An update to this article is included at the end

Journal of Manufacturing Systems 74 (2024) 329–345

Contents lists available at ScienceDirect

Journal of Manufacturing Systems

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

Technical paper

A novel collaborative agent reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for flexible job shop scheduling problem

Wenquan Zhang [a], Fei Zhao [b,*], Yong Li [c], Chao Du [d], Xiaobing Feng [a], Xuesong Mei [b]

[a] State Key Laboratory for Manufacturing Systems Engineering, Shaanxi Key Laboratory of Intelligent Robots, School of Mechanical Engineering, Xi'an Jiaotong University, Xi'an, 710049, China
[b] School of Physics, Xi'an Jiaotong University, Xi'an, 710049, China
[c] Xi'an Kunlun Industry (Group) Co., Ltd., Xi'an, Shaanxi Province, China
[d] Shaanxi Fast Auto Drive Group Co., Ltd., No. 129 West Avenue, High-tech Zone, Xi'an, Shaanxi Province, China

ARTICLE INFO

Keywords:
Soft actor–critic
Flexible job shop scheduling
Graph attention network
Transformer encoder
Deep Reinforcement Learning (DRL)

ABSTRACT

The Flexible Job Shop Scheduling Problem (FJSP), a classic NP-hard optimization challenge, has a direct impact on manufacturing system efficiency. Considering that the FJSP is more complex than the Job Shop Scheduling Problem (JSSP) due to its involvement of both job and machine selection, we have introduced a collaborative agent reinforcement learning (CARL) architecture to tackle this challenge for the first time. To enhance Co-Markov decision process, we introduced disjunctive graphs for the representation of state features. However, the representation of states and actions often leads to suboptimal solutions due to intricate variability. To achieve superior outcomes, we refined our approach to representing states and actions. During the solving process, we employed Graph Attention Network (GAT) to extract global state information from the disjunctive graph and used a Transformer Encoder to quantitatively capture the competitive relationships among machines. We configured two independent encoder–decoder components for job and machine agents, enabling the generation of two distinct action strategies. Finally, we employed the Soft Actor–Critic (SAC) algorithm and an integrated Deep Q Network (DQN) known as D5QN to train the decision network parameters of job and machine agents. Our experiments revealed that after just one training session, collaborative agents acquired exceptional scheduling strategies. These strategies excel not only in solution quality compared to traditional Priority Dispatching Rules (PDR) but also outperform results achieved by some metaheuristic and reinforcement learning algorithms. Additionally, they exhibit greater speed than OR-Tools. Moreover, the empirical findings on both randomized and benchmark instances underscore the remarkable robustness of our acquired policies in practical, large-scale scenarios. Notably, when confronted with the DPpaulli dataset, characterized by a considerable imbalance between the number of operations and machines, our approach achieved optimality in 11 out of 18 FJSP instances.

1. Introduction

Production scheduling constitutes a pivotal component of manufacturing systems, with particular significance in today's markets characterized by product diversity, small batch sizes, and customization. Effective production scheduling is paramount for maintaining competitiveness in the manufacturing industry, as it ensures on-time delivery performance and enhances overall corporate competitiveness [1]. In this context, the FJSP emerges as a critical production paradigm, facilitating effective resource allocation and task scheduling in diverse

Abbreviations: FJSP, Flexible Job Shop Scheduling Problem; JSSP, Job Shop Scheduling Problem; CARL, Collaborative agent reinforcement learning; GAT, Graph Attention Network; DQN, Deep Q Network learning; PDR, Priority dispatching rule; GA, Genetic algorithm; PSO, Particle swarm optimization; TS, Tabu search; ACO, Ant colony optimization; SA, Simulated annealing; DE, Differential evolution algorithm; ABC, Artificial bee colony; GW, grey wolf optimization algorithm; DRL, Deep Reinforcement Learning; MDP, Markov Decision Process; PPO, Proximal Policy Optimization; GNNs, Graph Neural Networks; CNN, Convolutional Neural Network; GCN, Graph Convolutional Network; DDQN, Double DQN; NDQN, Noisy DQN; PER, Priority Experience Replay; MLP, Multi-layer perceptron; FNN, Feedforward neural network

* Corresponding author.
E-mail address: ztzhao@mail.xjtu.edu.cn (F. Zhao).

https://doi.org/10.1016/j.jmsy.2024.03.012
Received 26 November 2023; Received in revised form 22 February 2024; Accepted 20 March 2024
Available online 8 April 2024
0278-6125/© 2024 The Society of Manufacturing Engineers. Published by Elsevier Ltd. All rights reserved.

production environments. FJSP is vital for reducing production cycles, optimizing resource utilization, and ensuring timely product deliveries. For the manufacturing industry, solving the FJSP is not only necessary but also a key strategy to gain a competitive edge in fiercely competitive markets [2].

Currently, approaches for solving FJSP can be broadly categorized into two classes: exact and approximation methods. The exact approaches consist of branch and bound method [3], branch-and-cut procedure [4], Mathematical Integer Linear Programming [5–9], and constraint programming [10–12]. These methods employ mathematical programming and traditional operations research methods to find optimal solutions within the solution space. However, these methods have limitations, rendering them suitable only for small-scale scheduling problems [13].

Approximation techniques solving FJSP encompass rule-based scheduling algorithms, metaheuristic methods, and machine learning approaches. Rule-based scheduling comprises job and machine selection rules aimed at minimizing the maximum completion time. Doh et al. [14] have introduced a practical priority scheduling approach that combines operation and machine selection rules with job sequencing. In light of heuristic methods' insensitivity to problem scale, making them viable for addressing large-scale problems, Saqlain et al. [15] have introduced a Monte Carlo Tree Search-based algorithm to handle intricate job scheduling problems in real-time shop floor environments. Nevertheless, attaining a high-quality combination of scheduling rules to address FJSP typically necessitates a multitude of comparative experiments. Hence, the selection of adaptive scheduling rules is currently a popular solution.

A wide range of metaheuristic methods has been employed in addressing FJSP, including genetic algorithm (GA) [16–18], particle swarm optimization (PSO) [19,20], tabu search (TS) [21], ant colony optimization (ACO) [22,23], simulated annealing (SA) [24,25], artificial bee colony (ABC) [26,27], differential evolution algorithm (DE) [28] and grey wolf optimization algorithm (GW) [29], all of which have demonstrated notable effectiveness. However, the utilization of metaheuristics often results in substantial computational complexity, leading to extended processing times and resource utilization. This complexity arises from the need to explore a vast search space to identify optimal solutions. Moreover, these approaches are susceptible to local optima, where they may converge to suboptimal solutions that fail to maximize the overall performance of the system [30]. In the contemporary landscape, reinforcement learning algorithms have become a novel approach for addressing FJSP, with current research efforts aimed at improving their suitability for this complex problem.

Recently, Deep Reinforcement Learning (DRL) methods offer a promising solution for tackling the Job Shop Scheduling Problem [31]. These methods treat the scheduling process as a Markov Decision Process (MDP) and use a neural network model with adjustable parameters. This model takes information about the production environment as input and assigns priorities to each possible scheduling action, like allocating an operation to a machine. This holistic approach streamlines the learning of scheduling while significantly reducing repetitive and redundant tasks. DRL models are rigorously trained on a dataset of production processes to acquire optimal behavior, adaptively maximizing the cumulative reward linked to production objectives within a specific state [32].

The effectiveness of these DRL methods heavily relies on the design of the state representation, serving functions such as precise problem modeling, capturing key features, managing complexity, enhancing learning performance, and ensuring generality. This has led to the development of a specialized state representation. Presently, scheduling states can be categorized into three main types. The first type utilizes production information or statistics related to it as states, encompassing aspects such as processing details, the production environment, and other pertinent production-related factors. For example, Wang et al. [33] delineate three matrices: a job processing time matrix, a job processing status matrix, and a machine allocation status matrix. These matrices are subsequently input into a neural network for policy learning. Luo et al. [34] employ ten distinct pieces of information, including metrics such as the number of machines, average machine utilization, and due date tightness, as states, to introduce an effective DQN. A primary limitation of these approaches is the continuous nature of information, which can result in the curse of dimensionality. The second category defines states based on quantity relationships among production information or statistics, a strategy that helps mitigate the challenge of expanding state spaces as problems scale up. For job shop scheduling, Zhao et al. [35] define six states by comparing estimated average idle time and estimated average remaining time. However, this approach has the potential drawback of information loss. The third classification transforms scheduling problems into graphs and defines states based on node and edge configurations within these graphs. Zhang et al. [36] model job shop scheduling problems using disjunctive graphs and propose a Proximal Policy Optimization (PPO) method to optimize Graph Neural Networks (GNN). Concerning adaptive job shop scheduling problems, Han and Yang [37] represent production information as multi-channel images and employ Convolutional Neural Network (CNN) to approximate state–action value functions. Park et al. [38] incorporate dynamic attributes into node features and design a customized GNN for message passing to learn operation embeddings. Chen et al. [39] extract disjunctive graph features using graph embedding techniques for downstream decision-making and introduce an attention-based model to generate solutions for JSSP. In the context of dynamic scheduling problems for flexible manufacturing systems, Hu et al. [40] model the problem using Petri nets and approximate the state–action value functions in DQN with Graph Convolutional Network (GCN). These methods consider the structural characteristics of the problem and effectively reflect the production environment. Nonetheless, a primary limitation lies in the relatively straightforward state representation, potentially resulting in limited exploration. In summary, our approach to designing states for the FJSP takes into account the trade-off between information loss and the size of the state space.

Moreover, the complexity of FJSP, which involves both operation sequencing and machine selection, presents significant challenges. Currently, the prevailing approach addresses these challenges by integrating operations and machines into a single combined action for the solution [41,42]. However, this approach is limited by existing single-agent reinforcement learning algorithms, which can only make single action decisions at each time step, thereby posing substantial obstacles in the design of effective learning mechanisms [36]. To address this challenge, this article introduces a CARL [43] scheduling optimization model for the first time, as illustrated in Fig. 1. Agent 1 initially selects candidate operations, and then Agent 2, based on the chosen operation, selects suitable machines that meet the criteria. Its advantage lies in fostering the sharing of information, knowledge, and experience among multiple agents, enabling them to collaborate and address complex scheduling problems more comprehensively and effectively. This approach not only reduces computational burdens and expedites the search for optimal solutions but also enhances learning performance and algorithm adaptability.

In addition, given the intricate one-to-many relationship between operations and machines, the utilization of neural networks for encoding the scheduling state becomes notably challenging. A critical challenge also lies in designing an effective network architecture capable of capturing and extracting crucial information from the raw scheduling data. This paper introduces an innovative approach, partitioning the FJSP with machine nodes into two distinct subgraphs. The subgraph encompassing operation nodes is encoded using the state-of-the-art GAT architecture [44], facilitating the learning of precedence constraints among operations. Subsequently, a quantitative assessment of operation priorities is conducted, incorporating machine competition details and raw machine characteristics through the improved DQN [45] agent.

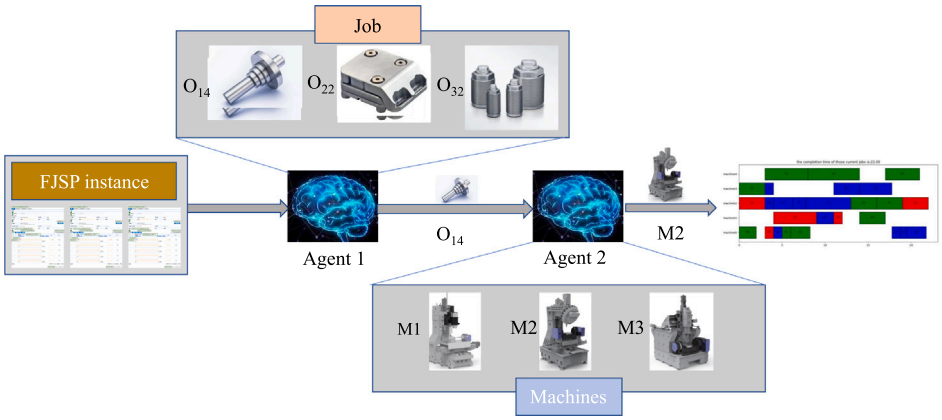


Fig. 1. CARL scheduling optimization model.

In pursuit of enhancing model generalization and addressing the performance of solving large-scale FJSP, this paper introduces structural framework of the CARL algorithm designed for solving FJSP. With the objective of minimizing the maximum completion time, this model integrates various components, including graph embedding techniques, an actor–critic framework employing maximum entropy reinforcement learning, and an ensemble of experience replay and competitive networks, forming an improved DQN agent structure. This comprehensive approach ensures the model's adaptability and the rapid generation of high-quality scheduling solutions within a short timeframe. The collaborative agent framework holds great promise in efficiently tackling large-scale FJSPs, effectively addressing complex scheduling challenges, and enhancing manufacturing efficiency.

The main contributions of this paper are listed as follows:

1. We have introduced an exceptionally efficient and innovative collaborative agent scheduling framework for FJSP for the first time. It overcomes the issue of poor generalizability observed in previous models and can resolve problems of any scale with just one training session.
2. We have devised dedicated states tailored for FJSP, effectively reflecting its crucial aspects, thereby augmenting the problem comprehension and decision-making capabilities of reinforcement learning agents.
3. We have pioneered the introduction of SAC with an entropy regularization term to increase the exploration of workpieces during the training process. Additionally, we have integrated multiple tricks into DQN to enhance the stability of the training process.
4. We partitioned the FJSP into two subgraphs: one subgraph utilizes graph attention networks to learn operation precedence constraints and relevant features, while the other incorporates a transformer encoder to quantitatively model the degree of competition among the same workpieces.

The rest of this paper is structured as follows. Section 2 defines FJSP, provides a disjunctive graph representation, and introduces relevant background knowledge related to our study, including GAT, Transformer, SAC, and D5QN. In Section 3, we introduce our scheduling framework, encompassing the scheduling environment and network training. Section 4 elaborates on the training process, hyperparameter adjustments, ablation study, and presents a series of comparative experiments conducted to assess the performance of CARL-FJSP. Finally, in Section 5, we summarize our conclusions and discuss potential avenues for future research.

2. Preliminaries

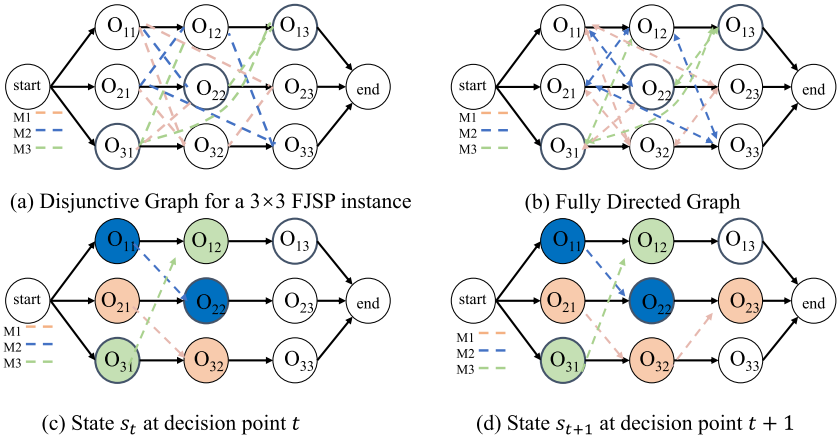
2.1. Flexible job-shop scheduling problem (FJSP)

FJSP is characterized by a set of jobs $J = \{J_1, J_2, J_3, \ldots, J_n\}$ that need to be processed on a set of machines $M = \{M_1, M_2, M_3, \ldots, M_m\}$. Each job $J_i \in J$ consists of p operations, and there are strict precedence constraints between these operations, denoted as $O_{i1} \rightarrow O_{i2} \rightarrow O_{i3} \rightarrow \cdots \rightarrow O_{ip}$. Each operation can be processed on one or multiple machines, and the processing time can vary depending on the machine. The following assumptions and constraints define the FJSP: (1) Each machine can process only one operation at a time without interruption. (2). There is no priority between operations of different jobs, but operations within the same job have precedence constraints, with each subsequent operation waiting for the completion of the preceding one. (3). All jobs can start processing at time 0. The objective of the FJSP is to minimize the maximum completion time, which corresponds to ensuring that the job with the longest processing time completes its tasks in the shortest possible time. This objective can be formally expressed using Eq. (1).

$$f = \min_{1 \le i \le n}(\max(C_i)) \tag{1}$$

2.2. Disjunctive graph

The disjunctive graph model, denoted as G = (N,C,D), is defined as follows: N comprises nodes representing the operations of all jobs, including the virtual start and end nodes; C represents the directed arcs connecting adjacent operations within the same job; $D = U_{K=1}^{M} D_k, k \in M$ consists of undirected arcs that connect adjacent operations processed on the same machine, with each machine forming a subset of disjunctive arcs. Therefore, the disjunctive graph G is a mixed graph composed of both undirected and directed arcs. Fig. 2 depicts the disjunctive graph model for a 3 × 3 FJSP instance, where Fig. 2(a) illustrates the disjunctive graph representation of this instance. The black solid line arrows denote the connecting arcs C, while the different colors of dashed lines represent disjunctive arcs D connecting adjacent operations processed on different machines. In scheduling problem solving, an undirected arc becomes two oppositely directed arcs, as in Fig. 2(b). After removing connection arcs C and virtual nodes from disjunctive graph G, m disjoint subgraphs represent task sequences for different machines. The goal in solving the scheduling problem is to select directions for oppositely directed arcs, efficiently ordering tasks on machines to ensure feasible solutions. The method described above initially adds all connection arcs and disjunctive arcs to the graph in the initial state, and subsequently selects the corresponding directed arcs, resulting in exceptionally high graph complexity and challenging problem solving. To simplify the algorithm and reduce problem-solving complexity, this study employs an arc-adding approach in constructing



(a) Disjunctive Graph for a 3×3 FJSP instance (b) Fully Directed Graph

(c) State s_t at decision point t (d) State s_{t+1} at decision point $t+1$

Fig. 2. Disjunctive graph model.

the disjunctive graph model for scheduling, as illustrated in Fig. 2(c) and (d). At decision point t, the action $a = (O_{32}, M_1)$ involving disjunctive arcs (O_{32}, O_{23}) is added to the graph, resulting in the description of the next state s_{t+1}. This approach eliminates the undirected disjunctive arcs in the initial state, significantly reducing computational complexity.

2.3. Graph attention network(GAT)

The GAT [46] is an architecture within the field of GNNs. It introduces attention mechanisms when dealing with graph data, allowing the network to assign different weights to relationships between different nodes. This personalized attention mechanism enables GAT to flexibly capture complex relationships among nodes in a graph, leading to significant achievements in various applications involving graph data.

To more comprehensively represent node features, a feature transformation is applied to each node h_i. Self-attention operations are performed on each node in the graph, computing attention weights for any two nodes. The formula for calculating the importance of node i to node j is as follows:

$$e_{i,j} = \text{LeakyReLU}\left(\vec{\mathbf{a}}^{\top}\left[\left(\mathbf{W}\vec{h}_i\right) \| \left(\mathbf{W}\vec{h}_j\right)\right]\right) \tag{2}$$

Here, h_i and h_j represent the features of nodes i and j, respectively. N_i denotes the set of neighboring nodes of node i (including node i itself), and $e_{i,j}$ is the attention weight obtained by a specific linear transformation of node features, W is the weight matrix, a is the attention parameter vector, LeakyReLU is the leaky rectified linear unit, and || denotes vector concatenation.

To make the coefficients easily comparable across different nodes, the softmax function is used to normalize them within the set. The normalization is performed in Eq. (3):

$$\alpha_{ij} = softmax(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in N_i}\exp(e_{ij})}, \forall i. \tag{3}$$

Finally, the normalized attention coefficients are linearly combined with their corresponding features to serve as the final output features for each node. This is expressed in Eq. (4):

$$\vec{h}'_i = \sigma\left(\sum_{j\in N_i}\alpha_{i,j}\mathbf{W}\vec{h}_j\right), \forall i. \tag{4}$$

Through this attention calculation, each node dynamically adjusts its representation based on the features of its neighbors, effectively capturing the relationships between nodes in the graph structure.

2.4. Transformer

The Transformer is a deep learning architecture by Vaswani et al. [47]. It was originally designed for natural language processing tasks, such as machine translation, but its attention mechanism has since been applied to various domains, including computer vision and graph processing. This architecture incorporates key components such as self-attention mechanisms, an encoder–decoder structure, multi-head attention, positional encoding, feedforward neural networks, and layer normalization with residual connections.

The self-attention mechanism enables the model to weigh the significance of different positions in the input sequence. The encoder processes the input, and the decoder generates the output, with multi-head attention capturing various aspects of the input sequence. Positional encodings are introduced to convey information about element positions, addressing the inherent lack of sequence order understanding in Transformers.

Both encoder and decoder layers feature feedforward neural networks, and layer normalization with residual connections ensures stable training. Transformers, known for their parallelization capabilities and proficiency in capturing long-range dependencies, have consistently achieved state-of-the-art performance across diverse natural language processing tasks.

2.5. D5QN algorithm principles

DQN [48] integrates deep learning and reinforcement learning, utilizing the Q function to approximate state–action values. This method effectively handles continuous value functions, a challenge for standard Q-Learning. During training, periodic replacement occurs between the separate target network $Q^{-}(\theta^{-})$ and the network $Q(\theta)$, ensuring stable training. At each time step t, the training target y_t is computed using Eq. (5).

$$y_t = r_{t+1} + \gamma \max_{a'} Q^{-}\left(s_{t+1}, a'; \theta^{-}\right) \tag{5}$$

Double Deep Reinforcement Learning (DDQN) is employed to address the issue of overestimation during the training of DQN. It consists of two DQN networks, where one Q network is responsible for selecting the action with the maximum Q-value, and the other network Q' is used to estimate the state–action value function $Q'(\theta)$, which is crucial or computing the training target y_t, as shown in Eq. (6).

$$y_t = r_{t+1} + \gamma Q'\left(s_{t+1}, \arg\max Q\left(s_{t+1}, a'; \theta\right); \theta'\right) \tag{6}$$

The Noisy DQN (NDQN) adds noise to the parameters of the Q network. At the beginning of each episode, gaussian noise is applied to each parameter of the $Q(s, a; w)$, transforming the function $Q(s, a; w)$ into $\tilde{Q}(s, a, \xi; \mu, \sigma)$. Here, μ and σ represent the mean and standard

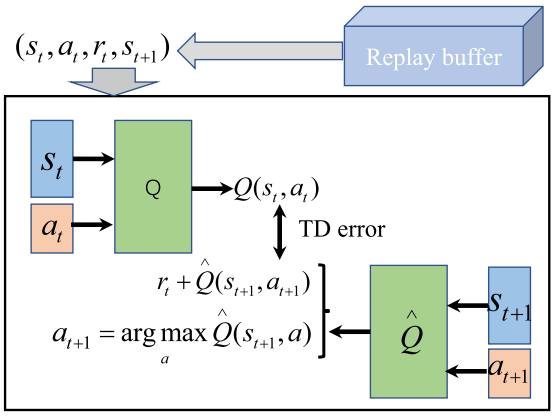


Fig. 3. Priority experience replay mechanism.

deviation, respectively, which are parameters of the neural network that need to be learned through experience. ξ is random noise, with each of its elements independently sampled from the standard normal distribution N(0, 1). This increases the exploration probability. The new action selection method is then implemented using $\tilde{Q}(s, a, \xi; \mu, \sigma)$, as shown in Eq. (7).

$$a_t = \arg\max_{a \in A} \tilde{Q}\ (s, a, \xi; \mu, \sigma) \tag{7}$$

Priority Experience Replay (PER) refers to the practice of conducting experience replay with certain priorities, increasing the chances of sampling important data to achieve improved training results, as depicted in Fig. 3.

Dueling DQN is an extension of the original DQN that divides the Q-network into two distinct branches. One branch calculates the action advantage function $A(s, a)$ independently of the state, while the other branch computes the state value function $V(s)$. These two branches are then combined to form the state–action value function.

$$Q(s, a) = V(s) + A(s, a) \tag{8}$$

The D5QN algorithm is proposed by integrating the DQN, Noisy DQN, PER DQN, Double DQN and Dueling DQN on the basis of the DQN algorithm. D5QN effectively leverages the advantages of each of these algorithms. It uses neural networks to model states, mitigates overestimation through DQN, introduces noise for increased exploration diversity, accelerates training with prioritized experience replay, and enhances Q-value training with competitive networks. Importantly, these optimization techniques do not negatively impact each other, achieving a complementary effect that combines strengths and mitigates weaknesses.

2.6. SAC algorithm principles

The SAC algorithm [49] is an advanced policy gradient method in reinforcement learning. It follows an actor–critic approach and is widely employed for both discrete and continuous control tasks. As a model-free, off-policy algorithm, it merges the advantages of both policy iteration and value iteration methods. The objective is defined as follows.

$$\pi^* = \arg\max_{\pi} \sum_t \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r\left(\mathbf{s}_t, \mathbf{a}_t\right) + \alpha \mathcal{H}\left(\pi\left(\cdot \mid \mathbf{s}_t\right)\right) \right] \tag{9}$$

where the temperature parameter α determines the relative importance of the entropy term compared to the reward, thereby controlling the stochasticity of the optimal policy.

The SAC update optimizes the Q-function and policy simultaneously, with the Q-function loss $J_Q\left(\phi_i\right)$ minimizing the mean squared error and the policy loss $J_\pi(\theta)$ maximizing the expected objective function, while an entropy loss $J(\alpha)$ encourages exploration. These objectives are captured by the following formulas: loss function for training the Q-function $J_Q\left(\phi_i\right)$, loss function $J_\pi(\theta)$ for training the policy, and entropy loss function $J(\alpha)$.

$$J_Q\left(\phi_i\right) = \mathbb{E}_{(s,a,r,s')} \left[\left(r + \gamma \max_{a'} Q_{\phi_i'}\left(s', a'\right) - Q_{\phi_i}(s, a) \right)^2 \right] \tag{10}$$

$$J_\pi(\theta) = \mathbb{E}_s \left[\mathbb{E}_{a \sim \pi_\theta(s)} \left[\alpha \log \pi_\theta(a \mid s) - Q_{\phi_i}(s, a) \right] \right] \tag{11}$$

$$J(\alpha) = \mathbb{E}_{a_t \sim \pi_t} \left[-\alpha \log \pi_t\left(a_t \mid s_t\right) - \alpha \mathcal{H}_0 \right] \tag{12}$$

where (s, a, r, s') denotes a tuple of state, action, reward, and next state, respectively. θ represents the parameters of the actor, ϕ_i represents the parameters of the Q-function, ϕ_i' represents the parameters of the target Q-network, γ is the discount factor, a denotes the action, $\pi_\theta(a \mid s)$ represents the policy distribution over actions given a state s. $\mathcal{H}_0$ is a baseline entropy used for stability during training, $\pi_t\left(a_t \mid s_t\right)$ represents the probability of selecting action a_t given the current policy s_t.

3. FJSP scheduling framework

The disjunctive graph of the FJSP in Fig. 3 provides a comprehensive overview of the scheduling conditions for the entire problem, encompassing factors such as precedence constraints among different operations of jobs, the assignment of operations to specific machines, and machine-based operation sequencing constraints. To optimize feature extraction from this graph, we have designed the structural framework of the CARL algorithm for solving FJSP, as illustrated in Fig. 4. This framework introduces encoder–decoder components for two different agents. The encoder identifies the environment and outputs the state for each agent based on the environment, and then the decoder generates actions based on these states. The encoder–decoder components are specifically designed to generate action policies for job operations and machine.

3.1. MDP formulation

FJSP involves a sequential decision-making task, requiring the handling of $|O|$ consecutive decision points. At each decision point, the SAC agent prioritizes eligible candidate operations and selects one using sampling or a greedy decoding strategy. Afterward, the D5QN agent prioritizes compatible machines and chooses one for executing the operation task using similar strategies. At each decision point, each agent needs to control one action. The collaborative reinforcement learning problem involving multiple agents is modeled as a Cooperative Markov Decision Process characterized by a five-tuple, known as co-MDP, defined by a quintuple (S, A, P, r, γ). Since the standard MDP reinforcement learning model for a single agent cannot be directly applied to a co-MDP, this paper introduces dual-layer action space consisting of the operation selection action space and the machine selection action space, where the action set is represented as $A = A_o \times A_\mathrm{m}$, with A_o being the subset of eligible candidate operation actions and A_m being the subset of compatible machine actions. Additionally, the paper defines specific state features and reward functions for the co-MDP. The corresponding co-MDP is defined as follows:

State: Based on the characteristics of FJSP, the scheduling state at each time step is composed of both the global state s_t^o and the local state s_t^m. The global state s_t^o is derived from the disjunctive graph $G(t) = (O, C \cup D_u(t), D(t))$, encompassing operation nodes classified into three categories: completed operations (denoted as $D_u(t) \in D$), operations in progress, and unplanned operations (designated by $D(t) \in D$). As each decision at time step (t) depends exclusively on the current production state, relevant operations, excluding those already completed and those not affecting future scheduling, are integrated into the global state. The local state s_t^m represents the status of all machines, with machines irrelevant to future scheduling indicating those

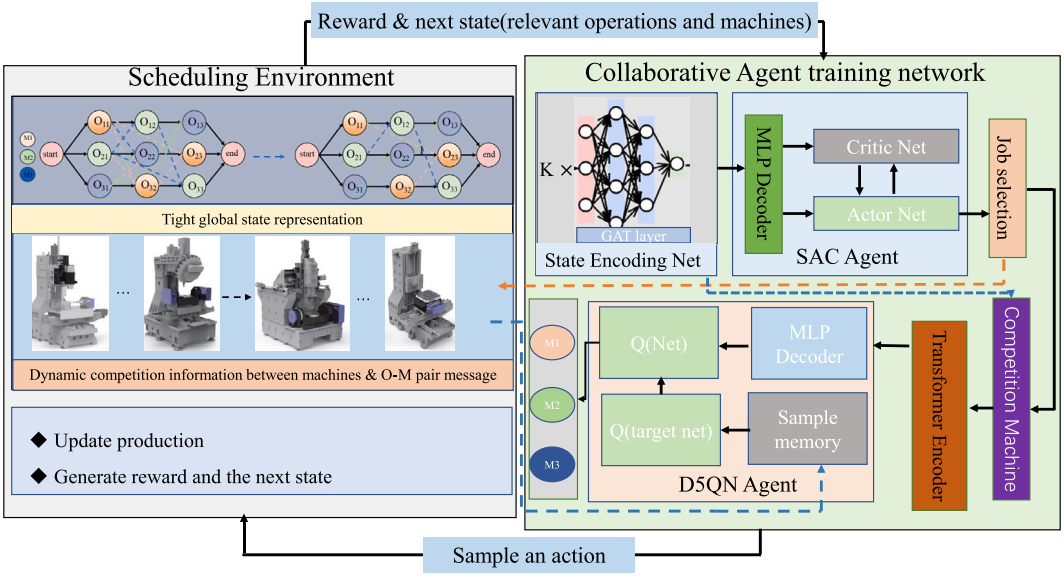


Fig. 4. Structural framework of the CARL algorithm for solving FJSP.

unable to handle remaining unplanned operations. Each machine node M_k is incorporated within the local state s_t^m. These descriptions are thoughtfully crafted to provide a succinct yet comprehensive portrayal of the static and dynamic attributes of scheduling-related entities. This state representation combines both global and local features of FJSP to offer a comprehensive view of the scheduling state. These descriptions effectively capture the essential information required for scheduling decisions at each time step. And the information about all machines, operations, and machine-operation pairs can be found in the appendix.

Action: At time t, the action $a_t \in |A_t|$ consists of two components: $a_t^o \in A_t^o$ and $a_t^m \in A_t^m$, which correspond to the choices made by the job agent for selecting a workpiece and the machine agent for selecting a machine, respectively. Here, A_t^o represents the set of workpieces available for selection by the job agent at time t, and A_t^m represents the set of machines available for selection by the machine agent at time t for processing the corresponding operation. As the job-shop scheduling process advances and more jobs are finished, the feasibility action set's size, denoted as $|A_t|$, decreases.

Transition: At time step t, the job-agent makes a decision a_t^o to select a specific operation node from the set of feasible operation nodes, while the machine-agent chooses an action a_t^m to select a machine corresponding to the chosen operation. Subsequently, the disjunctive arcs are updated based on the current action set a_t, resulting in a new disjunctive graph (as illustrated in Fig. 1 as the next global state s_{t+1}^o and local state s_{t+1}^m).

Reward: The reward is defined as the difference between the makespan of the partial schedule in states s_t and s_{t+1}. Mathematically, it can be expressed as:

$$R(a_t, s_t, s_{t+1}) = H_{\max}(s_t) - H_{\max}(s_{t+1}) \tag{13}$$

where $H_{\max}(s_i)$ represents the makespan of the partial schedule at state s_i. When the discount factor γ is set to 1, the cumulative reward in a solving episode, denoted as G, is computed as the sum of the rewards across all decision steps. The formula is as follows:

$$G = \sum_{t=0}^{|O|} R(a_t, s_t) = H_{\max}(s_0) - C_{\max} \tag{14}$$

For a specific problem instance, the initial makespan $H_{\max}(s_0)$ remains constant, implying that minimizing the makespan $C_{\max}$ and maximizing G (the objective function) are equivalent objectives. Therefore, the agent's goal is to minimize the makespan of the partial schedule at each step to achieve the maximum cumulative reward.

Policy: The scheduling process involves the utilization of $\pi(a_o, a_m|s)$ as well as $\theta = [\theta_o, \theta_m]$, signifying the scheduling strategies for the agents along with their corresponding strategy parameters. During the entire decision process, these strategies guide the job-agent and machine-agent to execute actions $\pi_{\theta_o}(a_o|s_t)$, $\pi_{\theta_m}(a_m|s_t, a_o)$ from the probability distributions over the sets of available operations A_t^o and compatible machines A_t^m, respectively.

3.2. Job operation encoder and decoder based on SAC

(1) Job operation encoder

In the co-MDP framework, the disjunctive graph offers a holistic view of the scheduling state, including critical details like priority constraints, machine-specific processing sequences, operation-compatible machine sets, and their respective processing times. To enhance scheduling performance, we employ GNNs, known for their iterative and nonlinear feature extraction capabilities, widely successful in practical applications. This suite of methods adeptly captures and integrates information related to both operations and machines, providing a potent toolbox for achieving efficient FJSP scheduling.

To extract the disjunctive graph features of the global state s_t^o at each time step t, a variant of the Graph Attention Network [44] is employed in this study. Utilizing the GAT model for state extraction from the disjunctive graph allows for the association of operations within the same job and the identification of the most critical operations based on their intrinsic attributes. This approach enhances the understanding of the current state by the agent and facilitates more efficient action generation. At each discrete time step t, the global state s_t^o is represented as nodes, denoted as $G(t) = (O, C \cup D_u(t), D(t))$. Specifically, for each operation $O_{ij} \in O_u$ with input features $h_{O_{ij}} \in \aleph^{d_o}$, we employ a graph attention network to compute the attention coefficients for O_{ij}. This allows us to model the relationships between O_{ij} and its preceding operation O_{ij-1} and succeeding operation O_{ij+1} (if it exists), as shown below.

$$e_{i,j,p} = \text{LeakyReLU}\left(\vec{\mathbf{a}}^{\top}\left[\left(\mathbf{W}h_{O_{ij}}\right) \| \left(\mathbf{W}h_{O_{ip}}\right)\right]\right) \tag{15}$$

where $\vec{\mathbf{a}}^{\top} \in \aleph^{2d'_o}$, and $W \in \aleph^{d'_o \times d_o}$ are linear transformations and all $|p - j| \leq 1$

It is worth noting that the computation here is similar to that in GAT, but we have restricted its scope. Since certain operations may not have predecessors (or successors) or may be deleted in some steps,

we apply dynamic masking to the attention coefficients of these predecessors and successors. Subsequently, we employ the softmax function to normalize the values of e_{ijk} for the selection of operation, resulting in normalized attention coefficients, denoted as α_{ijp}.

$$\alpha_{ijp} = Softmax(mask(e_{ijk})) \tag{16}$$

Finally, we obtain the output feature vector $h'_{O_{ij}} \in \aleph^{d'_o}$ by taking a weighted linear combination of the transformed input features $\mathbf{W}h_{O_{ij-1}}$, $\mathbf{W}h_{O_{ij}}$, and $\mathbf{W}h_{O_{ij+1}}$, followed by a nonlinear activation function δ:

$$h'_{O_{ij}} = \sigma\left(\sum_{p=j-1}^{j+1} \alpha_{i,j,p}\mathbf{W}h_{O_{ip}}\right) \tag{17}$$

To capture diverse relationships between operations, we employed multiple attention heads in the operation-process encoder. This approach has been demonstrated as effective in guiding agent to make accurate decisions and comprehending the intricacies of scheduling. Let K represent the number of attention heads in the attention layer, each employing distinct attention mechanisms with different parameters. Initially, the outputs of each attention head were integrated through an aggregation operation. Subsequently, an activation function δ was applied to obtain the layer's output.

$$h^t_{O_{ij}} = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{p=j-1}^{j+1} \alpha^k_{i,j,p}\mathbf{W}^k h_{O_{ip}}\right) \tag{18}$$

The entire graph pooling vector is given by the following formula:

$$h^t_G = 1/\mathcal{O}\sum_{O_{ij}\in\mathcal{O}} h^t_{O_{ij}} \tag{19}$$

(2) Job operation decoder

To enable the SAC agent to make decisions at discrete time step t based on the state s^o_t, a decoder has been implemented for the SAC agent to process the output from the encoder. This decoder is constructed using a multi-layer perceptron (MLP) and is responsible for producing the operation selection scores, computed according to the following formula. The entire graph pooling vector is given by the following formula:

$$sor^o_{t,O_{ij}} = MLP_{\theta_{\pi_o}}([h^t_{O_{ij}}, h^t_G]), O_{ij} \in \{1, 2, \ldots, \mathrm{O}\} \tag{20}$$

3.3. Machine encoder and decoder based on D5QN

Following SAC agent operation selection based on decoder scores, the subsequent step involves machine selection using the D5QN agent. Given multiple available machines for the same operation in the disjunctive graph, they compete to process unscheduled operations. To model their competitive relationships quantitatively, we define $P = \{M_k, M_q, \ldots, M_v\}$ as the set of currently available machines feature for a specific operation.

Here, we employ the Transformer architecture to model the competitive relationships among compatible machines in FJSP, reaping significant benefits. The Transformer architecture excels in sequence modeling, parallel processing, and generalization, making it a suitable choice for efficiently solving FJSP. Its self-attention mechanism effectively captures information among machine features, thus enhancing its applicability to the task.

The details are described in Fig. 5. Initially, we extract the information of the machines available for selection, denoted as matrix $H \in \mathbb{N}^{m\times d_{model}}$, where m is the number of candidate machines, and d_{model} is the dimension of machine features. We update H using a stack of N_{en} identical encoder layers, each consisting of two sublayers: a multi-head attention layer and a fully connected feedforward neural network (FNN). After each sublayer, a residual connection and layer normalization are applied. In the first layer, the query (Q), key (K), and value (V) matrices are derived from the input matrix H; in subsequent layer, they come from the output of the previous layer, enabling the model to capture and utilize information learned in earlier layers. The output of the final encoder layer is the encoding matrix Z, where each row z_i represents the encoded features of a machine. The attention mechanism utilizes an attention function to map the query matrix Q and the key–value pair matrices K and V to the output matrix. The attention function can be defined as follows:

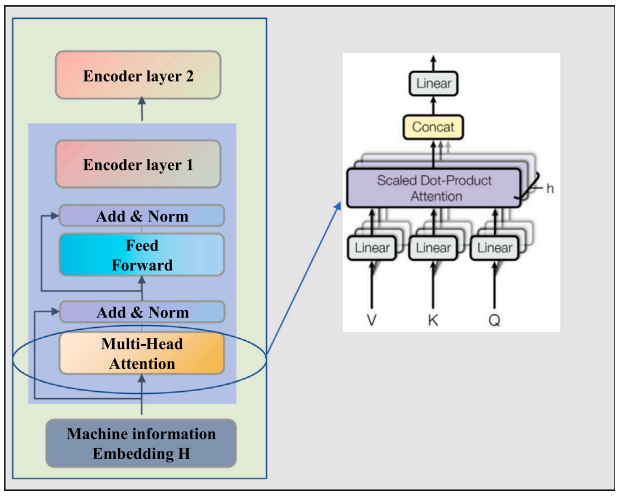


Fig. 5. Model competition machine information via a Transformer Encoder.

$$Attention(Q, K, V) = Softmax(\frac{(QK^T)}{\sqrt{d_k}})V \tag{21}$$

We use the multihead attention instead of the single-head one, W^Q_i, W^K_i, and W^V_i are learnable matrices. The multihead attention function can be defined as follows:

$$\begin{aligned} &\mathrm{MutiHead}(Q, K, V) = \left[\mathrm{head}_1, \ldots, \mathrm{head}_{\mathcal{M}}\right] W^O \\ &\text{where head}_i = \text{ Attention } \left(QW^Q_i, KW^K_i, VW^V_i\right) \end{aligned} \tag{22}$$

(2) Machine decoder

To enable the D5QN agent to take actions a^m_t based on the state s^m_t at discrete time step t, we employ a decoder to process the input from the encoder. The decoder is composed of a MLP and produces scores for machine selection using the following formula:

$$sor^m_{t,M_P} = MLP_{\theta_{\pi_m}}([Z^t_{M_{P_i}}, h^t_{O-M}]), P_i \in P \tag{23}$$

where h^t_{O-M} represents information related to the operation–machine pairs.

In order to ensure the legality of action selection by both agents, a similar approach as described in Ref. [50] is adopted. This involves setting the scores corresponding to already scheduled operations $sor^o_{t,O_{ij}}$ and operation–machine pairs that are infeasible due to machine unavailability sor^m_{t,M_P} to $-\infty$, thereby preventing the agents from selecting them. Subsequently, the Softmax function is applied to normalize $sor^o_{t,O_{ij}}$ and sor^m_{t,M_P}, resulting in the probability distributions for actions a^o_t and a^m_t, denoted a $p_i(a^o_t)$ and $p_k(a^m_t)$, respectively. Finally, the job agent selects the action with the highest probability based on a greedy decision, while the machine agent selects an action based on ϵ-greedy selection, as shown in Eq. (24).

$$a^m_t = \begin{cases} \arg\max\limits_{a^m_{t,k}} \left(p_k\left(a^m_t\right)\right), & \text{random } < \varepsilon \\ \text{random.choice } \left(a^m_{t,k}\right), & \text{random } > \varepsilon \end{cases} \tag{24}$$

3.4. Co-agent policy optimization

As shown in Fig. 4, to address the challenge of making multiple action decisions in FJSP, a collaborative reinforcement learning

Algorithm 1 Training CARL-FJSP Algorithm to dispatch

Input: Training episodes E_t, network update episodes E_s, batch size B;empty replay buffer D and fixed validation data ς_{val}; SAC algorithm: Training the actor network π_{θ_o} (parameter θ_o) and the critic network v_ϕ (parameter ϕ_1, ϕ_2). D5QN algorithm: Training the primary network Q_{θ_m} (parameter θ_m), target network $Q^-_{\theta^-_m}$(parameters $\theta^-_m = \theta_m$), execution network $\pi_{\theta^{old}_m}$, action selection probability parameter ε, Update network interval C.

Output: π_{θ_o} (parameter θ_o) , v_ϕ (parameter ϕ_1, ϕ_2), Q_{θ_m} (parameter θ_m)

1: Init θ_m 、 θ_o 、 ϕ_1 、 ϕ_2、 $\phi^{old}_o = \phi_o$ 、 $\theta^{old}_m = \theta_m$;
2: **for** $e = 1, 2, ..., E_t$ **do**
3: Draw B FJSP Instances from GenerateInstance
4: **for** $b = 1, 2, ..., B$ **do**
5: **for** $t = 1 \to T$ **do**
6: Based on $\pi_{\theta^{old}_o}$ using a greedy decision to execute $a^o_{b,t}$ under $s^o_{b,t}$ (c.f. Section 3.2) ;
7: Based on $\pi_{\theta^{old}_m}$ a probability ε to execute $a^m_{b,t}$ according to $s^m_{b,t}$ (c.f. Section 3.3) ;
8: Observe reward $r_{b,t}$ and next state $s_{b,t+1}$;
9: $y_{b,t} = \begin{cases} r_{b,t} + \gamma Q^-_{\theta^-_m}\left(s_{b,t+1}, \arg\max_{a'^m_{b,t}} Q_{\theta^{\text{old}}_m}\left(s_{b,t+1}, a'^m_{b,t}; \theta^{\text{old}}_m\right); \theta^-_m\right) \\ r_{b,t} \end{cases}$;
10: Store $(s_{b,t}, a_{b,t}, r_{b,t}, s_{b,t+1})$ in replay buffer D;
11: **end for**
12: **end for**
13: **for** $k = 1, 2, ..., E_s$ **do**
14: SAC cumulative lOSS: $J_Q(\phi)$, $J_\pi(\theta_i)$, $J(\alpha)$ (c.f. Section 2.6);
15: D5QN-loss:$\text{Loss}_{\text{D5QN}} = \left(y_b - Q_{\theta_m}\left(s^m_{b,t}, a^m_{b,t} \mid \theta_m\right)\right)$;
16: Every c steps reset $Q^-_{\theta^-_m} = Q_{\theta^{\text{old}}_m}$, soft update $\phi_o = 0.8 * \phi_o + 0.2 * \phi^{old}_o$;
17: Gradient descent updates all parameters;
18: **end for**
19: **if** Every Nr episodes **then**
20: Resample B instances to form the training data;
21: **end if**
22: **if** Every N_{val} episodes **then**
23: Validate π_θ on ς_{val};
24: **end if**
25: **end for**

scheduling optimization method is proposed. This method involves the design of two different agents for optimizing the selection of jobs and machines separately. The job agent and machine agent learn within the same environment. They interact with the environment based on their respective states, receive rewards, and then use their algorithms to calculate errors based on the rewards. These errors are used to update their network parameters through gradient descent, as outlined in Algorithm 1.

Algorithm 1, representing the CARL approach for addressing the FJSP, outlines a well-defined procedure. It begins with the random generation of B FJSP scheduling samples, taking into account workpiece processing, operations, and machines. These samples serve as the training dataset for two agents. Within the algorithm, lines 5–12 encapsulate the core scheduling process for a single FJSP sample. During this process, both agents collect experience tuples. Following the scheduling phase, the workpiece agent and machine agent independently compute errors, utilizing the SAC and D5QN algorithms (lines 13–18), and proceed to update their parameters through gradient descent. Subsequently, lines 19–21 involve periodic environmental resampling, performed at regular intervals based on a fixed distribution, for every Nr sets. Additionally, lines 21–24 encompass strategy validation on a fixed validation dataset ς_{val}, characterized by the same distribution as the training data. Upon the conclusion of model training, the final strategy is derived for the resolution of real-world FJSP instances.

4. Experimental settings

This section outlines the computational results of our study. We evaluated the performance of the proposed framework using both randomly generated instances and publicly available FJSP benchmarks.

4.1. Experimental settings

Dataset: We evaluated our framework using FJSP instances of different sizes. The training set consisted of 40,000 random FJSP instances, with validation and test sets comprising 100 instances each. Notably, training CARL model for large FJSP instances is time-consuming, and it is impractical to create a separate model for each instance size. Therefore, when designing the CARL algorithm, we give top priority to two fundamental aspects: the algorithm's robustness and its ability to efficiently generate high-quality solutions. Therefore, our research primarily focuses on training and validating the CARL model on medium-sized randomly generated instances, ranging in size from 10×5 to 20×5, to assess the method's performance and generalization capabilities. Furthermore, to further demonstrate the framework's robustness and scalability, we conducted direct testing on larger instances with dimensions of 20×10 and 30×20. In addition to direct testing on large instances, we also conducted experiments on recognized public FJSP benchmark datasets to explore their performance on cross-distribution tasks, including instances from Brandimart, Hurink and DPpaulli. The model is constructed by PyTorch 1.9, and the code is implemented using Python 3.9 on a Windows operating system. We

Table 1
The top 6 results in hyperparameter tuning.

CARL-FJSP		Hyperparameters	1	2	3	4	5	6
Agent 1	GAT	number of attention heads	4	4	6	2	6	6
		out feature dimension	8	8	16	8	32	16
		dropout rate	0.9	0.9	0.95	0.9	0.9	0.95
	SAC	learning rate	3e-4	3e-4	3e-3	3e-3	1e-3	3e-3
		tau	5e-3	5e-3	5e-3	5e-3	5e-3	5e-3
		decay ratio	0.9	0.95	0.95	0.95	0.95	1
Agent 2	Transformer Encoder	hidden size	32	32	64	64	128	128
		number of attention heads	4	4	8	2	4	8
		number of encoder layers	2	2	2	3	3	3
	D5QN	epsilon	0.8	0.8	0.85	0.8	0.8	0.8
		gamma	1	1	1	1	1	1
		learning rate	3e-4	3e-4	3e-3	3e-3	3e-4	3e-4
$Cmax$			334.6	336.8	338.4	346.7	361.6	367.0

Table 2
Heuristic scheduling rules.

Baseline rule	No.	Rule	Content
Operation selection rules	1	FIFO	select the earliest arriving job in the queue of a machine.
	2	MOPNR	select a job that has the greatest number of remaining operations to be done.
	3	LWKR	select a job that has the least total processing time remaining to be done
	4	SPT	select a job with the shortest processing time
	5	MWKR	select a job that has the most total processing time remaining to be done
Machine selection rules	1	SPT	select a machine with the shortest processing time for the operation of a job
	2	FIFO	select the earliest ready compatible machine.
	3	EET	select a machine that is idle at the earliest time

used the hardware Intel(R) Xeon(R) Platinum 8255C CPU, and a single Nvidia V100-SXM2-32 GB GPU.

Hyperparameters: In our study, we employed a random hyperparameter search approach for training the CARL model. The identification of suitable hyperparameters is a crucial step in optimizing DRL models. Given the extensive search space associated with hyperparameters, determining the optimal values can be a challenging task. Within the scope of our research, we utilized a random search methodology for hyperparameter optimization. Multiple rounds of hyperparameter tuning tests were conducted on the "mk09", and the performance of various hyperparameter configurations was assessed, with the outcomes of the initial six tests presented in Table 1. In subsequent experiments, we selected the hyperparameters from the first group due to their exceptional performance, as determined by our evaluation criteria.

Baseline Methods: To gauge the effectiveness of CARL strategies, we compared them with practical PDRs known for outstanding performance. In the context of FJSP, scheduling requires consideration of both job sequencing and machine allocation rules. We selected 5 prevalent operation selection rules and 3 widely adopted machine selection rules, as outlined in Table 2, resulting in a total of 15 distinct combinations. Following recommendations from Refs. [41,51], we fine-tuned our selections, ultimately settling on 10 sets of scheduling rule combinations. To ensure fair comparisons, we directly compared the learned strategies and manually crafted rules against baseline PDRs. Additionally, we benchmarked our results against Google OR-Tools, a widely-employed constraint programming solver known for its robust performance in solving industrial scheduling problems. To maintain fairness, a time limit of 1800 s was imposed on OR-Tools. The best or optimal solutions obtained from OR-Tools were used as a reference to evaluate the solution quality of both learned and manually crafted PDRs. To further assess the superiority of our algorithm compared to other algorithms on benchmark datasets, we compared the results

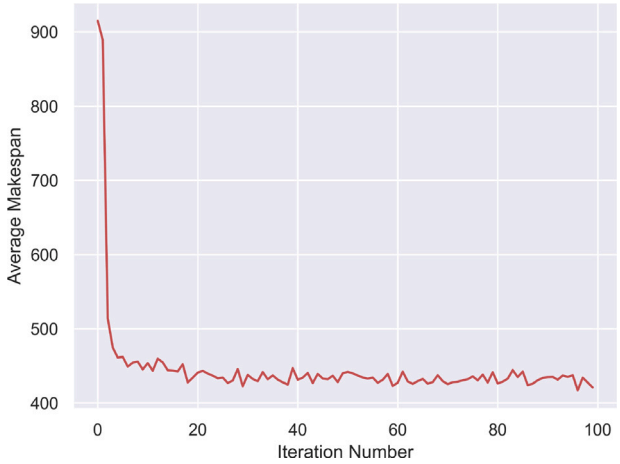


Fig. 6. Training curve on 15×10 instances.

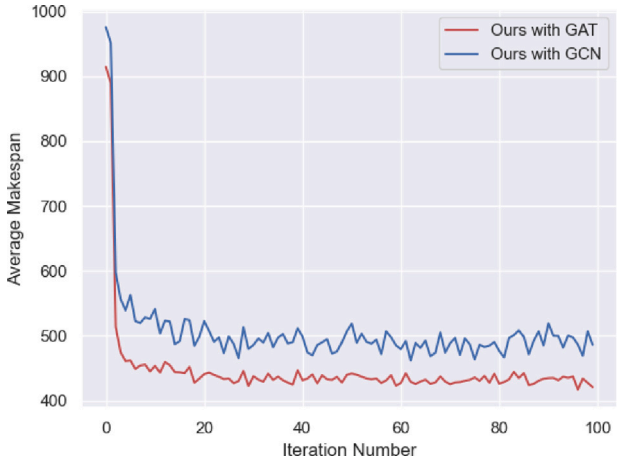


Fig. 7. Verification of the effectiveness of the GAT component.

Fig. 8. Verification of the effectiveness of the Transformer Encoder component.

generated by our uniquely trained CARL model with those produced by other heuristic, metaheuristic and RL [41] algorithms. In all subsequent tables, the bold font indicates the best performance on a particular dataset. RPD (Relative Percent Deviation) measures the relative percentage deviation from the optimal values for different algorithms. The calculation formula for RPD is provided in Eq. (25), where $C_{\max}$ represents the result of the respective algorithm, and $C_{\max}^{BS}$ is the minimum value across all algorithms.

$$RPD = (C_{\max}/C_{\max}^{BS} - 1) \times 100\% \tag{25}$$

4.2. Ablation study

(1) Influence of the GAT component: Firstly, the utilization of GNNs for feature extraction is imperative due to their remarkable adaptability to graphs of diverse scales and structures. GNNs efficiently capture and express intricate relationships within graph data, demonstrating versatility across graphs with varying sizes and structures. Secondly, to validate the efficacy of GAT component in feature extraction from disjunctive graph within the proposed methodology, we compare it with GCN for state extraction. The policy model is trained on 15 × 10 size. Both approaches employ Transformer Encoder technology for machine feature extraction and share identical reward function designs. As depicted in Fig. 7, the feature embedding facilitated by GAT in this study results in superior convergence of the policy model. This observation suggests that GAT enables the SAC agent to capture more effective state information, thereby enhancing its decision-making capabilities.

(2) Influence of the Transformer Encoder component: To validate the efficacy of the Transformer Encoder technique within our proposed methodology, we maintained uniformity in state feature representations and reward function definitions. We trained the policy model on 15 × 10 size, utilizing both the Transformer Encoder and an alternative model employing a FNN. As depicted in Fig. 8, our proposed approach exhibits enhanced convergence when leveraging the Transformer Encoder technology. This finding implies that the Transformer Encoder, through quantitative modeling of inter-machine competition, more effectively guides the agent in exploring the solution space.

4.3. Performance on synthetic instances

In order to expedite model training and enhance gradient direction optimization, we employed two pivotal techniques: batch training and learning rate decay. We grouped samples of identical size and systematically trained these groups in a specific order. We utilized various datasets with different dimensions, including 10 × 5, 10 × 10, 15 × 5, 15 × 10, and 20 × 5, to ensure consistent convergence of our model across diverse, independent datasets. Fig. 6 illustrates a training example with a problem size of 15 × 10. These curves depict the average completion times on 100 validation samples, offering a concise overview of the performance of our approach. Furthermore, we will evaluate the trained strategies, both in generating new samples and in their performance on larger-scale samples not encountered during the training process. Additionally, we will provide runtime analyses to assess the efficiency of our proposed method.

(1) Evaluation on Instances of Training Sizes: The results of the trained model's performance on 100 test instances generated with the same distribution are presented in Table 3. These results highlight the complexity of the FJSP. OR-Tools can provide optimal solutions within 1800 s for only 10 × 5 and 10 × 10 instances, and for 15 × 10 instances, it achieves optimal solutions in just 15% of cases. For larger instances, it can only provide approximate solutions. This necessitates the exploration of alternative approaches for broader scenarios. In comparison to the optimal baseline PDR method (MWKR + SPT) within the test set, our proposed approach exhibits an average performance improvement of 25.14% across all instance sizes. When contrasted with reinforcement learning models from the literature [41], with a model size of 20 × 10, our method surpasses the best model's performance by 67.39%. When compared to OR-Tools solutions, our method exhibits an average performance gap of 21.07%, approaching the performance of OR-Tools with minimal differences in solution quality.

(2) Generalization Performance on Large-Sized Instances: We evaluated the generalization of our trained CARL model to larger instances (20 × 10 and 30 × 20) to assess the performance of policies trained on smaller and medium-sized instances. The results are presented in Table 4, and a notable observation is the sensitivity of OR-Tools to instance size, as it can only provide approximate solutions for larger instance sizes. Compared to the optimal baseline PDR method (MWKR + SPT) within the test set, our approach exhibits an average performance improvement of 27.76% across all instance sizes. In contrast to reinforcement learning models with a size of 20 × 5, our method outperforms the optimal model, showcasing a performance improvement of 126.12%. These findings indicate that our model can effectively learn from smaller instances and demonstrates strong robustness in handling larger instances.

(3) Runtime Analysis: Runtime comparisons, as presented in Tables 3 and 4, demonstrate that scheduling rules achieve solution times within 1 s for various instance sizes, indicating limited sensitivity to instance size. In contrast, our approach exhibits solution times for FJSP instances ranging from 2 to 32 s. While it may not match the speed of PDR, it maintains competitiveness. This discrepancy can be attributed to the higher computational cost associated with neural network inference. However, when compared to OR-Tools, our

Table 3
Results of all methods on synthetic instances of training size.

Size		Ours	FIFO +FIFO	SPT +SPT	MOPNR +EET	MWKR +EET	LWKR +SPT	FIFO +SPT	FIFO + EET	MWKR + SPT	MOPNR +SPT	LWKR +EET	[41] 20×5	[41] 20×10	OR-Tools
10×5	Obj	**398.6**	567.85	515.9	558.36	548.54	562.85	479.54	571.01	484.12	484.16	795.5	572.01	552.72	326.24(100%)
	Gap	**22.2%**	76.10%	58.42%	72.83%	69.61%	72.53%	46.99%	75.03%	48.39%	48.41%	143.84%	75.33%	69.42%	-
	Time(s)	2.03	0.052	0.057	0.058	0.059	0.016	0.019	**0.014**	0.016	0.016	0.015	0.324	0.318	8.945
10×10	Obj	**403.53**	718.34	542.6	696.98	690.7	618.25	486.03	698.5	476.36	485.54	1205.99	675.5	692.44	329.31(100%)
	Gap	**22.5%**	120.63%	66.19%	113.99%	112.19%	87.74%	47.59%	112.11%	44.65%	47.44%	266.22%	105.13%	110.27%	-
	Time(s)	4.87	0.073	0.086	0.085	0.086	0.034	0.041	**0.031**	0.035	0.036	0.034	0.63	0.623	2.40
15×5	Obj	**540.02**	802.46	676.61	801.62	793.81	731.35	656.06	806.81	665.57	664.71	1056.3	798.24	789.93	462.27(27%)
	Gap	**16.82%**	74.41	46.62%	74.40%	72.61%	58.21%	41.92%	74.53%	43.98%	43.79%	128.50%	72.68%	70.88%	-
	Time(s)	3.36	0.053	0.056	0.056	0.057	0.028	0.033	**0.023**	0.028	0.029	0.026	0.498	0.514	1438.4
15×10	Obj	**491.32**	868.76	701.98	840.73	868.76	784.61	610.08	874.68	593.15	612.14	1513.02	841.55	832	377.13(15%)
	Gap	**30.28%**	131.69	86.47	124.29	121.20%	108.05%	61.77%	131.93%	57.28%	62.32%	301.19%	123.15%	120.61%	-
	Time(s)	6.62	0.112	0.116	0.121	0.123	0.061	0.073	**0.051**	0.059	0.063	0.057	1.008	1.008	1582.4
20×5	Obj	**679.25**	1044.95	836.1	1054	1023.98	895.44	810.23	1050.38	818.08	813.92	1299.5	1030.44	1025.5	598.15(0%)
	Gap	**13.56%**	75.74%	40.05%	77.26%	72.36%	49.70%	35.46%	75.60%	36.77%	36.07%	117.25%	72.27%	71.45%	-
	Time(s)	4.54	0.072	0.09	0.089	0.09	0.043	0.049	**0.034**	0.041	0.045	0.04	0.644	0.644	1800

(.%): The number of scheduling results that can be obtained within 1800s;
For OR-Tools, the solution and the ratio of optimally solved instances are reported.

Table 4
Results on the large-sized synthetic instances.

Size		Ours	FIFO +FIFO	SPT +SPT	MOPNR +EET	MWKR +EET	LWKR +SPT	FIFO +SPT	FIFO + EET	MWKR + SPT	MOPNR +SPT	LWKR +EET	[41] 20×5	[41] 20×10	OR-Tools
20×10	Obj	**579.63**	1089.61	825.4	1055.46	1041.89	925.18	725.69	1082.34	712.86	724.71	791.18	1044.53	1045.1	456.58(1%)
	Gap	**26.95%**	139.55%	80.83%	132.05%	129.09%	102.63%	58.94%	137.05%	56.13%	58.73%	73.28%	128.77%	128.90%	-
	Time(s)	9.42	0.161	0.176	0.181	0.178	0.087	0.107	0.075	0.09	0.097	**0.016**	1.412	1.369	1783.6
30×20	Obj	**674.53**	1671.18	1158.75	1586.53	1573.13	1202.55	856.6	1672.89	832.09	852.87	3525.68	1574.22	1620.11	598.15(0%)
	Gap	**12.77%**	180.31%	94.06%	166.05%	163.86%	101.04%	43.21%	179.68%	39.11%	42.58%	489.43%	163.18%	170.85%	-
	Time(s)	31.85	0.812	0.833	0.84	0.842	0.4	0.507	**0.348**	0.415	0.457	0.398	4.076	4.206	1802

(.%): The number of scheduling results that can be obtained within 1800s;
For OR-Tools, the solution and the ratio of optimally solved instances are reported.

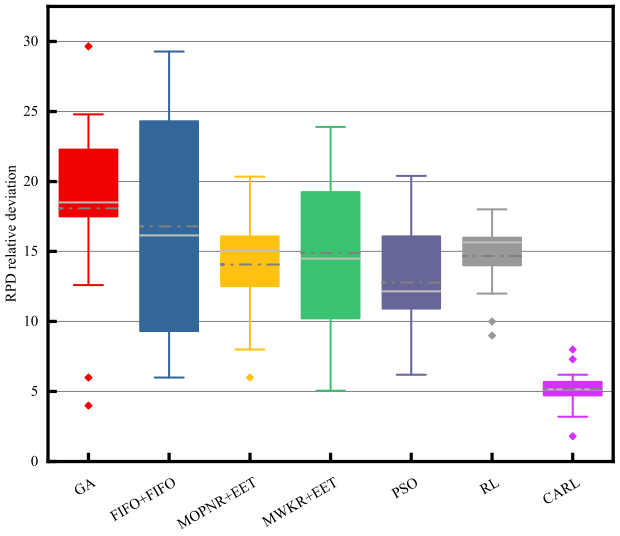


Fig. 9. RPD comparison box chart of different algorithms on the mk dataset. (the statistics are computed from 100 simulations).

method exhibits a significant advantage, as their runtimes approach our specified 1800-second limit for medium and large instances.

4.4. Performance on public benchmarks

In practical model deployment, data distributions often differ significantly from the training data, necessitating robustness in the model. Consequently, this subsection delves deeper into evaluating the performance of the model, which was trained on synthetic data, across three distinct categories of public benchmarks. These benchmarks exhibit distributions that are entirely disparate from the training instances. Each benchmark test set comprises instances of varying problem sizes. For instance, within the DPpaulli's benchmark, the number of jobs and machines varies within the ranges of 10 to 20 and 5 to 10, respectively,

Table 5
Result comparison on Brandimarte's instances between CARL algorithm and Other algorithms.

Instance(size)		Ours	FIFO +FIFO	SPT +SPT	MOPNR +EET	MWKR +EET	LWKR +SPT	FIFO +SPT	FIFO + EET	MWKR + SPT	MOPNR +SPT	LWKR +EET	[41] 20×5	[41] 20×10	GA	PSO	UB
MK01(10×6)	Obj	44	48	59	49	46	76	64	47	53	61	78	55	48	**42**	46	39
	RPD	12.8%	23.1%	51.3%	25.6%	17.9%	94.9%	64.1%	20.5%	35.9%	56.4%	100.0%	41%	23.1%	**7.7%**	17.9%	
MK02(10×6)	Obj	**31**	46	46	40	41	43	42	40	37	41	59	42	43	41	35	26
	RPD	**19.2%**	76.9%	76.9%	53.8%	57.7%	65.4%	61.5%	53.8%	42.3%	57.7%	126.9%	61.5%	65.4%	57.5%	34.6%	
MK03(15×8)	Obj	**207**	220	303	216	213	362	338	207	342	330	325	232	213	238	212	204*
	RPD	**1.5%**	7.8%	48.5%	5.9%	4.4%	77.5%	65.7%	1.5%	67.6%	61.8%	59.3%	13.7%	4.4%	16.7%	3.9%	
MK04(15×8)	Obj	**69**	78	76	80	71	181	169	76	167	174	108	82	75	74	71	60
	RPD	**15.0%**	30.0%	26.7%	33.3%	18.3%	201.7%	181.7%	26.7%	178.3%	190.0%	80.0%	36.7%	25.0%	23.3%	18.3%	
MK05(15×4)	Obj	**177**	186	226	191	186	279	251	194	280	269	250	205	185	188	185	172
	RPD	**2.9%**	8.1%	31.4%	11.0%	8.1%	62.2%	45.9%	12.8%	62.8%	56.4%	45.3%	19.2%	7.6%	9.3%	7.6%	
MK06(10×10)	Obj	**77**	99	89	102	108	123	108	99	104	104	185	110	103	115	98	58
	RPD	**32.8%**	70.7%	53.4%	75.9%	86.2%	112.1%	86.2%	70.7%	79.3%	79.3%	219.0%	89.7%	77.6%	98.3%	69.0%	
MK07(20×5)	Obj	**151**	214	214	219	212	230	232	212	217	217	273	215	214	183	176	139
	RPD	**8.6%**	54.0%	54.0%	57.6%	52.5%	65.5%	66.9%	52.5%	56.1%	56.1%	96.4%	54.7%	54.0%	31.7%	26.6%	
MK08(20×10)	Obj	531	531	664	**523**	535	651	587	531	587	592	736	525	523	523	557	523*
	RPD	1.5%	1.5%	27.0%	**0.0%**	2.3%	24.5%	12.2%	1.5%	12.2%	13.2%	40.7%	0.4%	0.0%	0.0%	6.5%	
MK09(20×10)	Obj	334	372	448	342	355	511	466	349	456	462	616	424	**333**	361	345	307
	RPD	8.8%	21.2%	45.9%	11.4%	15.6%	66.4%	51.8%	13.7%	48.5%	50.5%	100.7%	38.1%	**8.5%**	17.6%	12.4%	
MK10(20×15)	Obj	**245**	278	358	268	264	427	365	279	369	358	492	266	266	319	247	197
	RPD	**24.4%**	41.4%	81.7%	36.0%	34.0%	116.8%	85.3%	41.6%	87.3%	81.7%	149.7%	35.0%	35.0%	61.9%	25.4%	
Average	Obj	**186**	207	248	203	203	288	262	203	261	261	312	216	200	208	197	173
	RPD	**5.7%**	19.7%	43.4%	17.3%	17.3%	66.5%	51.4%	17.3%	50.9%	50.9%	80.3%	24.9%	15.6%	20.2%	13.9%	

UB column that is the best result from literature, and '*' denotes the result is optimal

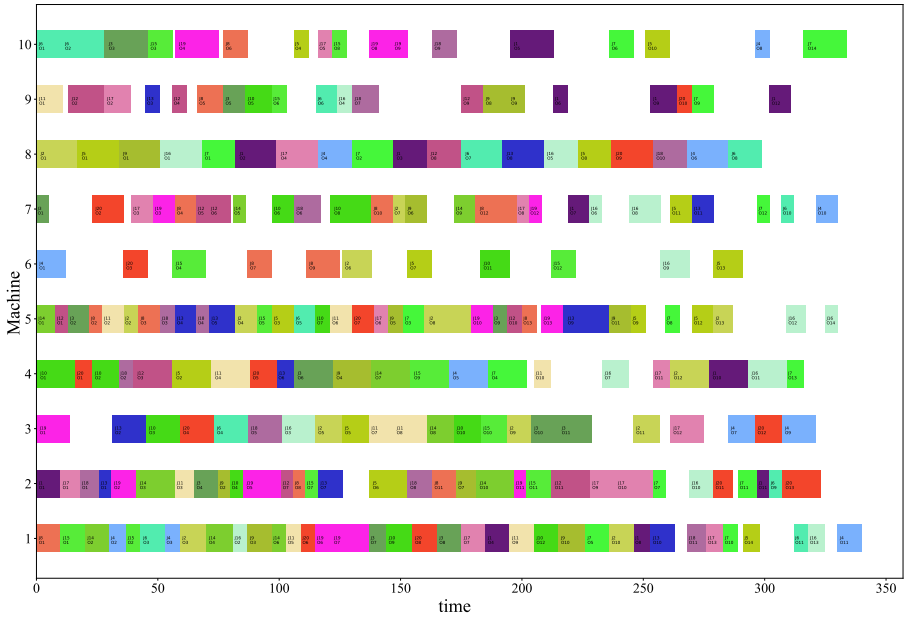


Fig. 10. The Gantt charts of our method on the mk09 dataset.

with a notably higher number of operations compared to machines. We opt to utilize the singularly trained CARL model for testing.

(1) Results on Brandimarte's benchmark:

The results presented in Table 5 encompass a comparative analysis of the performance across ten rule-based methods, two meta-heuristic algorithms (GA [52] and PSO [53]), a state-of-the-art reinforcement learning model [41], and our model. It is noteworthy that GA exhibits a slight superiority over our model in only two instances, namely, mk01 and mk08. This observation suggests the latent potential of meta-heuristic algorithms in achieving solutions of high quality. Against rule-based methods, our model outperforms results for all 9 FJSP instances, with MOPNR + EET showing a 1.5% improvement over our model specifically in the mk09 case. Compared to reinforcement learning models, our model surpasses 9 out of 10 FJSP instances, showcasing an overall average performance improvement of 9.9% compared to the 20 × 10 model. On average, our model exhibits only a marginal deviation of 5.7% from the optimal results reported in the literature.

To comprehensively evaluate the robustness of our model, we conducted 100 experiments on this dataset, systematically collecting and aggregating the results. We compared the performance of our model with three scheduling rules, meta-heuristic algorithms, and reinforcement learning models, and the comparative results are illustrated in

Table 6
Result comparison on Hurink's vdata instances between CARL algorithm and Other algorithms.

Instances(size)		Ours	FIFO +FIFO	SPT +SPT	MOPNR +EET	MWKR +EET	LWKR +SPT	FIFO +SPT	FIFO + EET	MWKR + SPT	MOPNR +SPT	LWKR +EET	[41] 20×5	[41] 20×10	UB
la1-5(10×5)	Obj	**544.4**	576.1	631.9	559.6	555.4	803.0	790.0	613.4	788.0	602.0	749.6	580.4	550.4	507*
	RPD	**7.4%**	13.6%	24.6%	10.4%	9.5%	58.4%	55.8%	21.0%	55.4%	18.7%	47.9%	14.5%	8.6%	
la6-10(15×5)	Obj	822.2	850.8	973.0	838.4	823.3	1263.	1146.0	842.8	1196.0	861.8	1061.4	845.6	**821.8**	794*
	RPD	3.6%	7.2%	22.5%	5.6%	3.7%	659.1%	44.3%	6.1%	50.6%	8.5%	33.7%	6.5%	**3.5%**	
la11-15(20×5)	Obj	**1062.8**	1089.6	1181.1	1080.6	1063.5	1464.6	1283.4	1090	1398.6	1094.8	1355.6	1072.8	1064	1040.8*
	RPD	**2.1%**	4.7%	13.5%	3.8%	2.2%	40.7%	23.3%	4.7%	34.4%	5.2%	30.2%	3.1%	2.2%	
la16-20(10×10)	Obj	**695**	716.6	728.7	719.1	714.0	1339.2	1002.6	716.4	1360	737.6	1138.0	715.6	703.2	679.8*
	RPD	**2.2%**	5.4%	7.2%	5.8%	5.0%	97.0%	47.5%	5.4%	100.1%	8.5%	67.4%	5.3%	3.4%	
la21-25(15×10)	Obj	858.4	874.1	981.3	847.9	866.9	1607.8	1232.6	894.4	1702.8	936.8	1277.2	848.2	**825.2**	777.2
	RPD	10.4%	12.5%	26.3%	9.1%	11.5%	106.9%	58.6%	15.1%	119.1%	20.5%	64.3%	9.1%	**6.2%**	
la26-30(20×10)	Obj	1100	1141.4	1376.7	1101.2	1119.8	2169.4	1603.6	1119.8	2136.6	1175.	1538.8	1107.8	**1085.4**	1054.2
	RPD	4.3%	8.3%	30.6%	4.5%	6.2%	105.8%	52.1%	6.2%	102.7%	411.5%	46.0%	5.1%	**3.0%**	
la31-35(30×10)	Obj	**1580.6**	1619.4	1786.7	1595.0	1589.5	2598.0	2136.0	1611.2	2639.6	1657.2	2322.4	1588.2	1614.0	1552.2*
	RPD	**1.8%**	4.3%	15.1%	2.8%	2.4%	67.4%	37.6%	3.8%	70.1%	6.8%	49.6%	2.3%	4.0%	
la36-40(15×15)	Obj	**970.8**	995.0	1003.7	986.4	986.8	1991.4	1589.6	1002.4	2039.2	1087.8	1601.2	983.6	983.4	950.8*
	RPD	**2.1%**	4.6%	5.6%	3.7%	3.8%	109.4%	67.2%	5.4%	114.5%	14.4%	68.4%	3.4%	3.4%	
Average	Obj	**954**	980.0	1083.0	966.0	961.8	1655	1348	986.0	1658.0	1019.0	1381.0	968	959	920
	RPD	**3.7%**	7.6%	17.8%	5.1%	4.6%	79.9%	46.5%	7.3%	80.3%	10.8%	50.1%	5.2%	3.8%	

UB column that is the best result from literature, and '*' denotes the result is optimal

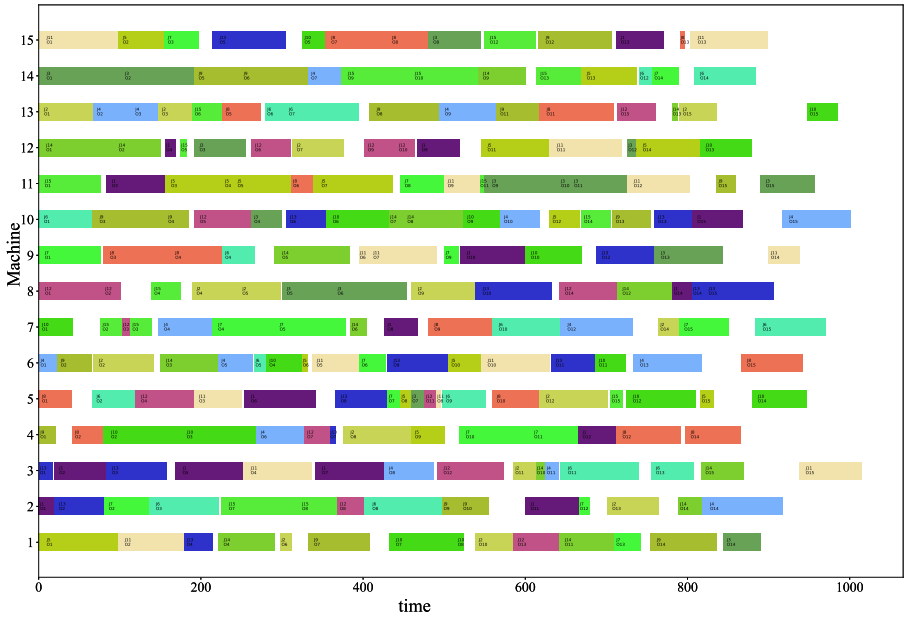


Fig. 11. The Gantt charts of our method on the vdata40 dataset.

Fig. 9. As depicted in Fig. 9, the outcomes consistently demonstrate that across various *RPD*, our trained CARL model outperforms the best scheduling rules by a substantial margin, exceeding 11.9%. When compared to the top-performing meta-heuristic algorithm (PSO), the difference is approximately 8.5%, and compared to the reinforcement learning model with a size of 20 × 10, the difference is about 17.4%. These findings underscore the robustness of our approach. Additionally, Fig. 10 showcases the scheduling Gantt chart generated by our approach for the mk09 instance.

These findings underscore the remarkable performance of our model in these instances, signifying its capability to furnish satisfactory solutions for combinatorial optimization problems. Furthermore, our model effectively strikes a balance between computational efficiency and robust generalization.

(2) Results on Hurink's benchmark:

Hurink's benchmark is a collection of datasets created based on real-world production scheduling problems. Its notable feature lies in its diversity of problem instances, encompassing various scales and complexities, from small to large instances, aimed at evaluating the scalability and applicability of algorithms. These datasets typically come with known optimal solutions, facilitating the assessment of algorithm performance. In this study, we selected datasets of two complexity classes, vdata and rdata, for testing. Instances of the same size were organized into groups, with each dataset divided into eight groups, each containing five instances. The experimental results are presented in Tables 6 and 7. Such a configuration contributes to a comprehensive evaluation of algorithm performance across different problem instances.

Table 7
Result comparison on Hurink's rdata instances between CARL algorithm and Other algorithms.

Instances(size)		Ours	FIFO +FIFO	SPT +SPT	MOPNR +EET	MWKR +EET	LWKR +SPT	FIFO +SPT	FIFO + EET	MWKR + SPT	MOPNR +SPT	LWKR +EET	[41] 20×5	[41] 20×10	UB
la1-5(10×5)	Obj	**549.8**	585.6	676.8	576	594.8	893.6	677	598.4	784.6	804.4	781.4	617.2	574.8	507.6
	RPD	**8.30%**	15.40%	33.30%	13.50%	17.20%	76.00%	33.40%	17.90%	54.60%	58.50%	53.90%	21.60%	13.20%	
la6-10(15×5)	Obj	**838.2**	865.6	1012	855.7	841.2	1256.4	1002.8	856.8	1090.4	1036.4	1129.8	878.8	839.6	794.2
	RPD	**5.50%**	9.00%	27.40%	7.70%	5.90%	58.20%	26.30%	7.90%	37.30%	30.50%	42.30%	10.70%	5.70%	
la11-15(20×5)	Obj	**1093**	1137.6	1223.7	1126.2	1095	1500.6	1327.4	1147	1354.4	1343.4	1416.4	1121.2	1105.4	1041
	RPD	**5.00%**	9.30%	17.60%	8.20%	5.20%	44.10%	27.50%	10.20%	30.10%	29.00%	36.10%	7.70%	6.20%	
la16-20(10×10)	Obj	**823**	864.1	887.8	860	842.4	1294.6	1063.6	830.2	1030.4	1042.8	1303.2	875.8	884.4	697
	RPD	**18.10%**	24.00%	27.40%	23.40%	20.90%	85.70%	52.60%	19.10%	47.80%	49.60%	87.00%	25.70%	26.90%	
la21-25(15×10)	Obj	958.2	1037.6	1140.9	1003.8	987.3	1816.4	1334.4	1043	1261.4	1398.2	1654.4	1004.6	**947.6**	807.2
	RPD	18.70%	28.50%	41.30%	24.40%	22.30%	125.00%	65.30%	29.30%	56.30%	73.20%	105%	24.50%	**17.40%**	
la26-30(20×10)	Obj	1182.4	1252	1441	1224.6	1191	2095.8	1772	1236	1585.4	1646.4	2068	1247.6	**1175.6**	1062
	RPD	11.30%	17.90%	35.70%	15.30%	12.10%	97.30%	66.90%	16.40%	49.30%	55.00%	94.70%	17.50%	**10.70%**	
la31-35(30×10)	Obj	1651	1692.2	1910	1656.3	1642	2650.6	2050.2	1683.4	2217	2111.8	2578	1637.4	**1612.8**	1553
	RPD	6.30%	9.00%	23.00%	6.70%	5.70%	70.70%	32.00%	8.40%	42.80%	36.00%	66.00%	5.40%	**3.90%**	
la36-40(15×15)	Obj	**1198.8**	1262	1311	1231	1231	2095.6	1612.8	1273	1579	1603.4	2132.4	1259.8	1212	1013
	RPD	**18.30%**	24.60%	29.40%	21.50%	21.50%	106.90%	59.20%	21.50%	55.90%	58.30%	111%	24.40%	19.60%	
Average	Obj	**1037**	1087	1200	1043	1053	1700	1355	1083	1363	1373	1633	1080	1044	934
	RPD	**11.00%**	16.30%	28.40%	11.70%	12.70%	82.00%	45.10%	15.90%	45.90%	47.00%	74.80%	15.60%	11.80%	

UB column that is the best result from literature

Table 6 provides a comprehensive evaluation of our model's performance on the vdata dataset, comparing it with the aforementioned 10 commonly used scheduling rules and a state-of-the-art reinforcement learning model. Within the vdata dataset, our model's performance falls slightly short of the optimal scheduling rule (MOPNR + EET) by a mere 1.3% in the 20 × 10 scheduling instances. However, for all other instance sizes, our model consistently outperforms these scheduling rules, demonstrating superior performance. Compared to the reinforcement learning model, our approach generally outperforms the 20 × 5 model across all instance sizes, with only a slight decrease of 1.3% in the average performance for the 15 × 10 instances. This highlights the robustness of our model. Specifically, in the context of 40 instances, our model exhibits only a minor performance gap of 0.9% compared to the optimal scheduling rule, whereas the gap widens considerably to 76.6% when contrasted with the least effective scheduling rule. These results underscore the substantial impact of different combinations of scheduling rules on the outcomes. Additionally, when compared to the optimal reinforcement learning model with a size of 20 × 10, our model exhibits a slight improvement in average performance. Furthermore, our model's performance remains highly competitive, with a modest performance disparity of within 3.7% when compared to the optimal results reported in the literature. These findings emphasize the remarkable performance of our model in handling diverse scheduling instances. Additionally, Fig. 11 illustrates the scheduling Gantt chart generated by our method for the vdata40 instances.

Table 7 provides a comprehensive evaluation of the performance of our model and other algorithms on the rdata dataset. In 30 × 10 scheduling instances of the rdata dataset, our model demonstrates slightly inferior performance compared to the optimal scheduling rule (MWKR + EET), with a marginal gap of 0.6%. However, for other instance sizes, our model excels, displaying significant performance advantages with performance gaps ranging from 0.7% to 63.8% when compared to scheduling rules. Furthermore, compared to reinforcement learning models with two different sizes, our model exhibits an average performance improvement of 2.7%. On average, our model demonstrates a performance gap of approximately 11.0% when contrasted with optimal solutions reported in the literature. These findings underscore the strong generalization ability of our approach across diverse scheduling instances, emphasizing its competitiveness compared to traditional scheduling rules and state-of-the-art solutions.

(3) Results on DPpaulli's benchmark:

DPpaulli's benchmark serves as a valuable dataset for assessing the efficacy of discrete manufacturing scheduling algorithms. Its distinguishing characteristics are rooted in the emulation of real-world scenarios, encompassing a wide array of problem instances with diverse scales and complexities. A noteworthy departure from our training data lies in the dataset's deliberate inclusion of instances where the number of operations significantly exceeds the number of machines. In contrast, our training data predominantly comprises instances where the number of operations is equivalent to or less than the number of machines. As a consequence, subjecting our model to testing on this dataset effectively underscores its robustness and adaptability, thereby accentuating its resilience in addressing challenging and unconventional scenarios.

We compared the results of our model with those of 10 commonly used scheduling rules, as shown in Table 8. Our model outperforms the results of most scheduling rules. Specifically, our model outperforms the best scheduling rules in 11 out of 18 instances. This dataset reveals substantial variations in the impact of scheduling rules, underscoring the challenges posed by these instances. In contrast, our model demonstrates robustness, maintaining competitive performance across diverse instances. Furthermore, we present the scheduling Gantt chart for the DPpaullii1a instance in Fig. 12 to offer additional insights into our model's performance and visualize the scheduling outcomes.

5. Conclusions and future work

This study introduces a CARL model tailored to address the FJSP, encompassing both operation selection and machine allocation. Our model is trained only once on various small and medium-sized datasets, showing great potential in generating datasets and in real-life production scenarios. The results of our experiments indicate that, when compared to the best scheduling rules, our model achieves a remarkable performance improvement of approximately 55.52% on larger problem instances without specific training. Across various benchmark datasets with distinct characteristics, our model consistently achieves satisfactory results, confirming its outstanding generalization capability and robustness. This achievement is attributed to our pioneering approach of employing cooperative DRL to address the FJSP challenge. The CARL model represents an integration of cutting-edge technologies. First, our model leverages graph structures to model the FJSP problem and

Table 8
Result comparison on DPpaulli's instances between CARL algorithm and dispatching rules.

Instances Size		Ours	FIFO +FIFO	SPT +SPT	MOPNR +EET	MWKR +EET	LWKR +SPT	FIFO +SPT	FIFO + EET	MWKR + SPT	MOPNR +SPT	LWKR +EET
10×5	DPpaullii1a	2980	3113	**2852**	3078	3179	3983	3123	3189	3313	3263	3718
	DPpaullii2a	**2483**	2526	2772	2973	2651	4388	3254	2839	3039	3236	4244
	DPpaullii3a	**2366**	2398	3128	2580	2432	4396	3016	3344	3105	3164	3760
	DPpaullii4a	3055	3000	**2889**	3018	3106	3917	3126	3102	3136	3206	3878
	DPpaullii5a	2422	**2419**	2485	2769	2602	3895	3020	3146	2977	2959	4141
	DPpaullii6a	**2330**	2502	3054	2527	2446	4233	3139	2993	3109	3157	3792
15×8	DPpaullii7a	2854	**2768**	2974	3052	3047	4073	3200	3087	3121	3149	3812
	DPpaullii8a	2327	**2319**	2632	2454	2452	3771	2943	3312	2933	3207	3999
	DPpaullii9a	**2171**	2240	2521	2368	2257	4321	3183	3087	3011	3263	3720
	DPpaullii10a	2819	2840	**2808**	3000	3018	4015	3110	3312	3202	3098	4145
	DPpaullii11a	2352	**2260**	2596	2544	2455	3734	3032	3104	3291	3038	3682
	DPpaullii12a	**2147**	2236	2255	2361	2233	3872	2720	2957	2855	2796	4016
20×10	DPpaullii13a	**2836**	2924	3487	2967	2838	4368	3095	3168	3054	3193	4482
	DPpaullii14a	**2351**	2374	2716	2578	2441	3930	3550	3315	3220	3165	4065
	DPpaullii15a	**2301**	2400	2505	2435	2306	4259	3339	3298	3181	3249	4482
	DPpaullii16a	**2815**	3050	2989	2843	2852	3899	3202	3178	3012	3078	3731
	DPpaullii17a	**2392**	2467	2441	2479	2416	4167	3200	3252	3301	3092	4209
	DPpaullii18a	**2242**	2328	2677	2422	2290	4025	2897	2976	2891	2978	4826

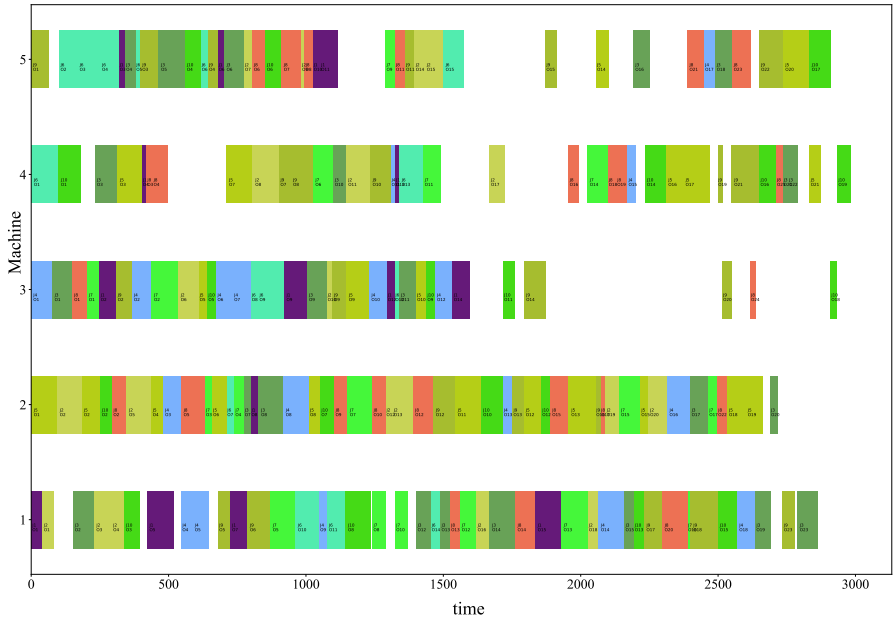


Fig. 12. The Gantt charts of our method on the DPpaullii1a dataset.

employs GATs for feature extraction. It also incorporates the SAC agent from Maximum Entropy Reinforcement Learning for machine selection. Second, our model utilizes self-attention mechanisms to precisely model the competition among various machines and employs parallel encoding to enhance computational speed. Finally, we introduce ensemble reinforcement learning, leveraging the D5QN agent for workpiece selection. Through the synergistic integration of these key techniques, we effectively overcome the complexity inherent in the FJSP, including workpiece selection and machine allocation. This leads to a significant reduction in problem-solving time, accompanied by the achievement of exceptional scheduling performance.

A promising direction for future exploration lies in the investigation of innovative disjunctive graph embedding strategies and advanced machine information modeling techniques. Furthermore, the application of diverse reinforcement learning agents, including SARSA and DQN for value-based approaches, alongside policy-based methodologies like PPO and A2C, could contribute valuable insights into the selection of job and machine.

CRediT authorship contribution statement

Wenquan Zhang: Conceptualization, Methodology, Writing – original draft. **Yong Li:** Data curation, Writing – original draft. **Chao Du:** Writing – review & editing. **Xiaobing Feng:** Visualization. **Xuesong Mei:** Funding acquisition.

Declaration of competing interest

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

Acknowledgments

This research was supported by the National Key R&D Program of China (Grant no. 2018AAA0101802), and the Science and Technology Major Project of Shaanxi Province, China under Grant number 2019ZDLGY01-05HZ.

Appendix

Raw features of operation nodes :

(1) The total number of jobs (n);

(2) Scheduling tag: This binary value (0 or 1) marks whether the operation is scheduled (1) or unscheduled (0);

(3) Estimated lower bound of completion time $C(O_{ij})$: Estimates the earliest possible completion time for the operation O_{ij};

(4) Minimum processing time: Represents the shortest processing time among all machines for the operation O_{ij};

(5) Average processing time $\bar{t}_{i,j}$: Signifies the mean processing time across all machines for the operation O_{ij};

(6) Remaining workload: Represents the cumulative average processing time of unscheduled operations of job J_i;

(7) $OP_i(t)$: The current number of completed operations for job J_i;

(8) $CJ_i(t)$: The completion rate of job at scheduling time t: $CJ_i(t) = \frac{OP_i(t)}{n_p}$;

(9) The average completion rate ($CO_{ave}(t)$) of the operation at time t: $CO_{ave}(t) = \frac{\sum_{i=1}^{n} OP_i(t)}{\sum_{i=1}^{n} n_p}$;

(10) The average completion rate ($CJ_{ave}(t)$) of the job at time t: $CJ_{ave}(t) = \frac{\sum_{i=1}^{n} CJ_i(t)}{n}$;

Raw features of machine nodes:

(1) The total number of machines (m);

(2) Working tag: the value is 0 if M_k is free otherwise 1;

(3) Remaining processing time: the time from T_s until the free time. (0 for free M_k);

(4) Completion time: $T_t(M_k)$ is the completion time for machine M_k before timestep t;

(5) Number of candidates that M_k can process;

(6) $CT_k(t)$: The completion time of the last operation allocated on machine M_k at scheduling time t;

(7) The machine utilization rate of machine k at scheduling time t: $U_k(t) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{OP_i(t)} X_{i,j,k} \cdot t_{i,j,k}}{CT_k(t)}$, where $X_{i,j,k}$ indicates that it equals 1 when the jth operation of job i is assigned to machine k; otherwise, it equals 0; $t_{i,j,k}$ is the processing time;

(8) The average utilization rate ($U_{ave}(t)$) of the machine: $U_{ave}(t) = \frac{\sum_{k=1}^{m} U_k(t)}{m}$;

Features of compatible operation–machine pairs:

(1) Processing time $t_{i,j,k}$;

(2) Ratio of $t_{i,j,k}$ to remaining workload of J_i;

References

[1] Liu Y, Wang L, Wang XV, Xu X, Zhang L. Scheduling in cloud manufacturing: state-of-the-art and research challenges. Int J Prod Res 2019;57(15–16):4854–79.
[2] Mao S, Wang B, Tang Y, Qian F. Opportunities and challenges of artificial intelligence for green manufacturing in the process industry. Engineering 2019;5(6):995–1002.
[3] Hansmann RS, Rieger T, Zimmermann UT. Flexible job shop scheduling with blockages. Math Methods Oper Res 2014;79:135–61.
[4] Kress D, Müller D, Nossack J. A worker constrained flexible job shop scheduling problem with sequence-dependent setup times. OR Spectr 2019;41:179–217.
[5] Özgüven C, Özbakır L, Yavuz Y. Mathematical models for job-shop scheduling problems with routing and process plan flexibility. Appl Math Model 2010;34(6):1539–48.
[6] Özgüven C, Yavuz Y, Özbakır L. Mixed integer goal programming models for the flexible job-shop scheduling problems with separable and non-separable sequence dependent setup times. Appl Math Model 2012;36(2):846–58.
[7] Jin L, Tang Q, Zhang C, Shao X, Tian G. More MILP models for integrated process planning and scheduling. Int J Prod Res 2016;54(14):4387–402.
[8] Wang C-N, Cheng Z-H, Phuc PNK, Nguyen VT. Scheduling optimization modelling: A case study of a woven label manufacturing company. Comput Syst Sci Eng 2021;38(2):239–49.
[9] Homayouni SM, Fontes DB. Production and transport scheduling in flexible job shop manufacturing systems. J Global Optim 2021;79(2):463–502.
[10] El Khayat G, Langevin A, Riopel D. Integrated production and material handling scheduling using mathematical programming and constraint programming. European J Oper Res 2006;175(3):1818–32.
[11] Zhang S, Wang S. Flexible assembly job-shop scheduling with sequence-dependent setup times and part sharing in a dynamic environment: Constraint programming model, mixed-integer programming model, and dispatching rules. IEEE Trans Eng Manage 2018;65(3):487–504.
[12] Novas JM, Henning GP. Integrated scheduling of resource-constrained flexible manufacturing systems using constraint programming. Expert Syst Appl 2014;41(5):2286–99.
[13] Demir Y, İşleyen SK. Evaluation of mathematical models for flexible job-shop scheduling problems. Appl Math Model 2013;37(3):977–88.
[14] Doh H-H, Yu J-M, Kim J-S, Lee D-H, Nam S-H. A priority scheduling approach for flexible job shops with multiple process plans. Int J Prod Res 2013;51(12):3748–64.
[15] Saqlain M, Ali S, Lee J. A Monte-Carlo tree search algorithm for the flexible job-shop scheduling in manufacturing systems. Flex Serv Manuf J 2023;35(2):548–71.
[16] Chaudhry I, Rafique A, Elbadawi I, Aichouni M, Usman M, Boujelbene M, Boudjemline A. Integrated scheduling of machines and automated guided vehicles (AGVs) in flexible job shop environment using genetic algorithms. Int J Ind Eng Comput 2022;13(3):343–62.
[17] Gocken T, Dosdogru A, Boru A, Gocken M. Integrating process plan and part routing using optimization via simulation approach. 2019.
[18] Huang X, Zhang X, Islam SM, Vega-Mejía CA. An enhanced Genetic Algorithm with an innovative encoding strategy for flexible job-shop scheduling with operation and processing flexibility. J Ind Manag Optim 2020;16(6):2943–69.
[19] Amiri F, Shirazi B, Tajdin A. Multi-objective simulation optimization for uncertain resource assignment and job sequence in automated flexible job shop. Appl Soft Comput 2019;75:190–202.
[20] Mohammadi S, Al-e Hashem SM, Rekik Y. An integrated production scheduling and delivery route planning with multi-purpose machines: A case study from a furniture manufacturing company. Int J Prod Econ 2020;219:347–59.
[21] Poppenborg J, Knust S, Hertzberg J. Online scheduling of flexible job-shops with blocking and transportation. Eur J Ind Eng 2012;6(4):497–518.
[22] Zhang S, Wong T. Studying the impact of sequence-dependent set-up times in integrated process planning and scheduling with E-ACO heuristic. Int J Prod Res 2016;54(16):4815–38.
[23] El Khoukhi F, Boukachour J, Alaoui AEH. The "Dual-Ants Colony": A novel hybrid approach for the flexible job shop scheduling problem with preventive maintenance. Comput Ind Eng 2017;106:236–55.
[24] Karimi S, Ardalan Z, Naderi B, Mohammadi M. Scheduling flexible job-shops with transportation times: Mathematical models and a hybrid imperialist competitive algorithm. Appl Math Model 2017;41:667–82.
[25] Wu X, Liu X, Zhao N. An improved differential evolution algorithm for solving a distributed assembly flexible job shop scheduling problem. Memet Comput 2019;11:335–55.
[26] Xie Z, Yang D, Ma M, Yu X. An improved artificial bee colony algorithm for the flexible integrated scheduling problem using networked devices collaboration. Int J Coop Inf Syst 2020;29(01n02):2040003.
[27] Cao Y, Shi H. An adaptive multi-strategy artificial bee colony algorithm for integrated process planning and scheduling. IEEE Access 2021;9:65622–37.
[28] Wang H, Sheng B, Lu Q, Yin X, Zhao F, Lu X, Luo R, Fu G. A novel multi-objective optimization algorithm for the integrated scheduling of flexible job shops considering preventive maintenance activities and transportation processes. Soft Comput 2021;25:2863–89.
[29] Zhu Z, Zhou X. An efficient evolutionary grey wolf optimizer for multi-objective flexible job shop scheduling problem with hierarchical job precedence constraints. Comput Ind Eng 2020;140:106280.
[30] Luo S. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. Appl Soft Comput 2020;91:106208.
[31] Wang L, Pan Z, Wang J. A review of reinforcement learning based intelligent optimization for manufacturing scheduling. Complex Syst Model Simul 2021;1(4):257–70.
[32] Li R, Gong W, Lu C. A reinforcement learning based RMOEA/D for bi-objective fuzzy flexible job shop scheduling. Expert Syst Appl 2022;203:117380.
[33] Wang L, Hu X, Wang Y, Xu S, Ma S, Yang K, Liu Z, Wang W. Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning. Comput Netw 2021;190:107969.
[34] Luo S, Zhang L, Fan Y. Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning. Comput Ind Eng 2021;159:107489.
[35] Zhao Y, Wang Y, Zhang J, Yu H, Tian Z. Application of improved Q learning algorithm in job shop scheduling problem. J Syst Simul 2022;34(6):1247–58.
[36] Zhang C, Song W, Cao Z, Zhang J, Tan PS, Chi X. Learning to dispatch for job shop scheduling via deep reinforcement learning. Adv Neural Inf Process Syst 2020;33:1621–32.

[37] Han B-A, Yang J-J. Research on adaptive job shop scheduling problems based on dueling double DQN. Ieee Access 2020;8:186474–95.
[38] Park J, Chun J, Kim SH, Kim Y, Park J. Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning. Int J Prod Res 2021;59(11):3360–77.
[39] Chen R, Li W, Yang H. A deep reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for the job-shop scheduling problem. IEEE Trans Ind Inf 2022;19(2):1322–31.
[40] Hu L, Liu Z, Hu W, Wang Y, Tan J, Wu F. Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinforcement learning with graph convolutional network. J Manuf Syst 2020;55:1–14.
[41] Song W, Chen X, Li Q, Cao Z. Flexible job-shop scheduling via graph neural network and deep reinforcement learning. IEEE Trans Ind Inf 2022;19(2):1600–10.
[42] Wang R, Wang G, Sun J, Deng F, Chen J. Flexible job shop scheduling via dual attention network based reinforcement learning. 2023, arXiv preprint arXiv:2305.05119.
[43] Allen J, Blaylock N, Ferguson G. A problem solving model for collaborative agents. In: Proceedings of the first international joint conference on autonomous agents and multiagent systems: part 2. 2002, p. 774–81.
[44] Veličković P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y. Graph attention networks. 2017, arXiv preprint arXiv:1710.10903.
[45] Van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double q-learning. In: Proceedings of the AAAI conference on artificial intelligence. vol. 30, 2016.
[46] Velickovic P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y, et al. Graph attention networks. Statistics 2017;1050(20):10–48550.
[47] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. Adv Neural Inf Process Syst 2017;30.
[48] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M. Playing atari with deep reinforcement learning. 2013, arXiv preprint arXiv:1312.5602.
[49] Haarnoja T, Zhou A, Abbeel P, Levine S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International conference on machine learning. PMLR; 2018, p. 1861–70.
[50] Nazari M, Oroojlooy A, Snyder L, Takác M. Reinforcement learning for solving the vehicle routing problem. Adv Neural Inf Process Syst 2018;31.
[51] Lei K, Guo P, Zhao W, Wang Y, Qian L, Meng X, Tang L. A multi-action deep reinforcement learning framework for flexible Job-shop scheduling problem. Expert Syst Appl 2022;205:117796.
[52] Feng Y, Zhang L, Yang Z, Guo Y, Yang D. Flexible job shop scheduling based on deep reinforcement learning. In: 2021 5th Asian conference on artificial intelligence technology. ACAIT, IEEE; 2021, p. 660–6.
[53] Zhang Y, Song W. Dual-file particle swarm optimization algorithm for flexible job shop scheduling. Comput Integr Manuf Syst 2022;59(11):294–301. http://dx.doi.org/10.3778/j.issn.1002-8331.2202-0298.

Update

Journal of Manufacturing Systems
Volume 75, Issue , August 2024, Page 333

DOI: https://doi.org/10.1016/j.jmsy.2024.04.009

Journal of Manufacturing Systems 75 (2024) 333–333

Contents lists available at ScienceDirect

Journal of Manufacturing Systems

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

Corrigendum

Corrigendum to "A novel collaborative agent reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for flexible job shop scheduling problem" [J Manuf Syst 74 (2024) 329–345]

Wenquan Zhang [a], Fei Zhao [a,*], Yong Li [b], Chao Du [c], Xiaobing Feng [a], Xuesong Mei [a]

[a] State Key Laboratory for Manufacturing Systems Engineering, Shaanxi Key Laboratory of Intelligent Robots, School of Mechanical Engineering, Xi'an Jiaotong University, Xi'an 710049, China
[b] Xi'an Kunlun Industry (Group) Co., Ltd., Xi'an, Shaanxi Province, China
[c] Shaanxi Fast Auto Drive Group Co., Ltd., No. 129 West Avenue, High-tech Zone, Xi'an, Shaanxi Province, China

The authors regret for the corrections to the published article:

The authors Wenquan Zhang, Fei Zhao and Xuesong Mei are affiliated with the State Key Laboratory for Manufacturing Systems Engineering, Shaanxi Key Laboratory of Intelligent Robots, School of Mechanical Engineering, Xi'an Jiaotong University, Xi'an, 710049, P.R. China.

The authors would like to apologise for any inconvenience caused.

DOI of original article: https://doi.org/10.1016/j.jmsy.2024.03.012.
* Corresponding author.
E-mail address: ztzhao@mail.xjtu.edu.cn (F. Zhao).

https://doi.org/10.1016/j.jmsy.2024.04.009

Available online 18 April 2024
0278-6125/© 2024 The Society of Manufacturing Engineers. Published by Elsevier Ltd. All rights reserved.