Mathematical Biosciences and Engineering

http://www.aimspress.com/journal/mbe

MBE, 21(1): 1445–1471.
DOI: 10.3934/mbe.2024062
Received: 11 October 2023
Revised: 30 November 2023
Accepted: 10 December 2023
Published: 28 December 2023

Research article

An actor-critic framework based on deep reinforcement learning for addressing flexible job shop scheduling problems

Cong Zhao and Na Deng*

School of Electronic and Electrical Engineering, Shanghai University of Engineering Science, Shanghai 201620, China

* **Correspondence:** Email: dengna@sues.edu.cn.

Abstract: With the rise of Industry 4.0, manufacturing is shifting towards customization and flexibility, presenting new challenges to meet rapidly evolving market and customer needs. To address these challenges, this paper suggests a novel approach to address flexible job shop scheduling problems (FJSPs) through reinforcement learning (RL). This method utilizes an actor-critic architecture that merges value-based and policy-based approaches. The actor generates deterministic policies, while the critic evaluates policies and guides the actor to achieve the most optimal policy. To construct the Markov decision process, a comprehensive feature set was utilized to accurately represent the system's state, and eight sets of actions were designed, inspired by traditional scheduling rules. The formulation of rewards indirectly measures the effectiveness of actions, promoting strategies that minimize job completion times and enhance adherence to scheduling constraints. The experimental evaluation conducted a thorough assessment of the proposed reinforcement learning framework through simulations on standard FJSP benchmarks, comparing the proposed method against several well-known heuristic scheduling rules, related RL algorithms and intelligent algorithms. The results indicate that the proposed method consistently outperforms traditional approaches and exhibits exceptional adaptability and efficiency, particularly in large-scale datasets.

Keywords: flexible job shop scheduling problems; deep reinforcement learning; actor-critic method; markov decision process; deep neural networks

1. Introduction

The manufacturing industry has been in a constant state of evolution. Recently, a paradigm shift from conventional mass production models to more adaptable, agile production systems has become evident [1]. In this innovative model, flexible job shops emerge as a pivotal element. Their adaptability is derived from versatile machine tools, robotics and advanced integrated information technology. Ac-

cordingly, to effectively arrange production resources and improve resource utilization, the FJSP has also become a hot topic in research and in practice [2].

As is widely acknowledged, the FJSP extends the classic job shop scheduling problem (JSP) and is classified as an NP-hard problem [3, 4]. The FJSP necessitates the identification of an optimal sequence for executing operations across jobs and astutely distributing these tasks among machines. Unlike its classical counterpart, the FJSP is characterized by its versatility, permitting the allocation of a job's operations to one or more machines with varying processing times. At present, the approaches to exploring the flexible job shop scheduling dilemma are categorized into two classes: exact and heuristic methods.

Exact methods encompass the strategies like mixed-integer programming and branch-and-bound. However, their applicability is restricted to diminutive problem sizes and they suffer from long computation times. Heuristic methods, encompassing established metaheuristic algorithms [5] and priority rule-based systems, such as genetic algorithms [6] and particle swarm optimization [7], face challenges in achieving global optimality in addressing the FJSP. These methods are particularly time-intensive when applied to larger instances and exhibit a suboptimal generalization performance. Priority rules are simple and can be reasonably allocated to machines to ensure the minimum cumulative completion time of jobs. However, choosing a single effective rule requires extensive domain knowledge and repeated experimentation.

Due to the inherent complexity of the FJSP, seeking efficient solutions often becomes a Herculean task, especially for large-scale instances. Consequently, the academic and industrial communities alike anticipate the advent of efficient heuristic and meta-heuristic solutions [8] to address this challenge. Lu [9] designed a hybrid optimization algorithm that merges iterative greedy techniques with local search strategies to resolve energy-efficient scheduling challenges in workshops.

Lately, the academic community has begun to acknowledge the profound capabilities of deep learning and RL in addressing intricate challenges [10, 11], particularly within expansive and high-dimensional decision spaces where conventional algorithms falter. RL, a subset of machine learning, enables agents to identify the optimal strategies via environment interactions. It is rewarded based on decision efficacy. Applying RL to the FJSP holds the promise of deriving policies to optimize job-to-machine allocations, thus minimizing the overall makespan.

RL operates independently of prior knowledge. Unlike the traditional optimization algorithms that necessitate intricate modeling, it autonomously uncovers potent strategies via exploration and exploitation mechanisms. After the initial training phase, the model is capable of perpetuating its learning and refining its strategies. Scholars have endeavored to conceptualize the development of priority rules in scheduling conundrums as a Markov decision process (MDP), utilizing end-to-end deep reinforcement learning (DRL) approaches taken to autonomously derive and instantiate scheduling rules. However, most existing RL methods focus only on the classic JSP, while the flexibility of the flexible job shop poses a significant challenge to the design of the learning mechanism. In the FJSP, both the system's state space in the system and the possible set of actions are enormous.

In this paper, an attempt is made to apply DRL to the FJSP to minimize the completion time. Our method uniquely leverages the actor-critic algorithm, a renowned RL paradigm combining value-based and policy-based advantages. To amplify solution effectiveness and learning efficacy, domain-specific enhancements are integrated. Benchmark dataset evaluations underscore our solution's superiority in terms of makespan and computational efficiency vis-à-vis extant methodologies.

Our contributions are detailed as follows:

1) A pioneering deep reinforcement learning methodology utilizing actor-critic constructs for the FJSP.

2) Adoption of state feature functions tailored for dynamic, continuous environments, supplemented by algorithmic actions rooted in parametric rules.

3) Development of a unique reward mechanism to gauge the impact of real-time scheduling on objectives.

4) Comprehensive hyperparameter sensitivity experiments conducted to fine-tune the solution.

The structure of this article is as follows. Section 2 introduces existing methods and techniques related to the FJSP. Section 3 describes the mathematical model and constraints on the FJSP, as well as introduces the RL algorithm and the standard actor-critic algorithm. Section 4 offers a comprehensive explanation of the algorithm used in our proposed solution for the FJSP. Section 5 introduces the DRL-AC framework, which constitutes a framework for deep reinforcement learning based on actor-critic architecture. Section 6 presents the experimental results that validate the proposed scheduling method. Section 7 summarizes the conclusions and outlines potential future work.

2. Related work

Over the past few decades, the FJSP has gained significant attention in the field of manufacturing scheduling and has become a research hotspot. At present, solutions to the FJSP mainly include mathematical planning methods, heuristic algorithms [12], meta-heuristic algorithms [5] and machine learning methods [13, 14]. Among them, heuristic algorithms and meta-heuristic algorithms are the most commonly used methods, including genetic algorithms [6], simulated annealing algorithms [15] and tabu search [16]. Such methods are prone to converging on local optima, particularly in instances of substantial problem sizes or intricate solution spaces, implying that the quality of their optimal solutions might not be optimal. Furthermore, the computational resources and time demanded by the algorithms tend to escalate exponentially with problem size, which impedes scalability in practical deployments. Therefore, researchers have formulated heuristic scheduling rules. Doh [17] presented a practical priority scheduling method that employs a combination of machine selection and job sequencing rules to resolve this problem. However, designing practical and efficient scheduling rules is not an easy task, which requires deep domain knowledge and long development time. When the size of the FJSP increases or the problem becomes more complex, simple scheduling rules may not be able to effectively capture all the characteristics and constraints of the problem, thus affecting the quality of the solution.

In recent years, machine learning methods have achieved an excellent performance in solving combinatorial optimization problems. RL, as a type of machine learning technique, has been applied to address several combinatorial optimization problems, including the job shop scheduling problem. The RL approach requires the agent to learn the optimal policy through its interactions with the environment and receiving rewards. With RL used to define the shop scheduling problem as a Markov decision process [18], the job scheduling process is viewed as the mapping from one sequence to another.

Some significant contributions in RL-based solutions for job shop scheduling problems include Shahrabi [19] who innovatively combined variable neighborhood search (VNS) with Q-learning to propose a method for solving job shop scheduling problems. Similarly, Wang and Yan [20] proposed an

adaptive scheduling algorithm that catered to the inherent uncertainties in manufacturing environments. Wang [21] developed a multi-agent model that iteratively updates Q-values and incorporates a system based on multi-agent knowledge, thus guiding equipment in policy selection. However, Q-learning methodologies [22], as applied to the FJSP, present some challenges. A glaring issue is the need to design a Q-value table that encapsulates all conceivable states and actions, causing an exponential surge in storage and computational demands.

The surge of DRL in various domains prompted its introduction in the shop floor scheduling domain as well. For instance, Luo [23] devised a deep Q network (DQN) to solve the FJSP, optimizing the scheduling policy for minimal delays. Another groundbreaking contribution by Li [24] involved a MOEA/D algorithm optimized with RL to solve the multi-objective FJSP. Song [11] addressed the FJSP by combining the operation selection and machine assignment into a composite decision and by taking an end-to-end DRL approach. Additionally, a graph neural network (GNN) was proposed to represent the complex relationships between operations and machines. Liu [25] proposed a training method that combines asynchronous updates with deep deterministic policy gradients. Yuan [26] designed a novel state representation using bidirectional scheduling features and applied the technique of masking invalid actions to narrow the search space.

From the extensive literature on the application of RL in job shop scheduling, several clear trends can be observed. First, while traditional job shop scheduling problems are a primary application area for RL methods, recent years have seen a growing focus on the FJSP, particularly as DRL demonstrates potential in addressing the complexity of the FJSP. Most existing studies [19, 21, 27] tend to rely more on a singular learning strategy (like Q-learning), which tends to form deterministic policies in the action space. Alternatively, using DQN [28, 29] and more abstract state representations requires carefully designed graph structures and feature encoding to capture the complex relationships between jobs and machines, resulting in significant computational resource demands.

Different from prior research, we focus on adopting an actor-critic architecture, combining value-based and policy-based methods and introduce parameterized rules instead of simply following fixed rules, allowing for more precise adjustments and more flexible decision-making. Furthermore, by using the beta distribution to predict optimal actions, our method demonstrates innovation in statistical decision-making and offers a balanced approach between exploration and exploitation. This approach empowers the agent to more effectively explore and exploit various scheduling strategies, thereby sustaining stability throughout the pursuit of the optimal solution.

3. Problem description and method

3.1. The flexible job shop scheduling problem

The FJSP discussed in this article can be defined as follows. There are n jobs $J = \{J_1, J_2, \ldots, J_n\}$ processed on m machines $M = \{M_1, M_2, \ldots, M_l\}$ and each job J_i consists of a sequence of operations, where O_{ih} represents the hth operation of job J_i. A distinguishing feature of the FJSP is that each operation O_{ih} has a set of candidate machines it can be processed on, rather than being restricted to a single machine, allowing for greater flexibility in scheduling. The jobs must follow a feasible sequence of operations. The machining process also adheres to the following constraints [30].

1) A machine can process only one job at a time.

2) An operation, once started, must proceed to completion without interruption.

3) The processing time for each operation is predetermined. The movement or setup times between operations are considered as part of the machine processing time.

4) If a machine is not assigned a task, it goes into a waiting state.

5) Machines are assumed to have a consistent performance across different jobs, irrespective of the specific job or operation they are handling.

Based on what is mentioned above and the model developed by Lu [30], Table 1 presents the symbols used for problem formulation.

Table 1. Explanation of the symbols used in problem statement.

Symbol	Explanation
Indices	
i, j	indices of jobs, i, $j = 1, 2, \ldots, n$.
h, k	indices of operations, h, $k = 1, 2, \ldots, o$.
m	index of machines, $m = 1, 2, \ldots, l$.
Parameters	
n	total number of jobs
l	total number of machines
o	total number of operations
t	current scheduling time
P_{ih}	processing time of operation O_{ih}
P_{ihm}	processing time of the hth operation of job J_i on machine M_m
C_{ihm}	completion time of the hth operation of job J_i on machine M_m
S_{ihm}	start time of the hth operation of job J_i on machine M_m
C_i	completion time of job J_i, $C_i \geq 0$
T_{preset}	preset completion time
t_i^m	time for job J_i to arrive at machine M_m
$n_{rest}(t)$	count of remaining operations for J_i at time t
L	a sufficiently large integer
J_i	ith job
O_{ih}	the hth operation of job J_i
Set	
Ω_{ih}	set of operations of job J_i on machine M_m

The decision variables are denoted as:

$$X_{ihm} = \begin{cases} 1, & \text{if } O_{ih} \text{ selects the machine } M_m; \\ 0, & \text{otherwise} \end{cases}$$

$$Y_{ihjkm} = \begin{cases} 1, & \text{if } O_{ihm} \text{ is processed before } O_{jkm}; \\ 0, & \text{otherwise} \end{cases}$$

In general, the scheduling objectives to be optimized are related to the completion time of the jobs. In this research, our objective is to minimize the makespan, as represented by C_{max}, which corresponds

to the maximum completion time among all operations. Based on the symbols and assumptions introduced earlier, the objective function of the FJSP as mentioned in this study is expressed as follows:

$$\min C_{\max} = \min(\max_{1\le i\le n}(C_i)) \tag{3.1}$$

$C_{max}(t)$ is the makespan at time step t, as illustrated in Figure 1. The objective function is subjected to the following constraints.

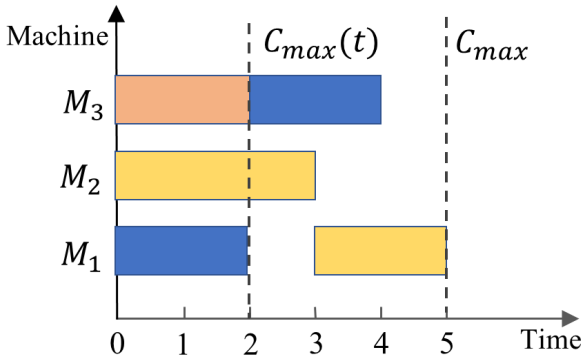


Figure 1. The schematic representation of $C_{max}(t)$ and C_{max}

$$\text{s.t.}\begin{cases} S_{ihm} + X_{ihm} \times P_{ihm} \le C_{ihm}, \forall i,h,m & (3.2)\\ S_{ihm} + P_{ihm} \le S_{jkm} + L(1 - Y_{ihjkm}), \forall i,h,m,j,k & (3.3)\\ C_{ih} \le C_{\max}, \forall i,h,m & (3.4)\\ \sum_{m\in\Omega_{ih}} X_{ihm} = 1, \forall i,h & (3.5)\end{cases}$$

Equation (3.2) represents the sequential constraint, which specifies the order in which operations must be performed for each job. Equation (3.3) denotes the resource constraint, which makes a machine capable of handling one operation at any given time. Equation (3.4) defines the time constraints, which ensure that each job must be completed within the total completion time. Finally, Eq (3.5) is the machine constraint, which stipulates that a particular operation can only be processed by one machine at a time.

3.2. Basic theory of reinforcement learning

Reinforcement learning is a machine learning approach that involves training an agent to make optimal decisions through its interactions with an environment. The core of RL is the Markov decision process (MDP). MDP can be described as a five-tuple ($\boldsymbol{S}$, $\boldsymbol{A}$, $\boldsymbol{P}$, γ, $\boldsymbol{R}$), where $\boldsymbol{S}$ denotes the state space, $\boldsymbol{A}$ denotes the action space, $\boldsymbol{P}$ denotes the state transition probability function, $\boldsymbol{R}$ denotes the reward function, and γ denotes the discount factor. Figure 2 shows the interaction between the agent as a subject and the environment as an object in MDP.

More specifically, the steps taken during an episode are denoted as the index $t = 0, 1, \ldots, T$, where $t = 0$ represents the initial state. At each time step t, the agent is in a state $S_t \in \boldsymbol{S}$, an action $A_t \in \boldsymbol{A}$ is

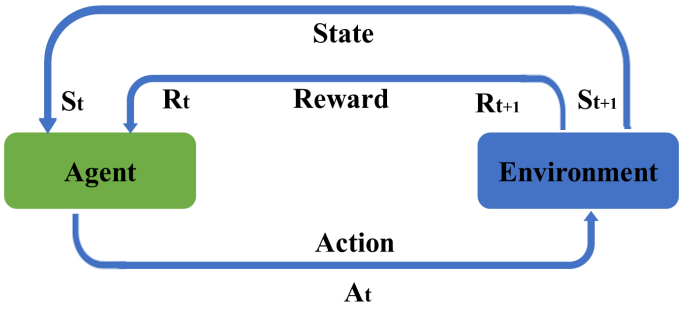


Figure 2. The interaction between the agent and the environment.

selected, and the environment transitions to a new state S_{t+1} according to the probability distribution $p(S_{t+1} \mid S_t, A_t)$, while also rewarding the agent based on the reward function $R(S_t, A_t, S_{t+1})$. In general, the agent seeks to maximize the expected return, which is represented by G_t and the return is the sum of the rewards. Herein, to ensure that the agent maximizes the sum of future returns, we introduce a discount factor $\gamma(0 \leq \gamma \leq 1)$ into the calculation. This factor weights the returns received by the agent in the future, and the resulting equation for the expected return is expressed as follows:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.6}$$

Of course, it is worth noting that the rewards that the agent expects to obtain in the future are dependent on the actions it takes at time step t. The mapping between the probabilities of selection from states to each action is called the policy. The ultimate goal of addressing a RL task is to find a policy that enables the agent to receive significant rewards over an extended period of time.

3.3. Standard actor-critic method

Among many reinforcement learning algorithms, this research chooses the actor-critic method, which combines value-based and policy-based learning to improve policies by estimating the state-value function. The actor, like the part of the athlete responsible for executing the decision, is responsible for generating the policy, while the critic, like the judge, is responsible for evaluating the policy. In the actor-critic algorithm, unlike others, the action selection in the actor network no longer depends on the value function, but on updating the policy.

In the critic network using the value-based policy evaluation method, a common estimation method is to use temporal-difference (TD) learning, and specifically, to update the estimated value of the current state by calculating the TD error between the estimated value of the current state and the estimated value of the next state, with the following update Eq (3.7):

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right] \tag{3.7}$$

Ultimately, the critic's target is to learn an accurate value function for evaluating long-term returns in arbitrary states. The actor and critic interact with each other, and this interaction accelerates the learning process because the actor can adjust the strategy through the critic's feedback. Also, the critic

can better estimate the value function through the exploration by the actor. Figure 3 illustrates the framework of the actor-critic method.

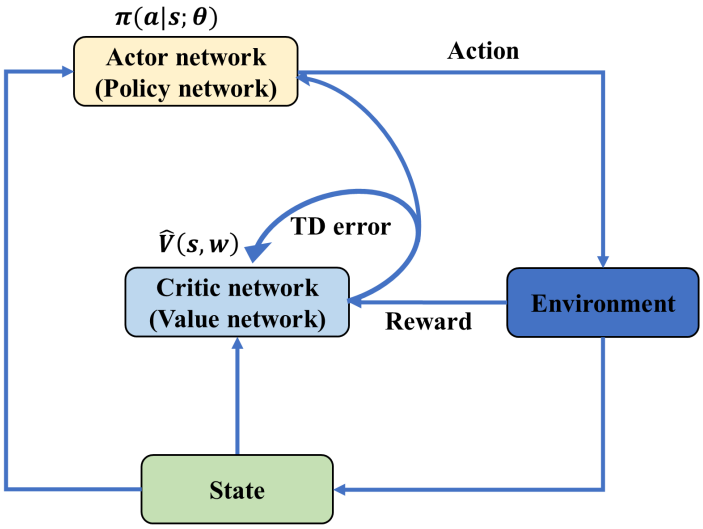


Figure 3. Actor-critic framework.

4. Solving the FJSP based on DRL-AC

4.1. Modeling flexible job shop scheduling as a Markov decision process

The FJSP focuses on how to allocate machines and time to each job task in an environment with multiple machines in order to meet certain optimization criteria. State: In the FJSP, the workshop's status at any given time can be represented by the working state of each machine and the queue of tasks waiting for processing. These states can change over time. Action: At a certain state, the workshop might need to decide which job should be processed on which machine. Decisions are guided by a set of heuristic rules that specify which job should be processed on which machine. Transition Probability: Based on a certain action, it is the probability of the workshop transitioning from the current state to the next. Reward: For each action or state transition, there is an associated reward or cost. In the FJSP, this could be the speed of task completion, machine utilization rate or other indicators related to the optimization goal. Markov property: In the MDP, the next state of the system only depends on the current state and the action taken, rather than on previous states or actions. Similarly in the FJSP, once the current workshop status and decision are given, the next state largely depends only on this information. Policy: Just like in the MDP, the goal of the FJSP is to find a policy, i.e., the best action to be taken in various states, to achieve the set optimization goals.

Due to these characteristics of the FJSP that align with the MDP framework, it is feasible to view the FJSP as an MDP problem and employ relevant solution methods to find optimal or near-optimal solutions.

4.2. State features

In RL-based job scheduling problems, some indicators of the production plant are usually defined as state features. As an important component in reinforcement learning, the state can directly affect the operational efficiency and accuracy of the whole scheduling algorithm, and inappropriate state feature definitions can make the algorithm difficult to train and use. The state features are defined based on the following principles: 1) The dimensions of the state should be minimized to reduce the size of the state space and lower computational complexity; 2) The state must be able to accurately and comprehensively describe the status of the workshop. Insufficient or inaccurate state information can have a detrimental effect on the algorithm's performance. 3) It is essential for the state to differentiate between various scheduling schemes. If identical state representations arise from different schemes, the agent will be unable to discern and decide differently. Hence, the state representation must incorporate a rich set of features that can effectively distinguish between diverse production setups and their corresponding decision results. 4) In FJSPs, time efficiency is one of the most important optimization objectives. Therefore, state features are typically based on time, such as the remaining time for jobs, the utilization rate of machines, and the completion rate of jobs, etc. These features are capable of reflecting the progress towards the achieving optimization goal. 5) Given the susceptibility of deep learning models to the scale of input data, normalizing state features to a range between 0 and 1 can enhance the learning efficacy of the model. This normalization not only mitigates the influence of disparate feature scales but also facilitates a swifter convergence of the algorithm.

To provide a continuous and comprehensive view of the production status of the shop floor, inspired by [31, 32], the following variables are taken into account in this study's state space analysis: the ratio between the operation's minimum and maximum processing times; machine utilization and average utilization; job completion rate and average job completion rate; and remaining operation time. The state is defined by consecutive parameters. These state parameters are consistent to minimize makespan, and they are therefore primarily time-based. To ensure generalization, the values of each value and vector element are set between 0 and 1. Each state has an initial value of 0 at $t = 0$. The detailed parameters and description of state features at time t are shown in Table 2.

(1) Processing time per operation $OP(t)$, as defined in (4.1)

$$OP(t) = P_{ih}(t) \tag{4.1}$$

(2) Utilization per machine $U(t)$, as defined in (4.2)

$$U(t) = \frac{\sum_{i=1}^{n}\sum_{h=1}^{o} P_{ihm}}{C_{\max}(t)} \tag{4.2}$$

(3) The average utilization rate of total machines $U_{ave}(t)$, which is defined in (4.3)

$$U_{ave}(t) = \frac{1}{l}\sum_{m=1}^{l}\frac{\sum_{i=1}^{n}\sum_{h=1}^{o} P_{ihm}}{C_{\max}(t)} \tag{4.3}$$

(4) The standard deviation of the utilization rate $D_{std}(t)$ of total machines at time t, as defined in (4.4)

$$D_{std}(t) = \sqrt{\frac{\sum_{m=1}^{l}(U(t) - U_{\text{ave}}(t))^2}{l}} \tag{4.4}$$

(5) Completion rate per job $JO(t)$, as defined in (4.5)

$$JO(t) = \frac{n_{\text{rest}}(t)}{o_i} \tag{4.5}$$

(6) Average completion rate of total jobs $JO_{ave}(t)$, which is defined in (4.6)

$$JO_{ave}(t) = \frac{\sum_{i=1}^{n}\frac{n_{\text{rest}}(t)}{o_i}}{n} \tag{4.6}$$

(7) The ratio between the minimum and maximum processing time of operation $Ra_{op}(t)$, as defined in (4.7)

$$Ra_{op}(t) = \frac{\min(P_{ih}(t))}{\max(P_{ik}(t))} \tag{4.7}$$

(8) The ratio of the minimum remaining processing time of a job to the maximum completion time at time t, as defined in (4.8)

$$MinRe(t) = \frac{\min\left(\sum_{k=h+1}^{o_i} P_{ik}\right)}{C_{\max}(t)} \tag{4.8}$$

Table 2. The state space parameters.

Parameter	State features	Description of features
s_t^1	$OP = (OP_1, OP_2, OP_3, \ldots, OP_o)$	Processing time per operation
s_t^2	$U = (U_1, U_2, U_3, \ldots, U_m)$	Utilization per machine
s_t^3	$JO = (JO_1, JO_2, JO_3, \ldots, JO_n)$	Completion rate per job
s_t^4	$Ra_{op}(t)$	The ratio between the minimum and maximum processing time of operation
s_t^5	$U_{ave}(t)$	Average utilization of total machines
s_t^6	$JO_{ave}(t)$	Average completion rate of total jobs
s_t^7	$D_{std}(t)$	Standard deviation of the utilization of total machines
s_t^8	$MinRe(t)$	Minimum remaining operating time

4.3. Action space selection

The action space can be described as the collection of all feasible actions that the agent is capable of performing in a given state. In job shop scheduling, the action space is usually composed of different heuristic rules of which the agent action selects the priority jobs to be scheduled according to the

heuristic scheduling rules [33]. The definition of the action space should be reasonable, meaning they can be assigned to a job and a machine without any conflicts or constraints. The actions in the action space have some features so that the agent can make wise choices among them.

Table 3. The defined action space.

Parameterized Description	Explanation	The priority rules for mapping
$\arg\min\{t_i^m\}$	The first arriving job J_i at scheduling moment t	FCFP(FIRST COME FIRST PROCESSED)
$\arg\max\{t_i^m\}$	The last arriving job J_i at scheduling moment t	FCLP(FIRST COME LAST PROCESSED)
$\arg\min\{\frac{P_{im}(t)}{\sum_{h=h'}^{o_i} P_{im}(t)}\}$	The job J_i with the shortest processing time at scheduling moment t	SPT(SHORTEST PROCESSING TIME)
$\arg\max\{\frac{P_{im}(t)}{\sum_{h=h'}^{o_i} P_{im}(t)}\}$	The job J_i with the longest processing time at scheduling moment t	LPT(LONGEST PROCESSING TIME)
$\arg\min\{(\frac{o_i - n_{\text{rest}}(t)+1}{o_i})\}$	The job J_i with the current minimum number of remaining operations at scheduling moment t	LOR(LEAST OPERATION REMAINING)
$\arg\max\{(\frac{o_i - n_{\text{rest}}(t)+1}{o_i})\}$	The job J_i with the current maximum number of remaining operations at scheduling moment t	MOR(MOST OPERATION REMAINING)
$\arg\min\{\sum_{k=h+1}^{o_i} P_{ik}(t)\}$	The job J_i with the shortest remaining processing time at scheduling moment t	SRPT(SHORTEST REMAINING PROCESSING TIME)
$\arg\max\{\sum_{k=h+1}^{o_i} P_{ik}(t)\}$	The job J_i with the most remaining processing time at scheduling moment t	MRPT(MOST REMAINING PROCESSING TIME)

Traditional scheduling methods rely on fixed heuristic rules, such as shortest processing time (SPT) or longest processing time (LPT). However, these rules often perform inconsistently under different conditions. In this research, parametric priority rules are introduced here where the parameters range from 0 to 1, a real number that can represent different priorities so that each discrete priority rule can be mapped to each specific value in the parameter space. The action space is dynamic, allowing the selection of actions according to different production states, and it consists of eight parameters that are normalized. It implies that we are not constrained to a handful of predefined rules; instead, we possess a continuous action space that opens up a broader spectrum of possibilities. Parametrized rules

enable a balance between exploration (trying new scheduling strategies) and exploitation (utilizing known effective strategies). Adjusting parameters allows for a seamless transition between different rules, avoids abrupt shifts and thereby enables the agents to fine-tune their scheduling strategies with greater precision. This action space includes priority rules, including rules based on processing times and operations, and each rule is divided into forward and reverse rules. The specific parameterized priority rules are shown in Table 3.

The agent's decision-making process employs the beta distribution to forecast the forthcoming optimal action. The β distribution, a continuous probability distribution characterized by two parameters, α and β, can embody various prioritization rules contingent on the values of these parameters. By examining all feasible actions within the current state, the agent constructs a probabilistic model, from which it selects actions based on a sampling method. The agent gauges the Euclidean distance between the parameters of each viable action and those of the optimal action, opting for the action that minimizes this distance.

4.4. Reward function

The reward plays a crucial role in the Markov decision process and serves as a feedback signal that the agent receives from the environment at each time step to guide the learning process of the agent. The Reward function should be set with the objective, and it should reflect whether the agent's performance converges to the optimal solution.

The objective of this study is to use the actor-critic framework to learn a strategy to optimize scheduling for the minimum makespan. However, in the shop job schedule, it is known only when the complete operation is performed, which belongs to the case of sparse reward in RL. There exist numerous approaches to address the issue of sparse rewards. For example, [34] proposes to maximize the average machine utilization instead of minimizing the completion time, and this research is inspired to rely on average machine utilization and job completion rate instead of the completion time to obtain the immediate reward because minimizing the completion time can be indirectly reflected as the average machine utilization $U_{ave}(t)$ and the average completion rate of job $JO_{ave}(t)$ to reach the equilibrium state. After the rule scheduling is completed, the budgeted completion time is compared by estimating the completion time and if it is not and the average machine utilization and the average completion rate of the job have improved, a positive reward is generated, otherwise, a negative reward is generated. Immediate rewards are applied to each step of the scheduling process during the judgment process and the reward r is determined based on the steps outlined in Algorithm 1:

5. Proposed DRL-AC model architecture

In the actor-critic based framework, both the actor network and the critic network are composed of deep neural networks (DNNs). The actor network is the action selection strategy that determines the best action for flexible shop scheduling and consists of one input layer, two hidden layers and two output layers. The input layer receives the state information, the hidden layer uses the ReLU function to enhance the accuracy and convergence of the actor network and each output layer uses the SoftPlus function to obtain the alpha and beta parameters of the beta distribution. The activation functions used in this study are the ReLU function and SoftPlus function, represented by Eqs (5.1) and (5.2):

Algorithm 1 Algorithm proposed for reward

Initialize environment
Initialize actor network and critic network
for episode = 1 to L **do**
    The agent observes the initial state and generates a feature vector that captures the relevant information about the state.
    for t = 1 to T **do**
        if scheduling continues **then**
            if scheduling time $C_{\max}(t) \leq T_{preset}$, $U'_{ave}(t) > U_{ave}(t)$ and $JO'_{ave}(t) > JO_{ave}(t)$ **then**
                generate the positive reward: $r_t \leftarrow 1$
            else
                generate the negative reward: $r_t \leftarrow -1$
            end if
        else
            generate the negative reward: $r_t \leftarrow -100$
        end if
    end for
end for

$$ReLU(x) = \max(0, x) \tag{5.1}$$

$$SoftPlus(x) = log(1 + e^x) \tag{5.2}$$

During the learning process, the critic network estimates the value of state s_t by comparing the chosen action with the average action taken in that state. The network uses linear layers and the ReLU function to extract useful features from the state. Specifically, it consists of an input layer, two hidden layers and an output layer, where the input layer receives the state information, the hidden layer is used to extract the features of the state and the output layer contains a neuron that is used to predict the value of the state. The two networks are optimized using Adam to optimize the objective function for the strategy executed by the agent. For this purpose, each network has its objective function to be optimized. The architecture and algorithm of the complete network are illustrated in Figure 4 and Algorithm 2 as follows:

In this study, the weights of the actor network and critic network are initialized, the initial state is sent to the actor network and the critic network and the Adam optimization algorithm is used to train the actor network during the training process. Finally, the actor network outputs eight pairs of two parameters of the beta distribution, α, β, and then samples the best action from the beta distribution. Also, the Euclidean distance is used to search for the best fitting action. The agent takes an action. That is to say, after the first work is scheduled, the state moves to the next state and the environment rewards the critic network according to the average machine utilization rate and the average job completion rate. The critic network will evaluate whether the job has taken good actions during the scheduling process, and then the actor network will modify the scheduling strategy according to the critic network. Once all the work is completed, the environment will give the final reward back, calculate the makespan and

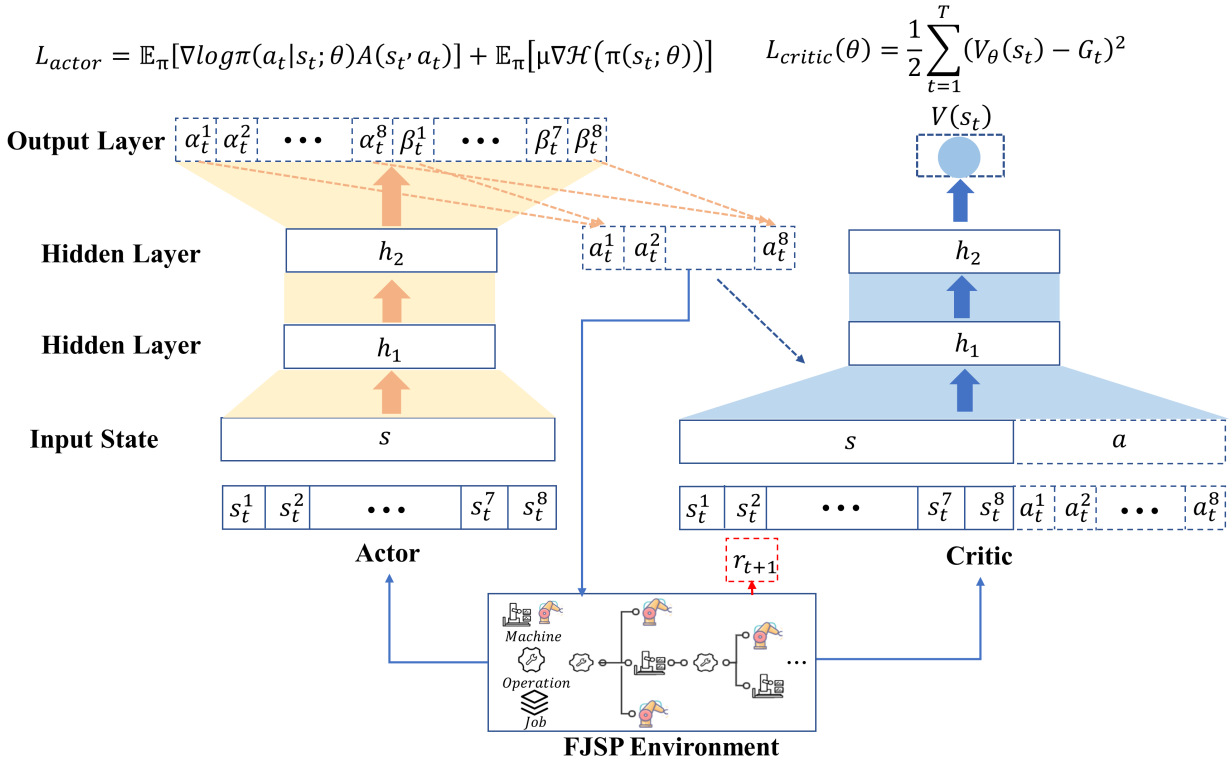


Figure 4. The proposed DRL-AC architecture.

adjust the state to the initial state until the number of iterations reaches the specified iteration threshold. The algorithm gradually approaches convergence, and the training is terminated. The algorithm learns the optimal scheduling strategy corresponding to each production state and can dynamically allocate production tasks in the flexible job shop production process. The update rules of the algorithm are based on the loss function. To minimize the difference between the predicted value and the target value, the critic-network part of the neural network is enabled to better estimate the value of the current state. Therefore, the critic network minimizes its squared error during training.

$$L_{critic} = \frac{1}{2}\sum_{t=1}^{T}(V_\theta(s_t) - G_t)^2 \tag{5.3}$$

The purpose of the actor network loss function is to maximize the expected value of the reward while minimizing the entropy of the strategy to facilitate exploration.

$$L_{actor} = \mathbb{E}_{\pi}[\nabla \log \pi(a_t|s_t;\theta)A(s_t, a_t)] + \mathbb{E}_{\pi}[\mu\nabla\mathcal{H}(\pi(s_t;\theta))] \tag{5.4}$$

$A(s_t, a_t)$ represents the advantage function, which measures how much better the action a is than the average value in the state s_t. Herein, TD error is used to approximate $A(s_t, a_t)$ as it is an unbiased estimate of $A(s_t, a_t)$, and the equation is $\delta_\theta = r + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)$. We also use another small method to improve the exploration [35] by adding the entropy of the strategy to the objective function of the strategy network minimization. In this function, $\mathbb{E}$ denotes the entropy function of the probability distribution and μ denotes the parameter influencing the super-control entropy regularization.

Algorithm 2 The proposed DRL-AC Algorithm for FJSP

Initialize environment
Initialize actor network and critic network
for episode = 1 to L **do**
    Acquire the initial state and extract the feature vector of the state
    for t = 1 to T **do**
        Determine action a_t based on α and β from the Beta distribution
        Advance the system time when the job is completed in
        Update the current system time
        Calculate the average machine utilization $U_{ave}(t)$ and the average completion rate of job $JO_{ave}(t)$
        Obtain the current state s_{t+1} and reward r_t
        Store state s_t, action, reward r_t, and the next state s_{t+1} in memory
        Sample a batch from the memory to train the actor and critical networks
        Update the critic network with the gradient $\nabla(V_\theta(s_t) - G_t)^2$
        Compute TD error $\delta_\theta = r + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)$ is used to approximate $A(s_t, a_t)$ according to the critic network
        Update the actor network using gradient $\nabla \log \pi(a_t|s_t;\theta)A(s_t, a_t) + \mu\nabla\mathcal{H}(\pi(s_t;\theta))$
    end for
end for

6. Results and discussion

This part shows the results of the simulation and computation. The algorithms run on Python 3.8 and Pytorch 1.10 based platforms. The experiments are performed on an Intel (R) Core (TM) i7-8550U @CPU1.80 GHz SERVER with RAM 8GB. To assess the effectiveness of the proposed framework, we tested it on the public FJSP benchmark test set and random instances. For the common FJSP datasets, we utilized the Brandimarte dataset, Kacem dataset and Hurink dataset, which encompass the problems ranging from small-scale to larger-scale. In experiments, the benchmark test cases we used are all static, all jobs are present and ready to be processed at the start time $t = 0$ and no other factors are involved. Thus, each data case can be both a training set and a testing set. With this work, the algorithm is to learn a maximally optimal strategy that minimizes makespan. The study designed experiments to validate the robustness of the proposed framework and the hyperparameters.

Table 4. Hyperparameter set for training.

Hyperparameter	Values
Number of episodes	1000
Number of input layer neurons of the actor-network	(o+m+n+5)
Number of input layer neurons of the critic-network	(o+m+n+13)
Number of neurons in each hidden layer	256
Number of hidden layers	2
Hyperparameters affecting entropy regularization (μ)	0.001
Learning rate (α)	0.001
Discount rate (γ)	0.9

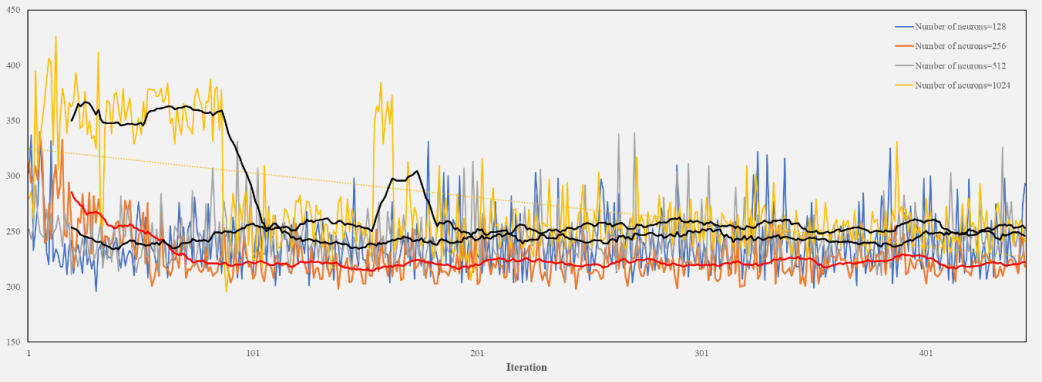
6.1. The impact of hyperparameters on model sensitivity

To demonstrate the effect of hyperparameters on the algorithm model, a comprehensive sensitivity analysis was performed on pivotal hyperparameters, including the number of neurons in hidden layers, the number of hidden layers and the learning rate, as shown in Figure 5.

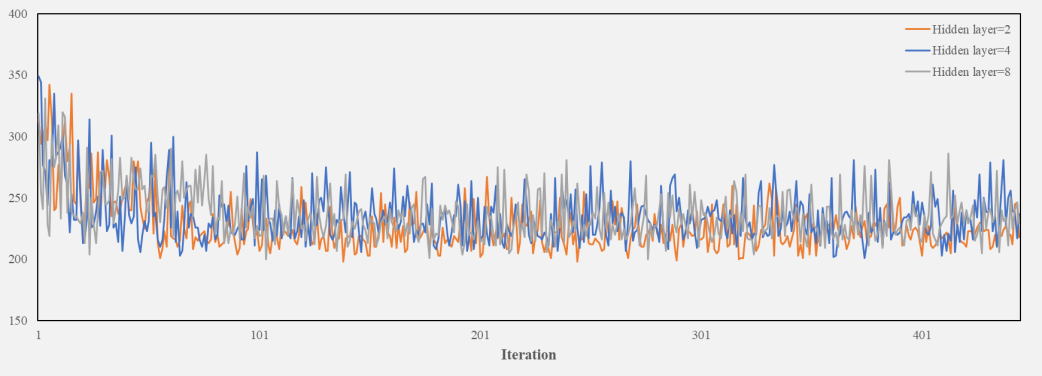
As shown in Figure 5(a), the number of neurons is adjusted from 128 to 1024. It is found that, when there are more neurons in the hidden layers, the algorithm has better convergence performance and converges to the minimum makespan. This improvement in performance can be attributed to the increased capacity of the network to capture more complex features and patterns within the data. However, it is also observed that beyond a certain threshold in the number of neurons, the performance of the model begins to oscillate, indicating overfitting, which can reduce the ability of the model to generalize to unseen data. Networks with a sparse hidden layer featuring too few neurons (like 128) might lack the necessary complexity for certain tasks, whereas those with an excessive count (like 1024) could be susceptible to overfitting and demand more computational resources. A network comprising 256 neurons achieves a balance, providing ample complexity for learning patterns while avoiding the shortcomings associated with larger networks. The results in Figure 5(b) show that the number of hidden layers has little effect on the scheduling results and that the number of hyperparametric hidden layers is not sensitive to the algorithm framework and does not require parameter optimization. This indicates that, within our algorithmic framework, this hyperparameter—the number of hidden layers—does not significantly impact performance. Figure 5(c) shows that the learning rate is 10^{-3}, and the results are relatively stable. This learning rate appears to provide a balance, ensuring sufficient learning while avoiding oscillation issues that may arise from too large a step size.

6.2. Comparison with related deep reinforcement learning methods and heuristic scheduling rules

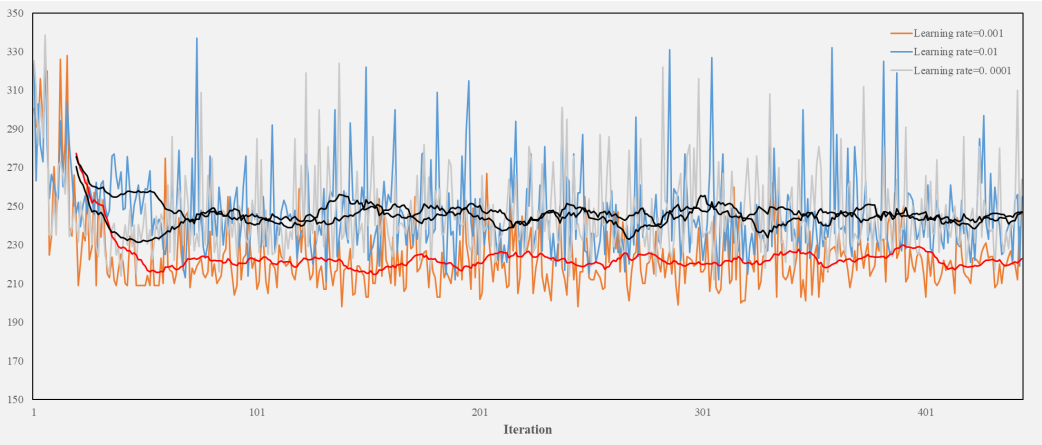
In this section, a comprehensive series of experimental tests were conducted to evaluate the performance and effectiveness of the proposed algorithms. The completion time of each algorithm was meticulously measured. The algorithms were then used to train a single model, with the aim to address the flexible job shop scheduling problems on various scales. Comparative analysis was performed with related static scheduling algorithms in line with reinforcement learning and five heuristic rules. The relative error (*RE*) for each instance was calculated using Eq (6.1).



(a) Number of neurons in the hidden layer of the instance



(b) Number of hidden layers of the instance



(c) Learning rate setting of the instance

Figure 5. The effect of hyperparameters.

$$RE = \frac{C_{max} - LB}{LB} \times 100\% \tag{6.1}$$

The smaller the RE, the more stability in algorithm performance, which is a common metric for job shop scheduling. Tables 5 and 7 show the scheduling results of different algorithms on the BRdata dataset, where $n \times l$ indicates that the instance contains n jobs and l machines, Flex. indicates the average number of machining processes of the flexible machine and LB and UB represent the optimal lower and upper bounds, respectively.

Table 5. The scheduling results of different reinforcement learning algorithms on the BRdata dataset.

			Opt. Makespan		DRL-AC		AC-S		DRL_SARSA		DRL_Q	
Instance	$n \times l$	**Flex.**	**LB**	**UB**	**Best**	**RE**	**Best**	**RE**	**Best**	**RE**	**Best**	**RE**
MK01	10×6	2.09	36	42	**41**	13.89%	49	36.11%	47	30.56%	53	47.22%
MK02	10×6	4.10	24	32	**28**	16.67%	47	95.83%	32	33.33%	32	33.33%
MK03	15×8	3.01	204	211	**206**	0.98%	257	25.98%	264	29.41%	255	25.00%
MK04	15×8	1.91	48	81	88	83.33%	**86**	79.17%	92	91.67%	102	112.50%
MK05	15×4	1.71	168	186	**175**	4.17%	222	32.14%	195	16. 07%	189	12. 50%
MK06	10×15	3.27	33	86	**93**	181.82%	149	351.52%	98	196.97%	112	239.39%
MK07	20×5	2.83	133	157	**213**	60.15%	227	70.68%	238	78.95%	232	74.44%
MK08	20×10	1.43	523	523	**525**	0.38%	581	11.09%	551	5.35%	527	0.76%
MK09	20×10	2.53	299	369	**361**	20.74%	473	58.19%	374	25.08%	384	28.43%
MK10	20×15	2.98	165	296	**277**	67.88%	405	145.45%	304	84.24%	279	69.09%

Table 6. The scheduling results of different scheduling rules on the BRdata dataset.

			Opt. Makespan		SPT		MOR		MOR+SPT		LOR		LPT	
Instance	**n×l**	**Flex.**	**LB**	**UB**	**Best**	**RE**	**Best**	**RE**	**Best**	**RE**	**Best**	**RE**	**Best**	**RE**
MK01	10×6	2.09	36	42	65	80.56%	83	130.56%	71	97.22%	89	147.22%	73	102.78%
MK02	10×6	4.10	24	32	44	83.33%	87	262.50%	46	91.67%	75	212.50%	61	154.17%
MK03	15×8	3.01	204	211	397	94.61%	462	126.47%	330	61.76%	423	107.35%	378	85.29%
MK04	15×8	1.91	48	81	109	127.08%	134	179.17%	188	291.67%	176	266.67%	204	325.00%
MK05	15×4	1.71	168	186	231	37.50%	195	16.07%	192	14.29%	238	41.67%	209	24.40%
MK06	10×15	3.27	33	86	128	287.88%	162	390.91%	101	206.06%	182	451.52%	254	669.70%
MK07	20×5	2.83	133	157	188	41.35%	341	156.39%	217	63.16%	261	96.24%	251	88.72%
MK08	20×10	1.43	523	523	670	28.11%	723	38.24%	603	15.30%	540	3.25%	625	19.50%
MK09	20×10	2.53	299	369	573	91.64%	503	68.23%	466	55.85%	459	53.51%	405	35.45%
MK10	20×15	2.98	165	296	536	224.85%	345	109.09%	423	156.36%	544	229.70%	531	221.82%

Figure 6 shows the scheduling Gantt chart for DRL-AC at MK01. AC-S method [36] is a method for minimizing the completion time of flexible job shops based on 3D disjunctive graph dispatching through a DRL and attention mechanism architecture, and its static dispatch data are used here. This paper also compares the use of RL on the FJSP combined with value-based approaches. DRL_SARSA is the algorithm combined with SARSA, and DRL_Q is the algorithm combined with Q-learning.

To further analyze the computational results, the box-line plots of the methods mentioned in this paper with the associated deep reinforcement learning methods and heuristic scheduling rules are plot-

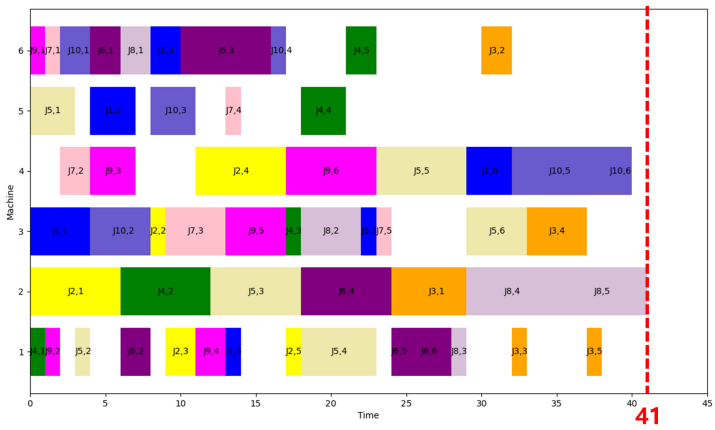


Figure 6. The scheduling Gantt chart for DRL-AC at MK01.

ted. Figures 7 and 8 showcase the boxplots of RE leveraging the Brandimarte dataset. The outcomes distinctly illustrate that the DRL-AC approach yields the minimal statistical RE figures and exhibits the tightest clustering of data trends. Such findings not only underscore the efficacy of our methodology in tackling scheduling challenges but also highlight its superior performance in stability—a critical measure of the robustness of a scheduling algorithm.

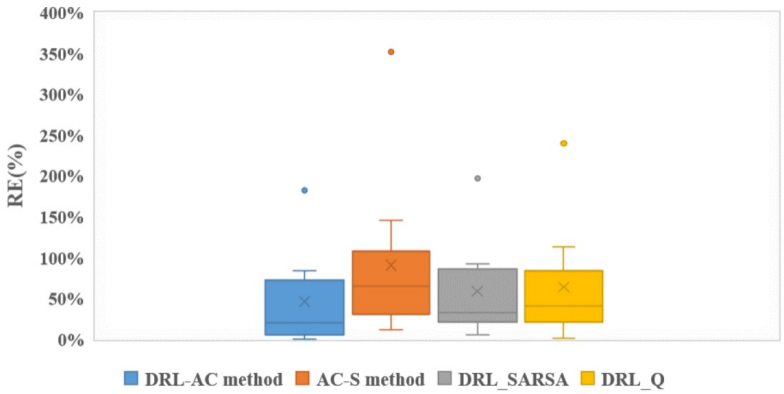


Figure 7. The RE boxplot of the four deep reinforcement learning-related algorithms.

Figure 9 illustrates how the three methods compare in terms of their convergence performance, value-based and value-and-strategy-based, on MK05, illustrating that the method proposed in this paper converges at a lower completion time than the value-based method based on Q-learning and SARSA methods.

6.3. Comparison with intelligent algorithms

In this subsection, the proposed algorithms are compared with classical algorithms from the existing literature, using benchmark instances from the Kacem dataset, which includes both partial and total flexible job shop scheduling problems. The dataset comprises four different problem sizes: 4×5, 8×8, 10×10 and 15×10. These correspond to scenarios with 4 jobs and 5 machines with 12 operations (partial flexibility), 8 jobs and 8 machines with 27 operations (partial flexibility), 10 jobs and

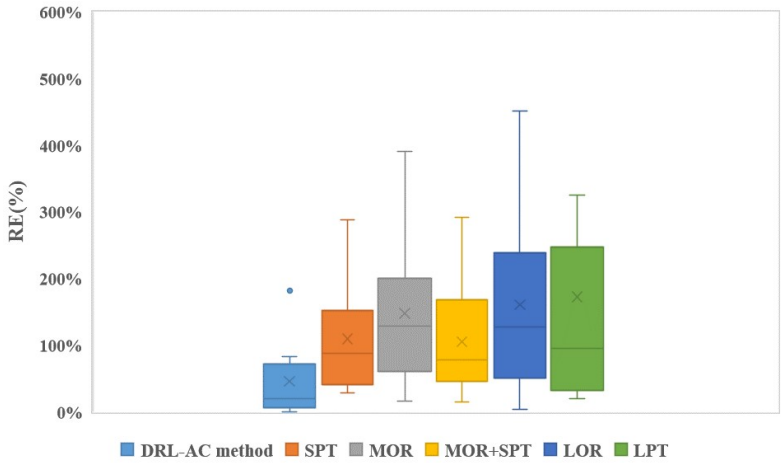


Figure 8. The RE boxplot for the proposed method and the heuristic scheduling rule method.

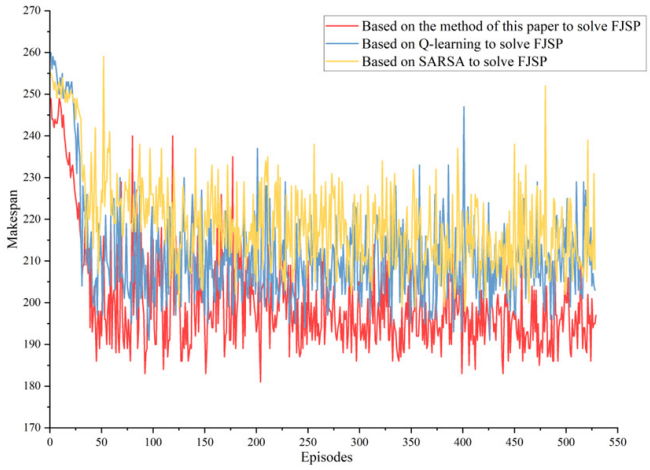


Figure 9. Convergence plots of the three methods on MK05.

10 machines with 30 operations (total flexibility) and 15 jobs and 10 machines with 56 operations (total flexibility), respectively. Partial flexibility implies that certain jobs cannot be processed on specific machines.

The proposed DRL-AC algorithm is compared with other classical algorithms from the literature. The results for completion times are presented in Table 7 and Figure 10. In Table 7, the symbol "-" indicates that a solution is not available in the literature, bold font denotes the best solution for each instance, "Random" represents a random scheduling algorithm, "classic GA" represents a classic genetic algorithm, "MATSPSO/PSO+TS" [37] is a method from the literature based on a hybrid multi-agent model combining Tabu Search (TS) and particle swarm optimization (PSO), "PSO+SA" [38] combines PSO and simulated annealing (SA) incorporating both local and global search strategies and "Kacem" [39] refers to a method proposed in the literature. "EDPSO" [7] refers to a distributed Particle Swarm Optimization algorithm proposed in the literature.

Table 7. Comparison of the algorithm with others on different instances.

	Random	classic GA	MATSPSO/PSO+TS	PSO+SA	Kacem	EDPSO	DRL-AC
4×5	**11**	13	-	**11**	16	**11**	**11**
8×8	20	16	15	15	19	17	**14**
10×10	9	**7**	**7**	**7**	16	8	8
15×10	15	-	-	**12**	23	-	**12**

Figure 10. Performance comparison of algorithms on different problem instances.

The algorithms Random, classic GA, MATSPSO/PSO+TS and EDPSO can only find a single best solution, while Kacem fails to find any optimal solutions. On the other hand, PSO + SA and the proposed DRL-AC find three optimal solutions, demonstrating that DRL-AC performs well on small-scale problems and outperforms the traditional GA. Across the four problem instances, the performance of DRL-AC is relatively stable, without extreme high or low values, indicating that the DRL-AC algorithm demonstrates good stability across problems of varying sizes.

Table 8. Comparison of the algorithm with others on larger-scale instances.

		TS		HA			DRL-AC		
n×l	Flex.	Makes	RE	Makespan	RE	CPU(s)	Makespan	RE	CPU(s)
		1073	0.19%	1071	0	1.73	1071	0	0.83
20*5	5	940	0.43%	936	0	2.73	936	0	0.58
		1041	0.29%	1038	0	4.34	1038	0	6.56
		1058	0.57%	1053	0.10%	41.43	1053	0.10%	15.29
20*10	5	1088	0.37%	1085	0.09%	35.29	1085	0.09%	15.21
		1076	0.65%	1070	0.09%	42.17	1073	0.37%	19.1
		1521	0.07%	1521	0.07%	58.25	1520	0	18.34
30*10	5	1659	0.12%	1658	0.06%	65.78	1657	0	21.4
		1499	0.13%	1499	0.13%	53.82	1498	0.07%	17.97

For larger-scale instances, tests were conducted on larger datasets from the Hurink dataset [40], specifically using datasets with a higher flexibility of five. The sizes tested were 20×5, 20×10 and 30×10, with three datasets each. TS denotes the traditional Tabu Search algorithm, while HA [41] refers to a hybrid algorithm that combines Genetic Algorithms with Tabu Search. As can be seen from Table 8, compared to TS, the hybrid algorithm and the algorithm proposed in this paper obtained the majority of the best solutions in terms of completion times. Furthermore, Figure 11 reveals that as the size of the instances grows, the difference in CPU time between the DRL-AC algorithm and the HA algorithm becomes increasingly significant, with the runtime for the HA algorithm showing a substantial growth. This underscores the enhanced capability of the algorithm presented in this paper to manage highly complex problems.

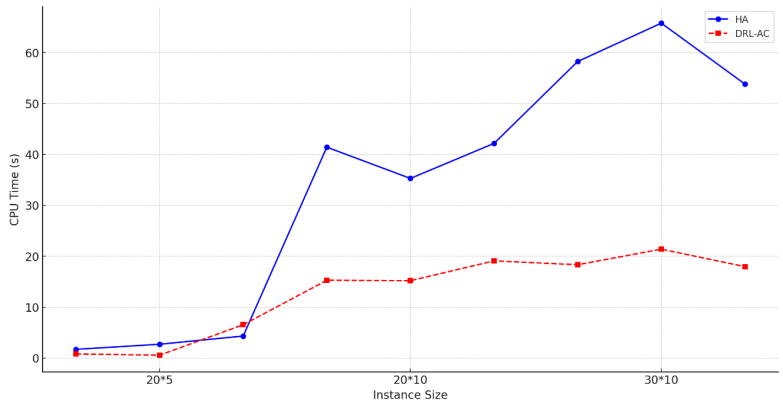


Figure 11. CPU time comparison of HA and DRL-AC across different instance sizes.

6.4. Discussion

The proposed DRL-AC method has significantly enhanced the performance of flexible job shop scheduling across multiple dimensions. A primary benefit of the proposed approach is its capacity for comprehensive analysis and assessment of the entire job suite prior to operation initiation. Furthermore, the designed action space enables the agent to tailor its decisions to the current state, offering a

level of adaptability not found in conventional methods. In conclusion, our reward function steers the learning process of the agent towards desired outcomes by aligning it directly with optimization goals, such as minimizing production time and maximizing machine utilization rates.

One of the principal strengths of DRL-AC is its adaptability to problems of varying scales, from small to large. On smaller datasets, like the Kacem dataset, the algorithm efficiently identifies optimal solutions, showcasing its precision and potential within constrained environments. As the model scales to larger instances, like those in the Hurink dataset, DRL-AC retains its performance levels without notable increases in computation time, suggesting that the algorithm can manage heightened complexity without any compromise on efficiency.

However, the performance of the algorithm comes at the expense of computational resources, particularly during the training phase. The training process is resource-intensive, necessitating significant computational power to realize the expected outcomes. Furthermore, hyperparameter tuning is a substantial undertaking, demanding meticulous experimentation and expertise to optimize algorithm performance. Despite these challenges, the DRL-AC algorithm distinguishes itself through its stable performance and scalability.

7. Conclusions

This study aimed to address the FJSP using a novel deep reinforcement learning methodology anchored in an actor-critic framework. By effectively transforming the FJSP into a Markovian decision process, we harmoniously integrated value-based and policy-based strategies. Our unique representations for states, actions and rewards enabled a more intuitive understanding and handling of the problem. The experimental evaluation comprehensively assesses the proposed reinforcement learning framework through simulations on standard FJSP benchmarks, comparing the proposed method with several well-known heuristic scheduling rules, relevant RL algorithms, and intelligent algorithms. The results showed that the proposed DRL-AC algorithm consistently outperformed existing scheduling rules. When pitted against traditional reinforcement learning algorithms, it showcased a commendable accuracy in convergence.

It is pertinent to note that, while our DRL-AC algorithm has demonstrated a superior performance over priority rules and conventional reinforcement learning schedulers, it has yet to achieve optimal scheduling results in certain cases. This gap underscores the inherent complexity of the FJSP and the challenges of devising a one-size-fits-all solution. For practitioners aiming to optimize their flexible job shop schedules, our research provides a promising avenue that goes beyond traditional methods. However, we believe the solution can be further enhanced. Hyperparameter tuning stands out as a potential area of improvement for our proposed framework. In future explorations, we intend to optimize these hyperparameters to further bolster the DRL-AC's performance. Additionally, delving into advanced policy-based strategies, like A3C and PPO, could provide insights into even more robust scheduling methods. Given the dynamic nature of manufacturing environments, it is imperative to account for an array of disturbances that might arise during real-world production processes. Future research will focus on expanding the adaptability of the model to these unpredictable scenarios, thus ensuring holistic shop floor productivity.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Conflict of interest

The authors declare there is no conflict of interest.

References

1. M. Parente, G. Figueira, P. Amorim, A. Marques, Production scheduling in the context of Industry 4.0: Review and trends, *Int. J. Prod. Res.*, **58** (2020), 5401–5431. https://doi.org/10.1080/00207543.2020.1718794
2. A. Ham, Flexible job shop scheduling problem with parallel batch processing machine, in *2016 Winter Simulation Conference (WSC)*, (2016), 2740–2749. https://doi.org/10.1109/WSC.2016.7822311
3. K. Gao, F. Yang, M. Zhou, Q. Pan, P. N. Suganthan, Flexible job-shop rescheduling for new job insertion by using discrete Jaya algorithm, *IEEE Trans. Cybern.*, **49** (2019), 1944–1955. https://doi.org/10.1109/TCYB.2018.2817240
4. C. Lu, X. Li, L. Gao, W. Liao, J. Yi, An effective multi-objective discrete virus optimization algorithm for flexible job-shop scheduling problem with controllable processing times, *Comput. Ind. Eng.*, **104** (2017), 156–174. https://doi.org/10.1016/j.cie.2017.01.030
5. N. Shahsavari-Pour, B. Ghasemishabankareh, A novel hybrid meta-heuristic algorithm for solving multi-objective flexible job shop scheduling, *J. Manuf. Syst.*, **32** (2013), 771–780. https://doi.org/10.1016/j.jmsy.2013.04.015
6. K. Hu, L. Wang, J. Cai, L. Cheng, An improved genetic algorithm with dynamic neighborhood search for job shop scheduling problem, *Math. Biosci. Eng.*, **20** (2023), 17407–17427.
7. M. Nouiri, A. Bekrar, A. Jemai, S. Niar, A.C. Ammari, An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem, *J. Intell. Manuf.*, **29** (2016), 603–615. https://doi.org/10.1007/s10845-016-1233-5
8. I.A. Chaudhry, A. A. Khan, A research survey: Review of flexible job shop scheduling techniques, *Int. Trans. Oper. Res.*, **23** (2016), 551–591. https://doi.org/10.1111/itor.12199
9. C. Lu, L. Gao, J. Yi, X. Li, Energy-efficient scheduling of distributed flow shop with heterogeneous factories: A real-world case from automobile industry in China, *IEEE Trans. Ind. Inf.*, **17** (2020), 6687–6696. https://doi.org/10.1109/TII.2020.2963792
10. Y. Feng, L. Zhang, Z. Yang, Y. Guo, D. Yang, Flexible job shop scheduling based on deep reinforcement learning, in *2021 5th Asian Conference on Artificial Intelligence Technology (ACAIT)*, (2021), 660–666. https://doi.org/10.1109/ACAIT53529.2021.9731322
11. W. Song, X. Chen, Q. Li, Z. Cao, Flexible job-shop scheduling via graph neural network and deep reinforcement learning, *IEEE Trans. Ind. Inf.*, **19** (2022), 1600–1610. https://doi.org/10.1109/TII.2022.3189725

12. M. Ziaee, A heuristic algorithm for solving flexible job shop scheduling problem, *Int. J. Adv. Manuf. Technol.*, **71** (2014), 519–528. https://doi.org/10.1007/s00170-013-5510-z

13. P. Priore, A. Gomez, R. Pino, R. Rosillo, Dynamic scheduling of manufacturing systems using machine learning: An updated review, *AI Edam*, **28** (2014), 83–97. https://doi.org/10.1017/S0890060413000516

14. Y. Li, S. Carabelli, E. Fadda, D. Manerba, R. Tadei, O. Terzo, Machine learning and optimization for production rescheduling in Industry 4.0, *Int. J. Adv. Manuf. Technol.*, **110** (2020), 2445–2463. https://doi.org/10.1007/s00170-020-05850-5

15. G. Chenyang, G. Yuelin, L. Shanshan, Improved simulated annealing algorithm for flexible job shop scheduling problems, in *2016 Chinese Control and Decision Conference (CCDC)*, (2016), 2191–2196. https://doi.org/10.1109/CCDC.2016.7531349

16. G. Vilcot, J. C. Billaut, A tabu search algorithm for solving a multicriteria flexible job shop scheduling problem, *Int. J. Prod. Res.*, **49** (2011), 6963–6980. https://doi.org/10.1080/00207543.2010.526016

17. H. H. Doh, J. M. Yu, J. S. Kim, D. H. Lee, S. H. Nam, A priority scheduling approach for flexible job shops with multiple process plans, *Int. J. Prod. Res.*, **51** (2013), 3748–3764. https://doi.org/10.1080/00207543.2013.765074

18. C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, X. Chi, Learning to dispatch for job shop scheduling via deep reinforcement learning, *Adv. Neural Inf. Process. Syst.*, **33** (2020), 1621–1632.

19. J. Shahrabi, M. A. Adibi, M. Mahootchi, A reinforcement learning approach to parameter estimation in dynamic job shop scheduling, *Comput. Ind. Eng.*, **110** (2016), 75–82. https://doi.org/10.1016/j.cie.2017.05.026

20. H. X. Wang, H. S. Yan, An interoperable adaptive scheduling strategy for knowledgeable manufacturing based on SMGWQ-learning, *J. Intell. Manuf.*, **27** (2016), 1085–1095. https://doi.org/10.1007/s10845-014-0936-1

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

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

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

24. R. Li, W. Gong, C. Lu, A reinforcement learning based RMOEA/D for bi-objective fuzzy flexible job shop scheduling, *Expert Syst. Appl.*, **203** (2022), 117380. https://doi.org/10.1016/j.eswa.2022.117380

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

26. E. Yuan, S. Cheng, L. Wang, S. Song, F. Wu, Solving job shop scheduling problems via deep reinforcement learning, *Appl. Soft Comput.*, **143** (2023), 110436. https://doi.org/10.1016/j.asoc.2022.110436

27. J. C. Palacio, Y. VM. Jiménez, L. Schietgat, B. van Doninck, A. Nowé, A Q-learning algorithm for flexible job shop scheduling in a real-world manufacturing scenario, *Procedia CIRP*, **106** (2022), 227–232. https://doi.org/10.1016/j.procir.2022.02.183

28. J. Popper, V. Yfantis, M. Ruskowski, Simultaneous production and AGV scheduling using multi-agent deep reinforcement learning, *Procedia CIRP*, **104** (2021), 1523–1528. https://doi.org/10.1016/j.procir.2021.11.257

29. J. Chang, D. Yu, Z. Zhou, W. He, L. Zhang, Hierarchical reinforcement learning for multi-objective real-time flexible scheduling in a smart shop floor, *Machines*, **10** (2022), 1195. https://doi.org/10.3390/machines10121195

30. L. Yin, X. Li, L. Gao, C. Lu, Z. Zhang, A novel mathematical model and multi-objective method for the low-carbon flexible job shop scheduling problem, *Sustainable Comput. Inf. Syst.*, **13** (2017), 15–30. https://doi.org/10.1016/j.suscom.2017.01.004

31. P. Burggräf, J. Wagner, T. Saßmannshausen, D. Ohrndorf, K. Subramani, Multi-agent-based deep reinforcement learning for dynamic flexible job shop scheduling, *Procedia CIRP*, **112** (2022), 57–62. https://doi.org/10.1016/j.procir.2022.01.026

32. S. Yang, Z. Xu, J. Wang, Intelligent decision-making of scheduling for dynamic permutation flowshop via deep reinforcement learning, *Sensors*, **21** (2021), 1019. https://doi.org/10.3390/s21031019

33. J. P. Huang, L. Gao, X. Y. Li, C. J. Zhang, A novel priority dispatch rule generation method based on graph neural network and reinforcement learning for distributed job-shop scheduling, *J. Manuf. Syst.*, **69** (2021), 119–134. https://doi.org/10.1016/j.jmsy.2022.12.008

34. B. A. Han, J. J. Yang, Research on adaptive job shop scheduling problems based on dueling double DQN, *IEEE Access*, **8** (2021), 186474–186495. https://doi.org/10.1109/ACCESS.2020.3029868

35. J. Bergdahl, *Asynchronous Advantage Actor-Critic with Adam Optimization and A Layer Normalized Recurrent Network*, Student thesis, (2017).

36. B. Han, J. Yang, A deep reinforcement learning based solution for flexible job shop scheduling problem, *Int. J. Simul. Modell.*, **20** (2021), 375–386. https://doi.org/10.2507/IJSIMM20-2-CO7

37. A. Henchiri, M. Ennigrou, Particle swarm optimization combined with tabu search in a multi-agent model for flexible job shop problem, in *Advances in Swarm Intelligence: 4th International Conference*, (2013), 385–394 .

38. W. Xia, Z. Wu, An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems, *Comput. Indust. Eng.*, **48** (2005), 409–425. https://doi.org/10.1016/j.cie.2004.11.002

39. I. Kacem, S. Hammadi, P. Borne, Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems, *IEEE Trans. Syst. Man Cybernetics*, **32** (2002), 1–13. https://doi.org/10.1109/TSMCC.2002.1000156

40. J. Hurink, B. Jurisch, M. Thole, Tabu search for the job-shop scheduling problem with multi-purpose machines, *Oper. Res. Spektrum*, **15** (1994), 205–215. https://doi.org/10.1007/BF01720537

41. X. Li, L. Gao, An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem, *Int. J. Prod. Econ.*, **174** (2016), 93–110. https://doi.org/10.1016/j.ijpe.2016.01.016

42. J. Stopforth, D. Moodley, Continuous versus discrete action spaces for deep reinforcement learning, in *Proceedings of the South African Forum for Artificial Intelligence Research*, (2019).

AIMS Press