Journal of Intelligent Manufacturing
https://doi.org/10.1007/s10845-024-02454-8

Design patterns of deep reinforcement learning models for job shop scheduling problems

Shiyong Wang[1] · **Jiaxian Li**[1] · **Qingsong Jiao**[2] · **Fang Ma**[3]

Received: 21 November 2023 / Accepted: 25 June 2024

Abstract
Production scheduling has a significant role when optimizing production objectives such as production efficiency, resource utilization, cost control, energy-saving, and emission reduction. Currently, deep reinforcement learning-based production scheduling methods achieve roughly equivalent precision as the widely used meta-heuristic algorithms while exhibiting higher efficiency, along with powerful generalization abilities. Therefore, this new paradigm has drawn much attention and plenty of research results have been reported. By reviewing available deep reinforcement learning models for the job shop scheduling problems, the typical design patterns and pattern combinations of the common components, i.e., agent, environment, state, action, and reward, were identified. Around this essential contribution, the architecture and procedure of training deep reinforcement learning scheduling models and applying resultant scheduling solvers were generalized. Furthermore, the key evaluation indicators were summarized and the promising research areas were outlined. This work surveys several deep reinforcement learning models for a range of production scheduling problems.

Keywords Production scheduling · Reinforcement learning · Smart manufacturing · Industry 4.0

Introduction

In a discrete manufacturing environment, producing a workpiece generally requires a set of machines to process a sequence of operations. An intuitive and efficient solution to this production requirement is the flow line. In a flow line, each operation is assigned a dedicated machine and these machines are arranged in the same sequence as the operations. Consequently, the workpieces of the same type can be naturally queued and processed one by another by the flow line. However, when a set of workpieces differing from each other in operation sequence and operation quantity, the flow line is no longer feasible as it hardwires the sequence. Although a flexible material handling system can reconfigure machine sequences dynamically, the workpieces coming from different sequences will scramble for a machine if they own the same operations that can be processed by the machine.

The above-mentioned problem has been recognized as the job-shop scheduling problem (JSSP). The key features of JSSP come from both the job side and the machine side:

1) The operation sequence of a job (e.g., a workpiece) is predefined and should be strictly obeyed;
2) A scheduling instance consists of a set of jobs different in operation sequence and quantity;
3) A machine can only process one operation at a time;
4) A machine is not allowed to preempt when processing an operation;
5) All machines turn on at the start of scheduling;
6) Transportation time of jobs and setup time of machines are negligible.

The scheduling solver is responsible for arbitrating the competition by determining the processing order of the competitive operations. Despite the constraints introduced by jobs and machines, multiple feasible solutions to a scheduling instance still exist, as the operations sharing the same

✉ Qingsong Jiao
739960684@qq.com

[1] School of Mechanical and Automotive Engineering, South China University of Technology, Guangzhou 510640, China

[2] Department of Electronic Business, South China University of Technology, Guangzhou 510640, China

[3] China National Electric Apparatus Research Institute Co., Ltd, Guangzhou 510300, China

Published online: 20 July 2024

machine can be queued in different ways if they come from different jobs. However, the feasible scheduling solutions generally differ in performance metrics such as makespan, tardiness, machine utilization, energy consumption, and carbon emission. This leaves space for a scheduling solver to optimize a given performance indicator or indicator combination. Therefore, production scheduling is an important production optimization technique.

JSSPs are inherently a subclass of NP-hard combinatorial optimization problems (Garey et al., 1976). For simple and small-scale problems, optimal solutions can be obtained using either the exact mathematical models or the exhaustive methods. On the other hand, approximate methods, e.g., heuristic rules (Panwalkar & Iskander, 1977) and meta-heuristics (Kato et al., 2018; Parjapati & Jain, 2015), are widely used to search for suboptimal solutions to complicated or large-scale problems as they can trade off efficiency and performance. The above-mentioned approximate methods are highly adaptable, but they cannot generalize solving experience to new problems, which means the solving experience cannot be reused to facilitate the solving of a different scheduling instance. To overcome this drawback, deep reinforcement learning (DRL) (Arulkumaran et al., 2017) based scheduling methods have attracted the researchers' interest due to their outstanding advantages, more specifically, fast computation and strong generalization ability (Li et al., 2023).

The DRL models have some common components, i.e., agent, environment, state, action, and reward. These components need specific designs when constructing DRL models for JSSP problems. In this paper, typical design patterns for the DRL scheduling models were identified and compared by reviewing representative literature. A DRL scheduling model needs training to become an applicable DRL scheduling solver. Therefore, the architecture and procedure of training deep reinforcement learning scheduling models and applying resultant scheduling solvers were established as well. Furthermore, the statistical analysis of the typic design patterns and pattern combinations was performed to highlight the popularity. The key evaluation indicators were summarized and the promising research areas were outlined to promote the research and application of DRL-based scheduling methods. This review provides insight to perfect the design of DRL scheduling models for the JSSP problems and inspires innovative DRL scheduling models for a wider range of scheduling problems.

Review process

This section describes in detail the review process including the objective, the search methods, the inclusion and exclusion criteria, and the search results. The results of this section screen out several related recent high-quality research articles for the follow-up analysis, which in turn enables us to answer the objective questions in the conclusion section.

Objective (review questions)

The question of this review is "What are the optional design patterns of DRL models when solving the JSSPs?" and it aims to shape a framework to facilitate the design and comparison of DRL models used to solve the JSSPs. Specific objectives are:

To recognize which design patterns have been applied to each component of a DRL model that is specially designed for solving the JSSPs.

To rank the popularity of the pattern combinations of the DRL models that are specially designed for solving the JSSPs.

To outline the further application of DRL models to solve production-related problems.

Search methods for identification of studies

Aveyard et al. (2016) state that it is important to use a systematic search strategy to retrieve all the relevant materials to answer the review questions. Similarly, Smith et al. (2011) also point out that an appropriate literature search method is the foundation of correct information retrieval and can determine whether the systematic review is successful or not. This comprehensive search method is not only essential to guarantee that the review author has identified and located the related primary research as many as possible, but can ensure that the sample is as unbiased and transparent as possible as well (Bettany-Saltikov, 2012). Therefore, both electronic searches and hand searches were applied to maximize the quantity of literature.

Search terms

In order to search as widely as possible, it is necessary to identify as many synonyms that have the same meaning as the key terms as possible (Bettany-Saltikov, 2012). What is more, wildcard characters are also recommended to be used to account for different spellings or terminologies to search as much literature as possible (Aromataris & Riitano, 2014).

As shown in Table 1, the search terms were divided into three parts. The "Methods" terms listed general concepts related to deep reinforcement learning, the "Algorithm" terms presented the implementation varieties of the DRL methods, and the "Problems" terms limited the application of DRL to the production scheduling problems. The keywords in each column were used with the Boolean operator OR, and then three columns were combined as "(Method OR Algorithm) AND Research subject".

Table 1 Search terms

Methods		Algorithms		Problems
deep reinforcement learning	**OR**	deep q-network	**AND**	production scheduling
DRL		DQN		shop scheduling
reinforcement learning		proximal policy optimization		job-shop scheduling problem
RL		PPO		JSSP
machine learning		actor-critic		flexible job-shop scheduling
artificial intelligence		AC		FJSP
intelligent		DDPG		

Scientific and technological database

The following electronic databases were searched: Web of Science, EI-Village, SCOPUS, IEEE/IEE, Springer, and Wiley, as these databases contain a variety of engineering-related articles and are accessible from the authors' facility.

Backward/forward search

Based on the review structure proposed by Webster and Watson (2002), backward/forward search is an important supplement to keyword search, capable of identifying interdisciplinary literature that extends beyond the scope of a user-defined search. Therefore, after applying the inclusion and exclusion criteria, the remaining papers were processed using a backward/forward search to find more related papers.

Inclusion and exclusion criteria

Inclusion criteria, also termed eligibility criteria, indicate what specific traits a study should have if it can be included in the review. By contrast, exclusion criteria refer to the attributes that make the studies unqualified for inclusion (Boland et al., 2017). The clear inclusion and exclusion criteria are beneficial for focusing on the review question and selecting studies more appropriately. For this review, studies are selected based on the following inclusion and exclusion criteria.

Regions, languages, and published date

This review did not limit the regions of the studies; therefore, studies all over the world were considered within the scope of inclusion. However, it was impractical to translate papers published in other languages into English. Considering the significant developments of DRL since 2013, only English literature published between 2013 and 2023 was included.

2.3.2 Types of studies

The papers categorized as "Article", "Conference paper" or "Meeting" were included to ensure a thorough investigation of original research findings, but those published in non-peer-reviewed journals or conferences were excluded to guarantee the research quality.

Types of methods and algorithms

DRL and its related algorithms (e.g., Deep Q-learning) are more sophisticated than traditional reinforcement learning and the related algorithms (e.g., Q-learning). Due to the high complexity of JSSPs, only papers that use DRL and its related algorithms were included.

Types of problems

JSSP is a specific type of production scheduling problem that can cover the majority of current production settings. Although the more complex flexible job-shop scheduling problem (FJSP) has already been identified, very few papers use DRL to solve FJSPs. Therefore, only JSSPs were included.

Types of data

To enable comparison and verification, the papers using publicly available or generated datasets were included, and those validated only in specific production scenarios were excluded.

Results of the search

A total of 67,081 studies were retrieved through several different databases (149 from SCOPUS, 8103 from EI-Village, 186 from Web of Science, 4726 from IEEE/IEE, 966 from Springer, and 52,951 from Wiley). As shown in Fig. 1 (adapted from the PRISMA Group (Moher et al., 2009), after applying the inclusion and exclusion criteria and backward/forward search, a total of 44 papers remained.

DRL-based scheduling for job-shop scheduling problem

The mathematical model of JSSP and the execution process of DRL scheduling models are presented in this section. The terms and symbols can then be used in the following

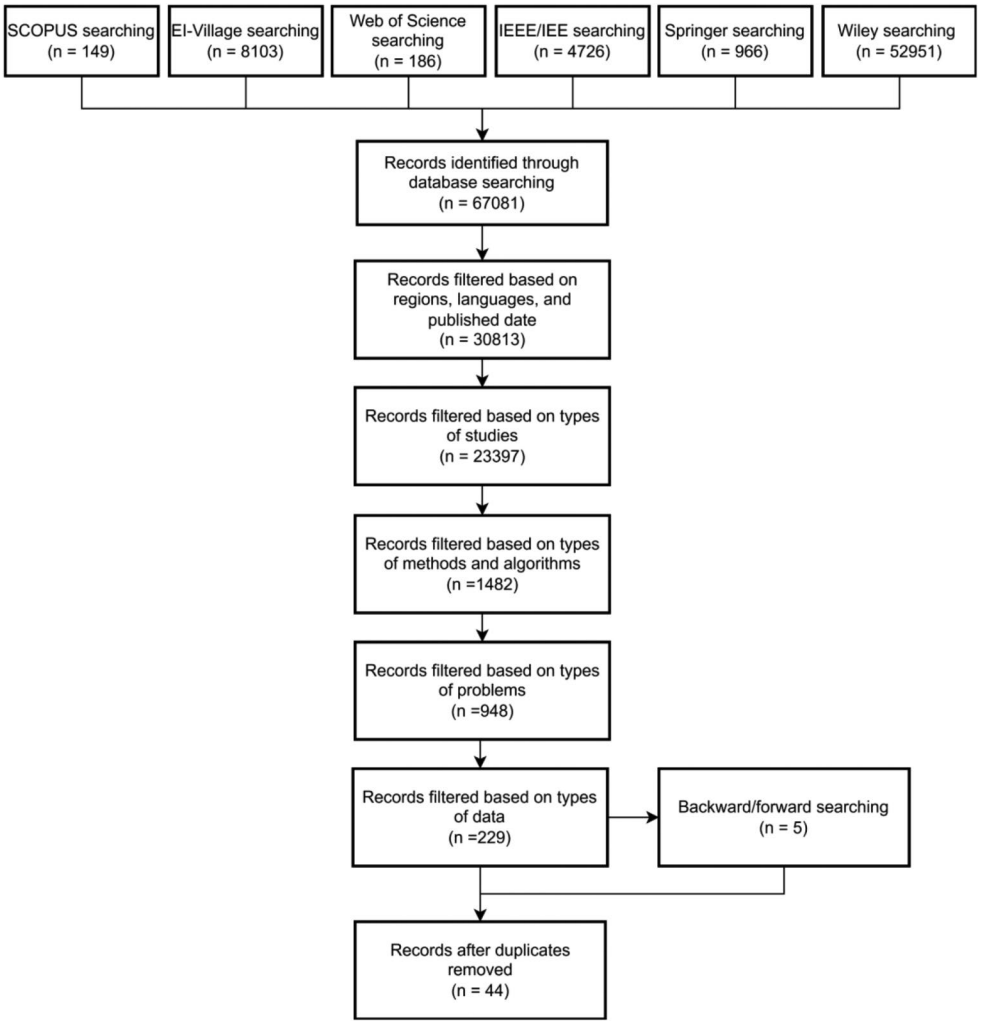


Fig. 1 Paper screening

sections. In this section, the formulation of JSSP is introduced firstly, to make sense of the scheduling concepts and principles that are used subsequently. Then a more detailed description of the DRL-based solving model for the scheduling problem formulated above is provided.

Formulation of job-shop scheduling problem

A JSSP scheduling instance deals with a job set $\mathcal{J}$ consisting of $|\mathcal{J}|$ jobs. A job $J_i \in \mathcal{J}$ has n_i operations, while $O_{i,j}$ denotes the j th operation of J_i. Therefore, all the operations belonging to $\mathcal{J}$ form an operation set $\mathcal{O}$ with $|\mathcal{O}| = \sum_{i=1}^{|\mathcal{J}|} n_i$ operations. A set of machines, $\mathcal{M}$, comprising $|\mathcal{M}|$ machines, is prepared to process the operations. Therefore, the JSSP size is generally defined as $|\mathcal{J}| \times |\mathcal{M}|$. The scheduling solvers aim to determine the start time $S_{i,j}$, and the completion time $C_{i,j} = S_{i,j} + p_{i,j}$ for each operation $O_{i,j}$, where the $p_{i,j}$ denotes the processing time of operation $O_{i,j}$.

Taking a 3×3 JSSP as an example, the M_k for and the $p_{i,j}$ of each operation are given in Table 2. The disjunctive graph shown in Fig. 2(a) utilizes the directed arcs to illustrate the operation order of the jobs. The undirected arcs are used to present the share of machines among different

Table 2 A JSSP scheduling instance

Job	Due time	Operation	Available M_k and corresponding $p_{i,j}$		
			M_1	M_2	M_3
J_1	20	$O_{1,1}$	2	--	--
		$O_{1,2}$	--	3	--
		$O_{1,3}$	--	--	5
J_2	15	$O_{2,1}$	3	--	--
		$O_{2,2}$	--	--	3
		$O_{2,3}$	--	3	--
J_3	25	$O_{3,1}$	--	3	--
		$O_{3,2}$	4	--	--
		$O_{3,3}$	--	--	3

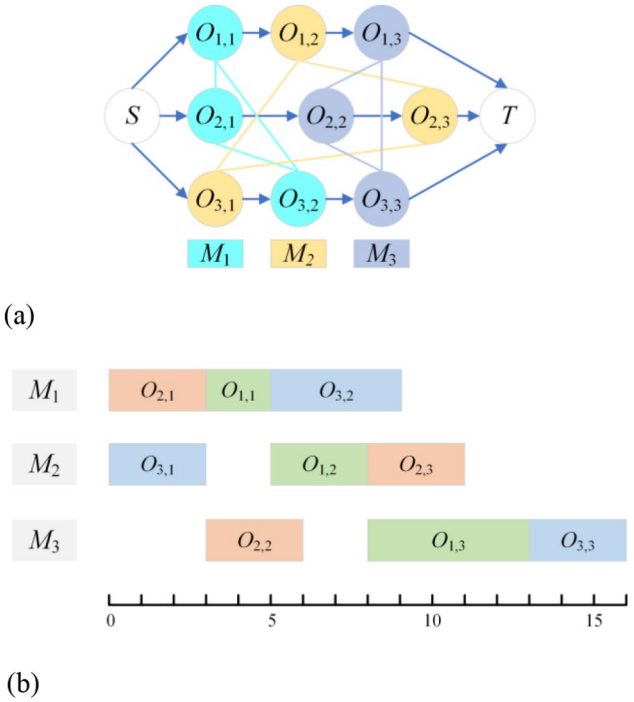


Fig. 2 Graph representation and scheduling solution of the JSSP instance in Table 2. (**a**) Disjunctive graph for the inter-operation relationship. (**b**) Scheduling solution Gantt chart

operations. Once the scheduling solver queues the sharing operations, the $S_{i,j}$ and $C_{i,j}$ of $O_{i,j}$ can be determined by a specific decoding algorithm. The result is generally visualized via a Gantt chart, as shown in Fig. 2(b), where the makespan, $C_{\max} = \max_i C_{i,n_i}$, is 16, which is the end time of the latest finished operation, $O_{3,3}$.

Architecture and procedure of deep reinforcement learning-based scheduling

A DRL scheduling model features a Markov decision process (MDP) (Sutton & Barto, 2018) with parametric equations, and the parameters are assigned values during model training process. Therefore, the DRL scheduling model becomes an applicable scheduling solver after training. The architecture (Li et al., 2023) and execution process of a DRL scheduling solver is illustrated in Fig. 3. A cycle between the Agent and the Environment is established through five links: perception, analysis, decision making, control, and execution. The Agent obtains raw information from the Environment via the perception module and concludes the state and reward through the analysis module. In the next step, the agent selects an action based on the state via the decision-making module and supervises the execution of the chosen action through the control and execution modules. Therefore, the Agent interacts with the Environment repeatedly updating the state, action, and reward; this way,

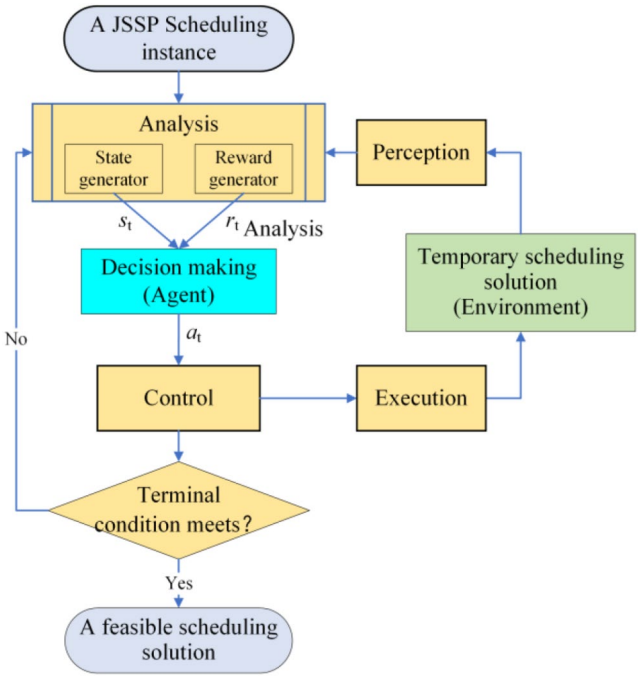


Fig. 3 The DRL production scheduling process

a feasible scheduling solution is generated after several iterations.

The Environment is a temporary scheduling solution (TSS), i.e., an initial scheduling solution, an incomplete scheduling solution, or a complete scheduling solution under optimization. The state is the Environment description while the current state s_t is generated by the state generation function included in the analysis link based on the data sampled from the Environment:

$$s_t = s\,(sensations) \tag{1}$$

where $sensations$ denotes the data sampled via the perception link.

The role of the decision link is to select an action a_t corresponding to the state s_t using the policy function π_θ:

$$p\,(a_t) = \pi_\theta\,(a_t|\,s_t) \tag{2}$$

where $p\,(a_t)$ is the probability of the action a_t being selected. The policy function π_θ is a probability distribution function with respect to state s_t and action a_t. After the action a_t is executed on the Environment, the TSS will be updated through the control and execution links. Finally, the state is transferred from the current s_t to the next s_{t+1}, and the reward r_t is produced.

The reward r_t indicates the change of the scheduling objective between the adjacent decision steps t and $t-1$

generated by the reward generation function in the analysis link:

$$r_t = r\left(sensations\right) \tag{3}$$

As shown in Eq. 4, the next state s_{t+1} is determined with the probability $p\left(s_{t+1}\right)$ defined as:

$$p\left(s_{t+1}\right) = p\left(s_{t+1}|\, s_t, a_t\right) \tag{4}$$

where $p\left(s_{t+1}|\, s_t, a_t\right)$ is a probability distribution function of components s_t, a_t, and s_{t+1}, reflecting the randomness of TSS. It should be noted that $p\left(s_{t+1}|\, s_t, a_t\right)$ reflects the inherent Environment characteristic – it is not controllable from the outside, i.e., by the Agent. Therefore, the Agent must find a well-matched policy other than change $p\left(s_{t+1}|\, s_t, a_t\right)$. By iterating the above-presented process until the stop criterion is satisfied, a feasible scheduling solution will be generated.

The training process resembles the above-described application process. However, training a DRL scheduling model does not focus on finding a feasible solution for a specific scheduling instance. Instead, it starts with a randomly initialized policy function π_θ, which is continuously optimized through the Agent-Environment interaction under several scheduling instances to sample a large amount of data (s_t, a_t, r_t, s_{t+1}). Consequently, these sampled data are used to optimize π_θ by updating the parameter set θ :

$$\theta_{i+1} = \theta_i + f\left(s_t, a_t, r_t, s_{t+1}\right) \tag{5}$$

By repeatedly sampling data and updating the parameters, θ will converge towards the optimal value θ^*, which corresponds with the optimal policy π_{θ^*}. In the application stage, π_{θ^*} is directly adopted, remaining unchanged, to generate a feasible solution for a given scheduling instance. However, as the optimal policy π_{θ^*} is obtained using training instances that differ from the application instances, π_{θ^*} may present a suboptimal solution rather than the optimal one.

Typical design patterns of DRL components

A DRL scheduling model mainly comprises five components: an Agent, an Environment, a state set, an action set, and a reward function. The typical design patterns of each component are described in the following subsections.

Design patterns of the Agent

The DRL is a sequential decision process consisting of multiple decision steps. Combining the state and action of each step results in a trajectory τ:

$$\tau = \left(s_1, a_1, s_2, a_2, \ldots, s_{T-1}, a_{T-1}, s_T\right) \tag{6}$$

where T denotes the termination step; s_1 and s_T are the initial and the terminate states, respectively.

The DRL optimization objective is to obtain the maximum cumulative reward $R\left(\tau\right)$:

$$R\left(\tau\right) = r_1 + r_2 + \cdots + r_{T-1} = \sum_{t=1}^{T-1} r_t \tag{7}$$

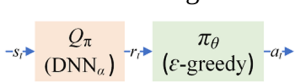
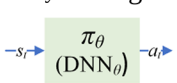
Currently, DRL algorithms can be divided into three categories: value-based (Wang et al., 2016; Van et al., 2016), policy-based (Sutton et al., 1999), and actor-critic (Bhatnagar et al., 2009). The design patterns of the agent are closely related to the DRL algorithm classification.

Value-based agent

A value function refers to either the state value function $V_\pi\left(s_t\right)$ or the state-action value function $Q_\pi\left(s_t, a_t\right)$, indicating the $R\left(\tau\right)$ expectation obtained from the state s_t and the state-action pair $\left(s_t, a_t\right)$, respectively. Therefore, $V_\pi\left(s_t\right)$ can be used for the comparison of states, and $Q_\pi\left(s_t, a_t\right)$ is further used for the action comparison of the same state.

The value-based agent consists of two parts: one is a policy function (e.g., the ϵ-greedy policy) and the other is a value function, as shown in the 'Structure' row of Table 3. The ϵ-greedy policy selects the action for which $Q_\pi\left(s_t, a_t\right)$

Table 3 Comparison of the DRL Agent

	Value-based agent	Policy-based agent	Actor-Critic agent
Structure			
Policy	Random policy & Value function as a DNN	Policy function as a DNN	Both policy function & Value function as DNNs
Learning style	Implicit learning	Explicit learning	Explicit learning
Typical algorithms	DQN (Mnih et al., 2015)	PPO (Schulman et al., 2017)	DDPG (Lilicrap et al., 2015), A3C (Mnih et al., 2016), A2C
Applicable problems	Discrete problems	Discrete problems	Discrete/Continuous problems

outputs the maximum value with probability 1-ϵ, while randomly picking up an action with probability ϵ. The greater ϵ value entices the Agent to explore unseen states, while the lower value urges the Agent to exploit the known optimal states. The optimal policy is eventually obtained by alternatively tuning ϵ and updating the value function represented as a deep neural network (DNN).

Policy-based agent

The value functions are not used in this design pattern. A DNN is designed to fit the policy function instead of a ϵ-greedy like random policy, as shown in Table 2 (see the 'Structure' row). Furthermore, the DNN is optimized by maximizing $R(\tau)$.

Actor-critic agent

This design pattern can be regarded as a synthesis of the value-based and the policy-based patterns. It uses two DNNs to fit the policy function and the value function respectively, and the two DNNs are updated alternately to achieve optimal output (see Table 2, 'Structure' row).

Comparison and discussion

A comparison of the three types of agents is shown in Table 2. Both the value-based and the actor-critic agent require two parts: a policy function and a value function. However, the former mechanism uses a random policy, while the latter uses a DNN to approximate the policy function.

Both the policy-based and the actor-critic agents use a DNN to approximate the policy function. However, they are trained differently; the former optimizes the policy function using the trajectory dependent $R(\tau)$ as the objective, while the latter utilizes the output of a value function, which is in turn optimized via the trajectory insensitive data (s_t, a_t, r_t, s_{t+1}).

Design patterns of the environment

The Environment is represented by the temporary scheduling solution, which updates and evolves during the DRL scheduling process; a deliverable feasible scheduling solution is determined when the stop criterion is satisfied. Currently, there are two primary design patterns of the Environment: partial solution-based Environment versus whole solution-based Environment.

Partial solution-based environment

The DRL scheduling model successively queues the operations of a given scheduling instance. As shown in Fig. 4(a), one operation is selected in the first decision step and another operation is selected in the second step, and so forth. Therefore, $|\mathcal{O}|$ steps are needed to determine a feasible scheduling solution. The partial solution design pattern is adopted by most studies (Wang et al., 2021b); therefore, design patterns of state, action, and reward are only elaborated for this pattern if not specifically stated.

Whole solution-based environment

A complete and feasible TSS is initialized randomly (Palombarini & Martinez, 2021) or by some traditional scheduling methods, e.g., metaheuristic algorithms (Gu et al., 2023) and heuristic rules (Chen & Tian, 2018). At each decision step, DRL scheduling model generates a new TSS by modifying the previous one, and both solutions are complete feasible solutions, as shown in Fig. 4(b). To solve the rescheduling problem, Palombarini and Martinez (2021) randomly initialized the Gantt chart, and then they used DRL to modify the positions and the start time of the selected operations. Magalhães et al. (2021) initialized the operation sequence with a meta-heuristic algorithm and used DRL to select operations and change their position to update the TSS.

Comparison and discussion

The partial solution-based DRL scheduling features an episodic task with a definite number of cycles, while the whole solution-based DRL scheduling is a continuous task. The number of cycles is determined by either the human

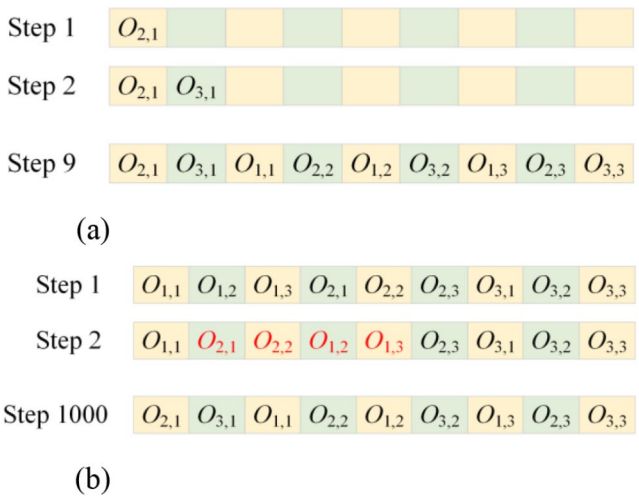


Fig. 4 Design pattern illustrations of the DRL Environment. (**a**) Partial solution-based Environment. (**b**) Whole solution-based Environment

experience or the testing results. The solution is incomplete during the partial solution-based DRL scheduling process; thus, the accurate performance indicators cannot be determined, making the reward design more difficult.

In contrast, all the TSSs in the whole solution-based DRL scheduling process are both complete and feasible, easing the reward function design. Additionally, the classical production scheduling methods such as genetic algorithms, simulated annealing algorithms, and distribution estimation algorithms, utilize complete TSSs. Therefore, it is easy to integrate these algorithms with the whole solution-based DRL scheduling (Du et al., 2022).

Design patterns of the state

In general, the MDP requires that the current state can fully describe the Environment evolution process. Although this requirement does not have to be strictly satisfied, the state design will greatly affect the decision-making quality. Currently, there are three main state design patterns: matrix-based, statistic-based, and graph-based.

Matrix-based state

As shown in Fig. 5(a), the information related to the TSS is classified and stored in several matrices, mainly describing the job and machine features. Matrices resemble the RGB channels of an image, meaning that the high-dimension state features can be extracted via convolutional neural networks (Liu et al., 2020; Wang et al., 2021a; Wu & Yan, 2023). Han and Yang (2020) used three matrices to describe the information regarding the processing time, scheduling results, and machine utilization; matrix height and width corresponded to the numbers of jobs and machines, respectively.

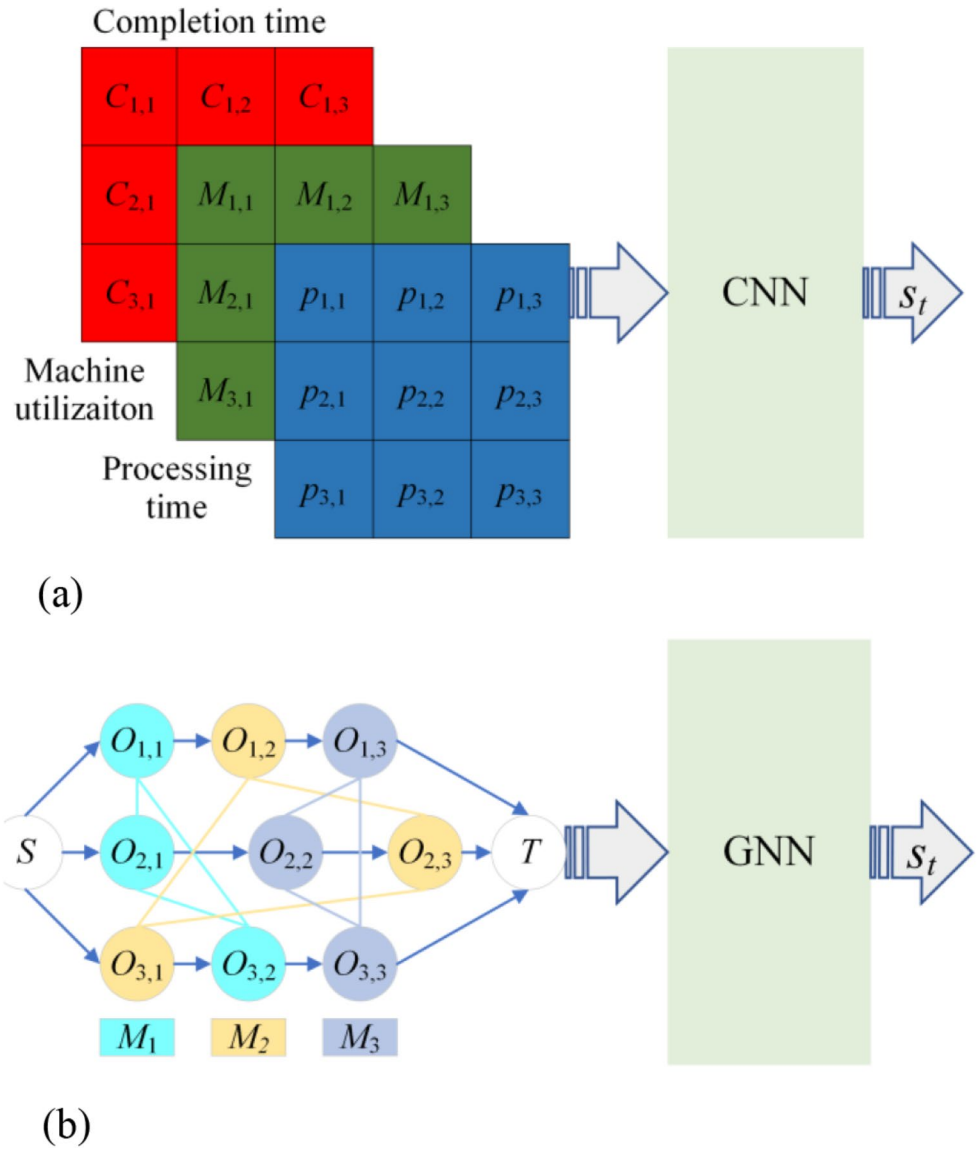


Fig. 5 Design pattern illustrations of the DRL state. (**a**) Matrix-based state representation and processing. (**b**) Graph-based state representation and processing

Statistic-based state

A set of statistical indicators describing the static and dynamic job and machine attributes are defined as states (Chang et al., 2022; Han & Yang, 2021; Luo et al., 2021b; Xu et al., 2022). Statistical indicators are roughly divided into three categories:

Gross indicators reflect the total quantity of jobs, machines, and other production factors in the scheduling environment within a certain period. Indicators such as the total number of machines, the total number of jobs, the total processing time, and the total tardiness, among others, reflect the initial overall scale of the scheduling problem. Furthermore, indicators such as the number of completed jobs, the number of remaining operations, the remaining job processing time, their delay time, and machine payloads reflect the current scheduling environment in terms of work hours and load distribution.

Relative indicators like the completion rate of each job, the delay rate, and the total machine utilization rate vary with the scheduling process. They reflect the scheduling progress in the form of ratios.

Average indicators such as the average processing time of remaining operations, the average job completion rates, and the average machine utilization rate balance the impact of the problem scale on dynamic indicators.

Statistical indicators and matrices can be used together. For example, Luo et al. (2021a) stated that statistical indicators could describe numerical information regarding the state, while the matrix could describe constraints among production resources.

Graph-based state

Generally, the whole solution-based DRL scheduling model uses the Gantt chart as the state (Ni et al., 2021), while its partial counterpart utilizes the disjunctive graph or its variants (Liu & Huang, 2023; Song et al., 2023). The graph-based state pattern extracts node and graph features as the high-dimension state features (Seito & Munakata, 2020; Zeng et al., 2022) based on graph learning methods, e.g., graph neural network (GNN) and attention mechanism, as shown in Fig. 5(b). Compared to the multilayer perceptron (MLP), GNN is more suitable for complicated problems since it provides better implicit inductive bias in terms of the extraction of state features (Hameed & Schwung, 2020).

Comparison and discussion

The matrix-based design pattern is intuitive and easy to calculate; however, the information type limits make it difficult to represent states of complicated problems. Additionally, the matrix size is related to the scheduling problem scale hindering the generalization of the DRL scheduling model to a scheduling instance with the different size.

The statistic-based design pattern describes various attributes which are quick to calculate. Compared to matrices, statistical indicators have the potential to fully represent the states of complicated scheduling problems. However, the redundant use of statistical indicators will waste computational resources. Further, different problems might require different indicators, weakening the generalization ability of statistic-based state design. Unfortunately, there are no reliable methods for indicator selection, meaning that statistical indicators are designed mainly based on experience and intuition.

Compared to the statistic-based design, the graph-based design pattern extracts rich state features from the graph-structured data. This way, the improper selection of statistical indicators is avoided, which contributes to the DRL generalization; however, two challenges remain. Firstly, there are no available graph presentations for complicated problems, e.g., flexible job-shop scheduling problem (Fattahi et al., 2007), since the disjunctive graph has limited expressiveness. Secondly, the adoption of graph neural networks requires significant computational resources and time during the training stage (Park et al., 2021b).

Design patterns of the action

Actions are used to update the TSS. In the partial solution-based DRL scheduling, executing an action corresponds to selecting an operation for TSS, while in the whole solution-based DRL scheduling, executing an action results in a new TSS. Currently, there are four action design patterns: rule-based, operation-based, attribute-based, and graph-based.

Rule-based action

In this pattern (Fig. 6(a)), the policy π_θ outputs the possibility of each rule for the state s_t :

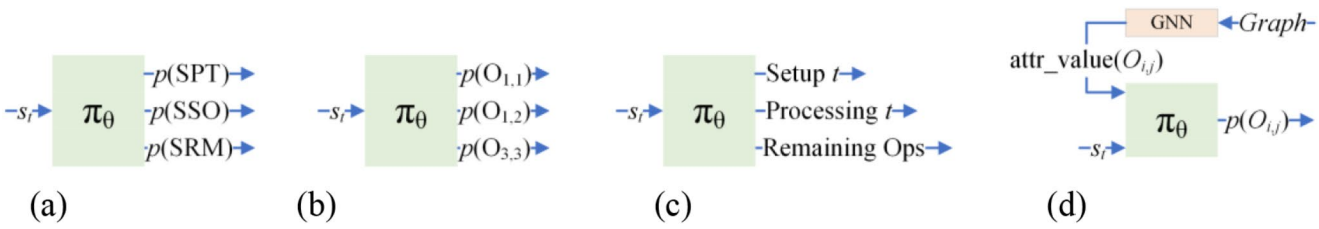


Fig. 6 Design pattern illustrations of the DRL action. (**a**) Rule-based (**b**) Operation-based (**c**) Attribute-based (**d**) Graph-based

$$[p\left(Rule_1\right), p\left(Rule_2\right), \ldots)] = \pi_\theta(s_t) \tag{8}$$

where $\sum_{l=1} p\left(Rule_l\right) = 1$. Consequently, the rule with the maximum probability will be selected as the action for the state s_t. Following the action execution, an operation will be selected. The number of rules is independent of the number of operations, jobs, or machines. The primitive or compound heuristic rules, such as the shortest processing time (SPT) (Lin et al., 2019; Zhao et al., 2021; Sun et al., 2023) and genetic-programming-based rules (G&P) (Li et al., 2022; Luo et al., 2021c) are generally used with the partial solution. In contrast, the manipulation methods such as crossover and mutation that are borrowed from genetic algorithms can be used to update the whole solution (Chen et al., 2020).

Operation-based action

In this pattern (Fig. 6(b)), the policy π_θ outputs the possibility of each operation for the state s_t:

$$\left[p\left(O_{1,1}\right), p\left(O_{1,2}\right), \ldots, p\left(O_{1n_1}\right), \ldots, p\left(O_{|\mathcal{J}|n_{|\mathcal{J}|}}\right)\right] = \pi_\theta\left(s_t\right) \tag{9}$$

where $\sum_{i=1}^{|\mathcal{J}|}\sum_{j=1}^{n_i} p\left(O_{i,j}\right) = 1$. In this all-operation design, the action set size is equal to the number of operations (Lee et al., 2020; Workneh & Gmira, 2023). Consequently, the operation with the maximum probability will be selected as the action for the state s_t.

It should be noted that at any decision step, only the first unscheduled operation of each job can be the scheduling candidate. This feature enables a variant of all-operation-based action design, i.e., the partial-operation-based design (Liao et al., 2023; Tassel et al., 2021; Turgut & Bozdag, 2020), which only selects feasible operations as π_θ outputs. Therefore, the size of the action set in the partial-operation design is no greater than the number of jobs. This is because only the unfinished jobs have unscheduled operations and each unfinished job contributes only one candidate operation (i.e., its first unscheduled operation)

Attribute-based action

In this pattern (Fig. 6(c)), some attributes are selected to describe operations, and each operation has a definite value set for the attribute set. The policy π_θ outputs the predicted value of each attribute under the state s_t:

$$[value\left(Attr_1\right), value\left(Attr_2\right), \ldots] = \pi_\theta(s_t) \tag{10}$$

where $value\left(Attr_i\right)$ denotes the output value of $Attr_i$. Thereafter, a distance between the operation value set and the predicted value set is calculated. This way, the operation whose value set is closest to the predicted value set will be selected as the action for the state s_t. The attributes generally have continuous value ranges (Park & Park, 2021a; Samsonov et al., 2021) and the number of attributes is independent of the number of operations, jobs, or machines.

Graph-based action

In this pattern (Fig. 6(d), the attributes of each operation are structured in the form of graph, e.g., the disjunctive graph, and their values are extracted using a graph learning method (e.g., GNN). Next, the policy π_θ outputs the possibility of a given operation $O_{i,j}$ for a state-operation pair (Chen et al., 2022; Elsayed et al., 2022; Yuan et al., 2023):

$$p\left(O_{i,j}\right) = \pi_\theta(s_t, attr\_value(O_{i,j})), i = 1 \ldots n, j = n_i \tag{11}$$

where $attr\_value\left(O_{i,j}\right)$ is the attribute value of the operation $O_{i,j}$.

Comparison and discussion

The operation-based action pattern is primitive. The rules and attributes have definite semantics, contributing to improve the interpretability of the DRL decision process. However, the rules and attributes are in most cases selected based on empirical knowledge; therefore, designing a set of rules or attributes with strong optimization ability and wide adaptation to various scheduling problems remains a challenge. The graph-based action uses the GNN to extract operation features based on the initially defined attributes and graph, lessening the attribute selection recline.

In the rule-based and attribute-based action patterns, the action should be mapped to operations in a way that an operation is selected based on the winning rule or the predicted attribute value set. In contrast, the operation-based and the graph-based action patterns directly output operation possibilities; therefore, an operation can be selected in a simpler way.

The rule-based, the attribute-based, and the graph-based action designs are all independent of the scheduling problem scale, which contributes to the generalization ability. In contrast, the operation-based action design is coupled with the number of jobs or operations. To overcome this problem, recurrent neural networks (RNNs) are often adopted in the literature (Monaci et al., 2021; Ren et al., 2020). However, the states are mapped to the serial number of operations in the operation-based action design. Consequently, the encoding method will affect scheduling performance, which contributes another factor to harm the generalization ability.

Design patterns of the reward

In the DRL, a reward is a scalar that can be either positive, negative, or zero, reflecting the immediate effect of the action executed in the current state. It must be related to the optimization objectives of the underlying scheduling problem so that maximizing cumulative reward corresponds to optimizing the objectives. Currently, there are three primary reward design patterns: temporary value-based, final value-based, and discrete value-based.

Temporary value-based reward

This pattern applies to the partial solution-based DRL scheduling, where the actual optimization objective values cannot be obtained until the last step since the TSS is not complete (i.e., there are unscheduled operations). Therefore, the reward in the partial solution-based DRL scheduling will be quite sparse if it relies on the actual optimization objective values, as the reward can only be calculated once for each episode other than each step. Reward sparsity will cause difficulties in the convergence of the DRL algorithm (van Ekeris et al., 2021; Zhao et al., 2022). To overcome this problem, the estimated optimization objective values are used instead, aiming to generate an immediate reward for each step.

Zhang et al. (2020) proposed a method for computing the lower-bound completion time $C_{\mathrm{LB}}(O_{i,j}, s_t)$ of operation $O_{i,j}$ at state s_t corresponding the step t. For the scheduled operations, $C_{\mathrm{LB}}(O_{i,j}, s_t)$ equals to the completion time determined by the scheduling decoding algorithm. For the unscheduled operation $O_{i,j+1}$, $C_{\mathrm{LB}}\left(O_{i,j+1}, s_t\right)$ is calculated as:

$$C_{\mathrm{LB}}\left(O_{i,j+1}, s_t\right) = C_{\mathrm{LB}}\left(O_{i,j}, s_t\right) + p_{i,j+1}, i = 1 \ldots n, j = n' \ldots n_i - 1 \tag{12}$$

where $C_{\mathrm{LB}}\left(O_{i,0}, s_t\right) = 0$ and n' denotes the number of scheduled operations. Using iterative calculations, the lower-bound completion time of every operation can be determined.

Next, the lower-bound makespan in the state s_t is determined as follows:

$$C_{\max}^{\mathrm{LB}}\left(s_t\right) = \max_i C_{\mathrm{LB}}\left(O_{i,n_i}, s_t\right), i = 1 \ldots n \tag{13}$$

The immediate reward r_t obtained in state s_t is defined as:

$$r_t = C_{\max}^{\mathrm{LB}}\left(s_t\right) - C_{\max}^{\mathrm{LB}}\left(s_{t+1}\right), t = 1 \ldots T-1 \tag{14}$$

Finally, the cumulative reward $R(\tau)$ can be obtained according to Eq. (7):

$$R(\tau) = C_{\max}^{\mathrm{LB}}\left(s_1\right) - C_{\max}^{\mathrm{LB}}\left(s_T\right) = C_{\max}^{\mathrm{LB}}\left(s_1\right) - C_{\max} \tag{15}$$

where $C_{\max}^{\mathrm{LB}}\left(s_1\right)$ is a constant for a given scheduling instance as it is determined in the initial state in which no operation has been scheduled; $C_{\max}^{\mathrm{LB}}\left(s_T\right)$ is determined in the last state where a complete scheduling solution is resolved so that $C_{\max}^{\mathrm{LB}}\left(s_T\right)$ is exactly the makespan $C_{\max}$. Therefore, maximizing $R(\tau)$ is consistent with minimizing $C_{\max}$.

Final value-based reward

This pattern suits the whole solution-based DRL scheduling, where the TSS is both complete and feasible so that the actual optimization objective values can be obtained and used to design the reward function. For example, taking the makespan $C_{\max}(t)$ as the optimization objective, the immediate reward r_t can be designed as:

$$r_t = C_{\max}\left(s_t\right) - C_{\max}\left(s_{t+1}\right), t = 1 \ldots T-1 \tag{16}$$

Ni et al. (2021) noticed that the above-presented reward function tended to neglect the long-term effect of an action in the way that an action may still positively contribute to $R(\tau)$ even if its r_t is lower or negative. Therefore, it is advised to reshape the immediate rewards; an example is described in Fig. 7.

$C_{\max}$ tends to fluctuate rather than monotonically change during the training as shown in Fig. 7. Reward reshaping first figures out a set of minimal points maintaining a strict monotonic decrease relationship:

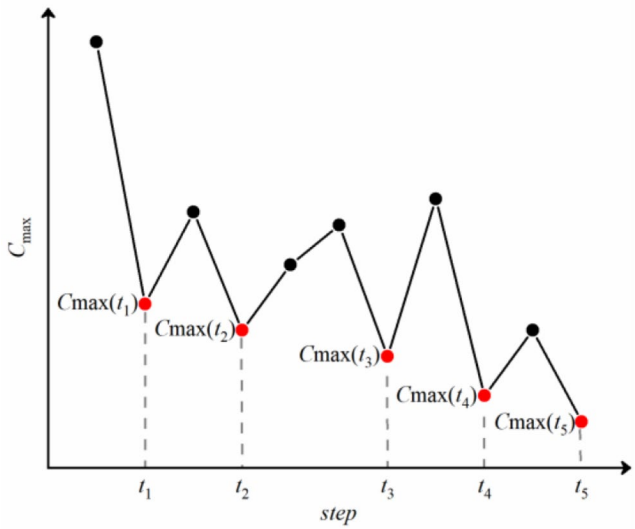


Fig. 7 History of $C_{\max}$

$[(t_1, C_{\max}(t_1)), (t_2, C_{\max}(t_2)) \ldots (t_a, C_{\max}(t_a))]$, where $t_1 < t_2 < t_3 < \cdots < t_a$ and $C_{\max}(t_1) < C_{\max}(t_2) < \cdots < C_{\max}(t_a)$. In the next step, it defines the immediate reward r_t as follows:

$$r_t = \frac{C_{\max}(t_p) - C_{\max}(t_{p+1})}{t_{p+1} - t_p}, t_p \le t < t_{p+1}, p = 0 \ldots a \tag{17}$$

where t_p denotes the time step when the pth minimal point appears so that several time steps (at least two) may expire from t_p to t_{p+1}.

Then, the cumulative reward $R(\tau)$ can be obtained via Eq. (7):

$$R(\tau) = C_{\max}(1) - C_{\max}(T) \tag{18}$$

where $C_{\max}(1)$ and $C_{\max}(T)$ denote the TSS makespan at the first and the last decision steps, respectively. Thus, $R(\tau)$ will be maximized by gradually reducing $C_{\max}$ of the TSS along with the DRL advance, which is consistent with the scheduling problem optimization objectives.

Discrete value-based reward

In this pattern, the reward value range is a set of discrete values. Therefore, the mapping rules of these discrete values to the optimization objectives are the essence of the reward design. Generally, both the direction and magnitude of objective variation are considered. Luo (2020) calculated the reward according to tardiness and machine utilization; the rewards + 1, -1, or 0 were given based on the value combination patterns of these two indicators.

Comparison and discussion

The temporary and final value-based rewards can be directly and tightly coupled with some optimization objectives. Therefore, the cumulative reward can be used to assess whether the reward design is reasonable. The reward function is reasonable if maximizing the cumulative reward corresponds to optimizing the scheduling objectives. On the other hand, the discrete value-based reward is loosely coupled with the optimization objectives since it uses only a few discrete values to code the changes of optimization objective. Therefore, the discrete value-based reward is quite a coarse-grained design pattern; however, it is applicable to both partial and whole solution-based DRL scheduling. On the contrary, the temporary value-based and final value-based rewards suit the partial solution and whole solution-based DRL respectively.

Statistical analysis of DRL design patterns

The authors so far identified three Agent design patterns, two Environment design patterns, three state design patterns, four action design patterns, and three reward design patterns, as summarized in Table 4.

A total of 216 different combinations can be constructed by the combination law. However, some of them are not applicable. Table 5 presents the design patterns used in the references, reflecting the popularity of each design pattern and pattern combination.

Statistical analysis of individual patterns

Figure 8 summarizes the number of times each design pattern occurs in Table 5. The value-based agent is the most popular because of the broad impact of DQN and its excellent performance on discrete tasks. In terms of the Environment, the partial solution-based design is adopted by most studies. Consequently, the temporary value-based reward is used in a greater proportion. As for the state, the statistic-based design is the most prevalent, while the rule-based and operation-based designs are the most common action design patterns. The popularity is affected by the characteristics which are compared in Sect. 3. Besides, we believe that the current research focuses on the development efficiency and interpretability of DRL-based scheduling models.

The end-to-end architecture for a partial solution-based Environment is more flexible than the iterative optimization required by the whole solution-based Environment. The statistic-based state design requires only an MLP network in most cases consuming less computational resources compared to GNNs and CNNs. Besides, the rule-based action design can provide a reliable interpretation of the output of the DRL scheduling model. The temporary value-based reward design can intuitively clarify the link between the optimization direction of the scheduling model and the scheduling objectives.

Table 4 A summary of design patterns of DRL scheduling model

Component	Pattern			
Agent	Value-based	Policy-based	Actor-Critic	
Environment	Partial solution-based	Whole solution-based		
State	Matrix-based	Statistic-based	Graph-based	
Action	Rule-based	Operation-based	Attribute-based	Graph-based
Reward	Temporary value-based	Final value-based	Discrete value-based	

Table 5 DRL design patterns shown in the references

Reference	Agent	Environment	State	Action	Reward
Chang et al. (2022)	Value-based	Partial solution-based	Statistic-based	Rule-based	Temporary value-based
Chen et al. (2020)	Value-based	Whole solution-based	Statistic-based	Rule-based	Final value-based
Chen et al. (2022)	Policy-based	Partial solution-based	Graph-based	Graph-based	Final value-based
Chen and Tian (2018)	Actor-Critic	Whole solution-based	Graph-based	Rule-based	Final value-based
Du et al. (2022)	Value-based	Whole solution-based	Statistic-based	Rule-based	Discrete value-based
Elsayed et al. (2022)	Actor-Critic	Partial solution-based	Graph-based	Graph-based	Temporary value-based
Gu et al. (2023)	Value-based	Whole solution-based	Statistic-based	Rule-based	Final value-based
Gebreyesus et al. (2023)	Policy-based	Partial solution-based	Graph-based	Graph-based	Temporary value-based
Hameed and Schwung (2020)	Actor-Critic	Partial solution-based	Graph-based	Graph-based	Temporary value-based
Han and Yang (2020)	Value-based	Partial solution-based	Matrix-based	Rule-based	Temporary value-based
Han and Yang (2021)	Policy-based	Partial solution-based	Statistic-based	Operation-based	Final value-based
Lee et al. (2020)	Value-based	Partial solution-based	Statistic-based	Operation-based	Temporary value-based
Li et al. (2022)	Value-based	Partial solution-based	Statistic-based	Rule-based	Temporary value-based
Liao et al. (2023)	Policy-based	Partial solution-based	Statistic-based	Operation-based	Temporary value-based
Lin et al. (2019)	Value-based	Partial solution-based	Statistic-based	Rule-based	Temporary value-based
Liu et al. (2020)	Actor-Critic	Partial solution-based	Matrix-based	Rule-based	Temporary value-based
Liu and Huang (2023)	Policy-based	Partial solution-based	Graph-based	Graph-based	Temporary value-based
Luo et al. (2021a)	Policy-based	Partial solution-based	Statistic-based	Operation-based	Temporary value-based
Luo (2020)	Value-based	Partial solution-based	Statistic-based	Rule-based	Discrete value-based
Luo et al. (2021b)	Value-based	Partial solution-based	Statistic-based	Rule-based	Discrete value-based
Luo et al. (2021c)	Policy-based	Partial solution-based	Statistic-based	Rule-based	Discrete value-based
Magalhães et al. (2021)	Value-based	Whole solution-based	Statistic-based	Operation-based	Discrete value-based
Monaci et al. (2021)	Policy-based	Partial solution-based	Statistic-based	Operation-based	Temporary value-based
Ni et al. (2021)	Policy-based	Whole solution-based	Graph-based	Rule-based	Final value-based
Palombarini and Martinez (2021)	Policy-based	Whole solution-based	Graph-based	Rule-based	Final value-based
Park and Park (2021a)	Actor-Critic	Partial solution-based	Statistic-based	Attribute-based	Temporary value-based
Park et al. (2021b)	Policy-based	Partial solution-based	Graph-based	Graph-based	Temporary value-based
Ren et al. (2020)	Actor-Critic	Partial solution-based	Statistic-based	Operation-based	Final value-based
Samsonov et al. (2021)	Actor-Critic	Partial solution-based	Statistic-based	Attribute-based	Final value-based
Seito and Munakata (2020)	Value-based	Partial solution-based	Graph-based	Graph-based	Final value-based
Song et al. (2023)	Value-based	Partial solution-based	Graph-based	Rule-based	Temporary value-based
Sun et al. (2023)	Value-based	Partial solution-based	Matrix-based	Rule-based	Temporary value-based
Tassel et al. (2021)	Policy-based	Partial solution-based	Statistic-based	Operation-based	Temporary value-based
Turgut and Bozdag (2020)	Value-based	Partial solution-based	Statistic-based	Operation-based	Temporary value-based
van Ekeris et al. (2021)	Policy-based	Partial solution-based	Statistic-based	Operation-based	Final value-based
Wang et al. (2021a)	Policy-based	Partial solution-based	Matrix-based	Operation-based	Temporary value-based
Workneh and Gmira (2023)	Value-based	Partial solution-based	Statistic-based	Operation-based	Temporary value-based
Wu and Yan (2023)	Policy-based	Partial solution-based	Matrix-based	Rule-based	Temporary value-based
Xu et al. (2022)	Value-based	Partial solution-based	Statistic-based	Rule-based	Temporary value-based
Yuan et al. (2023)	Policy-based	Partial solution-based	Graph-based	Graph-based	Temporary value-based
Zeng et al. (2022)	Value-based	Partial solution-based	Graph-based	Rule-based	Temporary value-based
Zhang et al. (2020)	Policy-based	Partial solution-based	Graph-based	Graph-based	Temporary value-based
Zhao et al. (2022)	Policy-based	Partial solution-based	Statistic-based	Operation-based	Temporary value-based
Zhao et al. (2021)	Value-based	Partial solution-based	Statistic-based	Rule-based	Temporary value-based

Statistical analysis of full pattern combinations

The 44 references shown in Table 5 involve only 28 pattern combinations with different ratios as shown in Fig. 9. The three most popular combinations are (in descending order):

- **FPC1** (the first full pattern combination): the policy-based agent with a partial solution design, a statistic-based state design, an operation-based action design, and a temporary value-based reward design.
- **FPC2**: the value-based agent with a partial solution design, a statistic-based state design, an rule-based action design, and a temporary value-based reward design.
- **FPC3**: the policy-based agent with a partial solution design, a graph-based state design, a graph-based action design, and a temporary value-based reward design.

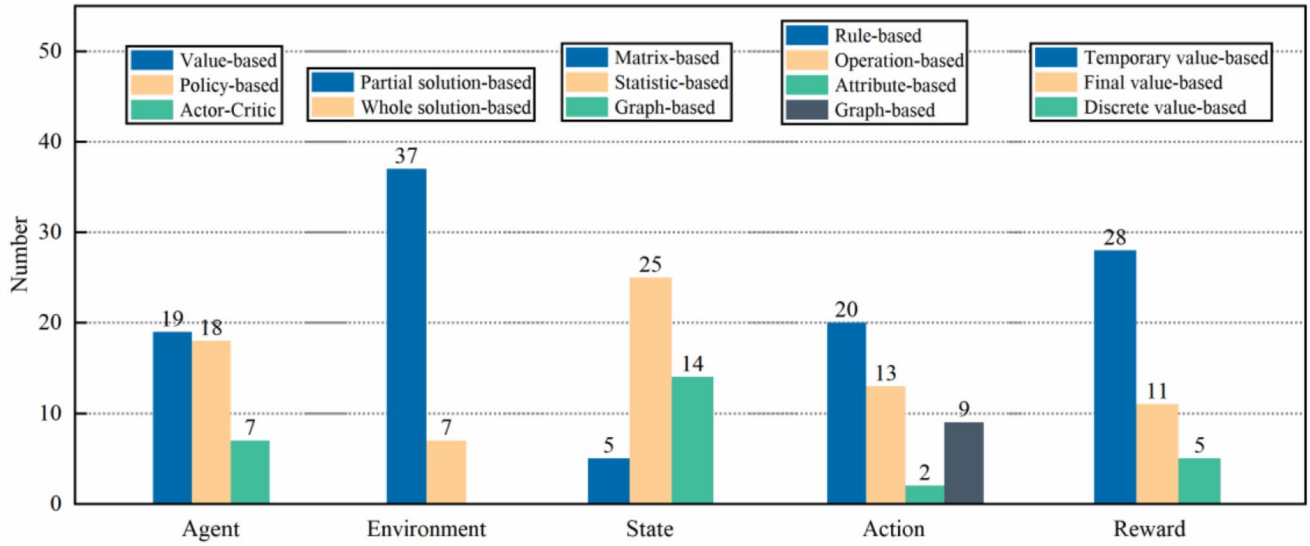


Fig. 8 Statistics of individual patterns

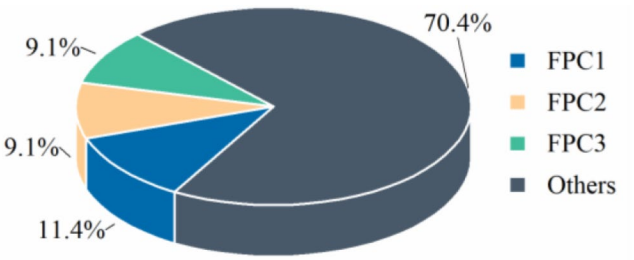


Fig. 9 Ratio of full pattern combinations

The remaining 25 pattern combinations account for approx. 70.4% and each of them is only adopted in one or two studies.

All the components in the dominant combinations FPC1, FPC2, and FPC3 utilize the popular design patterns as shown in Fig. 8. The partial solution-based Environment and the temporary value-based reward are most often used together. The Actor-critic-based agent has received less attention than the value-based or policy-based agent. However, the Actor-critic-based agent can take advantage of value functions to aid in policy optimization. It has also the ability to deal with both continuous and discrete problems. In the future, the Actor-critic-based agent has even greater potential to explore the performance of scheduling models.

Furthermore, from FPC1 to FPC3, it can be seen that the state and action design gradually evolved from straightforward simple patterns to graph-based patterns. Table 5 confirms that the publication dates of the graph-based design patterns are newer than the other patterns. Many research projects have demonstrated the superiority and generalization of graph-based design patterns (Zhang et al., 2020). However, very few graph presentations for the production scheduling problems are available currently, which means new graph presentations are needed to deal with the complex scheduling problems.

Evaluation indicators of DRL scheduling models

The effect of DRL production scheduling models is generally evaluated using the following four metrics, i.e., optimization objective, efficiency, convergence and stability, and generalization ability.

Optimization objective

Makespan, maximum/average tardiness, and maximum/average machine utilization are commonly used scheduling objectives. Their optimal extent determines the algorithm effectiveness. It is difficult to obtain the optimal value of the production scheduling objective since it is a type of NP-hard problem. Therefore, DRL scheduling methods are usually compared to heuristic rules, meta-heuristic algorithms, and exact optimization methods (e.g., integer programming and branch and bound) to evaluate their optimization ability. Experimental results provided in referenced studies show that the DRL scheduling methods are better than the heuristics rules and roughly equivalent to the meta-heuristics algorithms. Finally, they yield results that are rather close to the exact optimization methods.

Efficiency

The algorithm execution speed affects the practical production scheduling. As shown in Fig. 3, the DRL scheduling methods comprise several links:

$$t = N\left(t_s + t_r + t_\pi + t_{ce}\right) \tag{19}$$

Thus, the execution time t is equal to the number of episodes N times the sum of the state generation time t_s, the reward generation time t_r, the policy decision-making time t_π, and the action control and execution time t_{ce}.

The heuristic rules can quickly present a solution for a scheduling problem with no guarantee of the solution quality. The meta-heuristic and exact optimization methods will generally figure out a better solution when given a longer execution time which however is unacceptable for a real production scenario. DRL scheduling methods successfully trade off the efficiency and quality due to their generalization ability. In other words, the DRL scheduling methods can determine a better scheduling solution than the heuristic rules, in a shorter time than the meta-heuristic algorithms or the exact optimization methods.

Convergence and stability

The objective change curves concerning the training progress can be obtained from TSSs to describe the algorithm execution process. For a minimum optimization problem, a decline curve exhibits convergence, while a low amount of fluctuation presents stability. Experimental results have shown that the convergence and stability of DRL scheduling methods are comparable to that of the meta-heuristic algorithms.

Generalization ability

DRL scheduling models should be trained using a set of scheduling instances. After training, the DRL solvers can be applied to solve new scheduling instances. The execution time in the application stage is much lower than the training time. Such behavior suggests that the DRL model can utilize the learned experience to efficiently solve new scheduling instances without model rebuilding and retraining. In contrast, the meta-heuristic algorithms solve scheduling problems independently, indicating that the experience cannot be reused to facilitate the solution-seeking process.

Open issues

Among the available models and algorithms for solving production scheduling problems, deep reinforcement learning is unique as it can be generalized to new scheduling instances. It is also rather compatible with emerging IT technologies such as cloud computing, big data, and digital twins. Therefore, integrating DRL algorithms with smart manufacturing technologies to solve new scheduling problems or enable new properties seems a promising scheme.

Flexible job shop scheduling problems

Currently, DRL production scheduling studies mainly focus on the classical job shop scheduling problems. However, the flexible job shop scheduling (FJSP) is more in line with the consumption trends characterized by mixed-flow production of multi-variety, small-lot, and customized products (Kocsi et al., 2020), as the FJSP allows high flexibilities in both manufacturing resources and jobs. However, the FJSP also introduces great complexities. The jobs in a FJSP instance generally require different operations and/or different operation sequences and the operations have a many-to-many relationship with the machines, i.e., an operation can be processed by multiple machines and a machine can process multiple operations. Therefore, DRL scheduling methods for FJSP have both theoretical value and broad application prospects.

Multi-objective optimization problems

Production scheduling has an important role in multi-objective optimization (Mokhtari & Hasani, 2017); however, these optimization objectives are generally contradictory to each other. In other words, the improvement of one objective might degrade others. The key to multi-objective optimization problems is to achieve a trade-off between the objectives to maximize the overall performance. As one inherent property, DRL algorithms aim to maximize a cumulative reward. Therefore, the optimization objectives must be associated with the reward. However, the reward is a simple one-dimensional scalar which limits its ability to solve multi-objective optimization problems. Therefore, a significant breakthrough is needed to develop DRL-based multi-objective scheduling methods.

Multi-agent scheduling problems

The continuous advancement of vertical, horizontal, and end-to-end integration will significantly increase the complexity of the smart production system. Complex scheduling problems may also involve collaborative decision making and reliance from multiple parties (Ouelhadj & Petrovic, 2009). Therefore, it is difficult for a single agent to solve scheduling problems quickly, effectively, and economically. Consequently, the idea of multi-agent distributed computing and collaborative optimization provides an advanced

vision for solving complicated and large-scale problems. Although studies on multi-agent DRL algorithms were carried out (Waschneck et al., 2018; Lang et al., 2020), applying them to solve production scheduling problems remains challenging.

Adaptive scheduling problems

Many scheduling algorithms are not applicable to practical scenarios, especially to complex and large-scale problems, even if they can greatly optimize key performance indicators. There are multiple reasons behind this phenomenon; for example, some algorithms require complex parameter settings making it difficult for a user to carry out. The users expect a scheduling solver to run and evolve without their intervention. Therefore, the production scheduling solvers should be, in addition to optimization ability and efficiency, easy to use. It is possible to integrate DRL-based scheduling algorithms with smart manufacturing technologies such as digital twin (Fang et al., 2019) to reduce setting effort and improve the adaptability (Moon et al., 2021).

DRL scheduling specific generalization theory

Most advantages of DRL production scheduling models are related to their generalization ability (Gebreyesus et al., 2023), which is an important characteristic distinguishing deep reinforcement learning algorithms from meta-heuristic algorithms. The generalization ability empowers a trained DRL scheduling model to solve new scheduling instances efficiently and effectively, although training a DRL model may require both computational resources and time. However, currently available studies are exclusively focused on the generalization of small-scale instances to large-scale ones. Therefore, the complete DRL production scheduling-specific generalization theory is necessary. It will also enable new insights into the scheduling problems and the DRL-based scheduling methods.

DRL-based cloud manufacturing resource scheduling

Cloud manufacturing, as a service-oriented manufacturing mode, aims to provide consumers with on-demand manufacturing services. In an advanced cloud manufacturing environment, demand, supply, and production conditions change dynamically, such as changes in orders, machine breakdowns, or supply chain disruptions. Therefore, resource scheduling in cloud manufacturing environments is a complex problem. The DRL-based approach is able to continuously improve its performance through self-learning and self-adaptive mechanisms, demonstrating strong robustness to uncertainties and system perturbations, and learning how to make optimal decisions in dynamic and uncertain environments (Pahwa & Starly, 2021). Therefore, DRL has great potential in this area.

Conclusion

Based on the review results, the objective-related review questions can be answered. The related papers reflect that the design of each DRL component, i.e., the Agent, the Environment, the state, the action, and the reward, follows two to four patterns when solving the JSSP. However, the design patterns and pattern combinations have different levels of popularity. The features and popularity summarized in this review help developers form their specific design scheme to cope with the underlying JSSPs.

Furthermore, it is necessary to deeply integrate deep reinforcement learning algorithms with smart manufacturing technologies to solve complicated production scheduling problems. These include flexible job shop scheduling, multi-objective optimization, multi-agent scheduling, and self-adaptive scheduling. In addition, the DRL generalization ability requires further exploration in terms of the underlying mechanism, the production scheduling specific manifestation, and a comprehensive evaluation protocol.

This review mainly focuses on the DRL scheduling models for the job-shop scheduling problems due to the lack of literature for other production scheduling problems. However, our classification of design patterns and pattern combinations can broadly inspire the design of DRL models for advanced production scheduling problems.

Author contributions SW: Writing-Original Draft; Reviewing; Funding Acquisition; JL: Writing-Original Draft; Editing; Data Curation; QJ: Supervision; Conceptualization; FM: Supervision; Investigation.

Funding This work was supported by the National Key R&D Program of China (Grant No. 2020YFB1708500), the Guangdong Basic and Applied Basic Research Foundation (Nos. 2023A1515012975 and 2022A1515240061), the Open Project Program of Fujian Key Laboratory of Special Intelligent Equipment Measurement and Control, Fujian Special Equipment Inspection and Research Institute, China (No. FJIES2023KF12), and Characteristic and Innovative Project for Guangdong Regular Universities (Grant No. 2021KTSCX005).

Data availability This is a review paper and the statistical analysis presented in the paper is based on the published literature. All original data sources and analysis methods have been mentioned in the paper. For further information, please contact the corresponding author.

Declarations

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

References

Aromataris, E., & Riitano, D. (2014). Constructing a search strategy and searching for evidence. *American Journal of Nursing*, *114*(5), 49–56. https://doi.org/10.1097/01.NAJ.0000446779.99522.f6

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, *34*(6), 26–38. https://doi.org/10.1109/MSP.2017.2743240

Aveyard, H., Payne, S. A., & Preston, N. J. (2016). *A post-graduate's guide to doing a literature review in health and social care*. Open University.

Bettany-Saltikov, J. (2012). *How to do a systematic literature review in nursing: A step-by-step guide*. Open University.

Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M., & Lee, M. (2009). Natural actor–critic algorithms. *Automatica*, *45*(11), 2471–2482. https://doi.org/10.1016/j.automatica.2009.07.008

Boland, A., Cherry, M. G., & Dickson, R. (2017). *Doing a systematic review: A student's guide* (2nd ed.). SAGE Publications Ltd.

Chang, J., Yu, D., Hu, Y., He, W., & Yu, H. (2022). Deep reinforcement learning for dynamic flexible job shop scheduling with random job arrival. *Processes*, *10*(4), 760. https://doi.org/10.3390/pr10040760

Chen, X., & Tian, Y. (2018). Learning to perform local rewriting for combinatorial optimization. ArXiv preprint arXiv: 1810.00337. https://doi.org/10.48550/arXiv.1810.00337

Chen, R., Yang, B., Li, S., & Wang, S. (2020). A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem. *Computers & Industrial Engineering*, *149*, 106778. https://doi.org/10.1016/j.cie.2020.106778

Chen, R., Li, W., & Yang, H. (2022). A deep reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for the job shop scheduling problem. *IEEE Transactions on Industrial Informatics*, *19*(2), 1322–1331. https://doi.org/10.1109/TII.2022.3167380

Du, Y., Li, J. Q., Chen, X. L., Duan, P. Y., & Pan, Q. K. (2022). Knowledge-based reinforcement learning and estimation of distribution algorithm for flexible job shop scheduling problem. *IEEE Transactions on Emerging in Topics Computational Intelligence*, *7*(4), 1036–1050. https://doi.org/10.1109/TETCI.2022.3145706

Elsayed, E. K., Elsayed, A. K., & Eldahshan, K. A. (2022). Deep reinforcement learning-based job shop scheduling of smart manufacturing. *Computers Materials & Continua*, *73*(3), 5103–5120. https://doi.org/10.32604/cmc.2022.030803

Fang, Y., Peng, C., Lou, P., Zhou, Z., Hu, J., & Yan, J. (2019). Digital-twin-based job shop scheduling toward smart manufacturing. *IEEE Transactions on Industrial Informatics*, *15*(12), 6425–6435. https://doi.org/10.1109/TII.2019.2938572

Fattahi, P., Saidi Mehrabad, M., & Jolai, F. (2007). Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of Intelligent Manufacturing*, *18*, 331–342. https://doi.org/10.1007/s10845-007-0026-8

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flow-shop and job-shop scheduling. *Mathematics of Operations Research*, *1*(2), 117–129. https://doi.org/10.1287/moor.1.2.117

Gebreyesus, G., Fellek, G., Farid, A., Fujimura, S., & Yoshie, O. (2023). Gate-attention model with reinforcement learning for solving dynamic job shop scheduling problem. *IEEE Transactions on Electrical and Electronic Engineering*, *18*(6), 932–944. https://doi.org/10.1002/tee.23788

Gu, Y., Chen, M., & Wang, L. (2023). A self-learning discrete salp swarm algorithm based on deep reinforcement learning for dynamic job shop scheduling problem. *Applied Intelligence*, *53*, 18925–18958. https://doi.org/10.1007/s10489-023-04479-7

Hameed, M. S. A., & Schwung, A. (2020). Graph neural networks-based scheduler for production planning problems using reinforcement learning. ArXiv preprint arXiv: 2009.03836. https://doi.org/10.48550/arXiv.2009.03836

Han, B. A., & Yang, J. J. (2020). Research on adaptive job shop scheduling problems based on dueling double DQN. *Ieee Access : Practical Innovations, Open Solutions*, *8*, 186474–186495. https://doi.org/10.1109/ACCESS.2020.3029868

Han, B. A., & Yang, J. J. (2021). A deep reinforcement learning based solution for flexible job shop scheduling problem. *International Journal of Simulation Modelling*, *20*(2), 375–386. https://doi.org/10.2507/IJSIMM20-2-CO7

Kato, E. R. R., de Aguiar Aranha, G. D., & Tsunaki, R. H. (2018). A new approach to solve the flexible job shop problem based on an hybrid particle swarm optimization and random-restart hill climbing. *Computers & Industrial Engineering*, *125*, 178–189. https://doi.org/10.1016/j.cie.2018.08.022

Kocsi, B., Matonya, M. M., Pusztai, L. P., & Budai, I. (2020). Real-time decision-support system for high-mix low-volume production scheduling in industry 4.0. *Processes*, *8*(8), 912. https://doi.org/10.3390/pr8080912

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. In *2020 Winter Simulation Conference (WSC)* (pp. 3057–3068). IEEE. https://doi.org/10.1109/WSC48552.2020.9383997

Lee, S., Cho, Y., & Lee, Y. H. (2020). Injection mold production sustainable scheduling using deep reinforcement learning. *Sustainability*, *12*(20), 8718. https://doi.org/10.3390/su12208718

Li, Y., Gu, W., Yuan, M., & Tang, Y. (2022). Real-time data-driven dynamic scheduling for flexible job shop with insufficient transportation resources using hybrid deep Q network. *Robotics and Computer-Integrated Manufacturing*, *74*, 102283. https://doi.org/10.1016/j.rcim.2021.102283

Li, C., Zheng, P., Yin, Y., Wang, B., & Wang, L. (2023). Deep reinforcement learning in smart manufacturing: A review and prospects. *CIRP Journal of Manufacturing Science and Technology*, *40*, 75–101. https://doi.org/10.1016/j.cirpj.2022.11.003

Liao, Z., Chen, J., & Zhang, Z. (2023). Solving job-shop scheduling problem via deep reinforcement learning with attention model. *Advances and trends in Artificial Intelligence. Theory and applications* (pp. 201–212). Springer. https://doi.org/10.1007/978-3-031-36822-6_18

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *ArXiv Preprint arXiv*, 150902971. https://doi.org/10.48550/arXiv.1509.02971

Lin, C. C., Deng, D. J., Chih, Y. L., & Chiu, H. T. (2019). Smart manufacturing scheduling with edge computing using multiclass deep Q network. *IEEE Transactions on Industrial Informatics*, *15*(7), 4276–4284. https://doi.org/10.1109/TII.2019.2908210

Liu, C. L., & Huang, T. H. (2023). Dynamic job-shop scheduling problems using graph neural network and deep reinforcement learning. *IEEE Transactions on Systems Man and Cybernetics: Systems*, *53*(11), 6836–6848. https://doi.org/10.1109/TSMC.2023.3287655

Liu, C. L., Chang, C. C., & Tseng, C. J. (2020). Actor-critic deep reinforcement learning for solving job shop scheduling problems. *Ieee Access : Practical Innovations, Open Solutions*, *8*, 71752–71762. https://doi.org/10.1109/ACCESS.2020.2987820

Luo, S. (2020). Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Applied Soft Computing*, *91*, 106208. https://doi.org/10.1016/j.asoc.2020.106208

Luo, P. C., Xiong, H. Q., Zhang, B. W., Peng, J. Y., & Xiong, Z. F. (2021a). Multi-resource constrained dynamic workshop scheduling based on proximal policy optimisation. *International Journal*

of Production Research, *60*(19), 5937–5955. https://doi.org/10.1080/00207543.2021.1975057

Luo, S., Zhang, L., & Fan, Y. (2021b). Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning. *Computers & Industrial Engineering*, *159*, 107489. https://doi.org/10.1016/j.cie.2021.107489

Luo, S., Zhang, L., & Fan, Y. (2021c). Real-time scheduling for dynamic partial-no-wait multi-objective flexible job shop by deep reinforcement learning. *IEEE Transactions on Automation Science and Engineering*, *19*(4), 3020–3038. https://doi.org/10.1109/TASE.2021.3104716

Magalhães, R., Martins, M., Vieira, S., Santos, F., & Sousa, J. (2021). Encoder-decoder neural network architecture for solving job shop scheduling problems using reinforcement learning. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)* (pp. 1–8). IEEE. https://doi.org/10.1109/SSCI50451.2021.9659849

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Harley, T., Lillicrap, T. P., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd international Conference on International conference on Machine Learning - Volume 48* (pp. 1928–1937). JMLR.

Moher, D., Liberati, A., Tetzlaff, J., & Altman, D. G. (2009). Preferred reporting items for systematic reviews and meta-analyses: The PRISMA statement. *Annals of Internal Medicine*, *151*(4), 264–269. https://doi.org/10.1016/j.jclinepi.2009.06.005

Mokhtari, H., & Hasani, A. (2017). An energy-efficient multi-objective optimization for flexible job-shop scheduling problem. *Computers & Chemical Engineering*, *104*, 339–352. https://doi.org/10.1016/j.compchemeng.2017.05.004

Monaci, M., Agasucci, V., & Grani, G. (2021). An actor-critic algorithm with deep double recurrent agents to solve the job shop scheduling problem. *ArXiv Preprint arXiv*. https://doi.org/10.48550/arXiv.2110.09076. 2110.09076

Moon, J., Yang, M., & Jeong, J. (2021). A novel approach to the job shop scheduling problem based on the deep Q-network in a cooperative multi-access edge computing ecosystem. *Sensors (Basel, Switzerland)*, *21*(13), 4553. https://doi.org/10.3390/s21134553

Ni, F., Hao, J., Lu, J., Tong, X., Yuan, M., Duan, J., Ma, Y., & He, K. (2021). A multi-graph attributed reinforcement learning based optimization algorithm for large-scale hybrid flow shop scheduling problem. In *KDD 21: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (pp. 3441–3451). https://doi.org/10.1145/3447548.3467135

Ouelhadj, D., & Petrovic, S. (2009). A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, *12*, 417–431. https://doi.org/10.1007/s10951-008-0090-8

Pahwa, D., & Starly, B. (2021). Dynamic matching with deep reinforcement learning for a two-sided Manufacturing-as-a-Service (MaaS) marketplace. *Manufacturing Letters*, *29*, 11–14. https://doi.org/10.1016/j.mfglet.2021.05.005

Palombarini, J. A., & Martinez, E. C. (2021). End-to-end on-line rescheduling from Gantt chart images using deep reinforcement learning. *International Journal of Production Research*, *60*, 4434–4463. https://doi.org/10.1080/00207543.2021.2002963

Panwalkar, S. S., & Iskander, W. (1977). A survey of scheduling rules. *Operations Research*, *25*(1), 45–61. https://doi.org/10.1287/opre.25.1.45

Parjapati, S. K., & Jain, A. (2015). Optimization of flexible job shop scheduling problem with sequence dependent setup times using genetic algorithm approach. *International Journal of Mathematical Computational Natural and Physical Engineering*, *9*, 41–47. https://doi.org/10.5281/zenodo.1098021

Park, I. B., & Park, J. (2021a). Scalable scheduling of semiconductor packaging facilities using deep reinforcement learning. *IEEE Transactions on Cybernetics*, *53*(6), 3518–3531. https://doi.org/10.1109/TCYB.2021.3128075

Park, J., Chun, J., Kim, S. H., Kim, Y., & Park, J. (2021b). Learning to schedule job-shop problems: Representation and policy learning using graph neural network and reinforcement learning. *International Journal of Production Research*, *59*(11), 3360–3377. https://doi.org/10.1080/00207543.2020.1870013

Ren, J. F., Ye, C. M., & Yang, F. (2020). A novel solution to JSPS based on long short-term memory and policy gradient algorithm. *International Journal of Simulation Modelling*, *19*(1), 157–168. https://doi.org/10.2507/IJSIMM19-1-CO4

Samsonov, V., Kemmerling, M., Paegert, M., Lutticke, D., Sauermann, F., Gutzlaff, A., Schuh, G., & Meisen, T. (2021). Manufacturing control in job shop environments with reinforcement learning. In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART)* (pp. 589–597). https://doi.org/10.5220/0010202405890597

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. ArXiv preprint arXiv: 1707.06347. https://doi.org/10.48550/arXiv.1707.06347

Seito, T., & Munakata, S. (2020). Production scheduling based on deep reinforcement learning using graph convolutional neural network. In *Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART)* (pp. 766–772). https://doi.org/10.5220/0009095207660772

Smith, V., Devane, D., Begley, C. M., & Clarke, M. (2011). Methodology in conducting a systematic review of systematic reviews of healthcare interventions. *BMC Medical Research Methodology*, *11*(1), 15. https://doi.org/10.1186/1471-2288-11-15

Song, L., Li, Y., & Xu, J. (2023). Dynamic job-shop scheduling based on transformer and deep reinforcement learning. *Processes*, *11*(12), 3434. https://doi.org/10.3390/pr11123434

Sun, Z. Y., Han, W. M., & Gao, L. L. (2023). Real-time scheduling for dynamic workshops with random new job insertions by using deep reinforcement learning. *Advances in Production Engineering & Management*, *18*(2), 137–151. https://doi.org/10.14743/apem2023.2.462

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

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, *12*, 1057–1063.

Tassel, P., Gebser, M., & Schekotihin, K. (2021). A reinforcement learning environment for job-shop scheduling. *ArXiv Preprint arXiv: 2104 03760*. https://doi.org/10.48550/arXiv.2104.03760

Turgut, Y., & Bozdag, C. E. (2020). Deep Q-network model for dynamic job shop scheduling problem based on discrete event simulation. In *2020 Winter Simulation Conference (WSC)* (pp. 1551–1559). IEEE. https://doi.org/10.1109/WSC48552.2020.9383986

van Ekeris, T., Meyes, R., & Meisen, T. (2021). Discovering heuristics and metaheuristics for job shop scheduling from scratch via deep reinforcement learning. In *Proceedings of the Conference on Production Systems and Logistics (CPSL)* (pp. 709–718). https://doi.org/10.15488/11231

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., & de Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. *ArXiv Preprint arXiv*, 151106581. https://doi.org/10.48550/arXiv.1511.06581

Wang, L., Hu, X., Wang, Y., Xu, S., Ma, S., Yang, K., Liu, Z., & Wang, W. (2021a). Dynamic job-shop scheduling in smart manufacturing

using deep reinforcement learning. *Computer Networks*, *190*, 107969. https://doi.org/10.1016/j.comnet.2021.107969

Wang, L., Pan, Z., & Wang, J. (2021b). A review of reinforcement learning based intelligent optimization for manufacturing scheduling. *Complex System Modeling and Simulation*, *1*(4), 257–270. https://doi.org/10.23919/CSMS.2021.0027

Waschneck, B., Reichstaller, A., Belzner, L., Altenmüller, T., Bauernhansl, T., Knapp, A., & Kyek, A. (2018). Deep reinforcement learning for semiconductor production scheduling. In *2018 29th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)* (pp. 301–306). IEEE. https://doi.org/10.1109/ASMC.2018.8373191

Webster, J., & Watson, R. T. (2002). Analyzing the past to prepare for the future: Writing a literature review. *MIS Quarterly*, *26*(2), XIII–XXIII.

Workneh, A. D., & Gmira, M. (2023). Deep q network method for dynamic job shop scheduling problem. In *International Conference on Artificial Intelligence & Industrial Applications* (pp. 137–155). Springer, Cham. https://doi.org/10.1007/978-3-031-43524-9_10

Wu, X., & Yan, X. (2023). A spatial pyramid pooling-based deep reinforcement learning model for dynamic job-shop scheduling problem. *Computers & Operations Research*, *160*, 106401. https://doi.org/10.1016/j.cor.2023.106401

Xu, Z., Chang, D., Sun, M., & Lou, T. (2022). Dynamic scheduling of crane by embedding deep reinforcement learning into a digital twin framework. *Information*, *13*(6), 286. https://doi.org/10.3390/info13060286

Yuan, E., Cheng, S., Wang, L., Song, S., & Wu, F. (2023). Solving job shop scheduling problems via deep reinforcement learning. *Applied Soft Computing*, *143*, 110436. https://doi.org/10.1016/j.asoc.2023.110436

Zeng, Y., Liao, Z., Dai, Y., Wang, R., & Yuan, B. (2022). Hybrid intelligence for dynamic job-shop scheduling with deep reinforcement learning and attention mechanism. *ArXiv Preprint arXiv*, 220100548. https://doi.org/10.48550/arXiv.2201.00548

Zhang, C., Song, W., Cao, Z., Zhang, J., TanP. S., & Xu, C. (2020). Learning to dispatch for job shop scheduling via deep reinforcement learning. *ArXiv Preprint arXiv: 2010 12367*. https://doi.org/10.48550/arXiv.2010.12367

Zhao, Y., Wang, Y., Tan, Y., Zhang, J., & Yu, H. (2021). Dynamic job-shop scheduling algorithm based on deep Q network. *Ieee Access : Practical Innovations, Open Solutions*, *9*, 122995–123011. https://doi.org/10.1109/ACCESS.2021.3110242

Zhao, L., Shen, W., Zhang, C., & Peng, K. (2022). An end-to-end deep reinforcement learning approach for job shop scheduling. In *2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD)* (pp. 841–846). IEEE. https://doi.org/10.1109/CSCWD54268.2022.9776116