Li, Funing; Lang, Sebastian; Hong, Bingyuan; Reggelin, Tobias

A two-stage RNN-based deep reinforcement learning approach for solving the parallel machine scheduling problem with due dates and family setups

Journal of Intelligent Manufacturing

Provided in Cooperation with:
Springer Nature

Suggested Citation: Li, Funing; Lang, Sebastian; Hong, Bingyuan; Reggelin, Tobias (2023) : A two-stage RNN-based deep reinforcement learning approach for solving the parallel machine scheduling problem with due dates and family setups, Journal of Intelligent Manufacturing, ISSN 1572-8145, Springer US, New York, NY, Vol. 35, Iss. 3, pp. 1107-1140, https://doi.org/10.1007/s10845-023-02094-4

Journal of Intelligent Manufacturing (2024) 35:1107–1140
https://doi.org/10.1007/s10845-023-02094-4

A two-stage RNN-based deep reinforcement learning approach for solving the parallel machine scheduling problem with due dates and family setups

Funing Li[1] · Sebastian Lang[1,2] · Bingyuan Hong[3] · Tobias Reggelin[1]

Received: 11 April 2022 / Accepted: 9 February 2023 / Published online: 9 March 2023

Abstract
As an essential scheduling problem with several practical applications, the parallel machine scheduling problem (PMSP) with family setups constraints is difficult to solve and proven to be NP-hard. To this end, we present a deep reinforcement learning (DRL) approach to solve a PMSP considering family setups, aiming at minimizing the total tardiness. The PMSP is first modeled as a Markov decision process, where we design a novel variable-length representation of states and actions, so that the DRL agent can calculate a comprehensive priority for each job at each decision time point and then select the next job directly according to these priorities. Meanwhile, the variable-length state matrix and action vector enable the trained agent to solve instances of any scales. To handle the variable-length sequence and simultaneously ensure the calculated priority is a global priority among all jobs, we employ a recurrent neural network, particular gated recurrent unit, to approximate the policy of the agent. The agent is trained based on Proximal Policy Optimization algorithm. Moreover, we develop a two-stage training strategy to enhance the training efficiency. In the numerical experiments, we first train the agent on a given instance and then employ it to solve instances with much larger scales. The experimental results demonstrate the strong generalization capability of the trained agent and the comparison with three dispatching rules and two metaheuristics further validates the superiority of this agent.

Keywords Deep reinforcement learning · Parallel machine scheduling · Family setups · Recurrent neural network

✉ Sebastian Lang
sebastian.lang@ovgu.de

Funing Li
funing.li@st.ovgu.de

Bingyuan Hong
hongby@zjou.edu.cn

Tobias Reggelin
tobias.reggelin@ovgu.de

[1] Institute of Logistics and Material Handling Systems, Otto von Guericke University Magdeburg, Universitätsplatz, 2, 39106 Magdeburg, Germany

[2] Fraunhofer Institute for Factory Operation and Automation IFF, Sandtorstraße, 22, 39106 Magdeburg, Germany

[3] National-Local Joint Engineering Laboratory of Harbor Oil & Gas Storage and Transportation Technology/Zhejiang Provincial Key Laboratory of Petrochemical Pollution Control/School of Petrochemical Engineering and Environment, Zhejiang Ocean University, Zhoushan 316022, People's Republic of China

Introduction

As customer demands increase due to the globalization-driven intensification of competition, satisfying due dates of customer orders plays a more crucial role in today's business environment. A series of orders with due date violations is not desired and could make the manufacturer lose its earnings and reputation. Thus, effective scheduling that can minimize delays is in great demand in manufacturing systems.

Among several kinds of scheduling problems, the parallel machine scheduling problem (PMSP) is an essential problem since many real-world scheduling problems in various domains can be considered as a PMSP, such as in semiconductor wafer fabrication (Kim et al., 2010), offshore oil and gas industry (Abu-Marrul et al., 2021), and thin film transistor liquid crystal display (TFT LCD) manufacturing (Shin & Leon, 2004).

Due to the variety of orders flowing through a production system, different customer orders (jobs) require different

tools and preparation steps before they can be processed on a specific machine. In scheduling problems, such conditions can be modeled by considering, for instance, a family setups constraint. The family setups constraint postulates an additional time for preparing a machine when the family of the job to be processed is different from the previous one. The PMSPs with due dates and under family setups constraints are not trivial to solve, considering the following situation: In some cases, it is beneficial to process jobs from the same family in succession to reduce the number of occurrences of the setup time; in other cases, it is also necessary to switch to another family because there are jobs from that family that are reaching their due date.

Hence, the PMSP addressed in this paper takes family setups into consideration and takes minimizing total tardiness as the objective function. The total tardiness is defined as the cumulative delay of jobs. A mathematical definition is provided in section "Problem formulation". Moreover, the machines considered in this problem are uniform, which means that some machines process operations on jobs uniformly faster in comparison to others. This problem is NP-hard, since a simpler problem without consideration of different machine speeds and family setups has been proven to be NP-hard (Biskup et al., 2008).

As an attractive scheduling problem from both industrial and academical point of view, PMSP under diverse constraints has been intensively researched over the past decades. Of all the proposed solution methods, dispatching rules and metaheuristics are the most widely adopted. The mechanism of dispatching rules is to assign a priority to each job at each decision point and then select the job with the highest priority. The algorithm for the assignment of priorities is different for each specific rule. For example, shortest processing time (SPT) and earliest due date (EDD) are two classical dispatching rules that assign the highest priority to the job with a short processing time and earliest due date, respectively.

To address the scheduling problem with family setups constraint, Gavett (1965) develops the shortest setup time (SST) rule. This dispatching rule assigns the highest priority to the job that requires the shortest setup time. However, the SST rule might not be sufficient for the scheduling problem with due date- or flow time-related criteria since only the setup time is taken into account during scheduling. To minimize the mean tardiness of the scheduling problems with family setups, Wilbrecht and Prescott (1969) propose a modification of the SPT dispatching rule, in which the processing time of a job is substituted by the sum of the current setup time and its processing time when assigning priorities. Numerical experiments conducted by them show the superiority of this modification over SPT and SST in minimizing mean tardiness. Family-based dispatching rules are a particular category of family addressing dispatching rules, where a hierarchical approach is followed in prioritizing jobs (Pickardt & Branke, 2012). MAS_PAR is a specific approach of family-based dispatching rules that proposed by van der Zee (2015). The priority of each family is first assigned based on the number of jobs available in that family and the setup time of that family. After the family with the highest priority is selected, jobs in each family are then sequenced by the SPT rule. Although dispatching rules are characterized by their high time efficiency and low computational complexity (Rajendran & Holthaus, 1999), they are not able to search the solution space and only use their priority criterion for decision-making. Therefore, a high-quality solution cannot be guaranteed by dispatching rules.

Opposite to the dispatching rules, metaheuristics can realize higher solution quality, as they usually employ iterative, randomized search strategies for finding a solution. Balin (2011) adopts genetic algorithm (GA) to a non-identical PMSP for makespan minimizing. The results yielded by the GA significantly outperform those provided by a dispatching rule, namely the longest processing time (LPT). The particle swarm optimization (PSO) developed by Fang and Lin (2013) also indicates its superiority over EDD and weighted SPT (WSPT) in solving PMSP, where the objective functions comprising the total weighted job tardiness and the power cost minimizing. Metaheuristics have obtained outstanding results in addressing the PMSP with family setups. Cochran et al. (2003) present a GA with multi-population for identical PMSP under release dates and family setups constraints in order to minimize makespan and total weighted tardiness. GA is also utilized as the basic algorithm in an integrated metaheuristic developed by Zeidi and MohammadHosseini (2015), in which the simulated annealing (SA) method is also in addition applied as a local search procedure for minimizing the total cost of tardiness as well as earliness. Lee et al. (2010) propose a restricted SA approach on an identical PMSP with sequence-dependent setup times so that the maximum lateness is minimized. The same scheduling problem is also solved by Ying and Cheng (2010) through an iterated greedy (IG) approach. More recently, Báez et al. (2019) propose a hybrid metaheuristic algorithm that combines a greedy randomized adaptive search procedure (GRASP) and variable neighborhood search (VNS), which obtains promising results for minimizing the total completion time on identical parallel machines with sequence-dependent setup times. Moreover, several researchers apply metaheuristics to other variants of the PMSP with different constraints. Anghinolfi and Paolucci (2007) propose a hybrid metaheuristic method for minimizing the total tardiness of a PMSP, in which features of tabu search (TS), SA and VNS are integrated.

To achieve a higher computation efficiency, Afzalirad and Shafipour (2018) propose a genetic algorithm with a heuristic procedure (HGA) for addressing an unrelated PMSP with machine eligibility restrictions. Comparisons with a pure GA demonstrate the superiority of the proposed HGA. Armen-

tano and Yamashita (2000) propose two variants of TS to minimize the mean tardiness on identical parallel machines. The first variant substitutes the original short-term memory with a long-term memory, which can store the frequency of the moves executed throughout the search. The second variant makes use of influential moves. The improvements are indicated by large-scale instances of up to 10 machines and 150 jobs. In order to solve the dynamic scheduling problem, where all orders to be scheduled are impossible to be known in advance, Rolf et al. (2020) propose a GA-based method, which is capable to assign several given dispatching rules at any time point during the scheduling process. The combination of dispatching rules provided by this GA method can outperform any of the composite dispatching rule in solving the addressed hybrid flow shop scheduling problem. However, the drawback of metaheuristics is equally obvious since numerous iterations are required for evaluating and selecting an appropriate solution among several candidate solutions, which is a time-consuming process when dealing with large-scale instances.

In recent years, applying reinforcement learning (RL) to solve scheduling problems has attracted increasing attention (Kayhan & Yildiz, 2021). As one of the most promising branches of machine learning, RL demonstrates a powerful ability for decision-making in the context of complex and dynamic problems, such as in playing Go (Silver et al., 2017) or video games (Vinyals et al., 2019). The basic idea of RL is to model a problem as a Markov decision process (MDP), where an agent is trained to determine the optimal action at each decision point (Sutton & Barto, 2018). Although the training process may also be time-consuming, a well-trained agent can rapidly yield a remarkable result in other untrained situations. With the characteristics mentioned above, many researchers model the PMSP as MDP and then employ RL to solve it. Yuan et al. (2013) and Yuan et al. (2016) propose an agent trained by the Q-Learning algorithm to solve a dynamic PMSP and a variant with random breakdowns, respectively. The agent can select the most proper dispatching rule over three different rules, namely SPT, EDD and first in first out (FIFO). Moreover, the agent can also adjust the election percentage of the three rules depending on different objectives. Zhang et al. (2007) apply the Q-Learning algorithm on a dynamic PMSP with family setups and machine-job qualification constraints, in which five dispatching rules are utilized as actions of the agent to minimize the mean weighted tardiness. The agent trained by the proposed algorithm achieves a remarkable performance improvement over any of these five scheduling rules. Guo et al. (2020) train the agent with a multi-stage Q-Learning algorithm to balance three different objectives for a non-identical PMSP, which are specifically minimizing the number of tardy jobs, minimizing maximum tardiness and minimizing mean waiting time. The agent can select the most appropriate one based on the assessment of the state from three given scheduling rules, namely SPT, FIFO and minimum slack time (MST). Through a proper selection among these three rules, the proposed agent outperforms dispatching rules and a comparative agent trained by single-objective Q-Learning method. Zhang et al. (2012) address an unrelated PMSP under new jobs arrival constraint with R-Learning, which is an average-reward RL method. Minimizing mean weighted tardiness is selected to be the objective function. The agent can select four dispatching rules according to eight indicators of the production system. Computational experiments demonstrate that the proposed method can obtain better results than the four compared dispatching rules on every instances. Zhang et al. (2011) model a semiconductor test scheduling problem into an unrelated PMSP with multiple resource constraints. SARSA algorithm is utilized to minimize the total weighted unsatisfied demand in the scheduling horizon, in which five heuristics are derived as the actions. The superiority of the proposed algorithm is verified by a comparison with individuals and the Industrial Method (IM), which is an empirical heuristic used at that time. The previously mentioned traditional RL algorithm, such as Q-learning and SARSA, is characterized by the application of a lookup Q table, where the estimated Q function value of each state-action pair is stored. However, the number of state-action pairs in a real-world manufacturing environment might be so enormous that even constructing a huge Q table to store all the state-action pairs is almost impossible.

This limitation of traditional RL algorithms is overcome by deep learning techniques, i.e., deep neural networks (DNNs). RL algorithms employing DNNs are summarized under the term Deep Reinforcement Learning (DRL). In DRL, DNNs serve as a function approximator. Zhang et al. (2020) model the electric vehicles (EVs) charging scheduling problem based on the PMSP and the objective function aims to minimize the total time that EVs spend on charging. They adopt the deep Q-learning (DQL) algorithm, in which the Q table is substituted by a DNN called deep Q network (DQN). DQN is utilized here as a fully-connected neural network, also known as multilayer perceptron (MLP). Their experimental results demonstrate that the proposed agent trained by DQL can significantly reduce the time consumed for EV charging relative to the baseline. Zhou et al. (2020) investigate the usefulness of the DRL approach in solving PMSP. In particular, they apply the DQL algorithm on an unrelated parallel machine to minimize the maximum completion time of all jobs. A recent literature that utilized an RL-based approach to solve PMSP with family setups is proposed by Paeng et al. (2021). The representation of actions is defined as a tuple of a job family and a machine setup status, where machine setup status refers to the family of the job that previously processed by the machine. After executing an action, only the family of the job to be selected can be determined, and the selection of the specific job relies on a dispatch-

ing rule. Rodríguez et al. (2022) propose a multi-agent DRL approach on an identical parallel machine model to handle the uncertainty caused by multiple machine failures. Each agent is responsible for monitoring the state of a single machine, and it can trigger maintenance action to avoid any failures that might lead the machine to breakdown. This multi-agent method significantly reduces the breakdown times and prevents failures with respect to the traditional methods that are taken into comparison. DRL method is also widely used in other scheduling problems. Lang et al. (2020) present a multi-agent DQN approach for solving a flexible job shop scheduling problem with integrated process planning. DQN is used to train two agents. One agent is responsible for the selection of a predefined process plan before releasing a job. A process plan contains a set of operations, which a job must undergo, before being completed. A second agent is responsible for the dispatching of jobs to machines. DQN outperforms the GRASP metaheuristic in terms of minimizing the makespan and total tardiness. During inference, the DQN approach is able to beat the GRASP algorithm in terms of computational time. The authors further show that the agent is able to compute solutions in similar quality on problems that the agent have not seen during the training. Liu et al. (2020) propose a DRL approach based on the actor-critic mechanism to solve the job shop scheduling problem. The actor and the critic are both represented by a specific category of DNN, a convolutional neural network (CNN). The task of the CNN for actor and critic is to generate a probability distribution of each candidate action (which is two dispatching rules) and estimate the value of the current state, respectively. Luo (2020) develops an agent that is trained through a variant of DQN, which can select the optimal dispatching rule among six given rules depending on the current situation. This agent is utilized to minimize the total tardiness in a dynamic job shop scheduling environment, in which the arrival of new jobs is considered. The results show the superiority of the proposed agent over the composite dispatching rules and an agent that is trained by traditional Q-Learning. Table 1 summarizes the aforementioned work on utilizing RL for solving scheduling problems and the differences between these and our work. The problem properties of the works are indicated by the Graham notations (Graham et al., 1979), which is given in Table 2.

Without loss of generality, most of the RL-based approaches mentioned above employ dispatching rules for job selection, which is common according to the survey of Wang et al. (2021). In real-world scheduling, the performance can be impacted by several factors, but dispatching rules are restricted to specific criteria for priority assignment. In contrast, DNN provides the potential to incorporate several attributes of jobs (i.e., processing time, due date and family) and machines (i.e., the family of the previous job and the processing speed) to calculate a more complex and comprehensive priority. Moreover, the agent training process is time-consuming and cumbersome, especially when the scale of the instances used for training is large. If the agent could be trained on a small-scale instance and then applied to solve large-scale instances in real production without retraining, it would be a highly desirable property for the manufacturing industries.

With the motivations above, we propose a DRL approach with a novel representation of states and actions for addressing minimizing the total tardiness in PMSP with family setups in this paper. The proposed representation of states and actions enables the agent to generate a comprehensive way to calculate job priorities and select jobs directly depending on them. Furthermore, this process of job prioritization and selection is independent of the number of jobs, therefore our agent can handle arbitrary scaled instances of PMSP with family setups. The contributions of this paper can be listed as follows:

1. We implement a novel, variable-length state representation, where the number of rows of the state matrix is equal to the number of available jobs, and the number of columns is precisely the number of state features. Meanwhile, the index of each job is taken as the action representation, so the number of rows of the action matrix is also equal to the number of available jobs. This approach is first described by Lang et al. (2021), but has not yet been implemented and tested on a practical use case. The number of available jobs at time t is denoted as n_t and the number of state features is referred to as x, so the current state is a matrix of $n_t \times x$. This matrix is computed by the agent to a matrix of $n_t \times 1$, where each element denotes the priority of the corresponding job. This priority vector is converted into a probability distribution by the SoftMax function, according to which the job is selected. The job with higher priority has a higher probability of being selected, and the action to be executed is the index of the selected job. After a job is selected, the corresponding row is deleted to block the illegal action.
2. In order to handle the variable-length matrix of states and actions, the agent is represented by a recurrent neural network (RNN) (Elman, 1990), specifically, gated recurrent unit (GRU) (Cho et al., 2014), which is one of the most established models for processing variable-length sequence.
3. We apply the Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017) as the basic algorithm to train the agent. To accelerate the training process while remaining the policy unchanged, we design a two-stage training strategy under the inspiration of curriculum learning (Bengio et al., 2009). In the first training stage, we design the reward function according to whether the

Table 1 Comparison of RL methods for solving scheduling problems

Work	Problem type	Learning algorithm	State size	Actions
Yuan et al. (2013)	$P_m//L_{max} + \sum U_i$	Q-learning	4 for all instance scales	3 dispatching rules
Yuan et al. (2016)	$P_m/bkdwn/L_{max} + \sum U_i$	Q-learning	3 for all instance scales	3 dispatching rules
Zhang et al. (2007)	$R_m/S_{jk}/\frac{1}{n}\sum w_i T_i$	Q-Learning	Fixed by the instance scale	5 dispatching rules
Guo et al. (2020)	$Q_m//T_{max} + \sum U_i + \sum W_i$	Q-Learning	5 for all instance scales	3 dispatching rules
Zhang et al. (2012)	$R_m//\sum w_i T_i$	R-Learning	Fixed by the instance scale	4 dispatching rules together with select no job
Zhang et al. (2011)	$R_m/S_{jk}prec/TC$	SARSA	Fixed by the instance scale	5 dispatching rules
Zhou et al. (2020)	$R_m//C_{max}$	Deep Q-Learning with fully-connected neural network	Fixed by the instance scale	Index of services
Paeng et al. (2021)	$R_m/S_{jk}/\sum T_i$	Deep Q-Learning with fully-connected neural network	Fixed by the instance scale	Index of families, job selection inside the family through SPT
Rodríguez et al. (2022)	Predictive Maintenance	Proximal Policy Optimization with fully-connected neural network	Fixed by the instance scale	Index of all technicians (Jobs). Selection of invalid jobs is unavoidable
Lang et al. (2020)	$J/r_j/C_{max} + \sum T_i$	Deep Q-Learning with fully-connected neural network and recurrent neural network	Fixed by the instance scale	4 given operation sequence
Liu et al. (2020)	$J/r_j bkdwn/C_{max}$	Deep Deterministic Policy Gradient with convolutional neural network	Fixed by the instance scale	Combination of dispatching rules
Luo (2020)	$J/r_j/\sum T_i$	Deep Q-Learning with fully-connected neural network	7 for all instance scales	6 dispatching rules
Ours	$Q_m/S/\sum T_i$	Proximal Policy Optimization with with recurrent neural network	Variable during scheduling, corresponding to the number of available jobs	Index of all available jobs, thus job can be selected directly

Table 2 Graham notations

Machine environment		Optimality criteria	
Acronym	Meaning	Acronym	Meaning
P_m	Identical parallel machine	C_{max}	Makespan
Q_m	Uniform paralllel machine	L_{max}	Maximum lateness
R_m	Unrelated parallel machine	T_{max}	Maximum tardiness
		$\sum T_i$	Total tardiness
		$\sum U_i$	Total number of tardy jobs
		$\sum w_i T_i$	Total weighted tardiness
		$\frac{1}{n}\sum w_i T_i$	Mean weighted tardiness
		TC	Total cost

Job characteristics	
Acronym	Meaning
r_j	Release date
$prec$	Precedence constraints
S_{jk}	Sequence dependent setup times
$bkdwn$	Machine breakdown

job being assigned is of the same family as the previous job. The reward function of the second training stage is directly the negative of the objective function.

4. Numerical experiments on large scale instances with different parameter configurations demonstrate the robust generalization capability of the trained agent. The training process is also provided to validate the effectiveness of the proposed two-stage training strategy. Comparison with three dispatching rules and two metaheuristics further confirms the superiority of the proposed DRL approach.

The remainder of this paper is organized as follows. Section "Backgrounds" presents the background of RNN and GRU as well as RL and PPO. The PMSP considered in this paper is formulated in section "Problem formulation". The details of the proposed approach are established in section "Proposed approach". Section "Numerical experiments" provides the training process of the RL agent and its performance comparisons with considered alternatives with experiments. Finally, conclusions are drawn in section "Conclusion and future work".

Backgrounds

RNN and GRU

RNN is a particular form of artificial neural networks that is often used for problems with sequential data, such as music synthesis (Sigtia et al., 2014) and natural language processing (Yin et al., 2017). The input of such problems can be formalized as a sequence of vectors $(x_1, \ldots x_t, \ldots x_N)$ where x_t is the input at the time point t. According to this input sequence, the RNN generates a sequence of hidden states $(h_1, \ldots h_t, \ldots h_N)$ where h_t is defined by the following equation:

$$h_t = g(Wx_t + Uh_{t-1} + b) \tag{1}$$

where x_t is an m-dimensional external input vector at time t, the h_t and h_{t-1} denote the n-dimensional hidden states at time t and $t-1$, g is the activation function inside the RNN unit which is hyperbolic tangent function in our networks. W, U and b are the learnable parameters of the RNN. W and U are weight matrices, b is the bias vector with the shapes $n \times m$, $n \times n$ and $n \times 1$, respectively. By iterating through the input sequence to compute the hidden states, the relationships among sequential data can be captured. The architecture of the RNN unit is shown in Fig. 1a. Figure 1b shows the structure of the RNN unit being unfolded along the time steps. In order to show the transformation of hidden states, the RNN networks utilized in this paper are demonstrated in the unfolded structure.

An RNN comprising only hidden states is called simple RNN, and the gradient for updating the network can be calculated by backpropagation through time (Werbos, 1990). However, it is almost impossible for the simple RNN to capture long-term dependencies because the gradients tend to decay or blow up exponentially through time, which makes the learning process extremely unstable (Bengio et al., 1994).

To enable RNNs the storage of information over a long-range sequence, several variants with memory cells have been developed. Memory cells can be accessed (such as be written,

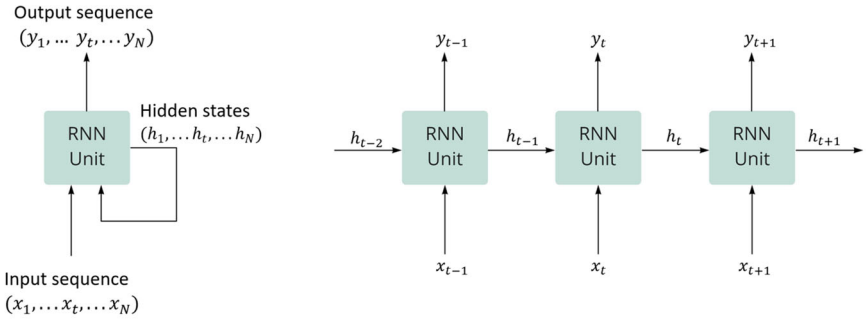


Fig. 1 The architecture of the RNN unit

(a) General architecture of the RNN unit

(b) The architecture of the RNN that are unfolded along the time steps

be read, be updated and so on) through their corresponding gate. Among those variants, GRU is a particular architecture whose memory cell contains a reset gate z_t and an update gate r_t. These two gates are presented by the following equations:

$$\begin{aligned} z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\ r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \end{aligned} \tag{2}$$

where σ is the sigmoid activation function, W_z, U_z and b_z are the weight matrices and the bias vector of the reset gate, while W_r, U_r and b_r correspond to the same for the update gate. With the integration of the memory cells, the GRU model can be formulized in the form:

$$\begin{aligned} h_t &= (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h_t} \\ \tilde{h_t} &= g(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h) \end{aligned} \tag{3}$$

where $\circ$ denotes the elementwise multiplication and h_t refers to the temporary output of the memory cell.

RL and PPO

RL approaches aim to teach an agent a policy of executing actions in order to maximize the cumulated reward that the agent receives from the interaction with its environment. This environment is generally modeled as an MDP. An MDP can be represented by a four-tuple representation (S, A, p, R), where S is a set of all possible states, A is a set of all actions that the agent can take, p is the state-transition function which provides the probability of a transition between every pair of states under each action, R is a reward function which generates a real value to each state-action pair. At each decision time point t, the agent observes the current state of the environment $s_t \in S$ and then conducts the action $a_t \in A$ according to the policy π, which is a mapping from states to actions. In response to the action, the environment changes to the next state s_{t+1} with transition probability $p(s_{t+1} \mid s_t, a_t)$ and receives an immediate reward calculated by $R(s_t, a_t, s_{t+1})$. For a stationary policy π, the expected cumulative reward upon taking an action a_t in state s_t is denoted as $Q_\pi(s_t, a_t)$, which is also known as the Q-function. The Q-function is defined as follows:

$$\begin{aligned} Q_\pi(s, a) = \mathbb{E}\Big[& r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \mid s = s_t, \\ & a = a_t, \pi \Big] \end{aligned} \tag{4}$$

where $0 < \gamma < 1$ is the discount factor that measures the relative importance between short-term and long-term rewards. The expected cumulative reward of this state can be estimated with the following equation:

$$V_\pi(s) = \sum_a \pi(s, a)\, Q_\pi(s, a) \tag{5}$$

where $\pi(s, a)$ is the probability that the policy π performs action a given state s. $V_\pi(s)$ is called the state-value function, and it can also be defined as the expected value of the following expression:

$$V_\pi(s) = \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \mid s = s_t, \pi \right] \tag{6}$$

Therefore, the optimal policy π^* maximizing the expected cumulative reward among all possible states can be defined as:

$$\pi^* = arg\max_\pi \mathbb{E}\left[V_\pi(s) \mid \pi \right] \tag{7}$$

In DRL methods, the policy π is represented by a DNN that contains a set of differentiable parameters θ. These parameters are updated as $\theta \leftarrow \theta_{old} + \eta \nabla E_{\pi_{\theta_{old}}}(V(S_1))$, where η is the learning rate and $\nabla E_{\pi_{\theta_{old}}}(V(S_1))$ is the derivative of the expected cumulative reward that can be obtained by the policy from the first state based on the old parameters. However, the updating process is quite unstable since it always encounters destructively large policy updates (Schulman et al., 2017). To prevent the updated policy π_θ from deviating significantly from the old policy $\pi_{\theta_{old}}$, the algorithm Trust Region Policy Optimization (TRPO) (Schulman

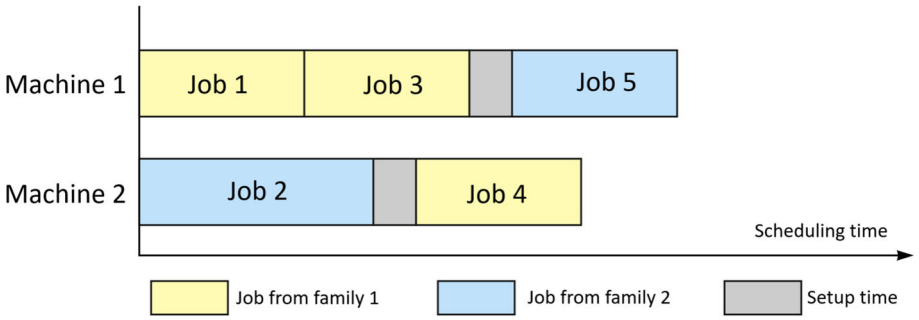


Fig. 2 An example of PMSP with family setups containing two machines and five jobs from two different families. The yellow and sky colors indicate the two families to which jobs belong to

et al., 2015) is proposed, where the difference between π_θ and $\pi_{\theta_{old}}$ is measured through the Kullback–Leibler (KL) divergence. However, the computation of the KL-divergence is computationally expensive. Against this background, Schulman et al. (2017) propose an extension of TRPO called PPO, where the policy updates are limited by a clipping function instead of the KL-divergence. Let $r_t(\theta)$ denote the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, where $\pi_\theta(a_t \mid s_t)$ and $\pi_{\theta_{old}}(a_t \mid s_t)$ refer to the probability of $\pi_{\theta_{old}}$ and π_θ for taking action a_t under the state s_t, respectively. The update rule of the PPO algorithm is shown in the following:

$$\max_{\theta} \mathop{\mathbb{E}}_{s,a\ \pi_{\theta_{old}}} \left[\mathbb{L}\left(s_t, a_t, \pi_{\theta_{old}}, \theta\right)\right] \tag{8}$$

and $\mathbb{L}$ is given by:

$$\begin{aligned}\mathbb{L}\left(s_t, a_t, \pi_{\theta_{old}}, \theta\right) = \min(&r_t(\theta)\, A^{\pi_{\theta_{old}}}(s_t, a_t),\\ &clip\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right) A^{\pi_{\theta_{old}}}(s_t, a_t))\end{aligned} \tag{9}$$

where $A^{\pi_{\theta_{old}}}(s_t, a_t) = Q_{\pi_{\theta_{old}}}(s_t, a_t) - V_{\pi_{\theta_{old}}}(s_t)$ is the advantage function that evaluates the advantage of a_t over all other possible actions and $0 < \epsilon < 1$ is the clip parameter. When the advantage function $A^{\pi_{\theta_{old}}}(s_t, a_t)$ is positive, the probability ratio $r_t(\theta)$ is limited to $1+\epsilon$; when the advantage function $A^{\pi_{\theta_{old}}}(s_t, a_t)$ is negative, the probability ratio $r_t(\theta)$ is limited to $1-\epsilon$. The policy update is therefore restricted and a stable improvement is also guaranteed.

Problem formulation

The PMSP with family setups constraint considered in this paper can be defined as follows. There are n independent jobs $J = \{J_1, J_2, \ldots, J_n\}$ and m uniform parallel machines $M = \{M_1, M_2, \ldots, M_m\}$. p_j and d_j refer to the processing time and the due date of the jth job J_j, respectively. Each job belongs to one of N_F families from the set $F = \{1, 2, \ldots, N_F\}$, and the family of the Jth job J_j is denoted as f_{J_j}. Each time the ith machine becomes idle, a job needs to be selected from all unassigned jobs and assigned to this machine. The family of the job that this machine just processed is the setup state of this machine and is represented as f_{M_i}. If the selected job belongs to a different family, then a positive and constant setup time S must be additionally considered, which is equal to 10 in the problem addressed in this paper. In addition, each machine M_i is characterized by an individual processing speed v_i, which acts as a coefficient of the processing time of a job $(1/v_i) \times p_j$ and thus, adjusts the processing time of jobs upwards or downwards. The objective function is to minimize the total tardiness TT, which is defined as followed:

$$TT = \sum_{j=1}^{n} \max\left(0,\ C_j - d_j\right) \tag{10}$$

where C_j is the completion time of J_j. Intuitively, it is therefore desired that each job J_j is completed before its due date d_j.

An exemplary schedule of a PMSP with family setups constraint is given in Fig. 2, which contains two machines and five jobs from two different families. There is no setup time between Job 1 and Job 3 as they share the same family. However, since Job 2 and Job 4 (as well as Job 3 and Job 5) belong to different families, an additional setup time must be considered.

To simplify the problem at hand, several predefined constraints should be satisfied as follows:

1. Each machine can immediately start to process a job after the setup is finished.
2. Each machine can process only one job at a time, and each job can be processed on only one machine.
3. There is no moving time for the jobs.
4. All the jobs are available at the beginning.

Then we mathematically describe the PSMP addressed in this paper based on the mixed integer formulation developed by Avalos-Rosales et al. (2015). The notations required for modeling are listed below.

1. *Parameters:*
   - n: total number of jobs

- m: total number of machines
- j, g: index of jobs, $j, g = 1, 2, \ldots, n$
- i, k: index of machines, $i = 1, 2, \ldots, m$
- J: the set of jobs
- M: the set of parallel machines
- p_j: the processing time of job J_j
- d_j: the due date of job J_j
- v_i: the processing speed of machine M_i
- S_{jg}: setup time for job J_g when it immediately follows job J_j (equal to 0 if J_g and J_j come from same family, equal to 10 otherwise)
- G: a sufficient large constant

2. *Decision variables:*

- C_j: the completion time of Job J_j
- X_{ijg}: 1 if J_j is a predecessor of J_g on machine M_i, 0 otherwise
- Y_{ij}: 1 if job J_j is assigned to machine M_i, 0 otherwise

Moreover, to support the problem formulation, a dummy job is introduced at the start and end on each machine and J' denotes the set of jobs includes J and the dummy jobs. The processing times and setup times related to the dummy jobs are considered 0. Our model can be therefore stated as:

Objective function:

$$Minimize \sum_{j=1}^{n} \max\left(0,\ C_j - d_j\right) \tag{11}$$

Subject to:

$$\sum_{i \in M} Y_{ij} = 1, \quad \forall j \in J \tag{12}$$

$$Y_{ig} = \sum_{j \in J', j \neq g} X_{ijg}, \quad \forall g \in J,\ \forall i \in M \tag{13}$$

$$Y_{ij} = \sum_{g \in J', g \neq j} X_{ijg}, \quad \forall j \in J,\ \forall i \in M \tag{14}$$

$$\sum_{j \in J} X_{i0j} \leq 1, \quad \forall i \in M \tag{15}$$

$$C_g - C_j + V(1 - X_{ijg}) \geq S_{jg} + \frac{p_g}{v_i}$$
$$\forall j \in J',\ \forall g \in J,\ j \neq g,\ \forall i \in M \tag{16}$$

$$C_0 = 0, \tag{17}$$

$$X_{ijg} \in \{0, 1\}, \quad \forall j \in J',\ \forall g \in J',\ j \neq g,\ \forall i \in M \tag{18}$$

$$Y_{ij} \in \{0, 1\}, \quad \forall j \in J',\ \forall i \in M \tag{19}$$

$$C_j \geq 0, \quad \forall j \in J \tag{20}$$

Objective 11 minimizes the total tardiness of the solution. Constraint 12 imposes that each job is assigned to one and only one machine. Constraints 13 and 14 ensure that each job on the machine it is assigned to has one and only one predecessor and one successor, respectively. Constraint 15 establishes that at most, one job is scheduled as the first job on each machine. Constraint 16 forbids overlapping among the jobs with respect to family setups and machine speeds. Constraint 17 sets the completion time of the dummy job at the start on each machine to 0. Constraints 18, 19 and 20 define the domain of the variables.

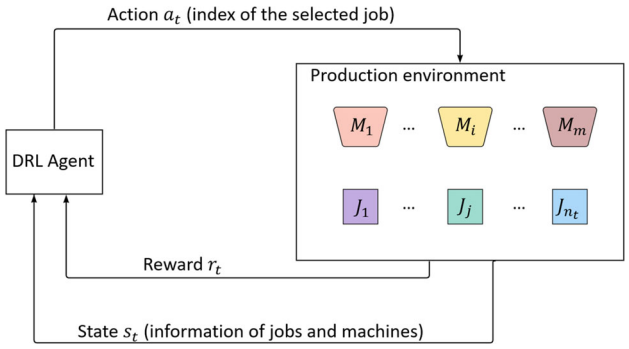


Fig. 3 The interaction mechanism between the agent and the production environment

Proposed approach

In order to employ DRL, we formulate the considered PMSP as MDP, which can be represented by a four-tuple representation (S, A, p, R) as described in section "Backgrounds". The interaction mechanism between the agent and the production environment is shown in Fig. 3. We first define the states and actions representation of the problem. For the representation of states and actions, we adapt a concept from a previous publication (Lang et al., 2021), which is characterized by a variable state and action space, thus being able to apply the agent's policy to a flexible amount of jobs. Thereafter, we describe our two-stage training strategy and the design of the reward function for each stage. Finally, we describe the PPO algorithm for the agent training as well as the DNN structure of the agent representation.

State representation

The state representation we propose is discrete and the state translates to the next at each decision time point, which is defined as every time a machine becomes idle. At each decision point, the priorities of all jobs are computed by the agent. According to these priorities, a job is selected and then allocated to the current idle machine. The overall scheduling framework is shown in Fig. 4.

In order to enable the agent to generate a comprehensive priority for each job, we design a two-dimensional state matrix whose number of rows is equal to the number of cur-

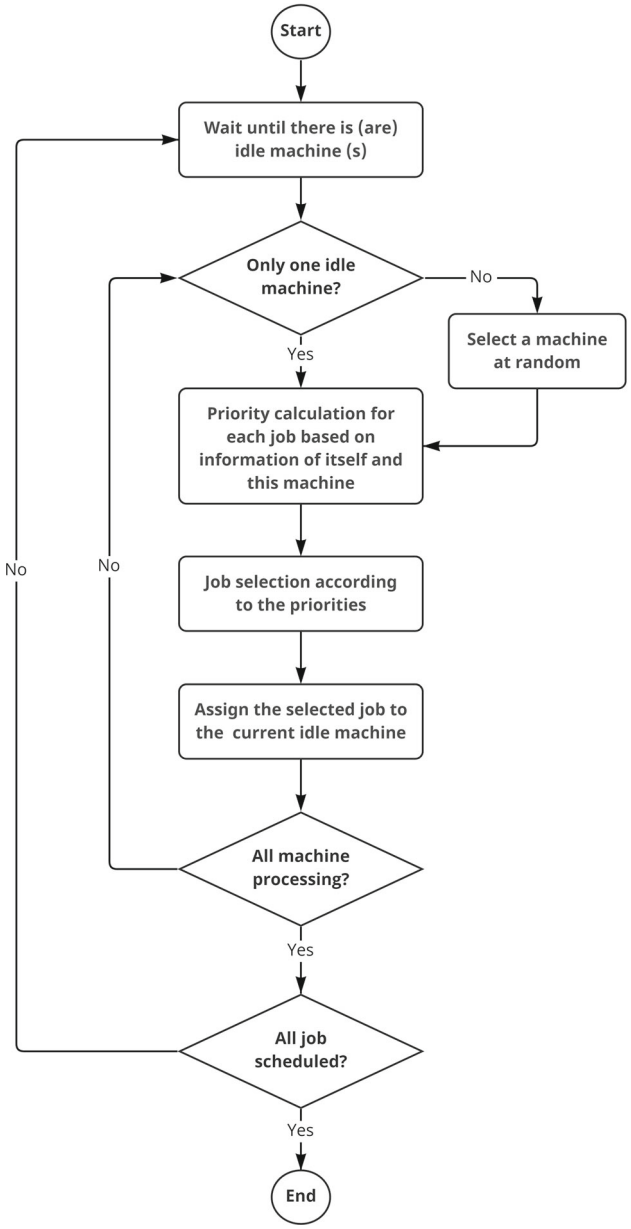


Fig. 4 Overall scheduling framework

rent available jobs and the number of columns is equal to the number of state features. The state features contain job features and machine features, where the job features consist of the processing time, the due date and the family of each job, while the machine features are the information of the current idle machine, which consists of the current time, the family of the previous processed job by this machine and the speed of this machine. All these features are listed in Table 3. Therefore, the agent can have access to all the information of all jobs and then conduct the priority calculation for each job with the consideration of the information of itself and that of other jobs as well as the information of the current idle machine.

Table 3 Job features and machine features in the state matrix

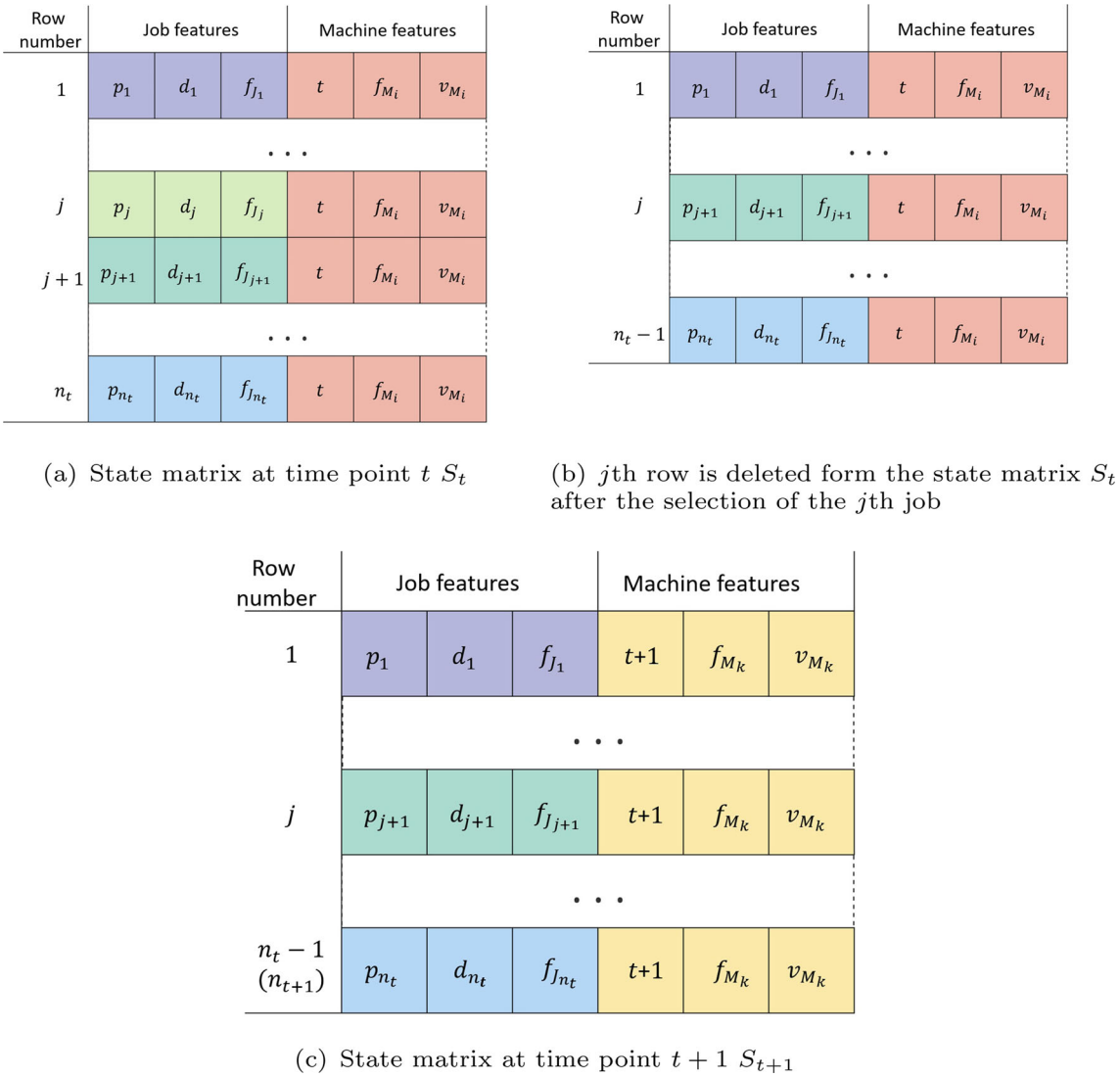
State features in the state matrix	
Job features	Processing time p_j
	Due date d_j
	Family f_{J_j}
Machine features	Current time t
	Family of the previous job f_{M_i}
	Speed of the machine v_i

The structure of the state matrix at decision time point t is illustrated in Fig. 5a. It is assumed that there are n_t jobs at time t, so the number of rows of the matrix is also n_t. In each row, the first three columns demonstrate the processing time, the due date, and the family of the jobs, respectively. The last three columns accommodate the time of the current decision point, the family of the previous processed job and the speed of the machine. Therefore, the data in the first three columns are different in each row, while the data in the last three columns are identical in each row.

A unique feature of our approach is that the length of the state matrix is variable. The majority of DRL-based solution techniques for selecting jobs that are described in the literature apply a mask (Tassel et al., 2021) on a state matrix of constant size in order to ensure that already processed jobs would not be selected again, so the shape of the state matrix can stay constant. However, the maximum number of jobs an agent can process is hence limited and if this limit is exceeded, the agent requires to be retrained. In our approach, in contrast, the row corresponding to the selected job is deleted from the state matrix after a job has been selected. Thus, our approach allows us to deploy a trained agent for sequencing a variable number of jobs waiting in a buffer to be processed. The concept of the variable state matrix is illustrated in Fig. 5. When the job J_j is selected and then assigned to the current idle machine M_i at time point t, the corresponding row is simultaneously deleted from the state matrix S_t as is demonstrated in Fig. 5b. This state matrix S_t will transfer to the state matrix S_{t+1} at the next decision point $t+1$, which is defined as the time when there is an idle machine in the environment again. This idle machine at the time point $t+1$ is assumed to be the kth machine, and its information together with the information of the remaining jobs are presented in the S_{t+1} that is given in Fig. 5c.

Action representation

To select the job directly, the action is the index of the available jobs. As it is shown in Fig. 5, the action at the time point t a_t is the index of the selected job j. Therefore, the size of the action space is equal to the number of the current avail-



(a) State matrix at time point t S_t

(b) jth row is deleted form the state matrix S_t after the selection of the jth job

(c) State matrix at time point $t+1$ S_{t+1}

Fig. 5 The changing process of state matrix when the jth job is selected at time point t ($a_t = j$). The first three columns in each state matrix are the job features, each row represents a different job and is indicated by a different color (purple, light green, dark green and blue). The last three columns are machine features, each row represents the same machine, indicated by the same color in all rows in the same matrix, and by another color in another matrix (red and yellow)

able jobs. To do so, we consider that the agent acts within a continuous action space. By this means, the agent has only a single output neuron. However, since the state matrix can be considered as a batch of states, the resulting batch of actions is an action vector whose length corresponds to the number of rows of the state matrix. Suppose there are n_t jobs that are unallocated at time t, then the size of the state matrix at this decision time point is $n_t \times 6$, because each job and all machines provide in total 6 attributes for describing a single job as the state. The agent forward propagates the matrix and thus computes a vector with n_t elements, where each element is the priority of the corresponding job. This priority vector will be passed to a SoftMax function in order to compute a probability distribution over the jobs that can be selected, where the job with the highest priority has the highest probability of being selected. Finally, the probability distribution samples the job to be selected by the idle machine. Same as the state matrix, the action space is reduced by one at the next decision point $t+1$, since the number of available jobs is n_{t+1}.

Reward design and the two-stage training strategy

Since the objective is to minimize the total tardiness, the invariance of the objective can be guaranteed if its negative is set directly to the reward function. However, using

the objective function as reward function results in sparse rewards, as reward for the agent is only provided at the end of an episode. Therefore, it is not possible for the agent to precisely determine the impact of selected actions, especially earlier actions, on the final objective. Thus, the agent might not be able to converge to a high-quality policy.

To address this issue, we develop a novel two-stage training strategy based on curriculum learning (Bengio et al., 2009). The basic idea of curriculum learning is to introduce an additional (or more) training stage with dense rewards before the agent is trained with the original sparse reward function. This training strategy achieves great success in handling the task with sparse rewards such as playing first-person shooting games (Wu & Tian, 2016) and training robotic arms (Zhou et al., 2021).

In our training strategy, the negative of the total tardiness represents the reward function of the second training stage. The design of the dense reward function of the first stage is based on the fact that a high-quality schedule will not be accompanied by a high number of setup changes, as each additional setup time can increase the probability of tardy jobs. Against this background, the reward function of the first training stage only considers whether the job being selected shares the same family as the previous job that has been processed on the same machine. If both jobs belong to the same family, i.e., no setup time is required, the agent obtains a positive reward. If the selected job has a different family than the previous one and at the same time, there are available jobs of the same family as the previous one, then the agent receives a negative reward. Otherwise, the reward is equal to zero. This procedure to calculate the reward at the first stage is given in Algorithm 1. Since for each decision point a job is assigned, this makes the reward function at the first stage dense and thus simplifies the convergence of the agent.

Algorithm 1 Reward calculation of the first training stage

Initialization: $s_t \leftarrow s_0, n_t \leftarrow n, reward \leftarrow 0$
while $n_t \neq 0$ **do**
  $a_t \leftarrow \pi_\theta(s_t)$
  $i \leftarrow a_t$
  if $f_i = f_M$ **then**
    $reward \leftarrow reward + 1$
  else
    for $j \leftarrow 0$ to n_t **do**
      if $f_j = f_M$ **then**
        $reward \leftarrow reward - 1$
        break
      end if
    end for
  end if
  $s_t \leftarrow s_{t+1}, n_t \leftarrow n_t - 1$
end while

The reward of the second stage is the total tardiness yielded by all jobs and is only provided at the end of each episode, i.e., when all jobs are scheduled. Moreover, if the total tardiness is equal to zero, the agent obtains an additional reward R, which is set to be 200 in this paper. The procedure to generate the reward of the second training stage is given in Algorithm 2. It is worth noting that although the tardiness is calculated during the scheduling process, the agent can only receive the total tardiness at the end of the scheduling process, since the value of the $Reward$ is assigned after the while loop. Before the second training stage, the agent is already able to achieve a suboptimal solution with the guidance of the first training stage. Therefore, compared to agents with random initialization, agents after the first training stage can converge and find the optimal solution more efficiently, instead of wasting time on aimless exploration.

Figure 6 demonstrates this two-stage training process with an exemplary instance containing only four jobs and one machine. In the first training stage, the relationship between every two sequential jobs is forwarded to the agent as the reward. It is clear that during a complete scheduling process, the agent can be rewarded multiple times. In our problem, one complete scheduling process is called one episode of training for the agent and the first training stage will last for a specific number of episodes, which is a hyperparameter defined in advance. After the first training stage, the agent is introduced into the second training stage, where the reward is only provided at the end of each episode, i.e., when a scheduling process is done. The number of episodes for the second training stage is also a pre-defined hyperparameter.

Algorithm 2 Reward calculation of the second training stage

Initialization: $s_t \leftarrow s_0$, $n_t \leftarrow n$, $reward \leftarrow 0$, $Tardiness \leftarrow 0$, $Total\ Tardiness \leftarrow 0$
while $n_t \neq 0$ **do**
  $a_t \leftarrow \pi_\theta(s_t)$
  $i \leftarrow a_t$
  if $t + p_i/v_M > d_i$ **then**
    $Tardiness \leftarrow (t + p_i/v_M - d_i)$
    $Total\ Tardiness \leftarrow Total\ Tardiness + Tardiness$
  end if
  $Tardiness \leftarrow 0$
  $s_t \leftarrow s_{t+1}, n_t \leftarrow n_t - 1$
end while
$Reward \leftarrow -Total\ Tardiness$
if $Total\ Tardiness = 0$ **then**
  $Reward \leftarrow Reward + R$
end if

Training algorithm and network structures of the proposed agent

In this paper, we adopt the PPO algorithm for updating the proposed agent. Since the PPO algorithm trains the agent through a batch of (S, A, R) tuples, the state matrices in this batch must be of the same size. To satisfy this, we first find

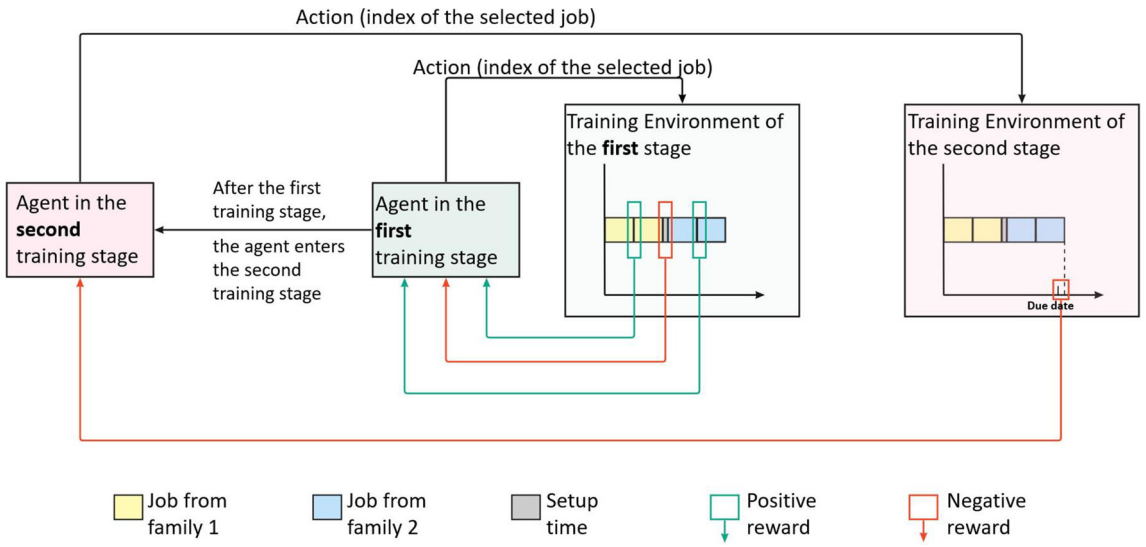


Fig. 6 The process of the two-stage training strategy

the matrix with the largest number of rows in a batch, and then apply zero padding to the other matrices in that batch so that they all have the same size as the largest one. Moreover, an actor DNN and a critic DNN must be built for the implementing the PPO algorithm. The actor DNN learns the decision policy, i.e., which action to execute given a specific state of the environment. More precisely, the actor takes the information required as input to calculate the priority of each job and then generates a probability distribution over all jobs in order to randomly select a job for the idle machine. Intuitively, jobs with a higher priority are more likely to be selected. The critic DNN approximates the state value function and thus estimates the cumulative future reward starting from a given state. The state value serves as the training signal for the actor DNN. The inputs of the critic DNN are the same as those for the actor DNN. The interaction between the actor-critic agent and its environment is illustrated in Fig. 7.

In DRL, agents are usually represented by a fully-connected DNN, which is also known as the multilayer perceptron (MLP). An MLP, however, is not able to analyze sequential relationships in input data streams. Considering that the agent calculates n priorities for n jobs waiting in a buffer before a job is selected to be produced, it seems evident that the agent should take into account its prioritization decisions of all other waiting jobs in the queue, when prioritizing one specific job. An MLP is hardly able to analyze such sequential relationships between data. To this end, RNNs are employed in the construction of both the actor and the critic DNN, which are dominant in sequence transduction and quite suited for handling variable-length sequences. To be more specific, the RNN used in the actor and critic architecture is the GRU model.

The actor model can be divided into a GRU part and an MLP part, where the GRU part consists of the input layer and a recurrent hidden layer while the MLP part consists of a fully-connected hidden layer and the output layer. The architecture of the actor model and the way it calculates the priorities of the jobs are demonstrated in Fig. 8. When calculating the priority of the jth job over all n jobs, the data of the jth row in the input matrix is firstly fed into the actor model through the input layer, and subsequently encoded by the recurrent hidden layer and then decoded by the fully-connected hidden layer, the priority of the ith job is finally given by the output layer.

What is worth to be noted is that the output of the recurrent layer is called the hidden state and the jth hidden state is denoted as h_j. The hidden state is not only used as input to the fully-connected layer, but also as input to the recurrent layer itself when calculating the priority of the $(j+1)$th job. It is self-evident that the $(j-1)$th hidden state $h_{(j-1)}$ is also taken into account by generating the jth hidden state h_j. The hidden state h_j can be considered as a summary of all the information from the 0th job to the jth job and a long-term dependence among the information sequences of jobs is therefore captured. Figure 9 presents an example of the GRU calculating priorities for an instance containing two jobs and one machine. For a clearer presentation, only the job features are shown in this example figure. It can be seen that at the beginning of the scheduling process, the memory cell of the GRU is in the initial state. Then the GRU summarizes the information of the first job into the first hidden state h_1. This hidden state, i.e., the summarized information of the first job, is further processed by the MLP into the priority of the first job, and simultaneously it is written into the memory cell. When calculating the priority of the second job, the

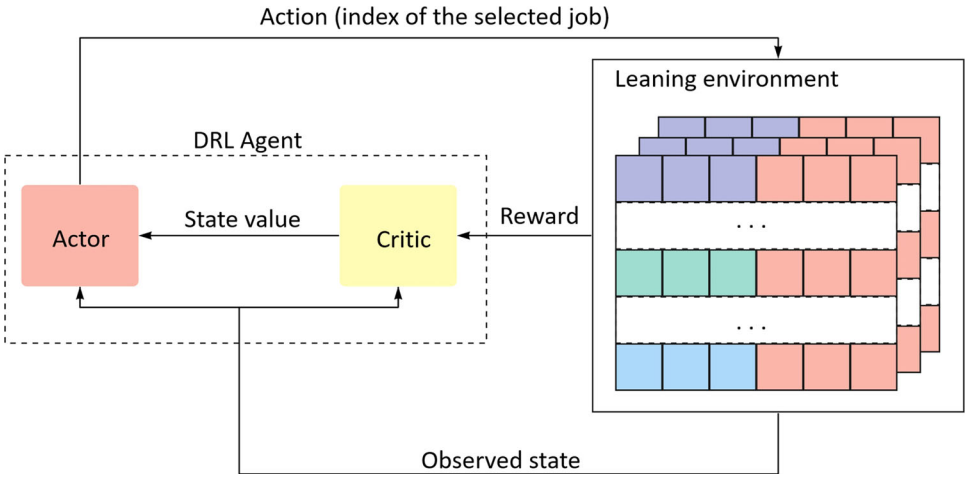


Fig. 7 The cooperation mechanism between DRL-Agent and its learning environment

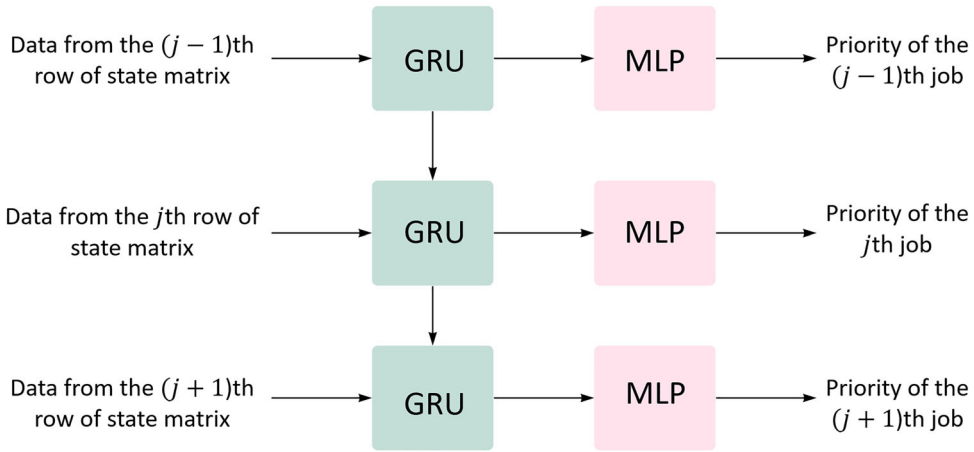


Fig. 8 Architecture of actor network by utilizing one-directional GRU

summarized information of the first job will be read from the memory cell by GRU and taken into account in the priority calculation. Then the information in the memory cell will be updated to the second hidden state h_2, which could be considered as the summarized information of the first and second jobs.

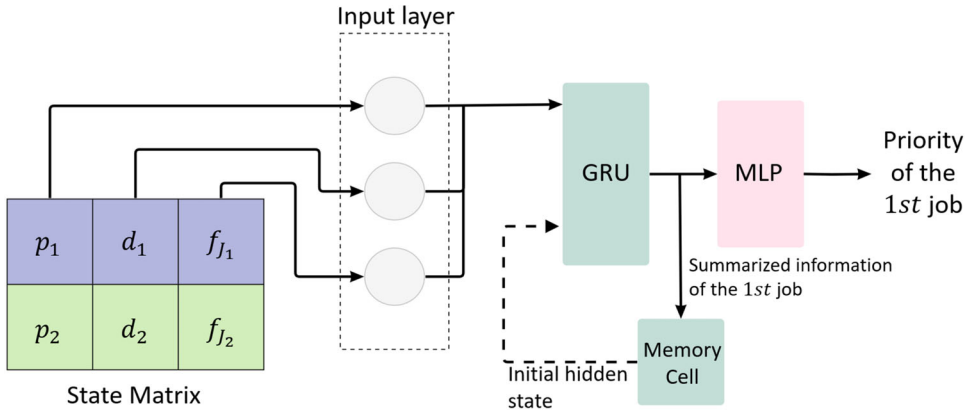
It is equally noticeable that the GRU processes the state matrix in a manner that is independent of the number of rows, which is thus independent of the number of jobs. When the number of jobs increases from two in the example to n in the practical application, only the number of times that the memory cell is updated increases correspondingly, while the architecture of the GRU remains unchanged. This characteristic can allow a trained agent to solve instances of arbitrary scale without time-consuming retraining.

However, only the information of the jobs before the jth job is involved in the calculation of the jth priority, which can lead to myopic behavior of the agent, since the agent only knows about the given priorities before the current job to be prioritized, but not about the priorities to be calculated after the current job. To handle this issue, the recurrent layer is set to be bidirectional (Yin et al., 2017), so the information sequences of jobs could be simultaneously fed into the neural network in both the forward and backward directions. As a result, the information from the $1st$ job until the nth job will be equally considered when calculating the priority of the jth job ($1 \leq j \leq n$). Figure 10 illustrates the computation of the priority of the actor model employing a bidirectional GRU.

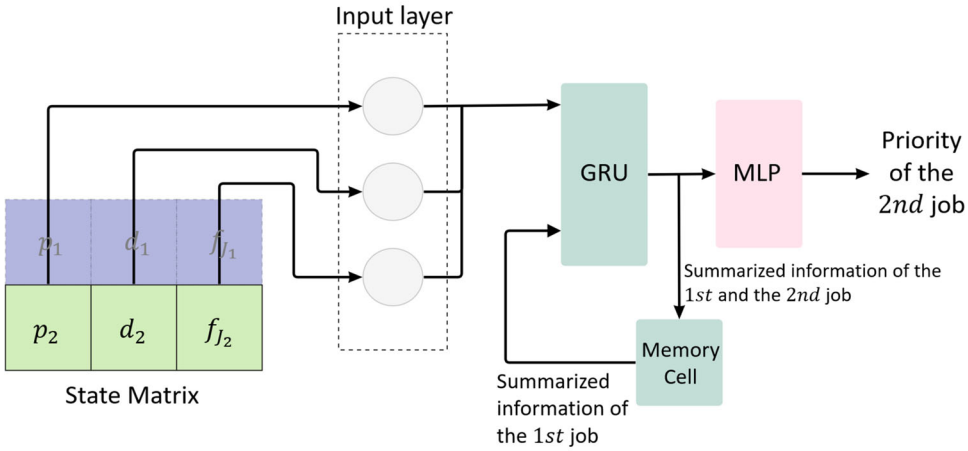
The input layer consists of six neurons (corresponding to the number of features for constructing a state—see Fig. 5a). Since the agent makes decisions within a continuous action space, thus to be flexible with respect to the number of jobs, the output layer contains only a single neuron that computes the priority of a job.

The recurrent hidden layer contains 32 bidirectional GRU cells, while the fully-connected layer consists of 64 neurons. The number of neurons of the fully-connected layer is exactly twice the number of GRU cells, as each neuron needs to receive information from both the forward and backward fed sequence (since GRU cells are bidirectional). The activation function used for this fully-connected layer is Relu.

When the data of each row in the input matrix has been encoded and decoded by the actor, a vector consisting of the



(a) Priority calculation process for the first job



(b) Priority calculation process for the second job

Fig. 9 Mechanism of GRU in calculating job priorities in an exemplary scheduling environment with one machine and two jobs

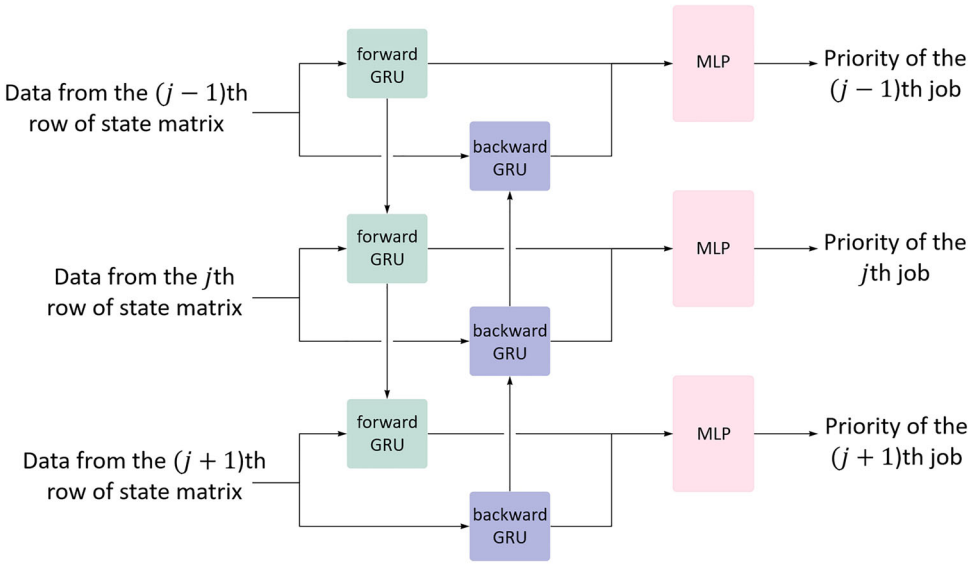


Fig. 10 Architecture of actor network by utilizing bi-directional GRU

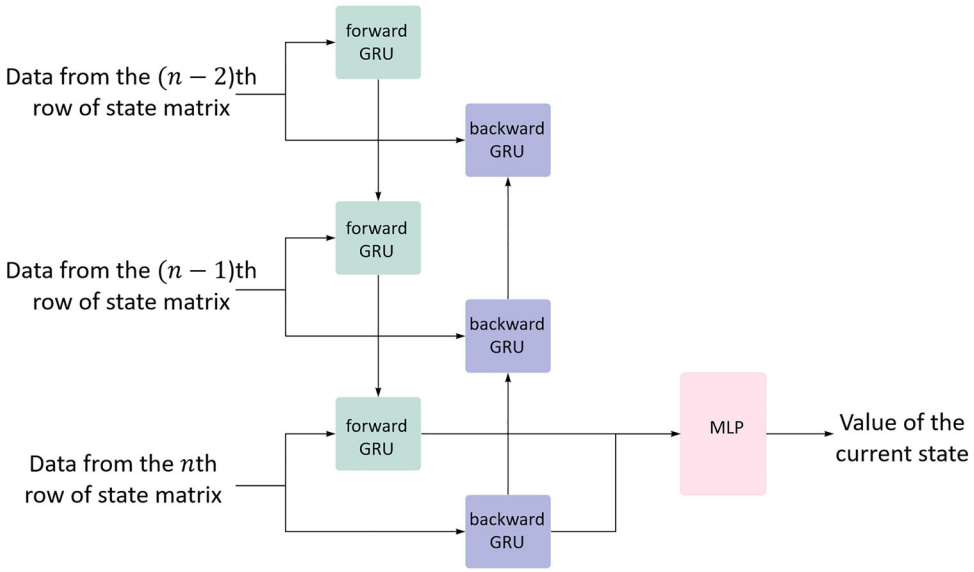


Fig. 11 Architecture of critic network by utilizing bidirectional GRU

priorities of all jobs is obtained. This priority vector is subsequently converted into a probability distribution by applying a SoftMax function. Finally, the agent selects the next job to be processed by sampling a random job index from the probability distribution.

The architecture of the critic network is almost identical to that of the actor network, it also contains a 2-layer GRU part and a 2-layer MLP part. The difference is that instead of each hidden state only the last hidden state h_n is forward propagated through the MLP part. Since the last hidden state is calculated based on all jobs, it can be transformed by the MLP into a summarized state value over all jobs in the buffer. In addition, by encoding the information of each job into the hidden state, we also employ bidirectional GRUs in order to give the critic network a global view. The structure of the critic model and the way it computes the value of the current state are illustrated in Fig. 11.

The number of neurons and GRU cells in each layer of the critic model is identical to the actor model. The numbers of neurons belonging to the input layer and output layer are the same as the number of state features and are equal to one, respectively. The first hidden layer is the recurrent layer with 32 GRU cells, while the second hidden layer is the fully-connected layer with 64 neurons. The activation function used for this fully-connected layer is also Relu.

In summary, the actor model encodes each row of the input matrix separately with the GRU layer and the MLP layer decodes the encoded sequence to the priority vector of the selectable jobs. Meanwhile, the critic model encodes the complete input matrix with the GRU layer, only after the GRU has encoded all the data of the input matrix, the MLP part decodes it to a real number representing the value of the state.

Table 4 Parameter settings of different production configurations

Parameter	Value
Total number of machines m	{10, 11, 12}
Total number of jobs n	{350, 400, 450, 500}
Total number of families of jobs N_F	{7, 8, 9}
Speed of machines v_i	{1, 1.25}
Processing time of a job p_j	Unif [5, 15]
Average tardiness factor r	{0.4, 0.6}
Relative range of due dates R	{0.1, 0.2}
Setup time S	10

Numerical experiments

In this section, we first provide the details of the training process of the agent. Based on the state-action representations and the network structure mentioned above, the agent can process PMSP instances with arbitrary scale, which enables us to train the agent more quickly on a smaller instance and then implement it on larger instances in the production environment. Then the performance of this trained agent on large-scale instances is demonstrated. To further validate the generalization ability of the proposed agent, this performance is compared with that of three dispatching rules and two metaheuristics. The instances with large scale used for validation are generated by the parameters listed in the Table 4.

The two indicators in Table 4, the average tardiness factor r and the relative range of due dates R, are utilized to generate the due date of each job according to the generation procedure used by Potts and Van Wassenhove (1985) for the single machine scheduling problem. The due date d_j of job J_j is generated using a uniform distribution over

Table 5 Parameter settings for the training environment

Parameter	Value
Total number of machines m	10
Total number of jobs n	75
Total number of families of jobs N_F	8
Number of machines with $v = 1.25$	5
Number of machines with $v = 1$	5
Processing time of a job p_j	Unif [5, 15]
Average tardiness factor r	0.1
Telative range of due dates R	0.25
Setup time S	10

Table 6 Hyperparameter settings for training

Hyperparameter	Value
Number of episodes in the first training stage	1500
Number of episodes in the second training stage	4500
Learning rate η	1e−4
Discount factor γ	0.99
Clip range ϵ	0.3
Number of steps per update	5
Batch size	5
Optimizer	Adam

$MP(1 - r - \frac{R}{2})$ and $MP(1 - r + \frac{R}{2})$. The indicator MP is calculated as $MP = \sum_{j=1}^{n} p_j/m + (N_s \cdot S)/m$, where N_s is the total number of times that the setup status is switched. Since the exact number of switches cannot be obtained until the scheduling is complete, and its maximum and minimum values are equal to the number of jobs n (each job does not belong to the same family as the previous one) and the number of families N_F (the setup status is switched only after all the jobs of a family have been processed), respectively. Hence the value of N_s is chosen as the average of the maximum and minimum values and be calculated as $N_s = \frac{n+N_F}{2}$.

The training process of the agent

We develop our agent in PyTorch and encode it with python. The proposed agent is trained in an RL environment that simulates a PMSP. We construct the environment based on OpenAI Gym (Brockman et al., 2016) and a Python-based Discrete-Event Simulation (DES) library called Salabim (van der Ham, 2018). The training process and the comparison process that follows are conducted on a PC with Intel Core i9-11900KF@3.50GHz CPU, 16GB RAM and a single Nvidia RTX 3080 GPU.

The training environment can be described by the parameters listed in Table 5, which contains 10 machines with 2 different speeds and 75 jobs from 8 families. The r and R are set to be 0.1 and 0.25, respectively. The agent is trained on this instance with the reward function Algorithm 1 for 1500 episodes (the first training stage) and then trained with the reward function Algorithm 2 for 4500 episodes (the second training stage). The hyperparameters used for the training process are given in Table 6, where the proposed network is updated by the optimizer Adam (Kingma & Ba, 2014). Compared to the classical stochastic gradient descent, Adam utilizes an adaptive learning rate and history-based updates, which makes it converge faster and require little tuning (Sun et al., 2019).

The training process of the first training stage is illustrated in Fig. 12, in which the abscissa is the number of episodes, and the ordinate is the average number of setups that the agent obtained in the previous 20 episodes. According to Algorithm 1, the agent obtains a negative reward if there are jobs with the same family as the machine setup state that are not selected. Meanwhile, the agent obtains a positive reward if a setup time is avoided. As a result, the curve of the number of setups declines smoothly with the training process, which indicates that agents have learned to schedule two jobs from the same family successively.

The agent is then trained in the second stage to reduce the total tardiness. The change curve of the total tardiness that the agent obtained during the second training stage is demonstrated in Fig. 13, where the abscissa is also the number of episodes, while the ordinate is the average total tardiness that the agent obtained in the previous 20 episodes. It can be seen that the total tardiness that the agent obtained starts at a relatively low level, because the agents have learned to schedule the jobs of the same family successively to avoid setup times after the first training stage, so the value of total tardiness is also reduced by a shorter makespan. Then the total tardiness increases in the beginning episodes, which is because in the second training stage, Algorithm 2 can only provide the agent a feedback after all jobs are scheduled, which results in the agent cannot accurately adjust the policy for priority calculation for each job, and larger total tardiness is generated in exploring the appropriate policy. However, the superiority of the proposed two-stage training strategy is that the additional first stage reduces the time for this exploration. It is evident that, as the training process proceeds, the curve of total tardiness decreases gradually and converges to a significantly lower level, indicating that the agent has learned how to select the most appropriate job based on information about the production environment as well as the job itself.

In order to validate this superiority, we also train a comparative agent only under the reward function Algorithm 2, to investigate whether the comparative agent could converge to a proper value with the same number of episodes without the

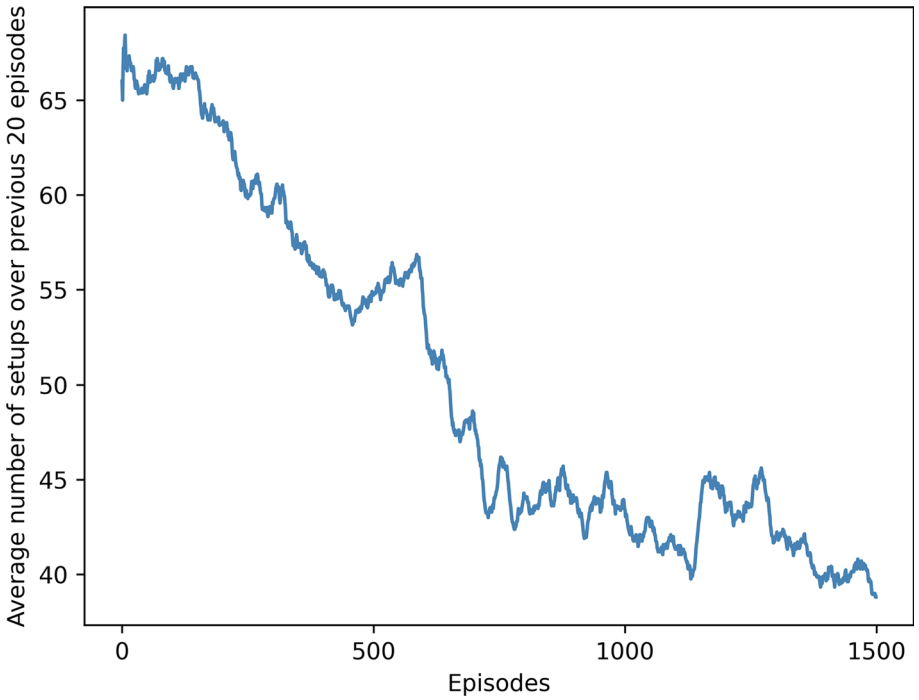


Fig. 12 Average number of setups over previous 20 episodes that the agent obtains in the first stage of training

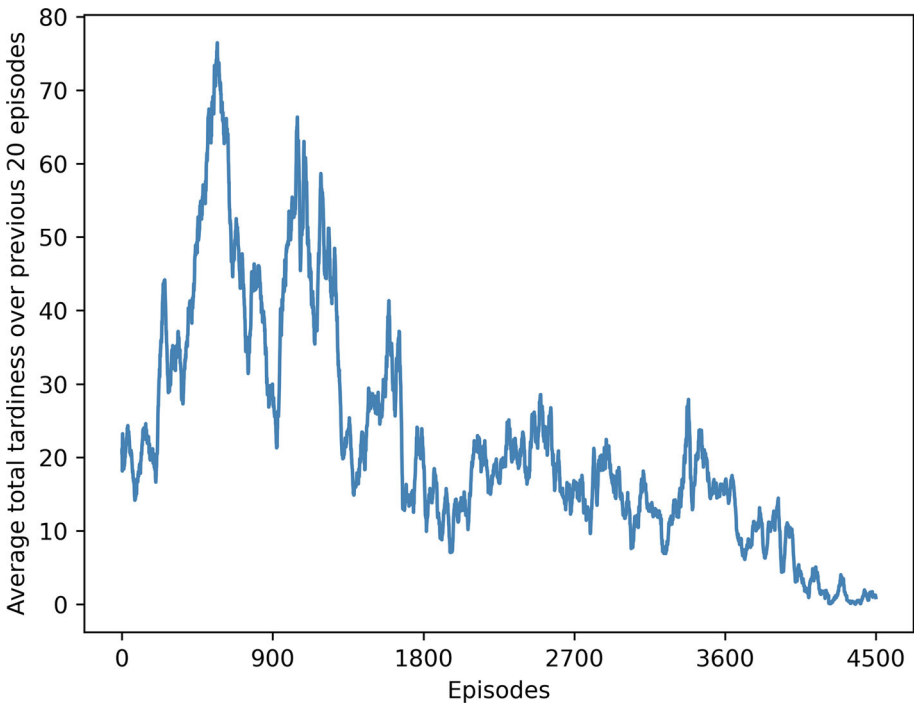


Fig. 13 Average total tardiness over previous 20 episodes that the agent obtains in the second stage of training

first training stage. The training process for this comparative agent trained only by Algorithm 2 is illustrated in Fig. 14, together with the training process for the agent trained by the proposed two-stage training strategy. To eliminate the explanation that a better result is obtained by the proposed agent because of being trained for more episodes, this comparative agent is trained only under the reward function Algorithm 2 for 6000 episodes, which is exactly equal to the total number of episodes that the proposed agent has been trained for (1500 episodes in the first stage and 4500 episodes in the second stage). The difference between the two curves is apparent. It can be observed that the curve of the comparative agent starts at a relatively high level, since the initial agent selects jobs in a stochastic behavior. Then the total tardiness that the comparative agent obtained decreases very slowly with strong fluctuations, because the solution space becomes

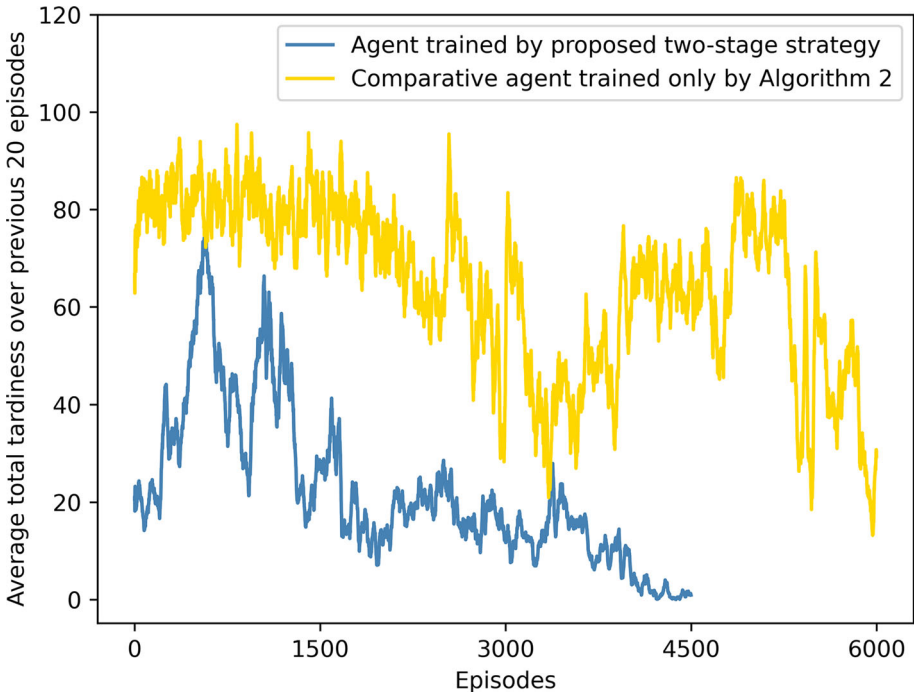


Fig. 14 Average total tardiness over previous 20 episodes obtained by the agent trained by 2-stage training strategy in the second stage of training (blue curve) and the average total tardiness over previous 20 episodes obtained by the comparative agent trained only under reward function Algorithm 2 (gold curve)

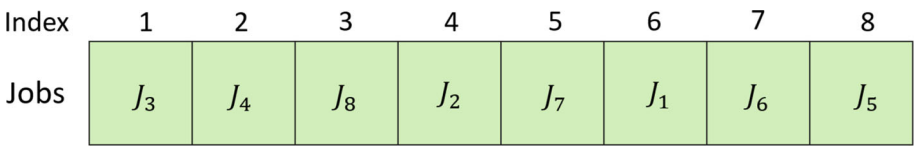


Fig. 15 A chromosome representation of a PMSP instance with 8 jobs

enormous as the instance scale gets larger, which is very time-consuming for exploration. In the finishing episodes of training, the curve of the comparative agent still cannot converge, which suggests that this agent has not learned a proper policy for job selection. In contrast, also trained under Algorithm 2, the proposed agent can find the optimal solution much more efficiently. This comparison indicates that the solution space for exploration was remarkably reduced by the first training stage and the effectiveness of the proposed two-stage training strategy is therefore confirmed.

Comparisons with dispatching rules and metaheuristics

In order to further confirm the effectiveness and generality of the proposed agent, we compare the agent trained in section "The training process of the agent" with three dispatching rules, including SPT, EDD and a family-based dispatching rule which is called MAS_PAR (van der Zee, 2015). SPT selects the job with the shortest processing time. EDD selects the job with the earliest due date. MAS_PAR is specially designed for the PMSPs under family setups constraint. This dispatching rule selects not only the particular job, but also decides which family should be scheduled at each pre-defined decision time point.

Table 7 Parameter settings for GA approach

Basic parameters for GA-Rules and GA-Classic	
Number of generations	50
Number of individuals	50
Crossover probability	0.8
Mutation probability	0.8
Specific parameters for GA-Rules	
Initial number of switches in each gene	5
Probability of add a new switch	0.2
Probability of drop a existing switch	0.1

Moreover, two metaheuristics are also taken into comparison. The first comparative metaheuristics is a GA developed by Rolf et al. (2020) for solving a hybrid flow shop scheduling problem, which is denoted as GA-Rules in this paper. The GA is utilized to assign four dispatching rules during the scheduling process, namely SPT, EDD, minimum slack time (MST) and smallest critical ratio (CR). Since CR is designed for flow shop scheduling problems, it is substituted by the previously mentioned MAS_PAR rule for addressing the PMSP in this paper. In GA-Rules, the number of switches of dispatching rules is a pre-defined hyperparameter, based

on which two specific mutation operators are developed. The first operator adds a new switch for each gene by mutation, while the second operator drops an existing switch for each gene by mutation.

We also derive a classic GA as another comparative metaheuristic, which is referred to as GA-Classic in this paper. The chromosome representation is a sequence of all jobs. Figure 15 gives an exemplary chromosome of a PMSP instance with 8 jobs, which will be assigned to machines in this sequence in a dispatching behavior. The parameter settings for both two GA approaches are listed in Table 7.

The PMSP instances used in the comparison are generated by different settings of the parameters in Table 4, namely the number of machines m, the number of jobs n, the number of families N_F, average tardiness factor r, and relative range of due dates R. The agent trained in Sect. "The training process of the agent" is implemented to solve these instances without retraining. Then the performance and the computation time are compared with those of the aforementioned three dispatching rules and two metaheuristics.

The performance of the agent could be better evaluated if the gap from the optimal solution of the agent and all comparative approaches on each instance could be computed. However, since the proposed PMSP is NP-hard, it is extremely time-consuming to compute the optimal solution on large-scale instances. Therefore, we prefer to use a lower bound in place of optimal solutions. We derive a lower bound LB for our problem based on the research of Azizoglu and Kirca (1998) and Schaller (2014). To calculate a lower bound for identical PMSP without family setups, Azizoglu and Kirca (1998) introduce the preemption relaxation, which allows jobs to be simultaneously processed on more than one machine. After sorting jobs by the SPT rule, the lower bound can be therefore computed as:

$$\sum_{j=1}^{n} \max\left\{0,\ \frac{\sum_{k=1}^{j} p_j}{m} - d_{[j]}\right\}, \tag{21}$$

where $d_{[j]} \leq d_{[j+1]}$ for all j. Then Schaller (2014) extends the above lower bound to the problem with family setups. The processing time p_j is substituted by a modified processing time mp_j, which is defined as $mp_j = p_j + \frac{S}{n_{f_j}}$, where n_{f_j} is the number of jobs from the family of the jth job and S is the setup time. The lower bound for identical PMSP with family setups is given by:

$$\sum_{j=1}^{n} \max\left\{0,\ \frac{\sum_{k=1}^{j} mp_j}{m} - d_{[j]}\right\} \tag{22}$$

In order to extend this lower bound to the proposed uniform PMSP with family setups, the number of machines m in Eq. 22 is replaced by the sum of all machine speeds. The lower bound LB for our problem can be calculated in the form of:

$$LB = \sum_{j=1}^{n} \max\left\{0,\ \frac{\sum_{k=1}^{j} mp_j}{\sum_{i=1}^{m} v_i} - d_{[j]}\right\} \tag{23}$$

Therefore, the performance of each approach can be measured by the gap to the lower bound Gap, which is defined as:

$$Gap(A) = \frac{TT(A) - LB}{LB} \times 100\% \tag{24}$$

where $TT(A)$ is the total tardiness obtained by the approach A.

The gap to lower bound for each approach on all instances are provided in Tables 8, 9, 10 and 11 where the minimum Gap is highlighted in bold font. Since the scheduling behavior of agents is characterized by a certain randomness, we repeat the scheduling of our agent independently 10 times on each instance. The mean value of the Gap obtained by the agent over the 10 repetitions is taken into comparison and the standard deviations are also given in Tables 8, 9, 10 and 11. Meanwhile, Table 12 demonstrate the computational time taken by each approach to solve the instance of each scale, since the computation time is primarily related to the instance scale rather than the due date setting.

First, it can be observed from Tables 8, 9, 10 and 11 that the proposed RL approach provides the minimum gap to the lower bound on each instance. And the advantage of the Gap of the RL agent over the Gap provided by all the other methods is significant, indicating a much higher solution quality. Moreover, Table 12 reports that the solutions of such high quality are provided in a very computationally efficient process. The largest instance with 500 jobs can be solved by the agent within 3 s, which takes the metaheuristics multiple times longer. Only the dispatching rules use less computational time than the trained RL agent, which is because of their simple procedures. However, the superiority of the results yielded by the RL agent over those of dispatching rules is significantly greater.

Figure 16 shows the comparison of the average Gap obtained by each method on all instances under each due date setting. It can be seen that the total tardiness generated by the RL method is remarkably lower for all due date settings. Meanwhile, compared to other methods, the performance of RL varies relatively little over the instances with different due date settings.

Figures 17, 18, 19 and 20 illustrate the influence of different parameter settings of the instances on the generalization performance of the trained RL agent. This illustration helps us to decide how to choose the parameter settings for the instance used to train the agent, according to the parameters of the instances in the real application, in order to achieve

Table 8 Mean value and standard deviation of the Gap(in %) of RL Agent and the Gap (in %) of comparative approaches when $r = 0.4$ and $R = 0.1$

N_F	m	n	RL		GA-Rules	GA-Classic	MAS_PAR	SPT	EDD
			Mean	Std					
7	10	350	**32.31**	4.27	120.08	206.82	167.39	476.63	568.39
		400	**28.79**	6.46	132.32	201.45	90.67	483.07	565.01
		450	**21.97**	4.92	117.20	223.47	79.68	507.64	577.98
		500	**18.49**	6.10	109.68	236.99	181.82	496.00	563.01
	11	350	**32.68**	3.78	131.79	202.15	170.52	476.17	558.07
		400	**29.29**	4.89	117.50	219.42	97.85	508.76	534.79
		450	**20.37**	6.97	103.06	220.36	85.80	490.67	584.89
		500	**15.87**	4.21	120.32	222.17	169.05	495.95	560.01
	12	350	**41.41**	4.13	155.17	212.91	182.66	480.99	531.89
		400	**37.65**	2.86	141.97	208.16	116.00	503.23	570.24
		450	**34.25**	4.28	131.49	226.88	103.02	510.53	573.10
		500	**24.54**	4.13	127.51	230.57	192.45	515.75	564.44
8	10	350	**41.52**	8.71	201.21	286.66	169.52	590.10	639.93
		400	**39.50**	5.76	171.13	281.28	124.44	544.55	647.21
		450	**35.12**	9.07	156.01	287.29	112.89	587.17	635.89
		500	**30.09**	4.86	130.83	284.02	96.31	577.75	620.28
	11	350	**44.54**	8.99	215.41	277.48	179.35	558.71	612.74
		400	**38.17**	6.77	180.02	270.06	134.55	561.46	612.96
		450	**31.48**	8.02	167.09	282.59	121.16	568.38	634.26
		500	**26.49**	4.76	137.93	261.22	102.93	556.80	625.50
	12	350	**54.11**	5.93	246.47	289.62	200.38	581.57	646.29
		400	**47.10**	4.28	204.60	291.86	157.52	613.91	628.51
		450	**43.81**	6.76	185.16	290.57	141.93	611.95	613.99
		500	**31.35**	6.63	159.88	297.40	121.81	589.01	653.62
9	10	350	**60.32**	10.85	175.50	236.01	253.24	526.12	580.64
		400	**43.42**	8.09	142.05	231.57	128.17	535.51	568.60
		450	**43.41**	7.40	130.40	250.36	172.24	515.23	584.00
		500	**32.36**	9.62	121.12	250.55	138.10	540.16	593.86
	11	350	**64.58**	10.48	192.95	238.22	267.71	520.37	558.19
		400	**38.07**	8.91	168.54	221.40	139.60	523.88	598.30
		450	**40.13**	5.89	153.84	236.57	180.80	517.19	565.23
		500	**29.88**	8.71	136.24	244.23	139.39	528.53	590.48
	12	350	**71.22**	9.93	223.13	232.34	265.56	522.98	560.52
		400	**47.80**	11.48	208.91	233.93	162.75	537.50	593.64
		450	**48.19**	7.01	173.83	256.01	201.80	532.23	591.46
		500	**41.02**	8.03	163.76	256.20	158.68	519.86	611.79

proper efficiency and performance. It can be seen that the agent performs better on instances with $R = 0.6$ and $r = 0.1$, where the due dates are relatively tighter and close to each other. What is also noteworthy is that the agent has better generalization performance on the instances with 7 families relative to instances with 8 families, even though this agent was trained on an instance with 8 families. This might be due to the fact that an instance with 7 families can be considered as a simplified instance with 8 families where the number of jobs of the 8th family is 0. Furthermore, the agent performs only slightly worse on the instances with 9 families, where the jobs from the 9th family are completely new to the agent, indicating a relatively robust generalization capability. Despite the fact that the agent's generalization performance fluctuates on instances with different parameter settings, it still produces the best solution on each instance among the solutions of all methods. A further experiment on investigating the difference in generalization capability of agents trained on instances of different scales is implemented in Appendix A, in which two new agents are trained on a larger instance and

Table 9 Mean value and standard deviation of the Gap(in %) of RL Agent and the Gap (in %) of comparative approaches when $r = 0.4$ and $R = 0.2$

N_F	m	n	RL		GA-Rules	GA-Classic	MAS_PAR	SPT	EDD
			Mean	Std					
7	10	350	**51.20**	5.00	152.63	250.43	205.16	555.70	644.39
		400	**49.34**	4.78	142.41	258.77	119.18	565.54	643.11
		450	**43.64**	5.36	151.53	279.41	107.89	596.41	660.75
		500	**39.86**	6.22	123.65	283.46	223.13	582.27	642.84
	11	350	**52.75**	3.45	165.92	245.56	208.67	555.26	632.68
		400	**48.62**	6.03	150.84	245.53	127.33	594.67	608.63
		450	**43.32**	6.13	156.04	278.24	114.90	576.98	668.74
		500	**37.74**	4.87	126.63	269.43	208.60	582.26	639.27
	12	350	**63.41**	5.05	192.80	259.95	222.53	560.66	602.92
		400	**58.55**	4.68	180.17	267.24	147.84	588.42	649.08
		450	**52.50**	5.35	161.45	286.03	134.58	599.72	655.11
		500	**45.33**	3.99	152.24	285.23	235.20	604.99	644.34
8	10	350	**67.24**	10.64	253.28	343.79	214.66	699.55	739.33
		400	**59.70**	6.86	214.77	320.68	162.34	646.40	747.92
		450	**57.09**	11.21	209.13	349.85	148.66	695.45	734.35
		500	**50.26**	4.18	168.76	355.94	129.11	683.61	715.53
	11	350	**66.00**	9.94	269.88	341.67	226.22	662.87	707.80
		400	**57.95**	7.31	228.17	341.62	174.13	666.01	708.30
		450	**54.21**	8.18	207.48	339.78	158.45	673.65	732.68
		500	**45.39**	4.84	177.15	334.56	136.79	659.29	721.60
	12	350	**79.14**	14.62	306.03	349.29	250.11	689.59	746.97
		400	**69.84**	6.32	256.75	352.59	200.53	726.95	726.28
		450	**62.86**	10.15	233.49	361.72	182.67	724.17	708.95
		500	**56.66**	4.45	202.77	359.86	158.68	696.61	754.34
9	10	350	**91.34**	9.94	215.45	278.61	302.66	615.05	661.84
		400	**63.73**	10.10	175.48	279.58	160.37	624.55	647.17
		450	**61.38**	9.81	163.24	289.59	212.05	600.29	663.98
		500	**55.04**	7.55	146.82	281.55	173.22	628.84	675.55
	11	350	**85.45**	11.69	235.71	285.72	319.47	608.30	635.97
		400	**55.27**	12.14	189.26	281.70	173.37	611.28	681.22
		450	**57.29**	9.19	198.88	284.13	221.89	602.48	642.55
		500	**48.46**	10.53	160.06	294.29	174.65	615.49	671.64
	12	350	**90.50**	12.21	270.34	285.70	316.86	611.45	638.57
		400	**69.90**	7.98	255.17	285.37	199.67	626.78	675.91
		450	**65.51**	8.71	234.63	299.03	245.95	619.67	672.44
		500	**60.11**	8.74	189.47	291.26	196.50	605.58	696.16

a smaller instance, respectively. The results demonstrate that all the agents outperform the best solution that the comparative approaches could provide, even though they are trained on different instances.

To summary, the RL agent trained on a much smaller instance outperforms the comparative dispatching rules and metaheuristics on all large-scale instances significantly. The superiority of this RL agent suggests that the knowledge learned by the agent on a small-scale instance is universal and valuable for solving large-scale problems.

Conclusion and future work

In this paper, we propose a DRL agent to minimize the total tardiness on PMSP with family setups. Novel variable-length representations for state and action are developed, which enables the agent to calculate a comprehensive priority for each job and then select the job according to these priorities every time a machine is idle. Moreover, the length-agnostic state and action representations allow a trained agent to solve instances of arbitrary scales without retraining. To capture the sequential relationships between jobs and handle

Table 10 Mean value and standard deviation of the Gap(in %) of RL Agent and the Gap (in %) of comparative approaches when $r = 0.6$ and $R = 0.1$

N_F	m	n	RL		GA-Rules	GA-Classic	MAS_PAR	SPT	EDD
			Mean	Std					
7	10	350	**27.73**	1.97	77.42	149.96	103.91	242.84	292.12
		400	**26.54**	2.65	73.60	158.63	58.81	246.28	288.77
		450	**19.54**	2.51	72.68	159.14	53.78	253.40	292.32
		500	**20.98**	3.37	76.76	166.91	103.58	247.52	286.44
	11	350	**28.38**	2.31	83.60	151.39	105.40	241.76	288.13
		400	**26.33**	2.23	76.07	153.70	62.74	257.12	275.89
		450	**21.35**	2.37	75.17	158.01	57.12	245.87	295.54
		500	**19.47**	2.34	71.05	160.34	97.71	248.54	283.63
	12	350	**33.35**	1.68	95.66	151.69	111.12	244.22	276.14
		400	**31.52**	1.46	92.90	159.31	72.40	254.54	291.67
		450	**26.09**	2.25	86.32	165.75	66.10	255.58	289.56
		500	**22.64**	2.38	88.56	168.48	109.10	257.30	286.16
8	10	350	**35.51**	4.48	118.62	191.32	100.10	287.74	317.90
		400	**34.46**	2.31	104.18	191.48	79.26	267.38	319.73
		450	**32.79**	4.76	96.14	190.59	72.90	285.28	314.14
		500	**28.85**	1.55	81.61	193.37	63.95	281.45	307.16
	11	350	**38.96**	4.86	125.57	194.72	105.55	274.09	306.16
		400	**32.49**	3.79	108.44	189.64	84.57	274.85	306.10
		450	**30.04**	3.95	99.10	191.92	77.28	277.04	313.99
		500	**26.98**	2.01	85.36	193.12	67.50	272.02	309.59
	12	350	**44.08**	4.84	140.49	190.77	116.38	283.59	321.20
		400	**39.29**	2.93	120.30	191.08	96.14	298.27	312.03
		450	**35.21**	4.79	110.39	195.14	87.84	296.93	304.98
		500	**31.82**	2.60	96.41	194.10	77.18	286.09	322.59
9	10	350	**48.31**	4.88	106.38	170.98	139.92	266.26	298.24
		400	**33.92**	4.20	89.42	164.80	81.38	269.88	291.87
		450	**34.30**	4.16	84.23	174.29	106.43	260.05	298.29
		500	**26.74**	4.51	79.90	175.28	84.18	270.34	300.67
	11	350	**45.49**	5.59	115.30	162.86	147.26	262.40	287.30
		400	**32.93**	6.49	96.44	170.18	87.39	264.56	305.31
		450	**33.53**	4.08	98.16	170.62	110.32	259.99	290.01
		500	**27.60**	3.34	86.24	176.88	85.62	264.52	299.13
	12	350	**47.98**	5.53	130.49	168.35	147.20	264.54	288.62
		400	**39.62**	5.16	124.36	166.53	99.44	270.35	303.40
		450	**38.86**	3.45	105.34	172.50	120.85	266.97	301.08
		500	**33.36**	3.87	98.88	180.24	95.66	260.72	309.26

the variable-length sequence, we utilize an RNN, particular GRU, to approximate the policy of the agent, which is dominant in sequence transduction. Finally, a two-stage training strategy is designed to train the agent efficiently under a sparse reward function.

Numerical experiments with different parameter settings are conducted to validate the effectiveness and generality of the proposed agent. The comparison between the proposed agent and the agent trained under a traditional one-stage strategy confirms the superiority of the two-stage training strategy. Moreover, the performances of the proposed agent on several untrained large-scale instances outperform the comparative dispatching rules and metaheuristics.

In future work, more dynamic and uncertain constraints such as new job insertions and machine breakdowns will be studied. Other objectives like total flow time and production costs are also considerable. Furthermore, it is worth to be noted that RNN is not the only network that can handle the variable-length sequence. The Transformer is a state-of-the-art model that is also capable of processing variable length sequences. We will investigate the Transformer and compare its performance with the agent proposed in this paper.

Table 11 Mean value and standard deviation of the Gap(in %) of RL Agent and the Gap (in %) of comparative approaches when $r = 0.6$ and $R = 0.2$

N_F	m	n	RL		GA-Rules	GA-Classic	MAS_PAR	SPT	EDD
			Mean	Std					
7	10	350	**37.07**	1.99	90.04	164.65	117.45	265.62	313.32
		400	**35.74**	2.94	84.99	178.98	69.70	269.87	310.30
		450	**30.19**	2.73	77.80	172.61	64.88	278.24	314.83
		500	**29.72**	2.94	75.95	180.60	117.42	271.65	308.33
	11	350	**37.90**	1.68	96.57	162.44	119.01	264.57	309.04
		400	**36.37**	2.36	89.03	165.04	73.87	281.44	296.61
		450	**30.08**	2.72	87.68	181.21	68.47	270.23	318.33
		500	**28.70**	2.54	82.99	185.16	111.12	272.65	305.17
	12	350	**43.37**	1.64	109.36	172.90	125.11	267.15	296.30
		400	**41.34**	2.12	109.39	178.10	84.15	278.72	313.52
		450	**37.04**	2.12	100.53	179.34	78.04	280.55	311.85
		500	**32.77**	2.77	89.12	181.70	123.27	281.97	308.01
8	10	350	**47.92**	4.10	137.08	218.20	115.75	317.19	344.68
		400	**43.94**	2.43	118.98	213.52	93.25	295.23	346.48
		450	**40.98**	4.60	113.56	218.30	86.43	314.03	340.13
		500	**38.19**	1.84	95.94	214.69	76.55	309.60	332.18
	11	350	**48.26**	5.26	144.61	218.07	121.56	302.58	331.91
		400	**42.48**	3.10	125.79	202.47	98.98	303.30	331.85
		450	**39.08**	4.66	127.74	214.29	91.14	305.22	340.02
		500	**36.07**	2.68	99.92	214.84	80.36	299.47	334.86
	12	350	**55.13**	7.41	160.77	216.06	133.18	312.84	348.32
		400	**49.13**	2.71	138.55	219.93	111.38	328.45	338.18
		450	**46.62**	3.69	127.58	218.41	102.49	326.56	330.28
		500	**41.43**	2.39	111.67	217.99	90.73	314.59	348.95
9	10	350	**57.91**	4.84	120.92	186.24	157.07	292.30	321.61
		400	**43.81**	4.97	115.87	184.19	94.14	295.52	314.25
		450	**43.47**	4.44	98.40	188.40	120.62	284.06	320.40
		500	**38.33**	4.57	89.40	194.18	97.42	294.81	322.70
	11	350	**54.41**	6.42	130.40	184.29	164.87	287.99	309.83
		400	**43.31**	4.83	123.97	182.10	100.62	289.79	328.69
		450	**41.04**	3.21	110.10	188.49	124.84	283.94	311.50
		500	**36.91**	4.54	93.43	189.79	98.95	288.54	321.04
	12	350	**58.49**	6.95	146.58	186.77	164.83	290.26	311.30
		400	**46.33**	6.03	138.99	187.16	113.51	296.02	326.62
		450	**47.02**	3.68	128.62	194.23	135.97	291.47	323.33
		500	**42.63**	4.29	107.28	198.05	109.64	284.49	331.95

Table 12 Comparison of the computational time(sec) of all approaches for solving instances with different scales

N_F	m	n	RL	GA-Rules	GA-Classic	MAS_PAR	SPT	EDD
7	10	350	1.54	15.95	49.19	0.12	0.13	0.12
		400	1.90	31.61	54.35	0.12	0.11	0.12
		450	2.37	13.91	60.14	0.14	0.14	0.14
		500	2.89	31.15	66.47	0.12	0.12	0.12
	11	350	1.49	15.34	48.67	0.10	0.11	0.10
		400	1.91	14.19	54.36	0.15	0.10	0.15
		450	2.36	14.36	60.09	0.16	0.10	0.16
		500	2.88	34.53	66.52	0.16	0.11	0.16
	12	350	1.50	14.57	48.75	0.12	0.11	0.12
		400	1.90	30.43	54.48	0.11	0.19	0.11
		450	2.37	14.67	60.23	0.14	0.11	0.14
		500	2.88	31.75	66.53	0.12	0.11	0.12
8	10	350	1.56	13.11	49.90	0.06	0.07	0.06
		400	1.91	22.64	55.15	0.07	0.08	0.07
		450	2.38	30.27	60.99	0.08	0.08	0.08
		500	2.90	17.37	67.71	0.08	0.08	0.08
	11	350	1.50	14.26	49.57	0.07	0.08	0.07
		400	1.91	13.90	55.44	0.07	0.08	0.07
		450	2.38	28.81	61.34	0.08	0.08	0.08
		500	2.89	16.00	67.49	0.08	0.08	0.08
	12	350	1.50	13.40	49.42	0.07	0.07	0.07
		400	1.91	13.05	55.23	0.08	0.07	0.08
		450	2.38	14.79	61.15	0.07	0.07	0.07
		500	2.89	14.66	67.62	0.07	0.08	0.07
9	10	350	1.56	16.07	49.85	0.06	0.07	0.06
		400	1.92	31.02	54.98	0.07	0.07	0.07
		450	2.37	30.73	60.82	0.08	0.08	0.08
		500	2.89	33.40	67.29	0.09	0.08	0.09
	11	350	1.49	14.87	49.17	0.07	0.07	0.07
		400	2.36	13.52	54.93	0.08	0.07	0.08
		450	2.38	32.49	60.99	0.09	0.08	0.09
		500	2.89	29.55	67.39	0.08	0.08	0.08
	12	350	1.60	13.33	49.24	0.07	0.08	0.07
		400	1.91	33.77	55.09	0.08	0.09	0.08
		450	2.37	34.62	60.87	0.09	0.08	0.09
		500	2.88	28.33	67.03	0.08	0.07	0.08

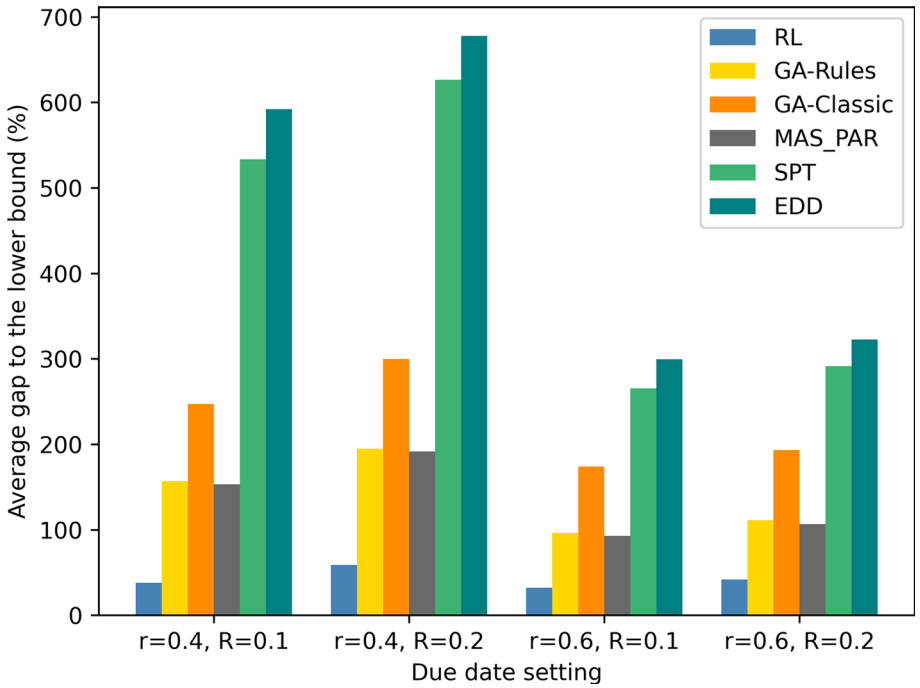


Fig. 16 The average $Gap(\%)$ obtained by RL agent and all comparative approaches over all instances under different due date settings

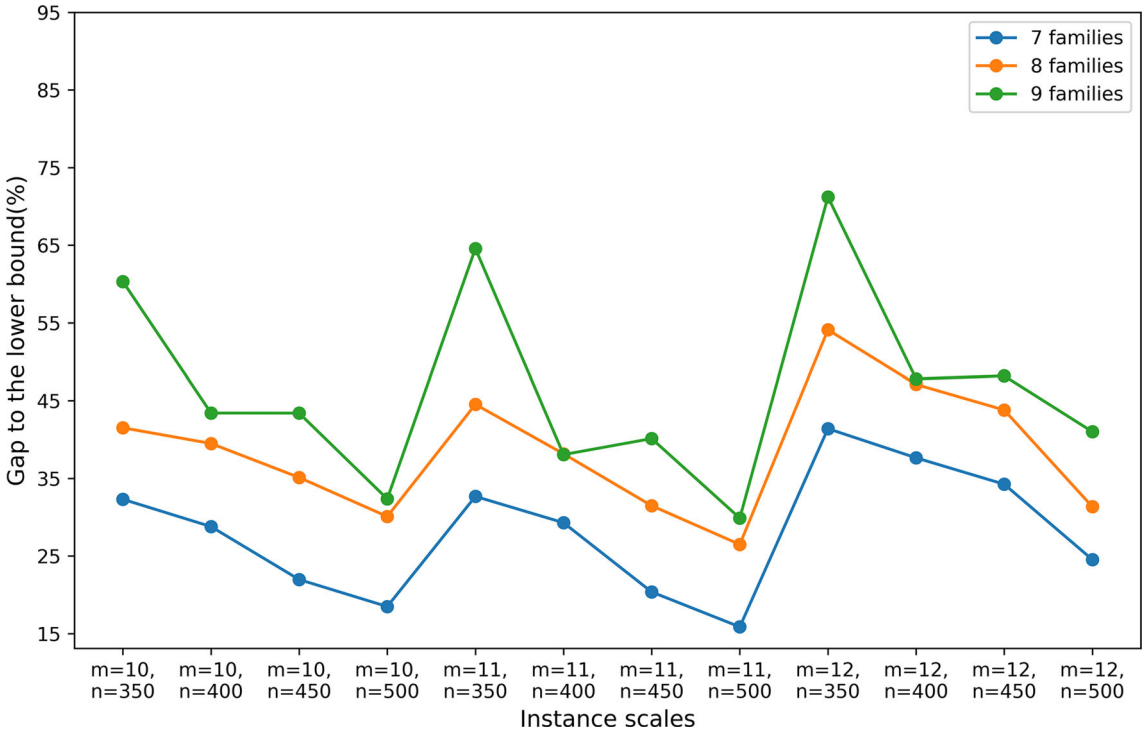


Fig. 17 $Gap(\%)$ obtained by the agent on instances with different number of machines m, number of jobs n and number of families N_F when $r = 0.4$ and $R = 0.1$

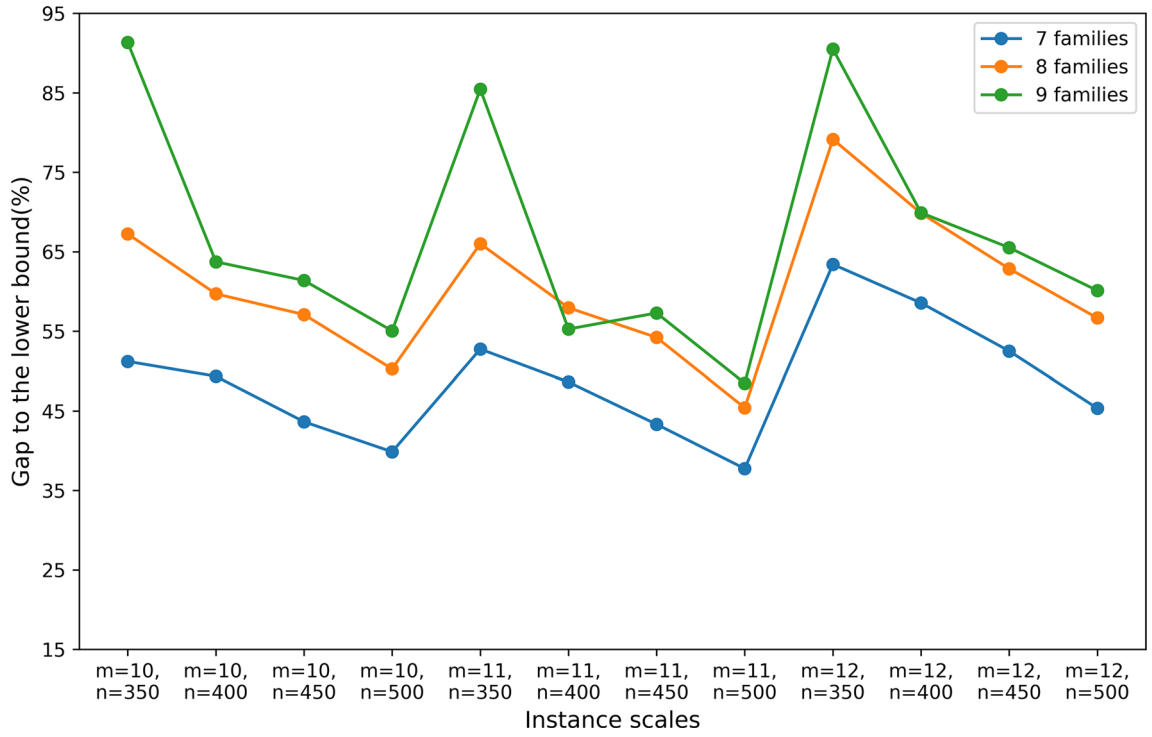


Fig. 18 $Gap(\%)$ obtained by the agent on instances with different number of machines m, number of jobs n and number of families N_F when $r = 0.4$ and $R = 0.2$

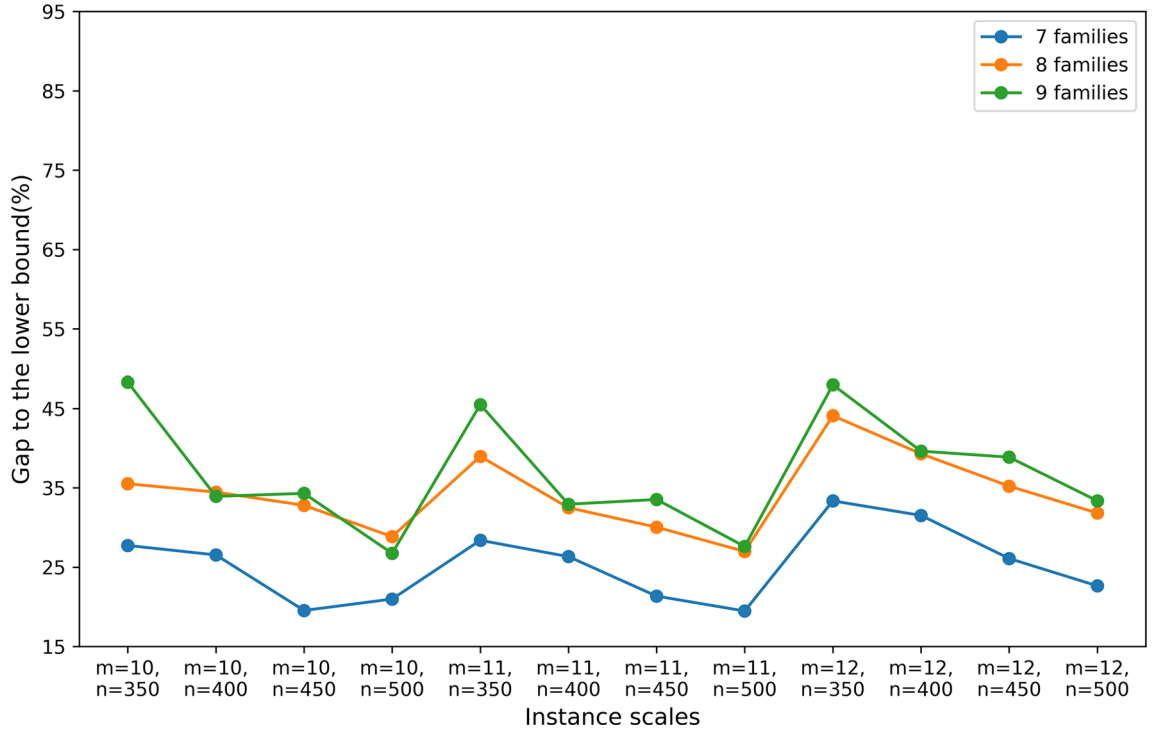


Fig. 19 $Gap(\%)$ obtained by the agent on instances with different number of machines m, number of jobs n and number of families N_F when $r = 0.6$ and $R = 0.1$

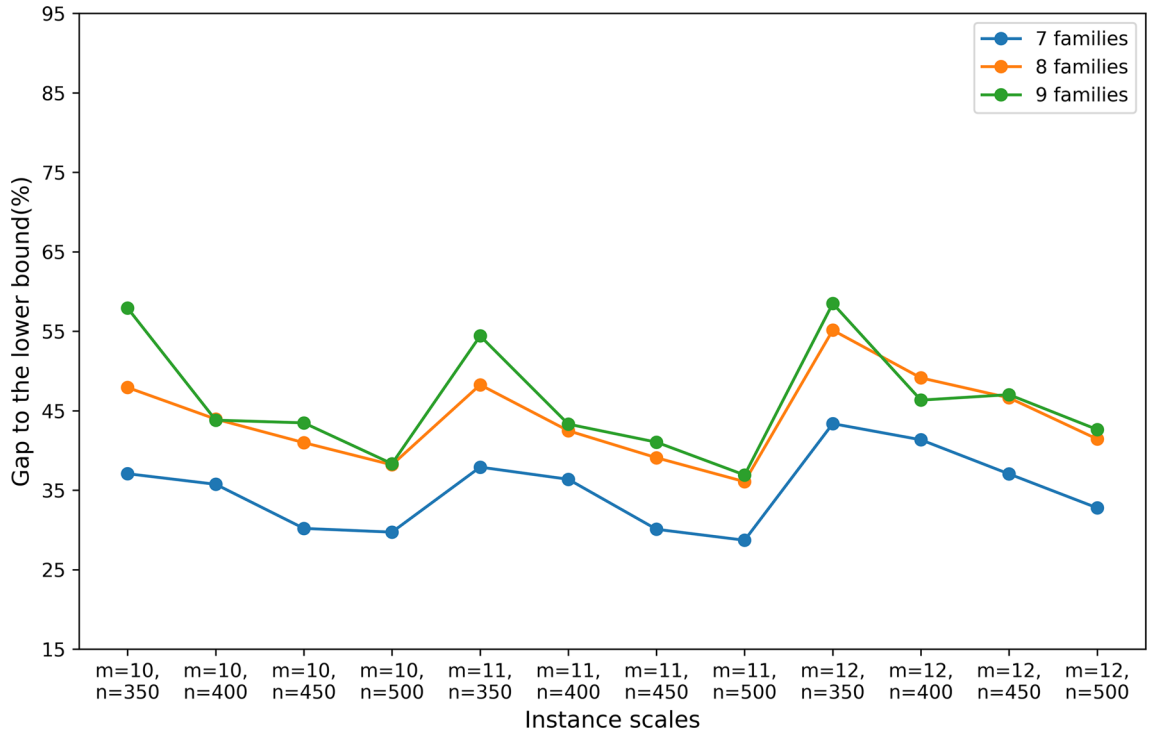


Fig. 20 $Gap(\%)$ obtained by the agent on instances with different number of machines m, number of jobs n and number of families N_F when $r = 0.6$ and $R = 0.2$

Acknowledgements Li is supported by the China Scholarship Council (CSC) to pursue a master's degree at Otto von Guericke University Magdeburg. The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany (BMBF) within the framework of the SENECA project (Grant Number: 01IS20019A). The authors acknowledge Benjamin Rolf from Otto von Guericke University Magdeburg for his assistance with the experiments.

Funding Open Access funding enabled and organized by Projekt DEAL.

Appendix A: Performance of the agents trained on instances of different scales

The training process demonstrated in section "The training process of the agent" is also implemented on a smaller instance and a larger instance. A comparison of the scales of the instances utilized in section "The training process of the agent" and the smaller and larger instances are given in Table 13, while the parameter settings used for training environment description that are not listed in this table are the same as in Table 5. As the instance utilized for training becomes larger and smaller, the number of episodes required for both two training stages increases and decreases, respectively. The number of episodes required to train an agent on the three previously mentioned instances are listed in Table 14. The other hyperparameters required for training are the same as given in Table 6.

Tables 15, 16, 17 and 18 illustrated the generalization performance of the agents trained on the smaller and the larger instances as well as of the agent trained in Section 5.1, in

Table 13 Comparison of the scales of the instance in section "The training process of the agent" and of the larger and smaller instances

Instance in section "The training process of the agent"	Number of machines m	10
	Number of jobs n	75
	Number of families N_F	8
Larger instance	Number of machines m	14
	Number of jobs n	100
	Number of families N_F	9
Smaller instance	Number of machines m	6
	Number of jobs n	40
	Number of families N_F	7

Table 14 The number of episodes required to train an agent on the instance in section "The training process of the agent" and on the larger and smaller instances

Number of episodes in the first training stage	
Instance in section "The training process of the agent"	1500
Larger instance	2500
Smaller instance	600
Number of episodes in the second training stage	
Instance in section "The training process of the agent"	4500
Larger instance	6000
Smaller instance	3900

Table 15 Mean value and standard deviation of the Gap(in %) of Agents trained on different instances and minimum Gap(in %) provided by all comparative approaches when $r = 0.4$ and $R = 0.1$

N_F	m	n	RL		RL-Large		RL-Small		Compar-Min
			Mean	Std	Mean	Std	Mean	Std	
7	10	350	32.31	4.27	38.41	4.69	**25.77**	6.50	*120.08*
		400	28.79	6.46	33.63	3.40	**25.89**	4.69	*90.67*
		450	21.97	4.92	30.15	3.30	**14.84**	4.33	*79.68*
		500	**18.49**	6.10	28.39	3.04	25.54	3.91	*109.68*
	11	350	32.68	3.78	39.09	4.99	**24.66**	5.12	*131.79*
		400	29.29	4.89	34.65	3.48	**26.93**	3.97	*97.85*
		450	20.37	6.97	29.01	4.50	**13.60**	4.94	*85.80*
		500	**15.87**	4.21	27.01	1.64	23.30	4.88	*120.32*
	12	350	41.41	4.13	48.34	4.97	**37.23**	3.92	*155.17*
		400	37.65	2.86	45.23	5.17	**30.62**	3.42	*116.00*
		450	34.25	4.28	41.03	3.61	**20.97**	5.96	*103.02*
		500	**24.54**	4.13	36.42	2.25	28.28	5.26	*127.51*
8	10	350	**41.52**	8.71	61.70	6.27	65.30	14.57	*169.52*
		400	**39.50**	5.76	57.44	5.11	72.63	11.82	*124.44*
		450	**35.12**	9.07	46.35	4.08	67.64	14.98	*112.89*
		500	**30.09**	4.86	39.48	3.29	70.54	11.34	*96.31*
	11	350	**44.54**	8.99	64.15	6.67	81.14	12.04	*179.35*
		400	**38.17**	6.77	57.94	6.78	66.33	13.38	*134.55*
		450	**31.48**	8.02	44.88	5.08	60.36	14.55	*121.16*
		500	**26.49**	4.76	38.99	4.96	58.03	14.98	*102.93*
	12	350	**54.11**	5.93	79.02	6.06	80.66	15.39	*200.38*
		400	**47.10**	4.28	69.75	5.95	92.01	11.07	*157.52*
		450	**43.81**	6.76	59.40	3.94	73.79	16.26	*141.93*
		500	**31.35**	6.63	49.00	5.15	65.67	14.23	*121.81*
9	10	350	60.32	10.85	**51.55**	5.75	109.53	17.90	*175.50*
		400	**43.42**	8.09	48.99	4.91	90.82	13.29	*128.17*
		450	43.41	7.40	**42.81**	4.75	123.85	10.97	*130.40*
		500	**32.36**	9.62	41.12	3.12	102.42	15.30	*121.12*
	11	350	64.58	10.48	**51.00**	6.73	107.60	14.71	*192.95*
		400	**38.07**	8.91	47.55	5.83	79.66	19.86	*139.60*
		450	**40.13**	5.89	43.05	4.82	122.09	10.10	*153.84*
		500	**29.88**	8.71	39.97	5.19	99.86	15.02	*136.24*
	12	350	71.22	9.93	**60.73**	7.50	118.63	12.74	*223.13*
		400	**47.80**	11.48	60.61	6.49	92.76	17.97	*162.75*
		450	**48.19**	7.01	52.17	5.19	124.84	10.73	*173.83*
		500	**41.02**	8.03	46.79	5.74	105.13	17.70	*158.68*

Table 16 Mean value and standard deviation of the Gap(in %) of Agents trained on different instances and minimum Gap(in %) provided by all comparative approaches when $r = 0.4$ and $R = 0.2$

N_F	m	n	RL		RL-Large		RL-Small		Compar-Min
			Mean	Std	Mean	Std	Mean	Std	
7	10	350	51.20	5.00	59.16	6.40	**44.80**	5.86	*152.63*
		400	49.34	4.78	50.63	6.31	**44.54**	6.57	*119.18*
		450	43.64	5.36	49.97	3.39	**32.04**	8.23	*107.89*
		500	39.86	6.22	48.42	3.08	43.57	5.95	*123.65*
	11	350	52.75	3.45	58.27	4.93	**44.17**	4.99	*165.92*
		400	48.62	6.03	56.00	4.86	**42.14**	6.88	*127.33*
		450	43.32	6.13	51.47	3.95	**30.60**	6.93	*114.90*
		500	37.74	4.87	47.04	2.99	43.24	5.87	*126.63*
	12	350	63.41	5.05	69.28	6.78	**53.61**	6.16	*192.80*
		400	58.55	4.68	62.03	5.74	**49.64**	4.54	*147.84*
		450	52.50	5.35	61.91	4.61	**38.69**	7.66	*134.58*
		500	**45.33**	3.99	56.59	3.80	49.94	6.09	*152.24*
8	10	350	**67.24**	10.64	90.43	8.39	96.45	17.59	*214.66*
		400	**59.70**	6.86	79.85	8.06	109.65	16.89	*162.34*
		450	**57.09**	11.21	70.72	4.75	92.39	19.15	*148.66*
		500	**50.26**	4.18	65.03	5.39	99.17	14.07	*129.11*
	11	350	**66.00**	9.94	92.29	8.98	112.21	15.11	*226.22*
		400	**57.95**	7.31	83.44	9.09	107.49	12.93	*174.13*
		450	**54.21**	8.18	69.92	6.04	98.08	18.19	*158.45*
		500	**45.39**	4.84	65.82	4.98	85.79	14.16	*136.79*
	12	350	**79.14**	14.62	104.98	10.16	110.25	17.29	*250.11*
		400	**69.84**	6.32	97.72	8.53	121.91	10.33	*200.53*
		450	**62.86**	10.15	84.05	6.63	102.04	20.16	*182.67*
		500	**56.66**	4.45	74.64	5.82	90.28	19.81	*158.68*
9	10	350	91.34	9.94	**69.29**	6.90	150.55	18.96	*215.45*
		400	**63.73**	10.10	68.42	6.27	125.69	13.45	*160.37*
		450	**61.38**	9.81	64.68	4.66	158.13	14.78	*163.24*
		500	**55.04**	7.55	60.87	4.52	118.68	21.36	*146.82*
	11	350	85.45	11.69	**72.35**	6.17	138.55	16.65	*235.71*
		400	**55.27**	12.14	73.06	4.49	112.50	16.60	*173.37*
		450	**57.29**	9.19	63.53	6.04	148.99	14.44	*198.88*
		500	**48.46**	10.53	60.88	6.72	122.78	19.00	*160.06*
	12	350	90.50	12.21	**83.51**	6.61	153.96	16.12	*270.34*
		400	**69.90**	7.98	83.48	5.94	119.08	22.55	*199.67*
		450	**65.51**	8.71	74.42	5.42	157.65	16.29	*234.63*
		500	**60.11**	8.74	69.01	5.97	128.63	18.37	*189.47*

which the agent trained on larger instance, the agent trained on smaller instance and the agent trained in Section 5.1 are denoted as RL-Large, RL-Small and RL, respectively. The scheduling behavior of RL agents is characterized by a certain randomness, we repeat the scheduling of all the three agents independently 10 times on each instance. Therefore, the value of the Gap obtained by the agent presented in Tables 15, 16, 17 and 18 is the mean value over the 10 repetitions. Meanwhile the standard deviations are also given in the tables.

The minimum value from the Gap obtained by all comparative approaches on each instance is also taken into comparison, which is referred to as Compar-Min. The minimum and maximum values of Gap generated by all approaches on each instance are highlighted in bold and italic, respectively. Figure 21 demonstrates the comparison of the average Gap of these three agents and the Compar-Min on instances with different number of families. It is worth noting that the three agents trained on instances with three different scales, outperform the best solution that the comparative approaches

Table 17 Mean value and standard deviation of the Gap(in %) of Agents trained on different instances and minimum Gap(in %) provided by all comparative approaches when $r = 0.6$ and $R = 0.1$

N_F	m	n	RL		RL-Large		RL-Small		Compar-Min
			Mean	Std	Mean	Std	Mean	Std	
7	10	350	27.73	1.97	31.81	2.18	**26.13**	2.72	*77.42*
		400	26.54	2.65	27.74	2.40	**24.40**	2.74	*58.81*
		450	19.54	2.51	24.98	2.82	**16.03**	5.13	*53.78*
		500	**20.98**	3.37	24.42	1.93	23.67	2.33	*76.76*
	11	350	28.38	2.31	32.47	2.80	**25.99**	2.77	*83.6*
		400	26.33	2.23	29.76	2.07	**22.10**	3.26	*62.74*
		450	21.35	2.37	25.52	3.03	**15.84**	3.57	*57.12*
		500	**19.47**	2.34	25.19	2.13	21.10	2.80	*71.05*
	12	350	33.35	1.68	35.27	3.53	**31.50**	3.54	*95.66*
		400	31.52	1.46	33.40	2.49	**26.88**	4.02	*72.4*
		450	26.09	2.25	31.71	3.09	**19.51**	2.91	*66.1*
		500	**22.64**	2.38	29.94	2.62	25.82	3.21	*88.56*
8	10	350	**35.51**	4.48	49.41	3.73	46.13	6.18	*100.1*
		400	**34.46**	2.31	44.48	3.23	44.53	7.77	*79.26*
		450	**32.79**	4.76	40.20	2.08	45.73	7.43	*72.9*
		500	**28.85**	1.55	35.27	2.44	49.25	5.12	*63.95*
	11	350	**38.96**	4.86	50.50	3.61	46.80	6.95	*105.55*
		400	**32.49**	3.79	46.15	3.80	49.99	4.88	*84.57*
		450	**30.04**	3.95	40.36	2.39	42.71	5.85	*77.28*
		500	**26.98**	2.01	36.15	2.16	44.04	4.49	*67.5*
	12	350	**44.08**	4.84	54.45	4.28	54.23	6.28	*116.38*
		400	**39.29**	2.93	53.49	2.97	55.38	5.80	*96.14*
		450	**35.21**	4.79	47.60	3.13	49.84	7.69	*87.84*
		500	**31.82**	2.60	41.50	2.27	48.36	4.76	*77.18*
9	10	350	48.31	4.88	**40.90**	3.67	73.43	6.11	*106.38*
		400	**33.92**	4.20	40.01	3.07	54.69	8.75	*81.38*
		450	**34.30**	4.16	37.00	2.97	70.13	5.05	*84.23*
		500	**26.74**	4.51	33.57	2.47	63.39	5.52	*79.9*
	11	350	45.49	5.59	**39.46**	3.77	69.00	7.84	*115.3*
		400	**32.93**	6.49	42.53	2.07	55.63	5.58	*87.39*
		450	**33.53**	4.08	35.50	3.57	64.48	6.29	*98.16*
		500	**27.60**	3.34	33.32	2.35	59.40	7.75	*85.62*
	12	350	47.98	5.53	**47.41**	2.73	75.59	5.05	*130.49*
		400	**39.62**	5.16	44.63	3.44	57.63	9.33	*99.44*
		450	**38.86**	3.45	41.25	2.91	72.28	5.99	*105.34*
		500	**33.36**	3.87	39.36	2.59	63.86	6.61	*95.66*

could provide, which verifies the generalization capability of the proposed two-stage RNN-based PPO algorithm.

Furthermore, the agent RL-Small trained on the instance with 7 families has the smallest Gap when solving all instances with 7 families. To be more specific, it outperforms the other two agents on 36 instances out of a total of 144 instances, which are all the instances with 7 families. However, its performance deteriorates significantly as the number of families increases. In contrast, the performances of the agents trained on instances with 8 and 9 families are relative stable over instances with different families.

The training process demonstrated in section "The training process of the agent" is also implemented on a smaller instance and a larger instance. A comparison of the scales of the instances utilized in section "The training process of the agent" and the smaller and larger instances are given in Table 13, while the parameter settings used for training environment description that are not listed in this table are the same as in Table 5. As the instance utilized for training

Table 18 Mean value and standard deviation of the Gap(in %) of Agents trained on different instances and minimum Gap(in %) provided by all comparative approaches when $r = 0.6$ and $R = 0.2$

N_F	m	n	RL		RL-Large		RL-Small		Compar-Min
			Mean	Std	Mean	Std	Mean	Std	
7	10	350	37.07	1.99	35.62	3.65	**33.67**	2.16	*90.04*
		400	35.74	2.94	33.84	2.40	**32.96**	2.40	*69.70*
		450	30.19	2.73	34.76	1.97	**24.28**	2.72	*64.88*
		500	**29.72**	2.94	33.12	2.40	30.14	2.74	*75.95*
	11	350	37.90	1.68	34.98	4.49	**33.46**	2.55	*96.57*
		400	36.37	2.36	34.66	3.22	**31.42**	2.70	*73.87*
		450	30.08	2.72	34.43	2.15	**24.28**	2.34	*68.47*
		500	**28.70**	2.54	33.61	2.33	31.39	3.18	*82.99*
	12	350	43.37	1.64	39.96	4.88	**39.60**	2.34	*109.36*
		400	41.34	2.12	38.43	3.39	**35.05**	2.12	*84.15*
		450	37.04	2.12	38.74	2.46	**28.20**	3.01	*78.04*
		500	**32.77**	2.77	38.52	2.06	34.84	2.90	*89.12*
8	10	350	**47.92**	4.10	59.02	3.48	68.67	5.31	*115.75*
		400	**43.94**	2.43	55.24	3.28	71.60	6.11	*93.25*
		450	**40.98**	4.60	50.94	2.79	68.24	5.94	*86.43*
		500	**38.19**	1.84	46.79	3.06	61.59	5.56	*76.55*
	11	350	**48.26**	5.26	59.92	3.64	70.62	5.48	*121.56*
		400	**42.48**	3.10	55.01	4.16	72.36	5.15	*98.98*
		450	**39.08**	4.66	50.80	2.94	66.17	4.88	*91.14*
		500	**36.07**	2.68	46.53	2.65	59.24	5.09	*80.36*
	12	350	**55.13**	7.41	64.74	3.84	76.27	4.54	*133.18*
		400	**49.13**	2.71	59.99	3.53	77.57	5.43	*111.38*
		450	**46.62**	3.69	56.49	3.34	67.62	5.20	*102.49*
		500	**41.43**	2.39	49.16	3.03	68.49	5.81	*90.73*
9	10	350	57.91	4.84	**49.13**	3.38	90.52	7.16	*120.92*
		400	**43.81**	4.97	48.01	3.34	91.58	5.53	*94.14*
		450	**43.47**	4.44	44.49	2.85	89.88	5.15	*98.40*
		500	**38.33**	4.57	41.77	2.75	87.18	7.52	*89.40*
	11	350	54.41	6.42	**50.30**	3.43	90.99	7.75	*130.40*
		400	**43.31**	4.83	45.99	3.45	87.43	7.49	*100.62*
		450	**41.04**	3.21	43.00	3.29	87.33	5.56	*110.10*
		500	**36.91**	4.54	41.94	3.13	87.95	7.09	*93.43*
	12	350	58.49	6.95	**54.87**	3.31	98.53	8.45	*146.58*
		400	**46.33**	6.03	52.46	3.36	94.98	6.25	*113.51*
		450	**47.02**	3.68	49.42	3.09	93.35	6.23	*128.62*
		500	**42.63**	4.29	46.76	3.59	93.74	5.68	*107.28*

becomes larger and smaller, the number of episodes required for both two training stages increases and decreases, respectively. The number of episodes required to train an agent on the three previously mentioned instances are listed in Table 14. The other hyperparameters required for training are the same as given in Table 6.

Tables 15, 16, 17 and 18 illustrated the generalization performance of the agents trained on the smaller and the larger instances as well as of the agent trained in section "The training process of the agent", in which the agent trained on larger instance, the agent trained on smaller instance and the agent trained in section "The training process of the agent" are denoted as RL-Large, RL-Small and RL, respectively. The scheduling behavior of RL agents is characterized by a certain randomness, we repeat the scheduling of all the three agents independently 10 times on each instance. Therefore, the value of the Gap obtained by the agent presented in Tables 15-18 is the mean value over the 10 repetitions. Meanwhile the standard deviations are also given in the tables.

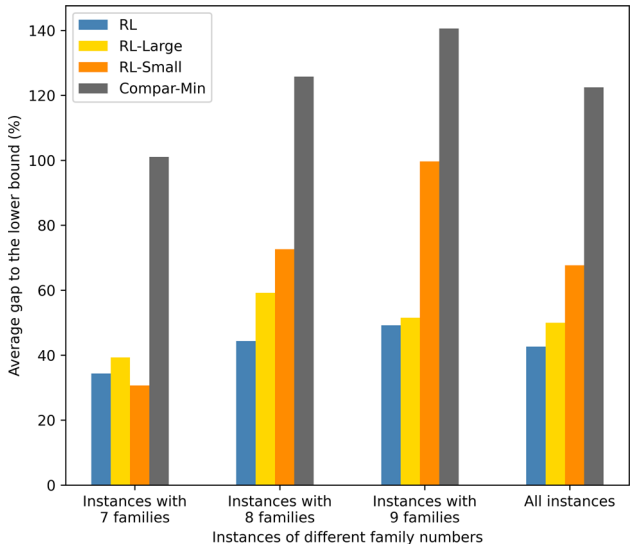


Fig. 21 The average $Gap(\%)$ obtained by RL agents trained on different instances and the the minimum $Gap(\%)$ of all comparative approaches over instances of different family numbers

References

Abu-Marrul, V., Martinelli, R., Hamacher, S., & Gribkovskaia, I. (2021). Matheuristics for a parallel machine scheduling problem with nonanticipatory family setup times: Application in the offshore oil and gas industry. *Computers & Operations Research, 128*, 105162.

Afzalirad, M., & Shafipour, M. (2018). Design of an efficient genetic algorithm for resource-constrained unrelated parallel machine scheduling problem with machine eligibility restrictions. *Journal of Intelligent Manufacturing, 29*(2), 423–437.

Anghinolfi, D., & Paolucci, M. (2007). Parallel machine total tardiness scheduling with a new hybrid metaheuristic approach. *Computers & Operations Research, 34*(11), 3471–3490.

Armentano, V., Yamashita, D. S., et al. (2000). Tabu search for scheduling on identical parallel machines to minimize mean tardiness. *Journal of Intelligent Manufacturing, 11*(5), 453–460.

Avalos-Rosales, O., Angel-Bello, F., & Alvarez, A. (2015). Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. *The International Journal of Advanced Manufacturing Technology, 76*(9), 1705–1718.

Azizoglu, M., & Kirca, O. (1998). Tardiness minimization on parallel machines. *International Journal of Production Economics, 55*(2), 163–168.

Báez, S., Angel-Bello, F., Alvarez, A., & Melián-Batista, B. (2019). A hybrid metaheuristic algorithm for a parallel machine scheduling problem with dependent setup times. *Computers & Industrial Engineering, 131*, 295–305.

Balin, S. (2011). Non-identical parallel machine scheduling using genetic algorithm. *Expert Systems with Applications, 38*(6), 6814–6821.

Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning. *Proceedings of the 26th annual international conference on machine learning*, (pp. 41–48).

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks, 5*(2), 157–166.

Biskup, D., Herrmann, J., & Gupta, J. N. D. (2008). Scheduling identical parallel machines to minimize total tardiness. *International Journal of Production Economics, 115*(1), 134–142.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. arXiv:1606.01540.

Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder–decoder approaches. arXiv:1409.1259.

Cochran, J. K., Horng, S.-M., & Fowler, J. W. (2003). A multi-population genetic algorithm to solve multi-objective scheduling problems for parallel machines. *Computers & Operations Research, 30*(7), 1087–1102.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science, 14*(2), 179–211.

Fang, K.-T., & Lin, B. M. (2013). Parallel-machine scheduling to minimize tardiness penalty and power cost. *Computers & Industrial Engineering, 64*(1), 224–234.

Gavett, J. W. (1965). Three heuristic rules for sequencing jobs to a single production facility. *Management Science, 11*(8), 166–176.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics, 5*, 287–326.

Guo, L., Zhuang, Z., Huang, Z., & Qin, W. (2020). Optimization of dynamic multi-objective non-identical parallel machine scheduling with multistage reinforcement learning. *2020 IEEE 16th international conference on automation science and engineering (CASE)*, (pp. 1215–1219).

Kayhan, B. M., & Yildiz, G. (2021). Reinforcement learning applications to machine scheduling problems: A comprehensive literature review. *Journal of Intelligent Manufacturing, 34*, 1–25.

Kim, Y.-D., Joo, B.-J., & Choi, S.-Y. (2010). Scheduling wafer lots on diffusion machines in a semiconductor wafer fabrication facility. *IEEE Transactions on Semiconductor Manufacturing, 23*(2), 246–254.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980 .

Lang, S., Behrendt, F., Lanzerath, N., Reggelin, T., & Müller, M. (2020). Integration of deep reinforcement learning and discrete-event simulation for real-time scheduling of a flexible job shop production. *Winter Simulation Conference (WSC), 2020*, 3057–3068.

Lang, S., Kuetgens, M., Reichardt, P., & Reggelin, T. (2021). Modeling production scheduling problems as reinforcement learning environments based on discrete-event simulation and openai gym. *IFAC-PapersOnLine, 54*(1), 793–798.

Lee, Z.-J., Lin, S.-W., & Ying, K.-C. (2010). Scheduling jobs on dynamic parallel machines with sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology, 47*(5), 773–781.

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.

Luo, S. (2020). Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Applied Soft Computing, 91*, 106208.

Paeng, B., Park, I.-B., & Park, J. (2021). Deep reinforcement learning for minimizing tardiness in parallel machine scheduling with sequence dependent family setups. *IEEE Access, 9*, 101390–101401.

Pickardt, C. W., & Branke, J. (2012). Setup-oriented dispatching rules-a survey. *International Journal of Production Research, 50*(20), 5823–5842.

Potts, C. N., & Van Wassenhove, L. N. (1985). A branch and bound algorithm for the total weighted tardiness problem. *Operations Research, 33*(2), 363–377.

Rajendran, C., & Holthaus, O. (1999). A comparative study of dispatching rules in dynamic flowshops and jobshops. *European Journal of Operational Research, 116*(1), 156–170.

Rodríguez, M. L. R., Kubler, S., de Giorgio, A., Cordy, M., Robert, J., & Le Traon, Y. (2022). Multi-agent deep reinforcement learning

based predictive maintenance on parallel machines. *Robotics and Computer-Integrated Manufacturing, 78*, 102406.

Rolf, B., Reggelin, T., Nahhas, A., Lang, S., & Müller, M. (2020). Assigning dispatching rules using a genetic algorithm to solve a hybrid flow shop scheduling problem. *Procedia Manufacturing, 42*, 442–449.

Schaller, J. E. (2014). Minimizing total tardiness for scheduling identical parallel machines with family setups. *Computers & Industrial Engineering, 72*, 274–281.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. *International conference on machine learning*, (pp. 1889–1897).

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv:1707.06347 .

Shin, H. J., & Leon, V. J. (2004). Scheduling with product family set-up times: An application in TFT LCD manufacturing. *International Journal of Production Research, 42*(20), 4235–4248.

Sigtia, S., Benetos, E., Cherla, S., Weyde, T., Garcez, A., & Dixon, S. (2014). RNN-based music language models for improving automatic music transcription. *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR)*, (pp. 53–58).

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017). Mastering the game of go without human knowledge. *Nature, 550*(7676), 354–359.

Sun, S., Cao, Z., Zhu, H., & Zhao, J. (2019). A survey of optimization methods from a machine learning perspective. *IEEE Transactions on Cybernetics, 50*(8), 3668–3681.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press.

Tassel, P., Gebser, M., & Schekotihin, K. (2021). A reinforcement learning environment for job-shop scheduling. arXiv:2104.03760.

van der Ham, R. (2018). salabim: Discrete event simulation and animation in python. *Journal of Open Source Software, 3*(27), 767.

van der Zee, D.-J. (2015). Family-based dispatching with parallel machines. *International Journal of Production Research, 53*(19), 5837–5856.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature, 575*(7782), 350–354.

Wang, L., Pan, Z., & Wang, J. (2021). A review of reinforcement learning based intelligent optimization for manufacturing scheduling. *Complex System Modeling and Simulation, 1*(4), 257–270.

Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE, 78*(10), 1550–1560.

Wilbrecht, J. K., & Prescott, W. B. (1969). The influence of setup time on job shop performance. *Management Science, 16*(4), 274–280.

Wu, Y., & Tian, Y. (2016). Training agent for first-person shooter game with actor-critic curriculum learning.

Yin, W., Kann, K., Yu, M., & Schütze, H. (2017). Comparative study of cnn and rnn for natural language processing. arXiv:1702.01923.

Ying, K.-C., & Cheng, H.-M. (2010). Dynamic parallel machine scheduling with sequence-dependent setup times using an iterated greedy heuristic. *Expert Systems with Applications, 37*(4), 2848–2852.

Yuan, B., Jiang, Z., & Wang, L. (2016). Dynamic parallel machine scheduling with random breakdowns using the learning agent. *International Journal of Services Operations and Informatics, 8*(2), 94–103.

Yuan, B., Wang, L., & Jiang, Z. (2013). Dynamic parallel machine scheduling using the learning agent. *IEEE International Conference on Industrial Engineering and Engineering management, 2013*, 1565–1569.

Zeidi, J. R., & MohammadHosseini, S. (2015). Scheduling unrelated parallel machines with sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology, 81*(9), 1487–1496.

Zhang, C., Liu, Y., Wu, F., Tang, B., & Fan, W. (2020). Effective charging planning based on deep reinforcement learning for electric vehicles. *IEEE Transactions on Intelligent Transportation Systems, 22*(1), 542–554.

Zhang, Z., Zheng, L., Hou, F., & Li, N. (2011). Semiconductor final test scheduling with sarsa (λ, k) algorithm. *European Journal of Operational Research, 215*(2), 446–458.

Zhang, Z., Zheng, L., Li, N., Wang, W., Zhong, S., & Hu, K. (2012). Minimizing mean weighted tardiness in unrelated parallel machine scheduling with reinforcement learning. *Computers & Operations Research, 39*(7), 1315–1324.

Zhang, Z., Zheng, L., & Weng, M. X. (2007). Dynamic parallel machine scheduling with mean weighted tardiness objective by q-learning. *The International Journal of Advanced Manufacturing Technology, 34*(9), 968–980.

Zhou, D., Jia, R., & Yao, H. (2021). Robotic arm motion planning based on curriculum reinforcement learning. *2021 6th International Conference on Control and Robotics Engineering (ICCRE)*, (pp. 44–49).

Zhou, L., Zhang, L., & Horn, B. K. (2020). Deep reinforcement learning-based dynamic scheduling in smart manufacturing. *Procedia CIRP, 93*, 383–388.