	3.71
	150	0.48	0.32	0.35	**0.13**	3.79	4.03	3.39	3.91	3.93	4.19	4.02
	180	1.60	0.38	1.42	**0.22**	3.27	3.02	2.61	3.41	2.83	3.18	2.97
	200	1.53	**0.18**	1.29	0.39	3.54	3.64	3.10	4.18	3.79	3.79	3.58
m	5	1.51	**0.36**	1.31	0.42	3.58	3.38	2.82	3.67	3.95	4.07	3.76
	8	1.27	**0.19**	1.02	0.38	3.26	3.23	3.16	3.82	3.94	4.28	3.71
	10	1.08	**0.14**	1.15	0.25	3.31	3.41	3.05	3.52	3.94	4.15	3.89
	12	1.41	**0.15**	1.04	0.32	3.63	3.64	3.24	3.97	4.36	4.71	4.35
	15	1.22	**0.18**	0.96	**0.18**	3.36	3.22	3.15	3.6	4.61	4.83	4.56
X	4	0.65	0.24	**0.19**	0.39	3.72	3.54	2.83	3.77	5.84	6.15	5.61
	6	1.61	**0.27**	1.86	0.33	4.87	4.81	3.94	5.17	4.75	4.94	4.64
	8	1.02	**0.21**	0.97	**0.21**	2.63	2.66	2.97	3.07	3.27	3.45	3.16
	10	1.92	**0.08**	1.37	0.31	2.49	2.49	2.60	2.86	2.78	3.10	2.81
Ave		1.30	**0.20**	1.10	0.31	3.43	3.38	3.08	3.72	4.16	4.41	4.05

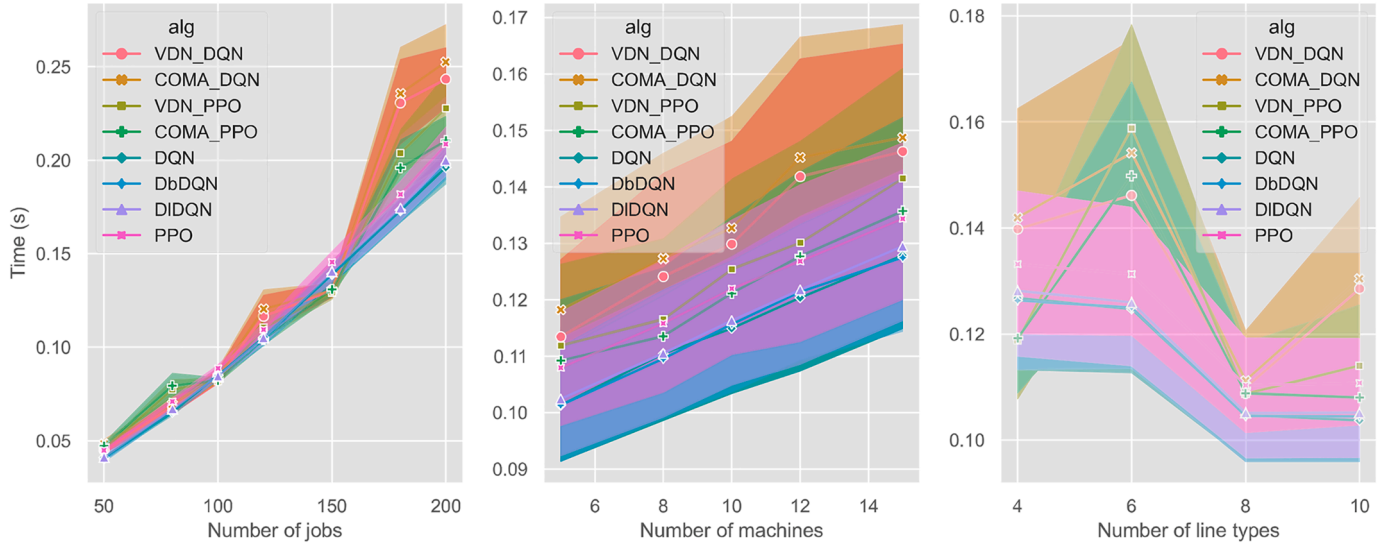


Fig. 21. The average computing time of the RL-based algorithms under different production configurations.

processing of each job on every machine, resulting in higher simulation time demands with more jobs or machines. Conversely, when the production lines increase, there is a noticeable decrease in computing time for most algorithms depicted in Fig. 21. This phenomenon can be attributed to the reduced number of jobs per production line type when more production line types are available. Consequently, as the scheduling state encompasses features of jobs associated with the current production line type, the computation time required for calculating scheduling states decreases with fewer jobs.

8. Conclusions

This study investigates the dynamic DRFSP with new job arrivals through the integration of MARL and DRL methodologies. The objective is to facilitate real-time decision-making for scheduling and reconfiguration in distributed reconfigurable workshops. To address the challenges associated with the simultaneous optimization of scheduling and reconfiguration, as well as coordination among reconfigurable workshops, we formulate the distributed reconfiguration and scheduling problem as MARL-based and DRL-based Markov decision processes, respectively. A reconfiguration mechanism is devised to appropriately coordinate the scheduling and reconfiguration processes. Additionally, a training paradigm involving centralized training and decentralized execution is introduced to train and coordinate the reconfiguration agents across distributed workshops. Experimental findings from training and test experiments showed the superiority of MADRL algorithms over popular DRL algorithms and efficient meta-heuristic algorithms across a wide range of test instances. This underscores the efficacy of amalgamating MARL and DRL methodologies in addressing the challenges posed by distributed and reconfigurable scheduling.

A limitation of this work is that it currently can cope with a small number of jobs, such as 60-200 jobs. For large-scale problems with over 5000 jobs, the proposed MADRL algorithms may not be able to scale. We will further investigate scaling and more efficient techniques, such as genetic programming [24], in the future to cope with large problems. Moreover, future work could focus on developing more efficient decision models, including enhanced reward functions, state representations, and action spaces. Additionally, further investigation into other coordination approaches, such as agent communication and hierarchical coordination mechanisms, will be valuable.

Funding

This work was supported by the National Natural Science Foundation of China (grant number 52405559), the National Key Research and Development Program of China (grant number 2022YFB3306000), and the Shanghai Technology Innovation Project (grant number 24YF2730400).

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work, the author(s) used ChatGPT to improve the language. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the published article.

CRediT authorship contribution statement

Shengluo Yang: Writing – review & editing, Writing – original draft, Methodology, Investigation, Conceptualization. **Zhigang Xu:** Writing – review & editing, Project administration, Funding acquisition. **Fangfang Zhang:** Writing – review & editing. **Yi Mei:** Writing – review & editing. **Quanke Pan:** Writing – review & editing. **Mengjie Zhang:** Writing – review & editing, Supervision.

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.

References

[1] C.E. Okwudire, H.V. Madhyastha, Distributed manufacturing for and by the masses, Science 372 (2021) 341–342.
[2] R. Ruiz, Q.K. Pan, B. Naderi, Iterated greedy methods for the distributed permutation flowshop scheduling problem, Omega-Int. J. Manage. Sci. 83 (2019) 213–222.
[3] X. Yan, H. Zuo, C. Hu, W. Gong, L. Gao, Distributed heterogeneous flow shop scheduling method for dual-carbon goals, IEEE Trans. Autom. Sci. Eng. 22 (2025) 7409–7420.
[4] X. He, Q.-K. Pan, L. Gao, J.S. Neufeld, J.N.D. Gupta, Historical information based iterated greedy algorithm for distributed flowshop group scheduling problem with sequence-dependent setup times, Omega 123 (2024) 102997.
[5] X. Chen, Y. Li, K. Wang, L. Wang, J. Liu, J. Wang, X.V. Wang, Reinforcement learning for distributed hybrid flowshop scheduling problem with variable task splitting towards mass personalized manufacturing, J. Manuf. Syst. 76 (2024) 188–206.
[6] X.-r. Tao, Q.-k. Pan, L. Gao, An iterated greedy algorithm with reinforcement learning for distributed hybrid flowshop problems with job merging, IEEE Trans. Evol. Comput. (2025) 1. -1.
[7] R. Li, L. Wang, W. Gong, J. Chen, Z. Pan, Y. Wu, Y. Yu, Evolutionary computation and reinforcement learning integrated algorithm for distributed heterogeneous flowshop scheduling, Eng. Appl. Artif. Intell. 135 (2024) 108775.
[8] M. Mahmoodjanloo, R. Tavakkoli-Moghaddam, A. Baboli, A. Bozorgi-Amiri, Distributed job-shop rescheduling problem considering reconfigurability of machines: a self-adaptive hybrid equilibrium optimiser, Int. J. Prod. Res. 60 (2021) 4973–4994.
[9] Y. Lei, Q. Deng, M. Liao, S. Gao, Deep reinforcement learning for dynamic distributed job shop scheduling problem with transfers, Expert Syst. Appl. 251 (2024) 123970.
[10] Y. Fu, K. Gao, L. Wang, M. Huang, Y.-C. Liang, H. Dong, Scheduling stochastic distributed flexible job shops using an multi-objective evolutionary algorithm with simulation evaluation, Int. J. Prod. Res. 63 (2024) 86–103.
[11] S. Hatami, R. Ruiz, C. Andrés-Romano, Heuristics and metaheuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times, Int. J. Prod. Econ. 169 (2015) 76–88.
[12] W.-Z. Wu, H.-Y. Sang, Q.K. Pan, Q.-Y. Han, H.-W. Guo, A cooperative discrete artificial bee colony algorithm with Q-learning for solving the distributed permutation flowshop group scheduling problem with preventive maintenance, Swarm Evol. Comput. 95 (2025) 101910.
[13] Z. Zhang, Y. Fu, K. Gao, Q. Pan, M. Huang, A learning-driven multi-objective cooperative artificial bee colony algorithm for distributed flexible job shop scheduling problems with preventive maintenance and transportation operations, Comput. Ind. Eng. 196 (2024) 110484.
[14] Z. Pan, L. Wang, J. Wang, Q. Zhang, A Bi-learning evolutionary algorithm for transportation-constrained and distributed energy-efficient flexible scheduling, IEEE Trans. Evol. Comput. 29 (2025) 232–246.
[15] R. Li, L. Wang, H. Sang, L. Yao, L. Pan, LLM-assisted automatic memetic algorithm for lot-streaming hybrid job shop scheduling with variable sublots, IEEE Trans. Evol. Comput. (2025) 1–13.
[16] Z. Liu, H. Sang, B. Zhang, L. Meng, T. Meng, A learning-based two-stage multi-thread iterated greedy algorithm for co-scheduling of distributed factories and automated guided vehicles with sequence-dependent setup times, IEEE Trans. Emerg. Top. Comput. Intell. (2025) 1–11.
[17] L.B. Wang, X. Hu, Y. Wang, S.J. Xu, S.J. Ma, K.X. Yang, Z.J. Liu, W.D. Wang, Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning, Comput. Netw. 190 (2021) 107969–107978.
[18] S. Luo, Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning, Appl. Soft Comput. 91 (2020) 106208.
[19] W. Gu, L. Duan, S. Liu, Z. Guo, A real-time adaptive dynamic scheduling method for manufacturing workshops based on digital twin, Flex. Serv. Manufac. J. (2024).
[20] Y. Zhang, H.H. Zhu, D.B. Tang, T. Zhou, Y. Gui, Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems, Rob. Comput. Integr. Manuf. 78 (2022) 102412.
[21] S.L. Yang, Z.G. Xu, Intelligent scheduling and reconfiguration via deep reinforcement learning in smart manufacturing, Int. J. Prod. Res. 60 (2022) 4936–4953.
[22] Y. Fu, Y. Wang, K. Gao, M. Huang, Review on ensemble meta-heuristics and reinforcement learning for manufacturing scheduling problems, Comput. Electr. Eng. 120 (2024) 109780.
[23] M. Xu, Y. Mei, F. Zhang, M. Zhang, Learn to optimise for job shop scheduling: a survey with comparison between genetic programming and reinforcement learning, Artif. Intell. Rev. 58 (2025).
[24] M. Xu, Y. Mei, F. Zhang, M. Zhang, Genetic programming and reinforcement learning on learning heuristics for dynamic scheduling: a preliminary comparison, IEEE Comput. Intell. Mag. 19 (2024) 18–33.
[25] Y. Liu, J. Fan, L. Zhao, W. Shen, C. Zhang, Integration of deep reinforcement learning and multi-agent system for dynamic scheduling of re-entrant hybrid flow shop considering worker fatigue and skill levels, Rob. Comput. Integr. Manuf. 84 (2023) 102605.
[26] Z. Wang, B. Cai, J. Li, D. Yang, Y. Zhao, H. Xie, Solving non-permutation flow-shop scheduling problem via a novel deep reinforcement learning approach, Comput. Oper. Res. 151 (2023) 106095.
[27] C.-L. Liu, T.-H. Huang, Dynamic job-shop scheduling problems using graph neural network and deep reinforcement learning, IEEE Trans. Syst. Man Cybern.: Syst. 53 (2023) 6836–6848.

[28] K. Lei, P. Guo, Y. Wang, J. Zhang, X.Y. Meng, L.M. Qian, Large-scale dynamic scheduling for flexible job-shop with random arrivals of new jobs by hierarchical reinforcement learning, IEEE Trans. Ind. Inform. 20 (2024) 1007–1018.
[29] Z. Liu, H. Mao, G. Sa, H. Liu, J. Tan, Dynamic job-shop scheduling using graph reinforcement learning with auxiliary strategy, J. Manuf. Syst. 73 (2024) 1–18.
[30] Y. Li, W. Gu, M. Yuan, Y. Tang, Real-time data-driven dynamic scheduling for flexible job shop with insufficient transportation resources using hybrid deep Q network, Rob. Comput. Integr. Manuf. 74 (2022) 102283.
[31] J. Qiu, J. Liu, Z. Li, X. Lai, A multi-level action coupling reinforcement learning approach for online two-stage flexible assembly flow shop scheduling, J. Manuf. Syst. 76 (2024) 351–370.
[32] J.P. Huang, L. Gao, X.Y. Li, C.J. Zhang, A novel priority dispatch rule generation method based on graph neural network and reinforcement learning for distributed job-shop scheduling, J. Manuf. Syst. 69 (2023) 119–134.
[33] S. Yang, J. Wang, Z. Xu, Real-time scheduling for distributed permutation flowshops with dynamic job arrivals using deep reinforcement learning, Adv. Eng. Inf. 54 (2022) 101776.
[34] J.-P. Huang, L. Gao, X.-Y. Li, A hierarchical multi-action deep reinforcement learning method for dynamic distributed job-shop scheduling problem with job arrivals, IEEE Trans. Autom. Sci. Eng. 22 (2025) 2501–2513.
[35] Y. Li, X. Li, L. Gao, Z. Lu, Multi-agent deep reinforcement learning for dynamic reconfigurable shop scheduling considering batch processing and worker cooperation, Rob. Comput. Integr. Manuf. 91 (2025) 102834.
[36] S. Yang, J. Wang, L. Xin, Z. Xu, Real-time and concurrent optimization of scheduling and reconfiguration for dynamic reconfigurable flow shop using deep reinforcement learning, CIRP J. Manuf. Sci. Technol. 40 (2023) 243–252.
[37] W. Gu, S. Liu, Z. Guo, M. Yuan, F. Pei, Dynamic scheduling mechanism for intelligent workshop with deep reinforcement learning method based on multi-agent system architecture, Comput. Ind. Eng. 191 (2024) 110155.
[38] X.H. Wang, L. Zhang, T.Y. Lin, C. Zhao, K.Y. Wang, Z. Chen, Solving job scheduling problems in a resource preemption environment with multi-agent reinforcement learning, Rob. Comput. Integr. Manuf. 77 (2022) 102324.
[39] N.M. Zhang, Y.L. Shen, Y. Du, L.L. Chen, X. Zhang, Counterfactual-attention multi-agent reinforcement learning for joint condition-based maintenance and production scheduling, J. Manuf. Syst. 71 (2023) 70–81.
[40] Duan, W., Lu, J., & Xuan, J. 2024. Group-aware coordination graph for multi-agent reinforcement learning. *arXiv preprint arXiv:2404.10976*.
[41] S.V. Albrecht, F. Christianos, L. Schäfer, Multi-Agent Reinforcement Learning: Foundations and Modern Approaches, MIT Press, 2024.
[42] Amato, C. 2024. An introduction to centralized training for decentralized execution in cooperative multi-agent reinforcement learning. *arXiv preprint arXiv: 2409.03052*.
[43] A. Kopic, E. Perenda, H. Gacanin, A collaborative multi-agent deep reinforcement learning-based wireless power allocation with centralized training and decentralized execution, IEEE Trans. Commun. 72 (2024) 7006–7016.
[44] J. Wang, Z. Ren, T. Liu, Y. Yu, C. Zhang, QPLEX: duplex dueling multi-agent Q-learning, in: International Conference on Learning Representations, Publishing, 2021.
[45] J. Zhang, Z. He, W. Chan, C. Chow, DeepMAG: deep reinforcement learning with multi-agent graphs for flexible job shop scheduling, Knowl. Based Syst. 259 (2023) 110083.
[46] S.H. Oh, Y.I. Cho, J.H. Woo, Distributional reinforcement learning with the independent learners for flexible job shop scheduling problem with high variability, J. Comput. Des. Eng. 9 (2022) 1157–1174.
[47] R. Chen, W. Li, H. Yang, A deep reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for the job-shop scheduling problem, IEEE Trans. Ind. Inform. 19 (2023) 1322–1331.
[48] M. Xu, Y. Mei, F. Zhang, M. Zhang, Niching genetic programming to learn actions for deep reinforcement learning in dynamic flexible scheduling, IEEE Trans. Evol. Comput. (2024) 1. -1.
[49] L.K. Li, X.J. Fu, H.L. Zhen, M.X. Yuan, J. Wang, J.W. Lu, X.L. Tong, J. Zeng, D. Schnieders, Bilevel learning for large-scale flexible flow shop scheduling, Comput. Ind. Eng. 168 (2022) 108140.
[50] J.F. Ren, C.M. Ye, Y. Li, A new solution to distributed permutation flow shop scheduling problem based on NASH Q-Learning, Adv. Prod. Eng. Manage. 16 (2021) 269–284.
[51] S. Yang, J. Wang, Z. Xu, Learning to schedule dynamic distributed reconfigurable workshops using expected deep Q-network, Adv. Eng. Inf. 59 (2024) 102307.
[52] Y. Li, X. Li, L. Gao, Real-time scheduling for production-logistics collaborative environment using multi-agent deep reinforcement learning, Adv. Eng. Inf. 65 (2025) 103216.
[53] Y. Gui, Z. Zhang, D. Tang, H. Zhu, Y. Zhang, Collaborative dynamic scheduling in a self-organizing manufacturing system using multi-agent reinforcement learning, Adv. Eng. Inf. 62 (2024) 102646.
[54] H. Kazemi, M.M. Mazdeh, M. Rostami, The two stage assembly flow-shop scheduling problem with batching and delivery, Eng. Appl. Artif. Intell. 63 (2017) 98–107.
[55] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (2015) 529–533.
[56] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
[57] E. Yuan, S. Cheng, L. Wang, S. Song, F. Wu, Solving job shop scheduling problems via deep reinforcement learning, Appl. Soft Comput. 143 (2023) 110436.
[58] P. Sunehag, G. Lever, A. Gruslys, W.M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J.Z. Leibo, K. Tuyls, T. Graepel, Acm, Value-decomposition networks for cooperative multi-agent learning based on team reward, in: 17th International Conference on Autonomous Agents and Multi Agent Systems, Stockholm, SWEDEN, (AAMAS). Publishing, 2018, pp. 2085–2087.
[59] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, S. Whiteson, Counterfactual multi-agent policy gradients, in: Proceedings of the AAAI Conference on Artificial Intelligence, Publishing, 2018.
[60] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.
[61] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, Publishing, 2016.
[62] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, N. Freitas, Dueling network architectures for deep reinforcement learning, in: Int. Conf. Mach. Learn., ICML. Publishing, 2016, pp. 1995–2003.
[63] Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., & Eslami, S. 2017. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*.
[64] J. Chen, H. Zhang, W. Ma, G. Xu, Real-time scheduling for two-stage assembly flowshop with dynamic job arrivals by deep reinforcement learning, Adv. Eng. Inf. 62 (2024) 102632.
[65] X. Wang, Y. Laili, L. Zhang, Y. Liu, Hybrid task scheduling in cloud manufacturing with sparse-reward deep reinforcement learning, IEEE Trans. Autom. Sci. Eng. 22 (2025) 1878–1892.
[66] B. Zhang, C. Lu, L.L. Meng, Y.Y. Han, H.Y. Sang, X.C. Jiang, Reconfigurable distributed flowshop group scheduling with a nested variable neighborhood descent algorithm, Expert Syst. Appl. 217 (2023) 119548.
[67] I.B. Park, J. Huh, J. Kim, J. Park, A reinforcement learning approach to robust scheduling of semiconductor manufacturing facilities, IEEE Trans. Autom. Sci. Eng. 17 (2020) 1420–1431.
[68] M. Wang, J. Zhang, P. Zhang, L. Cui, G. Zhang, Independent double DQN-based multi-agent reinforcement learning approach for online two-stage hybrid flow shop scheduling with batch machines, J. Manuf. Syst. 65 (2022) 694–708.