Computers & Operations Research 159 (2023) 106294

Contents lists available at ScienceDirect

Computers and Operations Research

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

A deep multi-agent reinforcement learning approach to solve dynamic job shop scheduling problem

Renke Liu [a,b,*,1], Rajesh Piplani [a], Carlos Toro [c]

[a] School of Mechanical and Aerospace Engineering, Nanyang Technological University, Singapore
[b] AIR Center, Hyundai Motor Group Innovation Center in Singapore
[c] Vicomtech Research Centre, San Sebastian, Spain

ARTICLE INFO

Keywords:
Job shop scheduling
Dynamic scheduling
Deep reinforcement learning
Multi-agent reinforcement learning

ABSTRACT

Manufacturing industry is experiencing a revolution in the creation and utilization of data, the abundance of industrial data creates a need for data-driven techniques to implement real-time production scheduling. However, existing dynamic scheduling techniques have been mainly developed to solve problems of invariable size, and are incapable of addressing the increasing volatility and complexity of practical production scheduling problems. To facilitate near real-time decision-making on the shop floor, we propose a deep multi-agent reinforcement learning-based approach to solve the dynamic job shop scheduling problem. Double deep Q-network algorithm, attached to decentralized scheduling agents, is used to learn the relationships between production information and scheduling objectives, and to make near real-time scheduling decisions. Proposed framework utilizes centralized training and decentralized execution scheme and parameter-sharing technique to tackle the non-stationary problem in the multi-agent reinforcement learning task. Several enhancements are also developed, including the novel state and action representation that can handle size-agnostic dynamic scheduling problems, a chronological joint-action framework to alleviate the credit-assignment difficulty, and knowledge-based reward-shaping techniques to encourage cooperation. Simulation study shows that the proposed architecture significantly improves the learning effectiveness, and delivers superior performance compared to existing scheduling strategies and state-of-the-art deep reinforcement learning-based dynamic scheduling approaches.

1. Introduction

Production scheduling has been extensively studied over decades and numerous techniques are developed to address the problem under different scenarios. The advancements in manufacturing keep updating the context of scheduling, bringing up new and unsolved problems; at the same time, research in optimization theory, computer science, and artificial intelligence keep coming up with novel approaches to solve the said problem. The constantly changing problems and methodologies make scheduling a vibrant domain to study under the umbrella of operation research.

However, some common simplifications in current scheduling research, such as the assumption of static environment and invariable problem specification, compromise the performance of the developed schedule in actual practice. In dynamic environments, a scheduling schedule/strategy developed for a specific problem instance may become infeasible due to changes in context and prerequisites. Therefore, the study of scheduling in the presence of real-time disruptions, termed dynamic scheduling, attracts increasing attention given the ever-increasing demand for agility, flexibility, and timeliness of production.

The revolutionary cyber-physical convergence of infrastructure as applied to the manufacturing industry inspires a novel field of scheduling research. The accessibility of real-time industrial data is greatly enhanced by the deployment of devices that constantly collect and exchange information on the shop floor (Messaoud et al., 2020). In addition, the introduction of high-performance computing systems such as edge computing and industrial cloud to manufacturing systems enables decentralized real-time data analytics (Chen & Ran, 2019). The abundance of both “raw material” and “tool” fulfills the prerequisite of deploying data-driven technologies to production planning and control.

Reinforcement learning (RL) algorithm (Sutton & Barto, 2018), which maps the observation of environment to action policy via reward-

* Corresponding author.
E-mail addresses: renke001@e.ntu.edu.sg (R. Liu), MRPiplani@ntu.edu.sg (R. Piplani), ctoro@vicomtech.org (C. Toro).
[1] ORCID: 0000-0003-4375-9140.

https://doi.org/10.1016/j.cor.2023.106294
Received 21 December 2021; Received in revised form 9 May 2023; Accepted 29 May 2023
Available online 2 June 2023

driven learning, is a demonstrated tool to tackle real-time control tasks. The incorporation of deep learning further enhances RL's ability to manage the complex state and action space, making it an ideal tool to interpret complex industrial data, therefore RL-based approaches are becoming popular in dynamic scheduling research.

Unfortunately, traditional RL algorithms are long-troubled by their poor scalability; the training becomes chaotic and inefficient when multiple agents are exploring the environment together. This challenge is even harder in scheduling problems as the consequence of action (reward) is typically sparse and delayed, making the credit assignment extremely hard to develop a cooperative policy for all agents. But the need for decentralized scheduling algorithms is crucial: system-level information may under-represent the complex shop floor environment; while a system-wide decision-maker can fail to address individual agents' needs. Algorithmic innovation is needed to fully utilize the power of decentralized data infrastructure, and to reach the full potential of RL.

In this research, we represent the job shop as a multi-agent system wherein each machine/agent is authorized to determine its order of operations. Furthermore, we propose a new deep multi-agent reinforcement learning (deep MARL) framework to schedule production in a system with random job arrivals that constantly change the problem specification, with the objective of minimizing cumulative tardiness.

The contributions of this research are: (1) an integrated deep MARL algorithm to perform dynamic job shop scheduling; (2) specialized state and action representations that can tackle size-agnostic scheduling tasks, which are scalable in dynamic environments; (3) chronological joint-action framework to alleviate the credit-assignment difficulty in scheduling problem, and (4) knowledge-based reward-shaping techniques to encourage cooperation among agents thus improving the system-level performance. Simulation studies show that under various scenarios, proposed approach provides competitive and consistent performance compared to contemporary approaches based on deep reinforcement learning (DRL) and existing real-time scheduling strategies.

The remainder of the paper is organized as follows. Section 2 reviews the latest dynamic scheduling research and the application of RL to both static and dynamic scheduling. Section 3 analyzes and formulates the research problem. Section 4 presents an approach for solving the dynamic job shop scheduling problem (DJSP); Section 5 then summarizes the results of numerical experiments. Finally, Section 6 presents a discussion and future directions for research.

2. Related works

2.1. Priority rules and metaheuristics for dynamic scheduling

Many attempts have been made to transfer static scheduling approaches to the dynamic environment. However, unlike static scheduling problems where a fixed number of jobs with known associated features, such as processing time and due date, are ready to be processed at the beginning of production, dynamic scheduling considers unpredictable events that render the pre-developed schedule almost immediately ineffective. A mechanism to absorb or adjust the schedule in response to these disruptions is necessary to maintain the high performance desired. Dynamic scheduling techniques derived from static scheduling research can be divided into three categories (Ouelhadj & Petrovic, 2008):

(1) Completely reactive: Scheduling can be implemented in real-time without a predetermined production plan; subsequently, the boundary between static or dynamic environments is eliminated. Priority rules are popular due to their ease of implementation. Extensive studies have been done, and it is generally agreed that no priority rule exhibits a decisive advantage compared to others (Sels et al., 2012).

(2) Robust-proactive: Popular metaheuristics techniques applied in static scheduling such as simulated annealing, tabu search, genetic algorithm, particle swarm optimization, or ant colony optimization are also widely applied to dynamic scheduling (Dehghan-Sanej et al., 2021; Li & Gao, 2020; Liu et al., 2017; Wang et al., 2019; Zhang et al., 2020). The solution development starts from a randomized schedule or population of schedules; the performance of candidates is evaluated in the dynamic environment, and those with better performance are reproduced and mutated in iterative evolution. Rescheduling process is usually not developed or optimized as the influence of dynamic events is considered "absorbed" by schedule. The schedules are disposable, making techniques under this category vulnerable to frequent changes in the problem specification.

(3) Predictive-reactive approaches: Techniques under the previous two categories cannot produce schedules with both speed and quality, but scheduling can be done in a hybrid manner to take advantage of the strength of both approaches. At the start, static scheduling techniques (usually metaheuristics) are applied to produce a centralized schedule. The schedule is then executed until a disruption occurs, at which point the schedule is repaired or replaced by a decentralized scheduling policy (Al-Behadili et al., 2017; Berger et al., 2019).

Lack of comprehensive visibility of system status and under-parametrized models are common shortcomings of traditional approaches. Priority rules usually take very few data features as input and make the decision based on simple computations. On the other hand, metaheuristic-based approaches do not use shop floor information as input at all; the development of a production schedule is objective-driven, randomized, and time-consuming evolution. It is questionable whether such techniques are suitable for a data-intensive and fast-changing manufacturing system.

2.2. Evolutionary algorithm and learning-based approaches

The ongoing industrial revolution brings new opportunities and challenges for scheduling research. The increasing volume and diversity of real-time industrial data provides a solid foundation for real-time data analytics, while at the same time increasing the difficulty of selection, preprocessing, and interpretation of such data. Some studies (Mei et al., 2017; Mirshekarian & Šormaz, 2016) have looked at the correlation between the data features and performance of the scheduling strategy.

Recent studies also aim to utilize a broader category of industrial data to generate high-quality schedules with negligible decisional latency by developing the scheduling strategy offline (as that process is time-consuming), and applying it online. Popular candidates other than RL include genetic programming (GP) and supervised learning.

Unlike other evolutionary approaches that produce a complete schedule to solve a static problem instance, GP develops a complex tree-represented priority rule through automatic and population-based evolution. It incorporates necessary shop floor information into the decision-making process, and enables low-latency response to dynamic events. Studies into GP-based scheduling cover a range of research topics such as representations of scheduling (Branke et al., 2015; Nguyen et al., 2013); interpretability of derived rules (Hildebrandt et al., 2010); feature selection (Shady et al., 2022), and comparative study of different architectures (Zhou et al., 2020; Zhou et al., 2019). However, the decision is produced by operating shop floor information through elementary mathematical and logic operations. The under-parametrization leads to lack of ability to interpret complex system dynamics.

Compared to evolutionary algorithms, supervised learning approaches assume that the knowledge can be learned from the experiences of sound scheduling decisions; and be represented by highly-parametrized algorithms through analytical or gradient-based

optimization. Many well-known classification models have been transferred to solve scheduling problems, such as decision trees to learn strategies from optimal schedules (Jun et al., 2019; Olafsson & Li, 2010); and support vector machine (SVM) to select the priority rule (Priore et al., 2010; Shiue, 2009). The most popular technique is the artificial neural network (ANN), which can be used for selecting a priority rule (Mouelhi-Chibani & Pierreval, 2010) or determining an optimal sequence of jobs (Gupta et al., 2019; Zang et al., 2019). However, the static architecture of neural networks does not meet the requirement for a flexible input/ output space that is conditioned on the constant-changing specification of a dynamic scheduling problem. If dynamic job arrivals are present, scheduling can only be done in a replacement-only way: bring one job into consideration to fill the vacancy resulting from the completion of a job (which exits the system). By doing so, jobs arriving on the shop floor later are implicitly assigned lower priority as they are held unattended until all jobs in front of the queue are cleared. In actual practice, jobs may arrive with different urgency or importance (weight), and preemption is sometimes necessary.

2.3. Reinforcement learning in scheduling research

The RL-based scheduling (in both static and dynamic environments) can be categorized by the following criteria:

(1) Algorithm: Except classic value-based algorithms such as Q-learning (Yan et al., 2022). DRL that utilizes ANN as state or policy representation is getting more popular for its effectiveness in handling high-dimensional, even continuous state/action space. Famous algorithms include DQN (Mnih et al., 2015), double DQN (Van Hasselt et al., 2016), and A3C (Mnih et al., 2016).
(2) Scheduling architecture: Scheduling can be performed in a centralized manner that all decision-making entities, such as machines or products, follow an identical strategy determined by a single RL agent. While in a decentralized scheduling system, RL agents attached to each decision-making entity schedule only the imminent operation for the entity, resulting in a MARL problem and additional complexity in training.
(3) State and action representation: State space includes the features associated with decision-making; their mapping to action space is learned via reward-driven training. State representations are dependent on the scheduling architecture and objective of scheduling, as the information needed for scheduling from a system (centralized) or individual (decentralized) perspective, or for achieving various goals can be rather different. Similarly, action representation can also be different depending on the research context and objective.

Most DRL-based studies use static and fixed-size job shop scheduling problem (JSP) instances, such as OR-library (Beasley, 1990) and Taillard's instances (Taillard, 1993). The design of representation and reward in these works is still of referential value to the study of dynamic scheduling, because the information associated with scheduling decisions is universally applicable. Examples include the use of a single DQN agent to select priority rules for all machines in a job shop (Lin et al., 2019); and deep actor-critic approach to select priority rules to solve JSP (Liu et al., 2020). Although some researchers mention the importance of taking dynamic events that change the problem specification into account, the effect of such events, e.g. job arrivals, are usually implemented by extending a small fixed-size problem to a larger one (Luo et al., 2021a).

Abstracted features coupled with indirect action is a popular approach to handle dynamic scheduling problems with variable size: the state space consists of descriptive information of jobs, machines, or system; while the action space consists of a set of priority rules. Models developed in this manner are independent of the problem size thus more practical and adaptable. Examples include the decentralized Q-learning approach for DJSP (Wang, 2020); and two-hierarchy deep DQN approach to solve dynamic flexible JSP (Luo et al., 2021b).

One-to-one mapping from jobs' information to the selection is also feasible. Gabel and Riedmiller (2012) develop a policy-gradient approach that calculates the probability of being selected for all jobs in the system, even for jobs not in the agents' queue, and prohibits each agent from taking infeasible action.

There are also studies focusing on job allocation and transfer within the system, such as the DQN approach to move products in a semiconductor manufacturing system (Waschneck et al., 2018); the "expert-guided" actor-critic approach that initializes replay memory with decisions made by priority rules to allocate jobs to servers (Qu et al., 2019), and DQN approach to manage the job transfer in automated production lines with stochastic processing time (Shi et al., 2020).

Using machines or work centers as the decisional entities is common, but RL agents can also be attached to jobs. Bouazza et al. (2017) authorize products, upon their arrival at work centers and machines, to select routing and sequencing rules by Q-learning algorithm. Baer et al. (2019) represent the flexible JSP as a Petri net, and use DQN to decide the route of each job to various machines.

Most existing DRL-based studies are single-agent approaches, under which a scheduling agent learns the mapping from the abstracted information of production to a system-wide scheduling strategy e.g. assigning a priority rule to all decisional entities in the system. The lack of local visibility results in the absence of capability to produce diversified strategies for each decisional entity to address their, usually different, needs of scheduling.

Researchers are also developing multi-agent scheduling approaches. However, existing studies such as (Liu et al., 2020; Luo et al., 2021a) simply let multiple agents explore the environment independently, without incorporating recent progress in MARL research.

In addition, the various specialized research contexts also jeopardize the comparability of existing studies. In this research, we use a generic job shop as the context for more transferable and comparable research.

3. Problem formulation

The key focus of this paper is the development of deep MARL architecture to manage a job shop facing unpredictable disruptions. Problem size is not constrained to simulate the production on an ongoing basis with a high frequency of dynamic events over a long time horizon.

Stochastic job arrival is considered the "**dynamic**" factor, which continuously changes the specification of the scheduling problem. Each time when a new job arrives at the system, its operations, together with all remaining operations that belong to other work-in-progress jobs in the system, form a new problem. The size of problem thus constantly changes over time, and a flexible scheduling strategy is needed to tackle this variable problem.

Here are two real-world examples to justify our research context: (1) For a company that operates with build-to-order strategy, the customer order with varied attributes (such as customized configuration and delivery date) may arrive at any time. Orders' arrival is unevenly distributed across time thus the decision strategy should consider both peak and non-peak periods; (2) In a factory, the unanticipated interruptions on the assembly line also affect the job scheduling in its dependent systems, such as warehouse or intra-logistics system. For those dependent systems, they must handle orders that come at a variable rate, with different urgency.

In both examples, it might be hard to divide the scheduling task into sub-problems of equal size or time horizon. A more desirable way might be considering all incomplete jobs, both old and new ones, and building a comprehensive production schedule.

Therefore, we build generic scenarios with random job arrival over a long duration in both training and testing stages. The former factor

enables the stochasticity and diversity of jobs; while the latter factor resembles the uneven and variable utilization of system capacity.

In a simulation-based training process, distributed agents attached to each machine are required to determine the order of operations when scheduling decisions are required i.e. when a machine becomes idle and at least two jobs are queuing to be processed, with the goal of minimizing the tardiness of jobs.

The constraints of the problem are: (1) preemption is not allowed; an operation is completed without interruption once started; (2) rework and re-entrance of the job are not modeled; (3) machine setup time and job movement time are ignored; (4) jobs are independent of each other, a machine can serve only one job at a time; (5) the queueing buffers are considered unlimited and not modeled.

3.1. Symbols

J and J_i	J is the set of all jobs,$J = \{J_i : i = 1, \cdots, n\}$
M and M_k	M is the set of all machines,$M = \{M_k : k = 1, \cdots, m\}$
O_i	Number of operations of J_i
Sqc_i	Operation sequence of J_i, $Sqc_i = [M_a, \cdots, M_z]$
$t_{i,k}$	Process time of J_i on machine M_k
t_i^j	Processing of j^{th} operation of J_i
J^k	J^k is the set of jobs that are being or queuing to be processed by M_k, $J^k = \{J_a, \cdots, J_z\}$
CO_k	Time that M_k have spent on the current operation
AM_k	Available time of M_k, $AM_k = \sum_{J_i \in J^k} t_{i,k} - CO_k$
D_i	Due date of J_i
NOW	Current time in the system
TTD_i	Time-till-due of J_i, $TTD_i = D_i - NOW$
WR_i	Work-remaining of J_i, $WR_i = \sum_{j=x}^{O_i} t_i^j$, x is the index of J_i's next operation
S_i	Slack time of J_i, $S_i = TTD_i - WR_i$
C_i	Completion time of J_i
T_i	Tardiness of J_i, $T_i = max(C_i - D_i, 0)$
T_{sum}	Cumulative tardiness, the objective of this research

3.2. Objective and scenarios

Tardiness is lateness in output or delivery; it is a customer satisfaction-oriented measurement that connects the optimization of operation with the customer's and supply chain partners' concerns. In a highly competitive market where companies put much effort into winning customers' loyalty, tardiness-related performance metrics become important in evaluating the quality of production scheduling. Therefore in this research, we use the cumulative tardiness T_{sum} as the objective to be minimized, it is calculated as.

$$T_{sum} = \sum_{i=1}^{n} T_i \tag{1}$$

The development and validation of scheduling strategy are simulation-based; three types of factors need to be specified to create the simulation environment:

(1) **Utilization rate**: this research focuses on scheduling in a moderate to highly utilized system, where a good scheduling strategy is needed most. The utilization rate of the system is used as the anchor variable to create scenarios for training and validation. Let $\mathbb{E}(t)$ and $\mathbb{E}(interval)$ denote the expected processing time of operations and the expected time interval between job arrivals, respectively. In a DJSP in which jobs visit all machines, the expected utilization rate of the job shop is calculated as.

$$E(utilization\ rate) = \frac{\mathbb{E}(t)}{\mathbb{E}(interval)} \times 100\% \tag{2}$$

Let a random variable X that follows exponential distribution denote the time interval between job arrivals: $X \sim \ \mathrm{Exp}(\beta), \beta = \mathbb{E}(interval)$; the value of β is explicitly specified in the random number generator to match an expected utilization rate. The deep MARL agents are trained and validated under **three levels of utilization rate**: 70%, 80%, and 90%.

(2) **Processing time**: drawing processing time from a uniform distribution is common practice in JSP research. Some researchers may use an empirical distribution that is specific to their context, but most researchers use uniform distribution as it can simulate a variety of scenarios, making the study more comparable and transferable. We have studied our approach's performance in a moderate processing time range: $t_{i,k} \sim \ U[1, 50]$.

(3) **Due date tightness**: Jobs may arrive with diversified delivery requirements. A tightness factor α_i is drawn from the uniform distribution to determine the remaining time till the job J_i's due date upon its arrival on the shop floor, by multiplying the tightness factor with the sum of processing times. The due date and initial slack/buffer time are determined accordingly. The initial TTD_i, D_i, and S_i are calculated as

$$TTD_i^{initial} = \alpha_i \times \sum_{j=1}^{O_i} t_i^j, \alpha_i \sim \ U[1, upper\ limit] \tag{3}$$

$$D_i = NOW + TTD_i^{initial} \tag{4}$$

$$S_i^{initial} = (\alpha_i - 1) \times \sum_{j=1}^{O_i} t_i^j \tag{5}$$

Agents will be demoralized to improve their performance in the presence of excessively loose due dates, whereas over-tight due dates may lead to excessive tardiness and confuse the agent by overwhelming it with punishment. The range of due date tightness is set to be: $\alpha_i \sim \ U[1, 3]$, which creates an initial slack time that equals the job's total processing time.

3.3. Analysis and problem specification

To facilitate the design of deep MARL architecture, simulation studies are carried out to observe the pattern of production in dynamic environments. The data are collected from 100 runs of simulation (each lasting for 2,000 units of time) under three scenarios; all machines in the system follow the first in first out (FIFO) rule. Three types of information are shown in Fig. 1: (1) row 1: the number of jobs in queue when sequencing decision is required, presented as the ratio that accounts for the total number of decisions. Decisions that affect more than one job are referred to as "active" decisions, else "passive"; (2) row 2: the tardy rate; and (3) row 3: the average tardiness.

Generally, the ratio of active decisions is proportional to the expected utilization rate, and inversely proportional to the number of machines in the system. Meanwhile, the tardy rate and average tardiness increase as the expected utilization rate increases, but their relationship with the number of machines appears to be non-monotonic. One hypothesis is that jobs are distributed less uniformly in a system with more machines, thereby spending more time queueing. On the contrary, a long sequence of operations offsets the unbalanced queuing time at each stage and eventually cancels the tardiness increase resulting from the non-uniform distribution of jobs.

Tardiness may exist even in a moderately-utilized system, which may be attributed more to the inevitable queuing rather than a poor scheduling strategy: jobs may join an empty queue but not be served immediately as the machine is busy processing another job. This would explain the moderately high tardy rate in systems where more than half the times the jobs are processed without any "competition" from other jobs.

A widely accepted notion is that the size of the manufacturing system does not significantly affect the relative performance of priority rules, and a job shop containing six machines and above can be considered complex (Holthaus & Rajendran, 1997). While Jain and Meeran (1999) state that the complexity of JSP is proportional to the ratio of number of

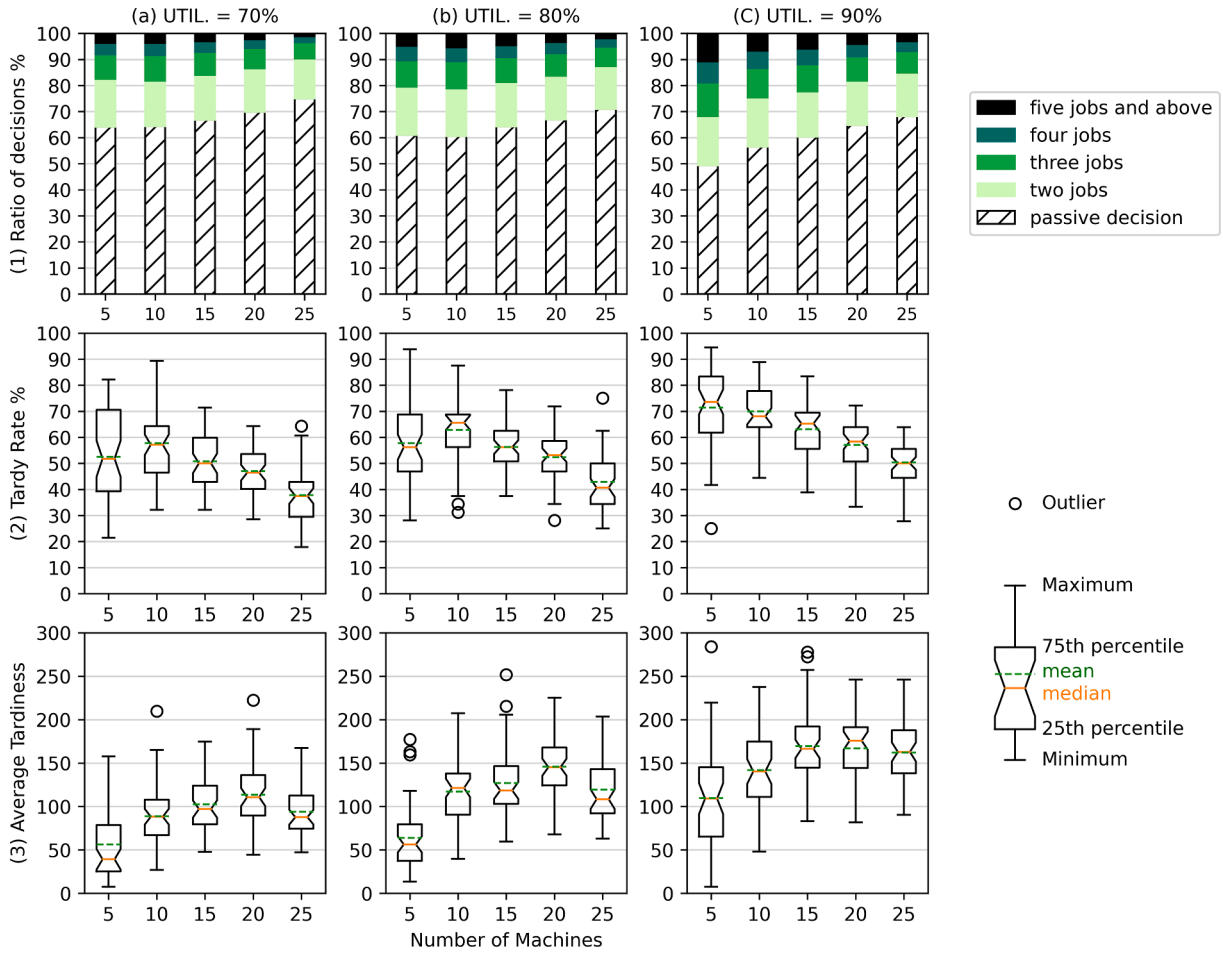


Fig. 1. Patterns of production under three levels of utilization rate.

jobs to machines; a JSP can be considered complex if the ratio is greater than 2.5 and "hard" if $n \geq 15, m \geq 10$ with a total number of operations greater than 200.

In this research, a **job shop consisting of 10 machines** is used for demonstration as it satisfies both thresholds of complexity. A high frequency of active decisions in a system is also enabled, thereby ensuring efficiency of training sample generation and justifying the practicality of problem setting. Meanwhile, the duration of simulation of such a system in the training and validation stages is long enough to create a job set that far exceeds the latter threshold in terms of number and the ratio of jobs to machines.

4. Deep MARL approach to solve DJSP

4.1. DRL and MARL preliminaries

RL tackles the sequential decision-making problem in a "trial and error" manner, which can be modeled as a Markov Decision Process (MDP) with a 5-element tuple representation (S, A, P, γ, R). The agent continuously interacts with the environment by following procedure: at each time step t, the agent observes the current state $s_t \in S$ and takes an action $a_t \in A$ according to a specific policy $\pi(S, A)$; a reward $r_t \in R$ is given after the agent transits to the new state s_{t+1}. The goal of RL agent is to maximize the long-term (usually discounted) cumulative reward G_t.

Modern DRL algorithms strengthen the traditional RL algorithm in the following ways: (1) use of ANN as the representation of the mapping from state to value or policy; (2) experience-replay mechanism that removes the correlation between training samples; and (3) decoupling action network (θ^A) and target network (θ^T) architecture. The former is used for actual control, while the latter is used to generate a stable target value for parameter optimization.

DQN is the very first algorithm that integrates the above enhancements. It parametrizes the value-estimation by ANN, and calculates the target value for updating the network as

$$y_i = r_t + \gamma \max_a Q\left(s_{t+1}, a|\theta^T\right) \tag{6}$$

However, DQN sometimes converges too quickly to a sub-optimal policy (over-optimism) due to estimation error (Van Hasselt et al., 2016). Double DQN uses the action that would have been taken, rather than the maximum state-action value in the next state, to reduce the over-estimation in training. The target value is calculated as

$$y_i = r_t + \gamma Q\left(s_{t+1}, argmax_a Q\left(s_{t+1}, a|\theta_i^A\right)|\theta^T\right) \tag{7}$$

The network is trained by minimizing the value of the loss $L\left(\theta_i^A\right)$ at iteration i.

$$L\left(\theta_i^A\right) = y_i - Q\left(s_t, a_t|\theta_i^A\right) \tag{8}$$

The task in this research is a fully cooperative multi-agent task, which can be described as a Decentralized Partial Observable MDP (Oliehoek & Amato, 2016). The concurrent exploration and evolution of agents lead to a non-stationary environment with additional complexity. The transition is affected not only by agent's own move but also other agents' actions that implicitly change the environment dynamics. Such an effect is hard to explain by the agent's own policy without knowing other agents' action/policy or the evolution of the environment. Common training schemes of MARL include (1) Concurrent scheme wherein all agents learn their own policy while treating other agents as part of

their environment, which is vulnerable to the non-stationary problem; (2) Centralized scheme that learns the mapping from joint state space to joint action space; and (3) Parameter sharing scheme, which lets homogeneous agents share part of or entire encoder and learn from the public experience pool.

Most recent research takes advantage of the mixed scheme; examples include the *centralized training and decentralized execution* approach, usually accompanied by parameter sharing technique to reduce the MARL to ordinary RL in the training phase while keeping the autonomy of agents in the execution phase (Gupta et al., 2017). And *extended actor-critic approach* that uses a centric critic (value-network) with global observability, to increase the efficiency of training actors (policy-networks) with local observability (Lowe et al., 2017). In this research, centralized training and decentralized execution scheme is adopted, and the neural network parameter is shared by all the agents due to the homogeneity of the scheduling task. This scheme is illustrated in Fig. 2.

In addition to the training scheme, reward mechanism also plays an important role in MARL. Early studies usually give all agents a joint reward signal in cooperative tasks so agents may learn to achieve a joint goal. But agents can be easily confused by reward signals that result from other agents' behavior (Sunehag et al., 2017).

A popular approach to address this issue is value decomposition i.e., value function factorization; famous algorithms such as VDN (Sunehag et al., 2017), QMIX (Rashid et al., 2020) and QTRAN (Son et al., 2019) decompose the joint state-action value into individual values for agents, thereby isolating the individual utilities from the joint reward signal. These approaches automate the reward-shaping and avoid the need for domain expertise (OroojlooyJadid & Hajinezhad, 2019). An implicit prerequisite of value decomposition method is the synchronous decision-making environment, where all agents act together at each step, for example, controlling multiple characters at each video game frame.

However, the decision-making in event-driven DSP is asynchronous; agents (machines) are required to make the decision when other machines are busy processing jobs, thus the joint action cannot be represented by a trivial collection of actions taken at identical time step; and disables the aggregation of individual action values. Even though, value decomposition approaches still provide inspiration to the design of deep MARL for DSP. Value of state or action is but an estimation of cumulative reward; value decomposition approaches tend to give each agent either a fraction or a transformed fraction of the joint value, which is akin to decomposing the joint reward into factorized reward. Thus direct decomposition of reward could be an alternative to the value decomposition.

Given these facts and inspired by the reward-difference methods that directly re-construct the joint reward (Tumer et al., 2002), a knowledge-based reward-shaping technique is proposed in this research to facilitate learning and foster cooperative behavior. It modifies the architecture of learning algorithm and decomposes the joint reward, whose details are introduced in section 4.3.

4.2. State and action representations

Neural network, as the core component of deep MARL algorithm, whose static architecture contradicts the variable specification of dynamic scheduling problem: the input and output space cannot be changed to include the job-specific information of an arbitrary number of queuing jobs.

As discussed in section 2.3, a common solution is using abstracted features to build a state space with a stable size. However, the capability of agents can be compromised due to the absence of job-specific information. Abstracted state features are collective descriptions of jobs; the transformation of original data dilutes the extreme value of the attribute of an individual job in a congested queue and loses the one-to-one correspondence between features and jobs.

While others avoid this challenge by demonstrating the algorithm on problem instances with a fixed size, this manner contradicts the objective of this research, whose focus is on scheduling with an arbitrary number of job arrivals over an unconstrained period.

A **minimal-repetition (MR)** approach is developed in this research. The state space contains job-specific information and selected extra information to enhance the agents' visibility, to build a one-to-one mapping from candidate jobs to the selection among them.

As discussed in Section 3.3, the decisions that affect five jobs and above account for only a small part of the active decisions; therefore, a local observation (state space) that covers four jobs (blue box in Fig. 3) is sufficient to include all jobs in most of the cases. On top of that, four well-known sequencing rules are used to select candidate jobs, whose information would be filled into the state space.

These rules focus on different aspects of jobs to ensure that the candidates possess at least one unique feature that makes them more urgent or more suitable than other jobs. The rationales are introduced as follows:

(1) **Shortest processing time (SPT)**: selects the job that can be processed in the shortest time, to minimize the slack time consumption for other queuing jobs.
(2) **Least work remaining (LWKR)**: selects the job that could exit system in the shortest time (estimated), to reduce the overall congestion level.
(3) **Minimal slack (MS)**: selects the most urgent job to protect it from being tardy or incurring too much tardiness.
(4) **Work in queue (WINQ)**: selects the job that exposes its succeeding operation to least congestion, to balance the workload of machines and avoid wasting machine idle time.

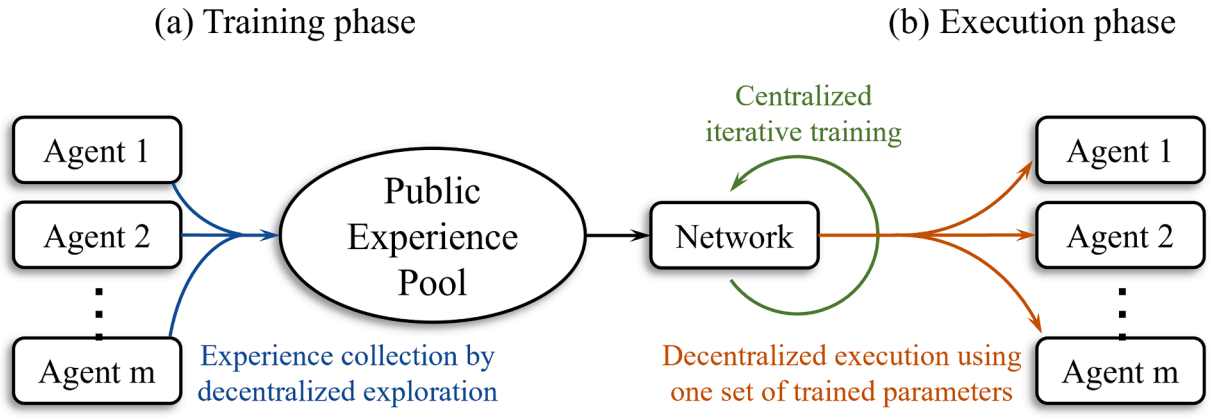
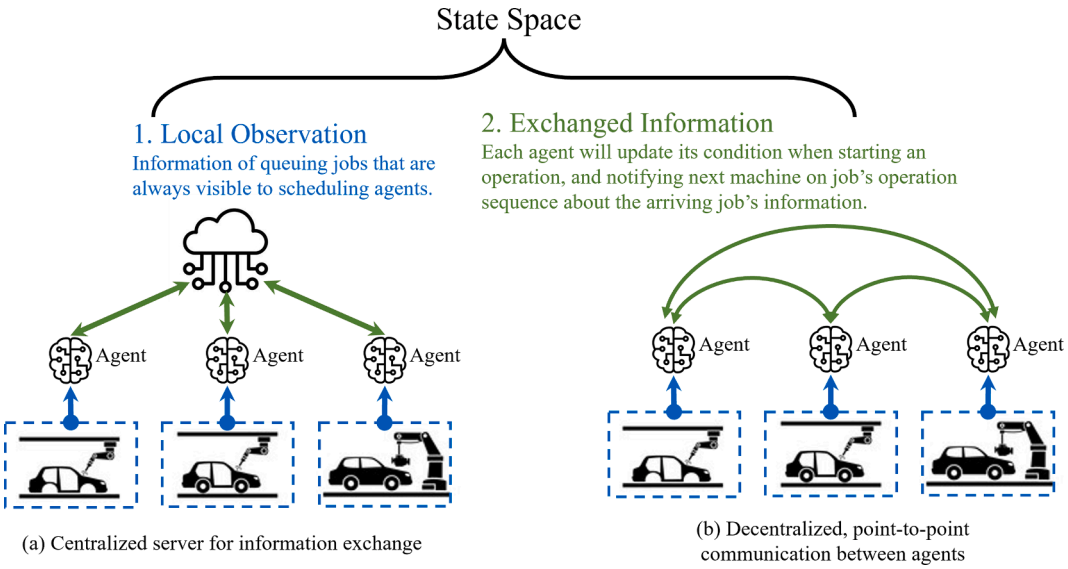


Fig. 2. Centralized training and decentralized execution scheme.



(a) Centralized server for information exchange

(b) Decentralized, point-to-point communication between agents

Fig. 3. Hypothetical information flow on the shop floor.

State features should be relevant to the scheduling objectives to prevent over-redundancy and computational waste. In addition to the information of candidate jobs (blue arrows in Fig. 3), selected information is exchanged between the agents (green arrows in Fig. 3) to enhance their observability, either through a centralized server or decentralized point-to-point communication. When an agent picks a job for processing, it immediately notifies the agent controlling the machine on which job will complete its next operation, sending the information about when the job will arrive and the job's attributes. To distinguish from the communication techniques (Foerster et al., 2016) in MARL research, the information exchange in this research is just an additional measure to augment observability, agents are not required to develop a communication protocol.

Some extra variables are introduced to facilitate the information exchange in system: (1) for job J_i, let the available time of its succeeding machine be denoted as SAM_i; (2) the time J_i have waited in the current queue, be denoted as CQ_i; and (3) the time till next arrival of job to machine M_k, be denoted as NA_k.

Five features are extracted for each candidate job: (1) $t_{i,k}$: the time expense of processing the candidate job, also the prolonged queuing time for other jobs (if selected); (2) WR_i: the minimal time before the completion of a job, showing how likely the congestion in the system can be reduced; (3) S_i: slack time, measurement of the urgency of a job; (4) SAM_i: a surrogate measurement of the queuing time of job's next operation; (5) CQ_i: the time that the agent has detained the job, indicating the accountability of agent in job's tardiness (if any).

The pseudocode of MR approach is presented in **Algorithm 1**. Jobs are included without repetition when the number of jobs in queue is greater than 3; otherwise, the jobs that conform to more rules would appear more times in state space.

Algorithm 1 Minimal-repetition approach to build the state space for M_k

Require: J^k
Initialize empty state tensor: $s_t \leftarrow [\ ]$
Initialize the buffer job set: $J^{buffer} \leftarrow J^k$
Initialize the rule set: $Rules \leftarrow [SPT, LWKR, MS, WINQ]$
Initialize the rule count: $x \leftarrow 1, y \leftarrow 1$
while $J^{buffer} \neq empty$ **and** $x \leq 4$ **do**
    Select J_a from J^{buffer} by x^{th} rule in $Rules$
    Append feature vector $[t_{a,k}, WR_a, S_a, SAM_a, CQ_a]$ to s_t
    $x \leftarrow x + 1$
    Delete J_a from J^{buffer}
end while
while $x \leq 4$ **do**

(*continued on next column*)

(*continued*)

    Select J_b from J^k by y^{th} rule in $Rules$
    Append feature vector $[t_{b,k}, WR_b, S_b, SAM_b, CQ_b]$ to s_t
    $x \leftarrow x + 1, y \leftarrow y + 1$
end while
if a job that will arrive at M_k is being processed elsewhere **then**
    Identify the job J_c that will arrive
    Append the feature vector $[t_{c,k}, WR_c, S_c, AM_k, NA_k]$ to s_t
else
    Append the dummy feature vector $[0, 0, 0, AM_k, 0]$ to s_t
end if
return s_t

The resultant state space is a 5×5 matrix consisting of the information of both queuing and arriving jobs (if any). The top four rows could contain duplicated elements when the number of queuing jobs is less than four.

The action space simply corresponds to the state space, in which actions are pointed to four slots of candidate jobs (possibly more than one slot direct to one job). The slot with the highest state-action value would be selected to be processed. Subsequently, the mapping from job-specific features to the direct selection of jobs can be learned.

4.3. Chronology joint action and knowledge-based reward shaping

A major challenge to DRL-based scheduling is the design of reward function. In the board games and video games where DRL has achieved great success, a trivial "victory" or "defeat" state would terminate the training and give an unbiased evaluation of the agent's policy. But in a dynamic scheduling problem aimed at minimizing cumulative tardiness, the production performance is a collective result of agents' behavior over a long period, during which a series of reward signals (realized tardiness of job) appear and need to be linked with actions which caused them.

The reward-shaping technique proposed in this work can be interpreted from a slack time consumption perspective. An example is presented in Fig. 4: a job with three operations arrives at the shop floor with an initial slack time as the buffer for queuing, the queuing at each stage of operation gradually consumes the slack time, and the eventual tardiness is the amount of over consumption of the slack time.

The tardiness of job results from decisions taken by the machines (agents) in the job's operation sequence, and agents cast their influence sequentially. The transition mechanism in traditional deep MARL

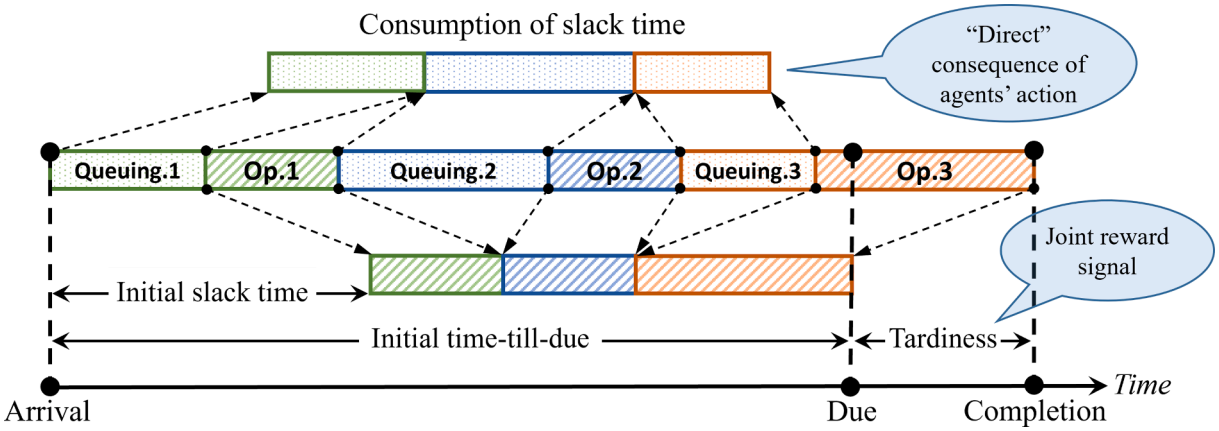


Fig. 4. Example production history.

algorithm should be revised to align with the sequential effect of agents' actions, to avoid misassignment of reward.

In most deep MARL tasks, all agents take action and move simultaneously to the next time step, therefore, the joint reward can be considered the direct result of all agents' actions in the previous time step. While in the scheduling problem, the influence of an agent's action is associated with a specific job and would only be realized upon the job's completion. This structure is termed as *chronological joint action* in this work; it changes the transition between states and the way to build experiences for training. Subsequently, parallel processes are needed to retain the incomplete experiences (only the state and action, without the reward) during the training.

The comparison between common MARL tasks and scheduling problems is presented in Fig. 5. Panel (a.1) and (a.2) show the difference in action-environment interaction. Panel (b.1) and (b.2) present the difference in the transition and experience-building process. Ω_t^k, r_t^k, a_t^k denotes the sub-state, sub-reward, and sub-action of the k^{th} agent (which controls M_k) at time step t, respectively.

A trivial reward mechanism, like most DRL studies, is to penalize agents for each tardy job, either by the actual value of tardiness or an arbitrary negative value (such as -1). While this manner is unbiased (Silver et al., 2021), it delivers unsatisfactory performance in actual practice, indicating that transformation of reward signal is needed to further identify the contribution of each agent.

A reward-shaping algorithm based on the queue time is developed in this work, wherein the joint reward is given upon a job's completion once the effect of all actions is realized. At that point, the history of production is backtracked, and the reward for each agent's action is calculated with regards to the time they detained the job. Rationales for the reward-shaping algorithm are:

(1) All agents along the trajectory are punished for a job's tardiness, but NOT awarded for its timeliness. Awarding timeliness may accelerate the convergence of the policy, but instill a preference for maximizing timeliness which may lead to a lower tardy rate but higher cumulative tardiness.
(2) Sequencing decision affects not only the queuing jobs, but also other agents by determining which agent's queue the job will subsequently join and when. If tardiness results, agents are held accountable for a portion of the queued time in their own queue, and a portion of the queued time at the succeeding machine. Agents learn to avoid over-crowding other agents' queue by reconstructing the queue time, with a validated shift-back ratio of $\alpha = 0.2$.
(3) As discussed in Section 3.3, queuing is inevitable, and tardiness is common given the training and validation scenario. To avoid overwhelming punishment, agents are not punished for the consumption of slack time beyond their control; the counting of queue time starts only when jobs can be selected for processing, as only the later queue time is a direct result of the agent's action.
(4) Agents are punished harder for detaining jobs with shorter slack time. The slack time of a job upon its arrival is converted to a criticality factor $\beta \in (0, 2)$ by a sigmoid function; the sensitivity of β to slack time is adjusted by a factor δ.
(5) The magnitude of queue time is adjusted by a factor φ, so most reward values fall within $[-1, 0]$, which is an appropriate range of input to the neural network. Square of time is used to punish excessively long waiting duration.

We next introduce extra notations associated with job J_i's production history and reward: (1) queue time before each operation, denoted as vector $Q_i = \left[Q_i^1, \cdots, Q_i^{O_i}\right]$; (2) slack time upon the job's arrival at each machine, denoted as vector $SA_i = \left[S_i^1, \cdots, S_i^{O_i}\right]$; and (3) the reward vector for each agent in the trajectory $R_i = \left[R_i^1, \cdots, R_i^{O_i}\right]$. The detailed reward-shaping procedure is presented in **Algorithm 2**.

Algorithm 2 Calculation of reward upon the completion of job J_i

Require: T_i, Q_i and SA_i
Initialize empty reward vector $R_i \leftarrow [\ ]$
if $T_i == 0$ **then**
  Fill reward vector with 0s:$R_i \leftarrow [0, \cdots, 0], |R_i| = O_i$
else
  for $j \leftarrow 1$ *to* O_i **do**
    Re-construct the queue time: $RQ_i^j = (1-\alpha) \times Q_i^j + \alpha \times Q_i^{j+1}$
    Calculate the criticality factor: $\beta = 1 - \left(S_i^j / \left(\left|S_i^j\right| + \delta\right)\right)$
    Get the reward: $R_i^t = -\left(\beta \times RQ_i^j / \varphi\right)^2$
    Clip the R_i^t to $[-1, 0]$
    Append R_i^t to R_i
  end for
end if
return R_i

An approximate search is done to determine the value of δ and φ, in the validated range $(2^6, 2^7)$. A larger δ results in higher sensitivity of reward to the slack time. The statistical distribution of slack time depends on the scenario: its upper limit ranges from 0 to 2 times the total process time; a lower limit can be found in Fig. 1, panels a3 to c3, as the slack time upon the completion becomes tardiness or earliness. An appropriate value of δ creates a smooth curve of β within the range of slack time.

φ is used to adjust the magnitude of reward to a suitable range so that most of the queue time would not be clipped; the change in its value results in only a minor change in the performance.

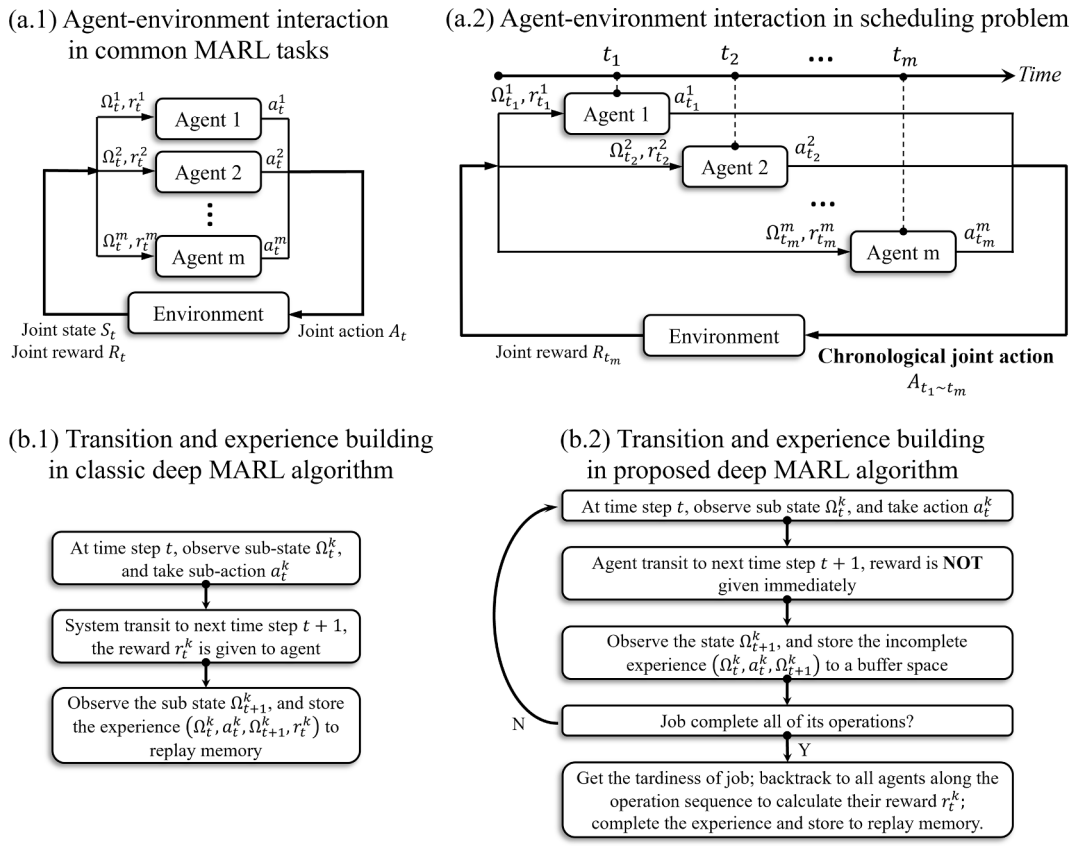


Fig. 5. Comparison between common MARL tasks and scheduling problem.

4.4. Parameters and training

The original values of features related to time (such as processing time $t_{i,k}$ and slack time S_i) fall beyond the ideal range for most activation functions, which leads to likely vanishing gradient problem. Meanwhile, the features in state space possess no time serial or graphical structure and are all measurements of time. Given these facts, ANN in this work uses Layer Normalization (Ba et al., 2016) technique to process the unstructured data, accompanied by a multi-layer perceptron (MLP) for the learning.

The parameters of training of agents and hyperparameters of MLP are listed in Table 1, the architecture of MLP is presented in Fig. 6.

The training simulates 100,000 units of time, with an average of 2800, 3200, and 3600 job arrivals on the shop floor under the three levels of utilization rate, respectively. The model development and experiment are implemented in Python programming language; the depending open-source libraries are listed in Table 2. The records of training loss under three scenarios are presented in Fig. 7.

Table 1
Parameters of training and hyperparameters of MLP.

Category	Parameter	Value
deep MARL	Discount factor (γ)	0.95
	Exploration rate (ε)	40% decay to 10%
	Minibatch size	64
	Replay memory size	1024
MLP	Activation function	tanh
	Loss function	Huber loss
	Optimizer	SGD, momentum = 0.9
	Learning rate	5×10^{-3} decay to 10^{-3}
	Hidden layers	$64 \times 48 \times 48 \times 36 \times 24 \times 12$

5. Experiment and results

The experiments are carried out in three stages:

(1) **Ablation and peer comparison in dynamic environments**: Except the state and action representations, multiple enhancements are incorporated to traditional MARL architecture in this work, such as doube DQN algorithm, centralized training and decentralized execution, parameter sharing, and reward-shaping. An ablation study is carried out to understand the contribution of those components to the overall performance. In addition, some DRL-based DJSP solutions are also included to verify the proposed approach's competitiveness.

(2) **Validation in dynamic environments**: Priority rules are widely applied in scheduling, especially in complex manufacturing systems, due to their low implementation cost, low decisional latency, and adaptability to different contexts. They are also used as benchmarks in almost all RL and DRL-based DJSP research. To validate the algorithm's effectiveness, this work also compares the proposed approach to a set of priority rules under three scenarios.

(3) **Generalization to static problem instances**: The production performance in dynamic environments is affected by both scheduling strategy's quality and the attributes of random events. The stochastic nature of problem makes it hard to compare different approaches on a "standardized" problem set; it also compromises the reproducibility and generalizability of any dynamic scheduling techniques. Therefore, we also generalize our model to the static environment, and measure its performance and decisional cost relative to metaheuristics.

In stage (1) and (2), the performance of proposed model, peer DRL-

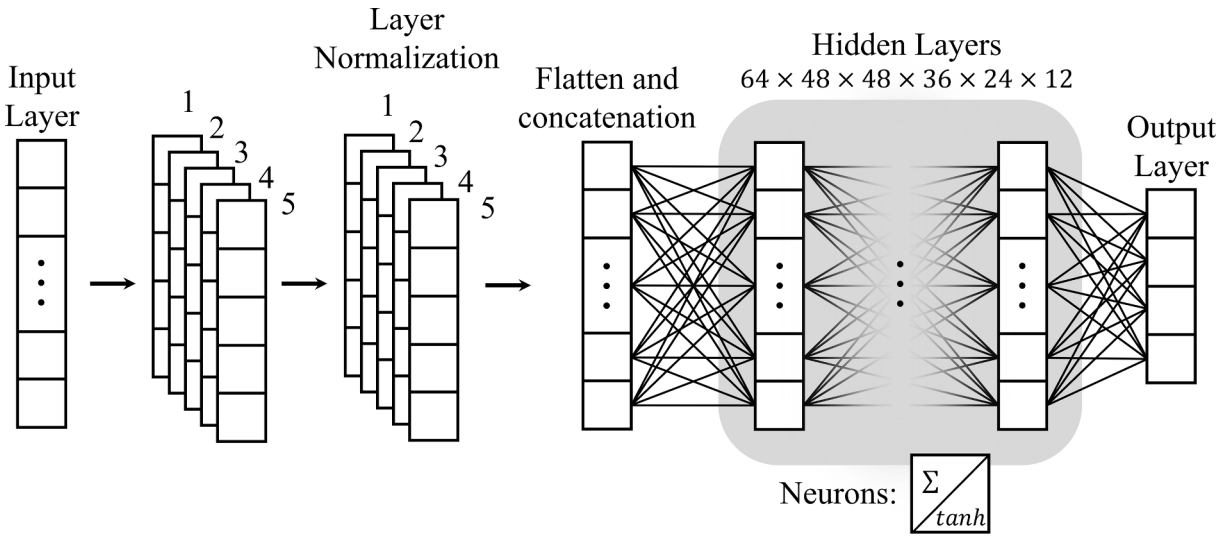


Fig. 6. Architecture of MLP.

Table 2
Dependent libraries.

Library	Description
SimPy (Matloff, 2008)	Discrete-event simulation library to build the model of manufacturing system
PyTorch (Paszke et al., 2019)	Machine learning library used for building the neural network in DRL algorithm
Matplotlib (Hunter, 2007)	2D graphics package used for image generation
NumPy (Harris et al., 2020)	Data analysis and processing

based approaches, and benchmark priority rules in **dynamic** environments are tested in 100 simulation runs under each scenario. A unique random event set is created in each run; the performance of scheduling strategies is tested in multiple simulation iterations that last for 2000 units of time, with an average of 56, 64, and 72 job arrivals under three levels of utilization rate (three scenarios), respectively. The process of experiments under each scenario is presented in Fig. 8.

In stage (3), a set of **static** problem instances are created using the identical setting used in previous two stages. The performance of the proposed approach and genetic algorithm are tested on these instances.

5.1. Ablation study and peer comparison in dynamic environments

Components of deep MARL model such as the training scheme, state and action representation, reward function, learning function, and architecture of encoder, etc. all contribute to the overall performance. In this section, components of the proposed architecture (termed deep MARL and reward-shaping, abbreviated as "deep MARL-RS" hereafter), including Double DQN algorithm, training scheme, and reward-shaping, are disabled (or replaced by classic frameworks) to investigate their individual utility.

Independent Q-learning that learns from the joint reward signal is probably the most commonly applied approach in MARL and serves as a strong benchmark (Rashid et al., 2020). We replace its tabular Q-learning algorithm with DQN and adopt the hyperparameters of ANN as well as the state and action representation used in the proposed deep MARL-RS algorithm. The resulting algorithm, termed independent DQN

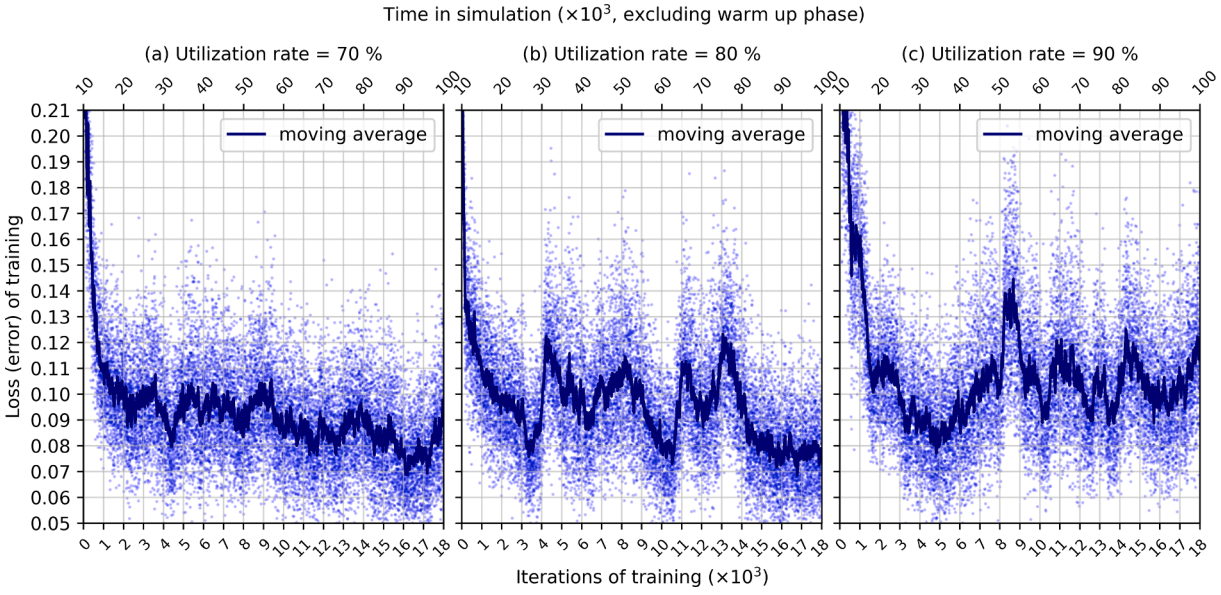
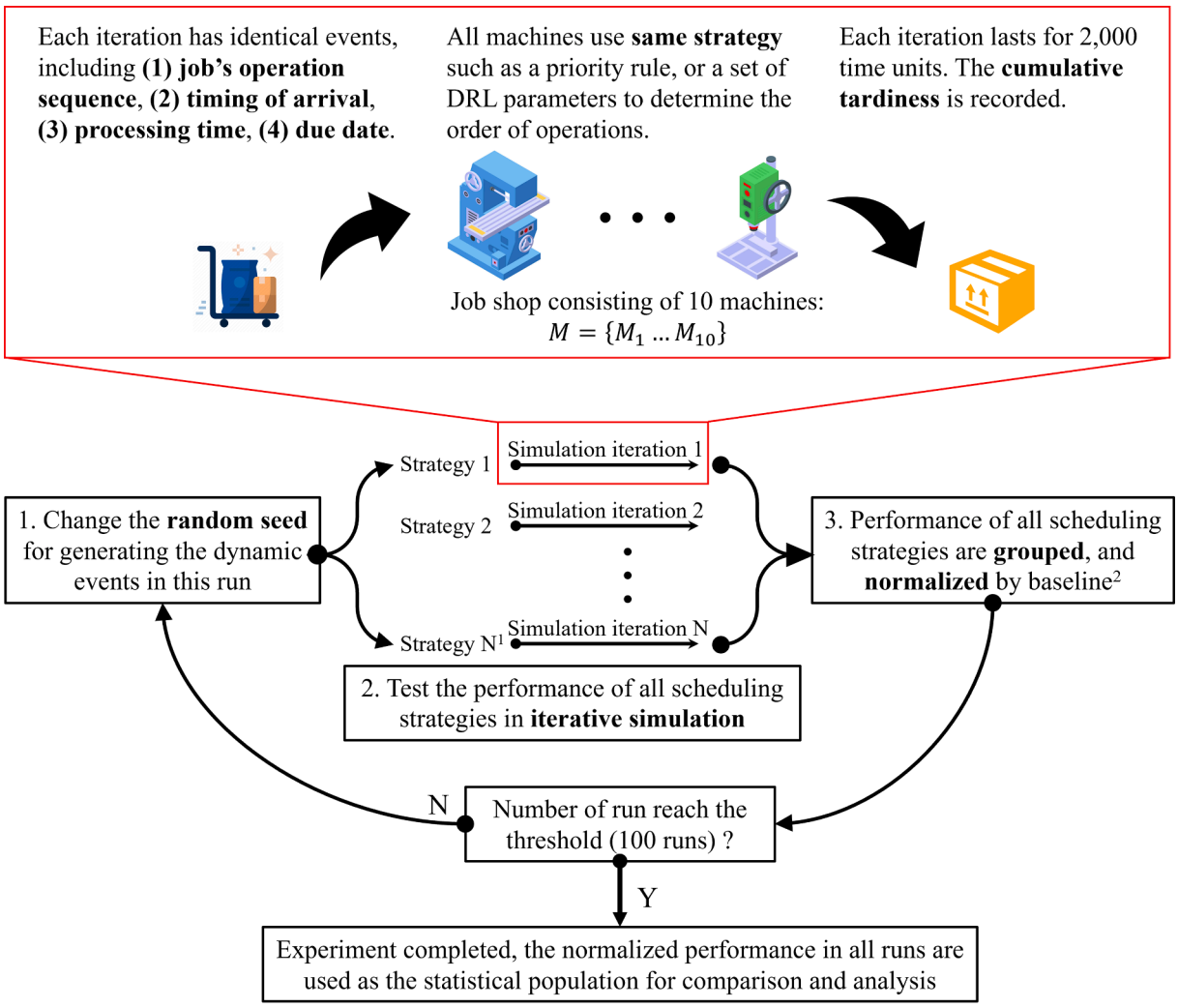


Fig. 7. Training loss.



[1] The number of strategies and respective number of iterations in a simulation run subject to the size of benchmark set. In Section 5.1, N = 8; while in Section 5.2, N = 21.
[2] Baseline is the performance of algorithm with minimal enhancements (I-DDQN algorithm in Section 5.1), or the priority rule that manages the job sequence in a most passive way (FIFO rule in Section 5.2).

Fig. 8. Experiment in dynamic environments (in one scenario).

(I-DQN), is used as the baseline in the ablation study.

Note that the proposed state and action representations, as the core components of our learning algorithm, cannot be replaced and measured. Therefore, a qualitative evaluation of state and action representations' contribution is implemented by comparing the proposed approach with some peer DRL-based benchmarks.

Recent RL-based dynamic scheduling techniques are included as the benchmark. However, it is unfair to compare against tabular algorithms (such as Q-learning or SARSA) as deep neural network has exhibited a decisive advantage in representing either the policy or value function. Therefore, we consider only DRL-based DJSP approaches, which leaves us very few studies to compare against. Furthermore, the scheduling problems in these studies are of fixed size, requiring the modification of their architecture to make them applicable to this study's context. The

modification[2], however, is minimal as we keep the hyperparameters of ANN, reward function, and action representation identical to the original design in the experiment stage[3]. The algorithms used in the ablation and comparative study are listed in Table 3.

Following metrics are used to measure the performance of the algorithm:

(1) Normalized cumulative tardiness (NCT): with I-DQN algorithm as the baseline, the normalized performance of DRL-based algorithms in each run is calculated as:

$$NCT = \frac{T_{sum}^{I-DQN} - T_{sum}^{algorithm}}{T_{sum}^{I-DQN}} \times 100\% \quad (9)$$

(2) Win rate: the percentage of runs in which the algorithm results in lowest total tardiness.

The experiment results are presented in Fig. 9. The solid black line shows the baseline performance (I-DQN), while the green dashed line shows deep MARL-RS's average performance.

Table 3
Algorithms in ablation study and peer comparison.

Algorithm	Name	Description
Proposed approach	deep MARL-RS	(1) Centralized training and decentralized execution with parameter sharing; (2) reward-shaping; (3) double DQN agents.
Ablation study	I-DQN*	**Baseline**: (1) Concurrent training without parameter sharing; (2) global reward; 3. DQN agents.
	I-DDQN	(1) Concurrent training without parameter sharing; (2) global reward; (3) double DQN agents.
	I-DDQN-RS	(1) Concurrent training without parameter sharing; (2) reward-shaping; (3) double DQN agents.
	P-DDQN	(1) Centralized training and decentralized execution with parameter sharing; (2) Global reward; (3) double DQN agents.
Peer approaches	N-2019 (Lin et al., 2019)	Single DQN agent with decentralized execution; action space consists of independent sets of priority rules for each agent.
	U-2020 (Liu et al., 2020)	Deep actor-critic, uses convolutional NN to process the data; action space consists of priority rules. Modified state space in experiment stage.
	O-2021 (Luo et al., 2021a)	Single DQN agent, direct selection of job. Modified state space in experiment stage.

[2] We didn't change any hyperparameters (including state and action representation, hyperparameters of ANN, reward function, and number of training epochs), or dataset (problem sets) in the **training stage**. All peer algorithms are trained as they were in original publications. However, in the **experiment phase**, some reproduced algorithms cannot directly solve our problem because they cannot handle the scheduling problem of variable size. Therefore, we applied the proposed "minimal-repetition" approach (see Section 4.2) to build a stable state space for U-2020 and O-2021 algorithm: MR approach is used to select a number of candidate jobs that match their input size; then fill the jobs' information in the state space. If the required number of jobs is greater than four, we will run the MR algorithm again over the remaining jobs until the state space is completed.

[3] Reproduced peer models used in this section may not represent the best performance that peer approaches may achieve, we strongly suggest our readers refer to the original publications for a complete understanding and comparison.

Experiment results show, albeit not as significantly, that double DQN still provides a 2 to 5% improvement compared to DQN. Meanwhile, reward-shaping and parameter sharing contribute most to the superior performance of the proposed approach; together they reduce around half of the tardiness as compared to the baseline; removing either one result in significant decline in performance. Quantified contributions of training scheme + parameter sharing, and reward-shaping are shown in Table 4.

The proposed deep MARL-RS approach also outperforms three benchmarks published in recent years in all scenarios. O-2021 delivers a performance that is very close to the baseline and I-DDQN, indicating that an independent multi-agent architecture alone, without special design in training scheme or reward, is comparable to the well-designed single-agent approach. U-2020 delivers good performance, but is constrained by the indirect job sequencing and a shallow ANN; N-2019, as the best peer approach, uses a partially shared encoder among agents with a branched output layer, and can be seen as a limiting case of parameter sharing technique. In addition, qualitative and indirect comparisons between DRL-based approaches can also be made as all these studies compare their approach to the universal benchmark: common priority rules.

Another observation is that peer approaches' performance improves as the expected utilization rate increases. The difference in training context is very likely the reason for this phenomenon.

RL algorithm's behavior is heavily influenced by the context it witnessed. All peer algorithms are trained with fixed-size problem instances, where all jobs are visible to and can be processed by the scheduling agent at the beginning of time horizon. In most of the problems, the number of jobs exceeds the number of available machines so the system is over-utilized (many jobs are queuing) most of the time. On the contrary, our algorithm is trained in a more balanced environment. Although the fluctuation of job arrival rate results in temporary over-utilization, in general, the system is not fully utilized and under-utilization also presents during training.

5.2. Validation in dynamic environments

As the most common real-time scheduling strategy, a set of priority rules with good record of minimizing tardiness are used to validate the performance of deep MARL-RS, as listed in Table 5. Note that some famous rules such as longest processing time (LPT), earliest release date (ERD) and most work remaining (MWKR) are discarded from the benchmark set because of their poor performance in minimizing tardiness. The criteria for selecting priority rules are based on (Đurasević and Jakobović, 2018; Sels et al., 2012; Xiong et al., 2017). Readers are advised to refer to these publications for complete comparative studies of priority rules in different contexts.

Metaheuristic-based approaches are not compared in dynamic environments due to the difficulty in implementation: most metaheuristics-based studies aim to solve dynamic problems with little or no rescheduling needs, and don't focus on minimizing the latency of decision-making process. However, the designated context in this work, as illustrated in Fig. 8, contains 100 problem instances under each scenario, resulting in 5600, 6400, and 7200 rescheduling points, respectively. A new scheduling problem is created at each rescheduling point; solving all these problems by metaheuristics leads to extremely high time and computational expense.

Following metrics are used to evaluate the performance of deep MARL-RS and priority rules:

(1) Normalized cumulative tardiness (NCT): with FIFO as the baseline, the normalized performance of rules or algorithms in each run is calculated as

$$NCT = \frac{T_{sum}^{FIFO} - T_{sum}^{rule\ or\ algorithm}}{T_{sum}^{FIFO}} \times 100\% \quad (10)$$

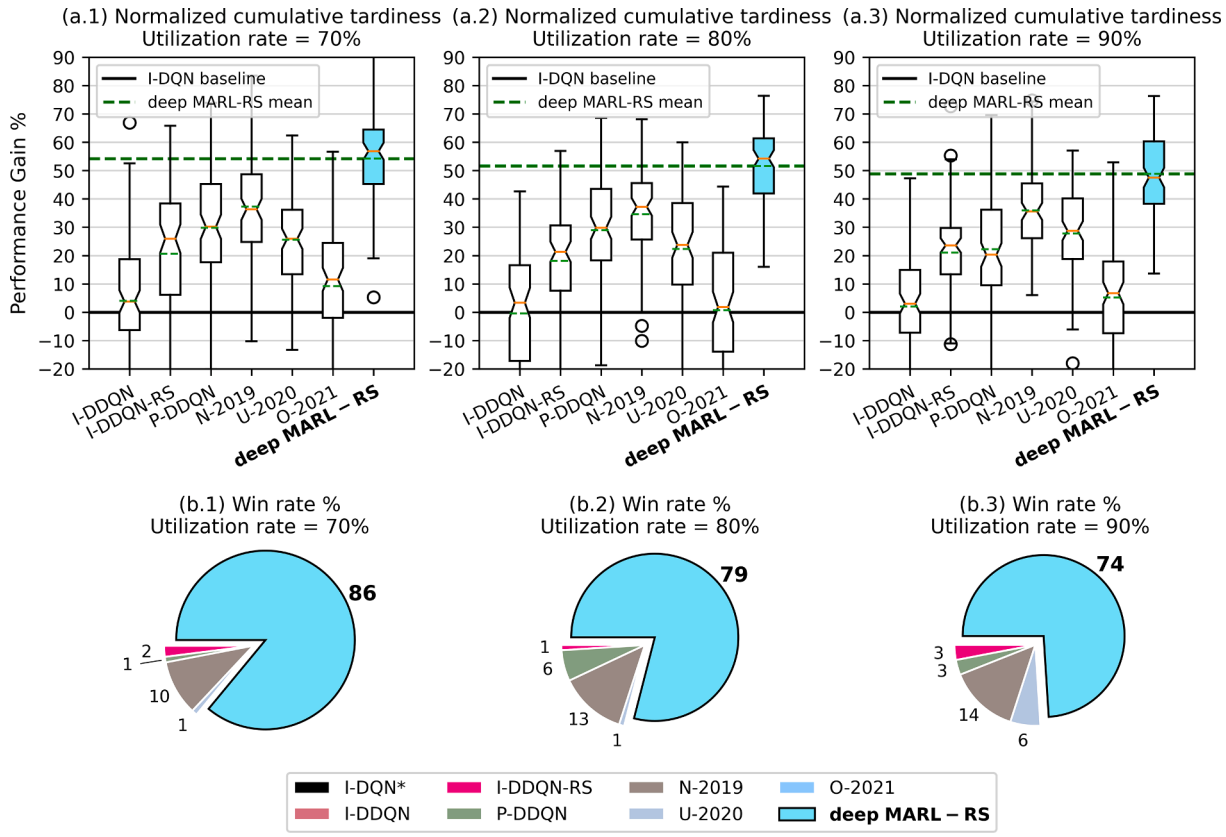


Fig. 9. Result of ablation study and peer comparison in dynamic environment.

Table 4
Quantified contribution of components (by percentage).

Category	Calculation	Contribution (by percentage) under three utilization rate levels		
		70%	80%	90%
Training scheme + Parameter sharing	$\frac{T_{sum}^{I-DDQN} - T_{sum}^{P-DDQN}}{T_{sum}^{I-DDQN}}$	24.44	27.73	19.00
	$\frac{T_{sum}^{I-DDQN-RS} - T_{sum}^{deepMARL-RS}}{T_{sum}^{I-DDQN-RS}}$	40.46	38.96	33.83
Reward-shaping	$\frac{T_{sum}^{I-DDQN} - T_{sum}^{I-DDQN-RS}}{T_{sum}^{I-DDQN}}$	15.00	16.75	17.97
	$\frac{T_{sum}^{P-DDQN} - T_{sum}^{deepMARL-RS}}{T_{sum}^{P-DDQN}}$	32.86	29.81	32.10

(2) Win rate: the percentage of runs in which the rule or algorithm results in lowest total tardiness. To better present how well the proposed deep MARL-RS approach performs relative to benchmark solutions, the result with and without deep MARL-RS are both shown.

The results of experiments are presented in Fig. 10: the solid black line shows the baseline performance (FIFO), below this line represents the zone of worse-than-doing-nothing performance, while the green dashed line is the mean of the normalized performance of deep MARL-RS.

As shown in Fig. 10, the proposed deep MARL-RS approach outperforms most priority rules in all scenarios; it provides 3–5% higher normalized performance gain (FIFO as the denominator) than the top benchmark rule (PTWINQS). A comparison of win rates with and without deep MARL agents shows that deep MARL-RS does not simply imitate the behavior of a specific rule, but maintains good performance across all validation instances and beats the other rules uniformly.

Another cross-scenario observation is that the advantage of composite rules and singular rules with more complex mathematical such as

Table 5
Benchmark priority rules.

Category	Abbreviation	Description
Singular	ATC	Apparent tardiness cost
	AVPRO	Average processing time per operation
	COVERT	Cost over time
	CR	Critical ratio
	EDD	Earliest due date
	LWKR	Least work remaining
	MDD	Modified due date
	MOD	Modified operational due date
	MS	Minimal slack
	NPT	Next operation's processing time
	SPT	Shortest processing time
	WINQ	Work in queue
Composite	CR + SPT	Sum of critical ration and processing time
	LWKR + SPT	Sum of work remaining and processing time
	LWKR + MOD	Sum of work remaining and modified operational due date
	PT + WINQ	Sum of processing time and work in queue
	PT + WINQ + S	Sum of processing time, work in queue, and slack time
	2PT + LWKR + S	Sum of processing time (doubled), work remaining, and slack time
	2PT + WINQ + NPT	Sum of processing time (doubled), work in queue, and Next operation's processing time
Baseline	FIFO*	First in first out

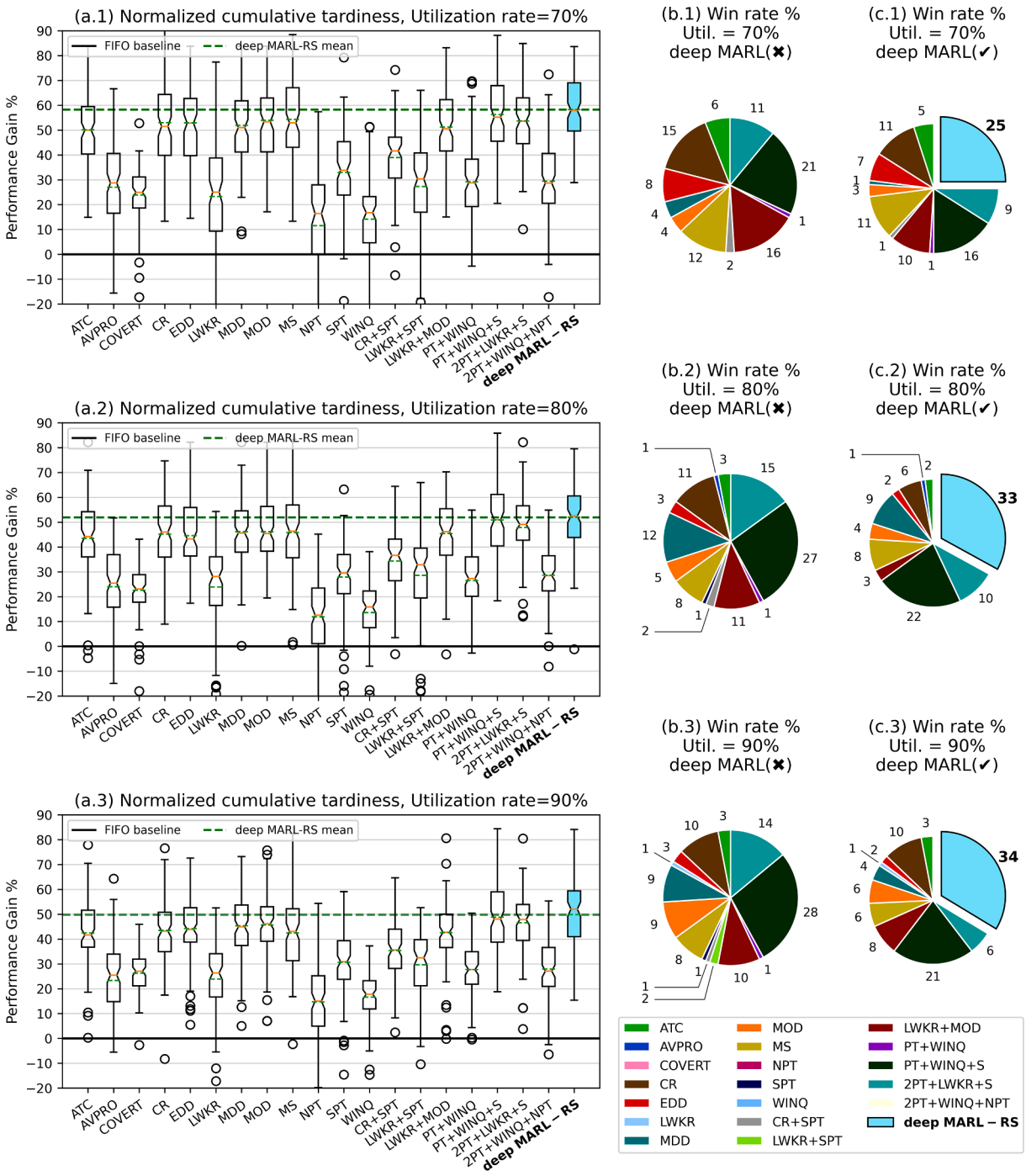


Fig. 10. Result of validation experiment in dynamic environment.

MOD and MDD is consistent across three expected utilization levels. Additionally, their win rate increases as the utilization level increases, indicating their superiority in a congested system; this indirectly proves the necessity of complex parametrization of shop floor information in scheduling.

5.3. Generalization to static instances and computational efficiency

A dynamic scheduling problem can be divided into a series of static problems, whose boundaries are drawn by the occurrence of unforeseen events; those events change the specification of the scheduling problem (needs to consider the new job, and problem size fluctuates over time) and trigger the rescheduling process.

The long-term performance of production scheduling is therefore a cumulative result of individual scheduling decisions at each rescheduling point; measuring the quality and cost of decisions at each rescheduling point (solving a static problem) can be considered a surrogate evaluation of the proposed approach.

Due to the absence of a widely-applied problem library with tardiness-related objectives, problem instances used in this section are randomly created using the parameters illustrated in Section 3.2 (processing time and due date tightness).

Metaheuristics are used as the "practical optimal" solutions in most JSP research due to the NP-hardness of problem. We tested our approach and Genetic Algorithm (GA) in problems of three classes of specification: (1) 5 jobs and 5 machines; (2) 10 jobs and 10 machines; (3) 15 jobs and

10 machines.

Ten instances are created under each complexity; the details of instances and the schedules produced by GA are presented in the appendix. Parameters of GA are listed in Table 6.

The comparison between GA and deep MARL-RS is presented in Fig. 11. In addition to the normalized cumulative tardiness (FIFO as baseline), we also include the number of tardy jobs as a complementary metric.

The gap between GA and deep MARL-RS model is amplified by the complexity of problem. Deep MARL-RS delivers almost identical performance on small-size problems; its performance deteriorates relative to GA when problem size increases. In general, GA provides around 5–15% higher performance gain on larger problem instances.

Decisional latency is also an important factor that determines whether an approach can make near real-time scheduling decisions and be applied to highly dynamic environments. It's noteworthy that priority rules and deep MARL-RS model can schedule for only one machine when a scheduling decision is required, while GA produces a complete schedule containing all operations in the problem. Therefore we must consider both the time for single scheduling decisions and the cumulative decisional cost during the production. The CPU times of different approaches are listed in Table 7.

CPU time is affected by many factors, such as the programming language and processing power of devices. However, the difference between deep MARL-RS and GA is in order of magnitude, which should be credited more to the structural difference between algorithms.

All metaheuristics comprise iterative evaluation and reproduction processes, which consume most computational resources on manipulating and evaluating inferior candidates that wouldn't contribute to the performance improvement. Despite the rapid advancement in dynamic scheduling research, much effort from both algorithm and hardware perspectives is needed to reconcile the efficiency and quality of decision-making in data-intensive and highly-dynamic environments.

6. Conclusion and future work

As one of the hottest topics of artificial intelligence and machine learning, DRL exhibits great potential in real-time control, and researchers have gradually extended its applications from the digital world to cyber-physical systems. An integrated deep MARL approach is proposed in this research to solve the scheduling problem with dynamic job arrivals; as an attempt to build the scheduling system for the factory of the future, which will be characterized by unlimited real-time data accessibility, and requires low-latency and high-quality decision-making.

In addition to incorporating popular techniques such as centralized training and decentralized execution and parameter sharing, we develop novel state and action representations to overcome the constraint of fixed-size problems in traditional scheduling research. Knowledge-based reward-shaping is developed to decompose the aggregated performance of production to individual and learnable rewards for distributed agents; the framework of the classic deep MARL algorithm is also adapted to the chronological-joint-action structure of the scheduling problem. The utility of the proposed components is tested and validated in an ablation experiment; the overall effectiveness of our approach convincing outperforms contemporary approaches and a set of well-tested priority rules.

Some research gaps can be identified for future work:

(1) The study of dynamic scheduling is still in its infancy, resulting in absence of a well-established context or problem set, which leads to difficulties in benchmarking algorithms. A possible reason is the complexity of dynamic events: researchers may focus on events that are unique to their own domain and context but produce incomparable instances of problems that differ from traditional flow shop scheduling problem, JSP, or flexible JSP.
(2) The ANN used in this work is a lightweight MLP with only 10,000 + parameters. Even though it delivers satisfactory results and guarantees the efficiency of training and application, it is observed that a "deeper" MLP does provide better performance at the cost of difficulty in convergence during training. The model used for validation is trained with conservative hyperparameter settings to ensure its reproducibility; better performance can be expected by incorporating cutting-edge deep learning techniques, to achieve higher stability and capability.
(3) The selection, abstraction, and preprocessing of industrial data rely on interdisciplinary expertise in machine learning and manufacturing domain. The state features used in this work are handcrafted based on literature review and iterative experiments, but the ever-increasing volume and complexity of data will eventually surpass human capability, not to mention the possibly inaccurate and irrelevant data collected in real manufacturing systems. An automatic data-cleansing and prioritization mechanism is needed to augment the actual application of learning-based scheduling, to avoid the wastage of human labor and computing resources.
(4) In addition to hyperparameter-tuning, the curriculum design of RL also plays an important role in achieving high training efficiency and performance. It has been observed in this research that models trained in different contexts (training problem instances) exhibit distinct specialties. A quantitative study of the use of training problem instances would contribute to the development of better RL-based dynamic scheduling approaches.

Declarations.

- Funding: This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.
- Conflicts of interest/Competing interests: The authors declare that they have no conflicts of interest/competing interests.
- Code availability: Dependent open-source Python libraries are stated in section 4.4. Codes are available in: https://github.com/RK0731/Deep-MARL-for-Dynamic-JSP

CRediT authorship contribution statement

Renke Liu: Conceptualization, Methodology, Software, Validation, Writing – original draft, Formal analysis, Investigation. **Rajesh Piplani:** Conceptualization, Methodology, Validation, Writing – review & editing, Supervision. **Carlos Toro:** Validation, 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.

Table 6
Genetic algorithm parameters.

Category	Value/Description		
Problem size ($n \times m$)	5×5	10×10	15×10
Population size	100	200	300
Generation	100		
Crossover operator	Position based crossover		
Crossover rate	0.8		
Mutation operator	Swapping		
Mutation ratio	0.1		
Elitism	Applied		
Encoding	Operation-based		

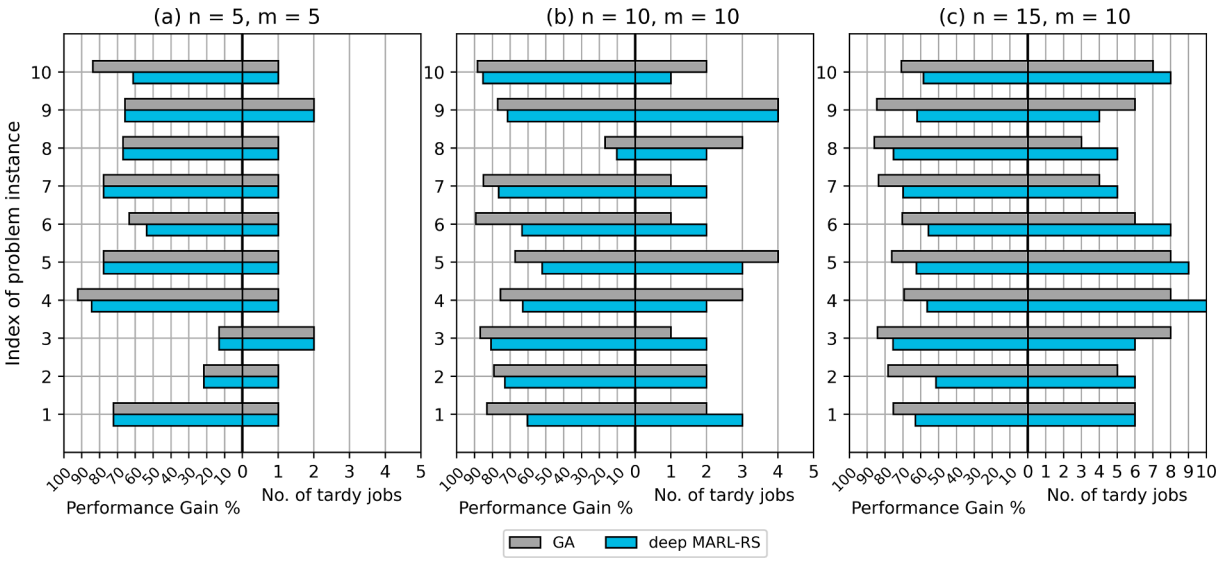


Fig. 11. Performance on static problem instances.

Table 7
Average CPU time for scheduling decision.

Problem Size ($n \times m$)	Category	Priority rule (FIFO)	Deep MARL-RS	Genetic Algorithm
5×5	Single decision	$9.46 \times 10^{-7}s$	$2.11 \times 10^{-4}s$	3.26s
	All decisions	$2.37 \times 10^{-5}s$	$5.28 \times 10^{-3}s$	
10×10	Single decision	$9.87 \times 10^{-7}s$	$2.12 \times 10^{-4}s$	31.43s
	All decisions	$9.87 \times 10^{-5}s$	$2.12 \times 10^{-2}s$	
15×10	Single decision	$9.96 \times 10^{-7}s$	$2.14 \times 10^{-4}s$	62.80s
	All decisions	$1.49 \times 10^{-4}s$	$3.21 \times 10^{-2}s$	

Appendix A. Supplementary material

Supplementary data to this article can be found online at https://doi.org/10.1016/j.cor.2023.106294.

References

Al-Behadili, M., Ouelhadj, D., Jones, D., 2017. Multi-objective particle swarm optimisation for robust dynamic scheduling in a permutation flow shop. Cham.

Ba, J. L., Kiros, J. R., Hinton, G. E., 2016. Layer normalization. arXiv preprint, arXiv: 1607.06450.

Baer, S., Bakakeu, J., Meyes, R., Meisen, T., 2019. Multi-agent reinforcement learning for job shop scheduling in flexible manufacturing systems. In: *2019 Second International Conference on Artificial Intelligence for Industries (AI4I)*. IEEE, 2019.

Beasley, J.E., 1990. OR-Library: distributing test problems by electronic mail. J. Oper. Res. Soc. 41 (11), 1069–1072.

Berger, S.L.T., Zanella, R.M., Frazzon, E.M., 2019. Towards a data-driven predictive-reactive production scheduling approach based on inventory availability. IFAC-PapersOnLine 52 (13), 1343–1348. https://doi.org/10.1016/j.ifacol.2019.11.385.

Bouazza, W., Sallez, Y., Beldjilali, B., 2017. A distributed approach solving partially flexible job-shop scheduling problem with a Q-learning effect. IFAC-PapersOnLine 50 (1), 15890–15895. https://doi.org/10.1016/j.ifacol.2017.08.2354.

Branke, J., Hildebrandt, T., Scholz-Reiter, B., 2015. Hyper-heuristic evolution of dispatching rules: a comparison of rule representations. Evol. Comput. 23 (2), 249–277.

Chen, J., Ran, X., 2019. Deep learning with edge computing: a review. Proc. IEEE 107 (8), 1655–1674. https://doi.org/10.1109/JPROC.2019.2921977.

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. Intel. 101 https://doi.org/10.1016/j.engappai.2021.104207.

Đurasević, M., Jakobović, D., 2018. A survey of dispatching rules for the dynamic unrelated machines environment. Expert Syst. Appl. 113 https://doi.org/10.1016/j.eswa.2018.06.053.

Foerster, J., Assael, I.A., de Freitas, N., Whiteson, S., 2016. Learning to Communicate with Deep Multi-Agent Reinforcement Learning. Advances in Neural Information Processing Systems (NIPS 2016), 29.

Gabel, T., Riedmiller, M., 2012. Distributed policy search reinforcement learning for job-shop scheduling tasks. Int. J. Prod. Res. 50 (1), 41–61. https://doi.org/10.1080/00207543.2011.571443.

Gupta, J.K., Egorov, M., Kochenderfer, M., 2017. Cooperative Multi-agent Control Using Deep Reinforcement Learning. In: *Autonomous Agents and Multiagent Systems: AAMAS 2017 Workshops, Best Papers*. São Paulo, Brazil.

Gupta, J.N.D., Majumder, A., Laha, D., 2019. Flowshop scheduling with artificial neural networks. J. Oper. Res. Soc. 1–19 https://doi.org/10.1080/01605682.2019.1621220.

Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Oliphant, T.E., 2020. Array programming with NumPy. Nature 585 (7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2.

Hildebrandt, T., Heger, J., Scholz-Reiter, B., 2010. Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach. Proceedings of the 12th annual conference on Genetic and evolutionary computation.

Holthaus, O., Rajendran, C., 1997. Efficient dispatching rules for scheduling in a job shop. Int. J. Prod. Econ. 48 (1), 87–105. https://doi.org/10.1016/S0925-5273(96)00068-0.

Hunter, J.D., 2007. Matplotlib: a 2D graphics environment. Comput. Sci. Eng. 9 (3), 90–95. https://doi.org/10.1109/MCSE.2007.55.

Jain, A.S., Meeran, S., 1999. Deterministic job-shop scheduling: past, present and future. Eur. J. Oper. Res. 113 (2), 390–434. https://doi.org/10.1016/S0377-2217(98)00113-1.

Jun, S., Lee, S., Chun, H., 2019. Learning dispatching rules using random forest in flexible job shop scheduling problems. Int. J. Prod. Res. 57 (10), 3290–3310. https://doi.org/10.1080/00207543.2019.1581954.

Li, X., Gao, L., 2020. In: A hybrid genetic algorithm and tabu search for multi-objective dynamic jsp. Springer, Berlin Heidelberg, pp. 377–403. https://doi.org/10.1007/978-3-662-55305-3_18.

Lin, C.-C., Deng, D.-J., Chih, Y.-L., Chiu, H.-T., 2019. Smart Manufacturing scheduling with edge computing using multiclass deep Q network. IEEE Trans. Ind. Inf. 15 (7), 4276–4284. https://doi.org/10.1109/TII.2019.2908210.

Liu, C.-L., Chang, C.-C., Tseng, C.-J., 2020. Actor-critic deep reinforcement learning for solving job shop scheduling problems. IEEE Access 8, 71752–71762. https://doi.org/10.1109/ACCESS.2020.2987820.

Liu, F., Wang, S., Hong, Y., Yue, X., 2017. On the robust and stable flowshop scheduling under stochastic and dynamic disruptions. IEEE Trans. Eng. Manag. 64 (4), 539–553. https://doi.org/10.1109/TEM.2017.2712611.

R. Lowe Y. Wu A. Tamar J. Harb P. Abbeel I. Mordatch Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. arXiv preprint 2017 arXiv: 1706.02275.

Luo, B., Wang, S., Yang, B., Yi, L., 2021a. An improved deep reinforcement learning approach for the dynamic job shop scheduling problem with random job arrivals. J. Phys. Conf. Ser. 1848 (1), 012029.

Luo, S., Zhang, L., Fan, Y., 2021b. Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning. Comput. Ind. Eng. 159 https://doi.org/10.1016/j.cie.2021.107489.

Matloff, N., 2008. Introduction to discrete-event simulation and the simpy language, Vol. 2.

Mei, Y., Nguyen, S., Xue, B., Zhang, M., 2017. An efficient feature selection algorithm for evolving job shop scheduling rules with genetic programming. IEEE Trans. Emerg. Top. Comput. Intell. 1 (5), 339–353. https://doi.org/10.1109/TETCI.2017.2743758.

Messaoud, S., Bradai, A., Bukhari, S.H.R., Quang, P.T.A., Ahmed, O.B., Atri, M., 2020. A survey on machine learning in Internet of Things: algorithms, strategies, and applications. Internet of Things 12, 100314. https://doi.org/10.1016/j.iot.2020.100314.

Mirshekarian, S., Šormaz, D.N., 2016. Correlation of job-shop scheduling problem features with scheduling efficiency. Expert Syst. Appl. 62, 131–147. https://doi.org/10.1016/j.eswa.2016.06.014.

Mnih, V., Badia, A.P., Mirza, M., et al., 2016. Asynchronous methods for deep reinforcement learning[C]//International conference on machine learning. PMLR 1928–1937.

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. https://doi.org/10.1038/nature14236.

Mouelhi-Chibani, W., Pierreval, H., 2010. Training a neural network to select dispatching rules in real time. Comput. Ind. Eng. 58 (2), 249–256. https://doi.org/10.1016/j.cie.2009.03.008.

Nguyen, S., Zhang, M., Johnston, M., Tan, K.C., 2013. A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. IEEE Trans. Evol. Comput. 17 (5), 621–639. https://doi.org/10.1109/TEVC.2012.2227326.

Olafsson, S., Li, X., 2010. Learning effective new single machine dispatching rules from optimal scheduling data. Int. J. Prod. Econ. 128 (1), 118–126. https://doi.org/10.1016/j.ijpe.2010.06.004.

Oliehoek, F.A., Amato, C., 2016. The decentralized POMDP framework. In: Oliehoek, F. A., Amato, C. (Eds.), A Concise Introduction to Decentralized POMDPs. Springer International Publishing, pp. 11–32. https://doi.org/10.1007/978-3-319-28929-8_2.

OroojlooyJadid, A., Hajinezhad, D., 2019. A review of cooperative multi-agent deep reinforcement learning. arXiv e-prints, arXiv:1908.03963.

Ouelhadj, D., Petrovic, S., 2008. A survey of dynamic scheduling in manufacturing systems. J. Sched. 12 (4), 417. https://doi.org/10.1007/s10951-008-0090-8.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., 2019. Pytorch: an imperative style, high-performance deep learning library. Adv. Neural Inf. Proces. Syst.

Priore, P., Parreño, J., Pino, R., Gómez, A., Puente, J., 2010. Learning-based scheduling of flexible manufacturing systems using support vector machines. Appl. Artif. Intell. 24 (3), 194–209. https://doi.org/10.1080/08839510903549606.

Qu, S., Wang, J., Jasperneite, J., 2019. Dynamic scheduling in modern processing systems using expert-guided distributed reinforcement learning[C]//2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE, pp. 459–466.

Rashid, T., Samvelyan, M., De Witt, Farquhar, G., Foerster, J., Whiteson, S., 2020. Monotonic value function factorisation for deep multi-agent reinforcement learning. J. Mach. Learn. Res. 21 (1), 7234–7284.

Sels, V., Gheysen, N., Vanhoucke, M., 2012. A comparison of priority rules for the job shop scheduling problem under different flow time-and tardiness-related objective functions. Int. J. Prod. Res. 50 (15), 4255–4270.

Shady, S., Kaihara, T., Fujii, N., Kokuryo, D., 2022. A novel feature selection for evolving compact dispatching rules using genetic programming for dynamic job shop scheduling. Int. J. Prod. Res. 1–24 https://doi.org/10.1080/00207543.2022.2053603.

Shi, D., Fan, W., Xiao, Y., Lin, T., Xing, C., 2020. Intelligent scheduling of discrete automated production line via deep reinforcement learning. Int. J. Prod. Res. 58 (11), 3362–3380. https://doi.org/10.1080/00207543.2020.1717008.

Shiue, Y.-R., 2009. Data-mining-based dynamic dispatching rule selection mechanism for shop floor control systems using a support vector machine approach. Int. J. Prod. Res. 47 (13), 3669–3690. https://doi.org/10.1080/00207540701846236.

Silver, D., Singh, S., Precup, D., Sutton, R.S., 2021. Reward is enough. Artif. Intell. 299, 103535 https://doi.org/10.1016/j.artint.2021.103535.

Son, K., Kim, D., Kang, W.J., Hostallero, D.E., Yi, Y., 2019 May. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In: *International conference on machine learning*. PMLR, pp. 5887–5896.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W.M., Zambaldi, V., Jaderberg, M., ... Graepel, T., 2017. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*.

Sutton, R.S., Barto, A.G., 2018. Reinforcement Learning: An Introduction, second ed. The MIT Press.

Taillard, E., 1993. Benchmarks for basic scheduling problems. Eur. J. Oper. Res. 64 (2), 278–285. https://doi.org/10.1016/0377-2217(93)90182-M.

Tumer, K., Agogino, A.K., Wolpert, D.H., 2002, July. Learning sequences of actions in collectives of autonomous agents. In: Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1, pp. 378–385.

Van Hasselt, H., Guez, A., Silver, D., 2016. Deep reinforcement learning with double q-learning. Thirtieth AAAI conference on artificial intelligence.

Wang, Y.-F., 2020. Adaptive job shop scheduling strategy based on weighted Q-learning algorithm. J. Intell. Manuf. 31 (2), 417–432. https://doi.org/10.1007/s10845-018-1454-3.

Wang, Z., Zhang, J., Yang, S., 2019. An improved particle swarm optimization algorithm for dynamic job shop scheduling problems with random job arrivals. Swarm Evol. Comput. 51, 100594 https://doi.org/10.1016/j.swevo.2019.100594.

Waschneck, B., Reichstaller, A., Belzner, L., Altenmüller, T., Bauernhansl, T., Knapp, A., Kyek, A., 2018. Optimization of global production scheduling with deep reinforcement learning. Proc. CIRP 72, 1264–1269. https://doi.org/10.1016/j.procir.2018.03.212.

Xiong, H., Fan, H., Jiang, G., Li, G., 2017. A simulation-based study of dispatching rules in a dynamic job shop scheduling problem with batch release and extended technical precedence constraints. Eur. J. Oper. Res. 257 https://doi.org/10.1016/j.ejor.2016.07.030.

Yan, Q., Wang, H., Wu, F., 2022. Digital twin-enabled dynamic scheduling with preventive maintenance using a double-layer Q-learning algorithm. Comput. Oper. Res. 144 https://doi.org/10.1016/j.cor.2022.105823.

Zang, Z., Wang, W., Song, Y., Lu, L., Li, W., Wang, Y., Zhao, Y., 2019. Hybrid deep neural network scheduler for job-shop problem based on convolution two-dimensional transformation. Comput. Intell. Neurosci. 2019, 7172842. https://doi.org/10.1155/2019/7172842.

Zhang, R., Song, S., Wu, C., 2020. Robust scheduling of hot rolling production by local search enhanced ant colony optimization algorithm. IEEE Trans. Ind. Inf. 16 (4), 2809–2819. https://doi.org/10.1109/TII.2019.2944247.

Zhou, Y., Yang, J.-J., Zheng, L.-Y., 2019. Hyper-heuristic coevolution of machine assignment and job sequencing rules for multi-objective dynamic flexible job shop scheduling. IEEE Access 7, 68–88. https://doi.org/10.1109/ACCESS.2018.2883802.

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. https://doi.org/10.1080/00207543.2019.1620362.