Knowledge-Based Systems xxx (xxxx) xxx

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Flexible job shop scheduling via deep reinforcement learning with meta-path-based heterogeneous graph neural network

Lanjun Wan [a,*], Long Fu [a], Changyun Li [a], Keqin Li [b]

[a] School of Computer Science, Hunan University of Technology, Zhuzhou 412007, China
[b] Department of Computer Science, State University of New York, New Paltz, NY, 12561, USA

ARTICLE INFO

Keywords:
Deep reinforcement learning
Flexible job shop scheduling problem
Graph neural network
Heterogeneous graph
Markov decision process

ABSTRACT

The flexible job shop scheduling problem (FJSP) is an important production scheduling problem in intelligent manufacturing. How to model the complex FJSP more accurately and improve the efficiency and generalization of scheduling policies is an urgent challenge to be solved. Therefore, a new end-to-end deep reinforcement learning (DRL) method combined with the meta-path-based heterogeneous graph neural network (MHGNN) is proposed to effectively solve FJSP. The task of solving FJSP is decomposed into two subtasks of operation selection and machine allocation. This dual-task FJSP is represented by introducing a heterogeneous graph and modeled as the dual Markov decision process (DMDP). A MHGNN is proposed to embed the global scheduling states of the dual-task FJSP. A heterogeneous graph neural network (GNN) framework is designed for the dual-task FJSP to efficiently encode the operation nodes and machine nodes, where the extracted embedded feature information is used to represent the operation selection policy and the machine allocation policy. Two policy networks are designed to efficiently predict the operation selection policy and the machine allocation policy, and a soft double-actors critic algorithm is proposed to train these two policy networks, where the trained policies can be used to efficiently solve FJSP instances of different scales. The experiments on three public benchmarks show that the proposed method outperforms the well-known heuristic scheduling rules and some advanced methods for solving FJSP. In particular, when solving large-scale FJSPs, the proposed method performs more prominently in terms of solution quality and solution time.

1. Introduction

With the development of economic globalization, traditional flexible manufacturing is facing significant challenges and finding it difficult to meet the requirements of the market [1]. To cope with these difficulties, the flexible job shop scheduling has gradually become the focus of manufacturing enterprises. FJSP is an NP-hard combinatorial optimization problem. Compared with the job shop scheduling problem (JSSP), FJSP relaxes the restrictions on machines so that each operation can be processed on multiple compatible machines for each job, which makes FJSP more flexible and sophisticated.

Recently, the methods used to solve FJSP can be broadly classified into exact algorithms, heuristic algorithms, meta-heuristic algorithms, and DRL methods [2]. The exact algorithms [3–5] seek the optimal solution by exhausting all possible scheduling schemes. They often model the FJSP mathematically and solve it using a mathematical programming model. However, as the scale of FJSP increases, the computational complexity of the exact algorithms grows exponentially. The heuristic algorithms [6–8] utilize heuristic rules and search strategies to solve FJSP, which has certain randomness and adaptability. They can find the approximate optimal solutions in a short time when solving FJSP, but they cannot guarantee to search for the global optimal solution. The meta-heuristic algorithms [9–11] improve the solution performance by combining multiple basic heuristic algorithms or adaptively modifying the parameters of the heuristic algorithm, and they are suitable for solving FJSPs in different scenarios. However, they require many computational resources when solving large-scale FJSPs, and they are easy to fall into local optimal solutions without jumping out. The DRL methods [12–15] can learn the decision process of the scheduling policies from the original input data without defining specific scheduling rules in advance by combining deep neural network and reinforcement learning (RL) techniques, and they have good performance in solving FJSPs of different scales. Because of these advantages of the DRL methods, a DRL approach is studied to solve FJSP in this paper.

In the face of more complex FJSPs, how to effectively solve them using DRL methods faces the following challenges: (i) how to reasonably allocate each operation to one of the compatible machines;

* Corresponding author.
E-mail address: wanlanjun@hut.edu.cn (L. Wan).

https://doi.org/10.1016/j.knosys.2024.111940
Received 21 August 2023; Received in revised form 31 January 2024; Accepted 10 May 2024
Available online 16 May 2024

(ii) how to extract more effective global scheduling information using deep neural networks; and (iii) how to make the trained scheduling policies efficiently solve FJSPs of different scales. In view of this, a new end-to-end DRL approach is proposed to efficiently solve FJSP. First, the operation selection and machine allocation in FJSP are separated into two different decisions instead of an integrated decision, which is convenient for the decision-making agent to perform operation selection and machine allocation. Second, a heterogeneous GNN framework based on encoder–decoders is designed to process the structure information of the heterogeneous graph representing the global scheduling states of FJSP, where a GNN is used as the encoder and two policy networks are used as two decoders. Finally, a new soft double-actors critic algorithm is proposed to efficiently train policy networks, which seeks to maximize the reward while increasing the exploration space of the actions, and this facilitates the generation of more scheduling policies for the agent to select.

The main contributions of this paper are as follows.

- The task of solving FJSP is decomposed into two subtasks of operation selection and machine allocation. This dual-task FJSP is represented by introducing a heterogeneous graph and modeled as the DMDP with the specially designed states, actions, and rewards.
- A MHGNN is proposed for embedding the global scheduling states of the dual-task FJSP. A heterogeneous GNN framework is designed for the dual-task FJSP to efficiently encode the operation nodes and machine nodes, where the extracted embedded feature information is used to represent the operation selection policy and the machine allocation policy.
- Two policy networks are designed for efficient predictions of the operation selection policy and the machine allocation policy. A soft double-actors critic algorithm is put forward to train these two policy networks, where the trained policies can be used to efficiently solve FJSP instances of different scales.
- A large number of experiments are conducted on three public benchmarks to verify the efficiency, stability, and generalization of the proposed method by comparing it with the well-known heuristic scheduling rules and some advanced approaches for solving FJSP.

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 introduces the preliminaries. Section 4 describes the proposed method for solving FJSP. Section 5 presents the experimental results and analysis. Finally, Section 6 provides the conclusions and the future work.

2. Related work

2.1. Exact algorithms for solving FJSP

The exact algorithms have been applied early to solve small and medium-scale FJSP instances. Demir and İşleyen [3] proposed five different mathematical models for FJSPs with completion time performance measurements and proved that the mixed-integer linear programming (MILP) model has the smallest makespan (*i.e.*, the maximum completion time) in solving FJSP. Gran et al. [4] presented a mixed integer programming model to minimize the total process time and makespan, which can be well applied to solve small-scale FJSP instances. Meng et al. [5] put forward four MILP models and a constrained planning model to minimize the makespan for the distributed FJSP. These models can obtain the optimal solution for small and medium-scale FJSP instances. The exact algorithms can usually obtain an exact solution in a reasonable time when solving small and medium-scale FJSP instances. However, due to their high computational complexity, the solution time increases exponentially, thus they are not suitable for solving large-scale FJSP instances.

2.2. Heuristic algorithms for solving FJSP

In recent years, heuristic algorithms have been successfully applied to solve FJSP in a specific scenario. Zhang et al. [6] proposed a new genetic planning algorithm to measure the importance of the machine allocation rules and operation sequencing rules. This algorithm can effectively solve dynamic FJSP (DFJSP) with the arrival of new jobs by utilizing computational resources reasonably. Ai et al. [7] proposed a new heuristic algorithm for solving FJSP with due windows. A mathematical planning model is used to model FJSP, and the model is solved with the genetic algorithm and quadratic programming method respectively, which can minimize the weighted earliness or tardiness. Zhang et al. [8] used a multi-objective MILP model to model the energy-efficient FJSP and introduced a new heuristic algorithm to solve the model efficiently, which effectively solves FJSP with transportation time and sequence-dependent set-up time. The heuristic algorithms can generate high-quality approximate solutions in a faster time when solving FJSP, but they rely on the design of heuristic rules. The formulation of heuristic rules relies on rich knowledge and experience in the scheduling field. In addition, the heuristic rules have significant limitations, and the same heuristic rules may not always lead to satisfactory scheduling schemes in different scenarios.

2.3. Meta-heuristic algorithms for solving FJSP

Recently, meta-heuristic algorithms have been widely used to solve FJSPs in different scenarios. Yang et al. [9] offered an improved dragonfly algorithm and adopted a dynamic opposite learning strategy to enhance the searching ability, which can effectively solve FJSP. Sun et al. [10] introduced an improved hybrid meta-heuristic algorithm to enhance the ability to search for the global optimal solution of FJSP by combining the variable neighborhood search algorithm and the genetic algorithm, which can effectively minimize the makespan of FJSP. Wei et al. [11] proposed a migrating birds optimization algorithm for the DFJSP with machine breakdowns. The algorithm introduces game theory to optimize the two objectives of productivity and stability of DFJSP, which can effectively solve multi-objective DFJSP. These meta-heuristic algorithms have achieved good results in solving FJSP, but in the face of FJSPs in different scenarios, the reasonable setting of parameters in the meta-heuristic algorithms is a challenge that requires extensive experimental tuning. Moreover, as the scale of FJSP instances increases, the underlying algorithms need to go through a large number of iterations, which will consume a large amount of computational time, and the computational complexity of the meta-heuristic algorithms becomes a significant limiting factor.

2.4. DRL methods for solving FJSP

Recently, the application of DRL has provided new ideas for solving FJSP [16]. DRL, as one of the most active AI research areas, combines the perceptual ability of deep learning [17] with the decision-making ability of RL [18], which has been widely applied to solve combinatorial optimization problems. Luo [12] put forward a deep Q-network (DQN)-based DRL approach and designed multiple composite scheduling rules for DQN training, which can minimize the total tardiness of DFJSP with new job insertions. Feng et al. [13] designed a DRL method based on proximal policy optimization (PPO) to learn the scheduling policy for minimizing the makespan of FJSP. Luo et al. [14] introduced a DRL method based on two-hierarchy DQN for DFJSP with new job insertions, and the two DQNs are hierarchically trained. The trained model is used to optimize the average machine utilization rate and the total weighted tardiness. Liu et al. [15] proposed a DRL method based on double DQN (DDQN), which is able to efficiently solve DFJSP with constant job arrivals. These DRL methods model FJSP as a Markov decision process (MDP), extract multiple general state features as the input of states, and design multiple scheduling rules as actions for

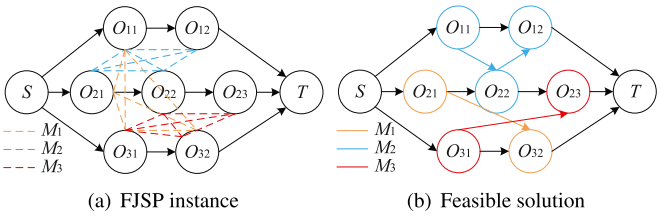
decision-making on operations and machines. They have shown good performance in solving FJSP. However, the state information for the entire process of flexible job shop scheduling is compressed, and the use of structural information in flexible job shop scheduling is also extremely limited.

The above problems can be effectively solved by using graphs (*e.g.*, the disjunctive graph) to represent scheduling states of FJSP, using GNN to process the complicated graph structural information and represent the extracted embedded feature information as scheduling policies, and training and optimizing the scheduling policies via the DRL. Currently, the fusion of GNN and DRL has attracted considerable attention from researchers [19]. Zeng et al. [20] developed a DRL method to solve FJSP. A disjunctive graph is used to represent the local scheduling states of FJSP, GNN is introduced to extract state features from the disjunctive graph, and the trained policy is used to minimize the makespan of FJSP. Lei et al. [21] proposed a hierarchical DRL framework based on the disjunctive graph to solve large-scale DFJSPs with random job arrivals. The higher-layer agent divides the DFJSP into multiple FJSPs, and the lower-layer agents based on GNN are used to learn scheduling policies for FJSP. Lei et al. [22] put forward an end-to-end DRL framework that uses a disjunctive graph to represent the local scheduling states of FJSP and uses GNN to process the structural information of the disjunctive graph. A multi-PPO algorithm is used to train the operation selection policy and the machine allocation policy separately. Lei et al. [23] designed a multi-agent hierarchical RL framework based on the disjunctive graph, which can minimize the makespan of large-scale DFJSPs with random job arrivals. The DFJSP is transformed into multiple static FJSPs by the agent based on DDQN, where the operation selection and machine allocation are controlled through the GNN-based agent and multi-layer perceptron (MLP)-based agent respectively. The above research solves FJSP through the fusion of GNN and DRL, which fully utilizes the advantages of GNN in processing complicated graph structure information and the powerful feature perception and decision-making capabilities of DRL. Specifically, first, the disjunctive graph is used to represent the FJSP, where the corresponding features are set for each type of node and edge in the graph to represent the structure information of the FJSP. Second, GNN is used as the encoder to extract the feature information of each embedded node, and the policy network is designed as the decoder to process the output of the encoder. Finally, the policy network is trained by the DRL method, and the scheduling strategy trained by this end-to-end DRL method can efficiently solve FJSP instances of different scales. However, the global scheduling states of the entire FJSP are composed of the scheduling states of the operations and those of the machines. When the disjunctive graph is used to represent the scheduling states of the FJSP, there is only the representation of operation nodes but no representation of machine nodes. Therefore, when GNN is used to process the structural information of the disjunction graph, it is difficult to extract the features of machine nodes and lean toward feature extraction of operation nodes, which makes it difficult for the trained scheduling policies to maximize the utilization of machine resources to more rationally allocate the available machines to the operations.

3. Preliminaries

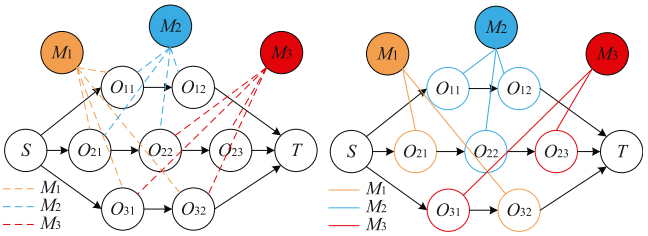
3.1. Description of FJSP

A FJSP instance contains n jobs and m machines. The set of jobs is denoted as $J = \{J_1, J_2, \ldots, J_n\}$, and the set of machines is denoted as $M = \{M_1, M_2, \ldots, M_m\}$, where the job J_i consists of n_i consecutive operation $O_i = \{O_{i1}, O_{i2}, \ldots, O_{in_i}\}$ with sequence constraints. For the job J_i, the next operation O_{ij} cannot start until its previous operation $O_{i(j-1)}$ has been completed. The operation O_{ij} can be processed on any of a set of available machines M_{ij} and the processing time of O_{ij} on its available machine M_k is denoted as P_{ijk}. Once an operation starts processing, it cannot be interrupted, and each machine can only process one operation at a time. the goal of the FJSP is to minimize the makespan $C_{\max} = \max\{C_{in_i}\}$, where C_{in_i} is the completion time of the job J_i and $1 \le i \le n$.



(a) FJSP instance (b) Feasible solution

Fig. 1. Disjunctive graph representation of a FJSP instance.



(a) FJSP instance (b) Feasible solution

Fig. 2. Heterogeneous graph representation of a FJSP instance.

3.2. Heterogeneous graph representation of FJSP

The disjunctive graph [24] is commonly used to represent JSSP instances, but with the deepening of research, some researchers gradually began to use it to represent FJSP instances. The disjunctive graph for representing FJSP can be defined as the triple $G = (\mathcal{O}, \mathcal{C}, \mathcal{D})$, where $\mathcal{O} = \{\mathcal{O}_{ij} | \forall i, j\} \cup \{S, T\}$ is the set of real operation nodes and dummy operation nodes. The real operation node $\mathcal{O}_{ij}$ denotes the j-th operation of the i-th job. $\{S, T\}$ is the set of dummy operation nodes, where S and T represent the dummy start and end operations with zero processing time, respectively. $\mathcal{C}$ is a set of directed conjunctive arcs, where a path formed by connecting S to T with a solid line is used to represent the sequence constraints among the operations of a job. $\mathcal{D}$ is a set of undirected disjunctive arcs, where multiple operations that can be processed on the same machine are connected with dashed lines. The disjunctive graph representation of a 3×3 FJSP instance is provided in Fig. 1.

For FJSP, the constraints on machines are relaxed, which makes it extremely complicated to use the disjunctive graph to represent FJSP. Therefore, a heterogeneous graph [25] is used to represent FJSP instances. It introduces a set $\mathcal{M}$ of machine nodes in the original disjunctive graph G. The set $\mathcal{D}$ of disjunctive arcs in G is replaced by a set E of undirected O-M arcs connecting operation nodes to machine nodes. Here, the heterogeneous graph is defined as $HG = (\mathcal{O}, \mathcal{M}, \mathcal{C}, E)$. The heterogeneous graph representation of a 3×3 FJSP instance is provided in Fig. 2.

The heterogeneous graph representation of FJSP has the following advantages. Firstly, the heterogeneous graph can effectively handle multiple types of nodes, *i.e.*, operation nodes and machine nodes. Secondly, only after determining which operations will be processed on which machine, the corresponding O-M arcs will be added and other unrelated O-M arcs will be eliminated, which greatly reduces the density of the graph and can effectively reduce the computational and storage costs.

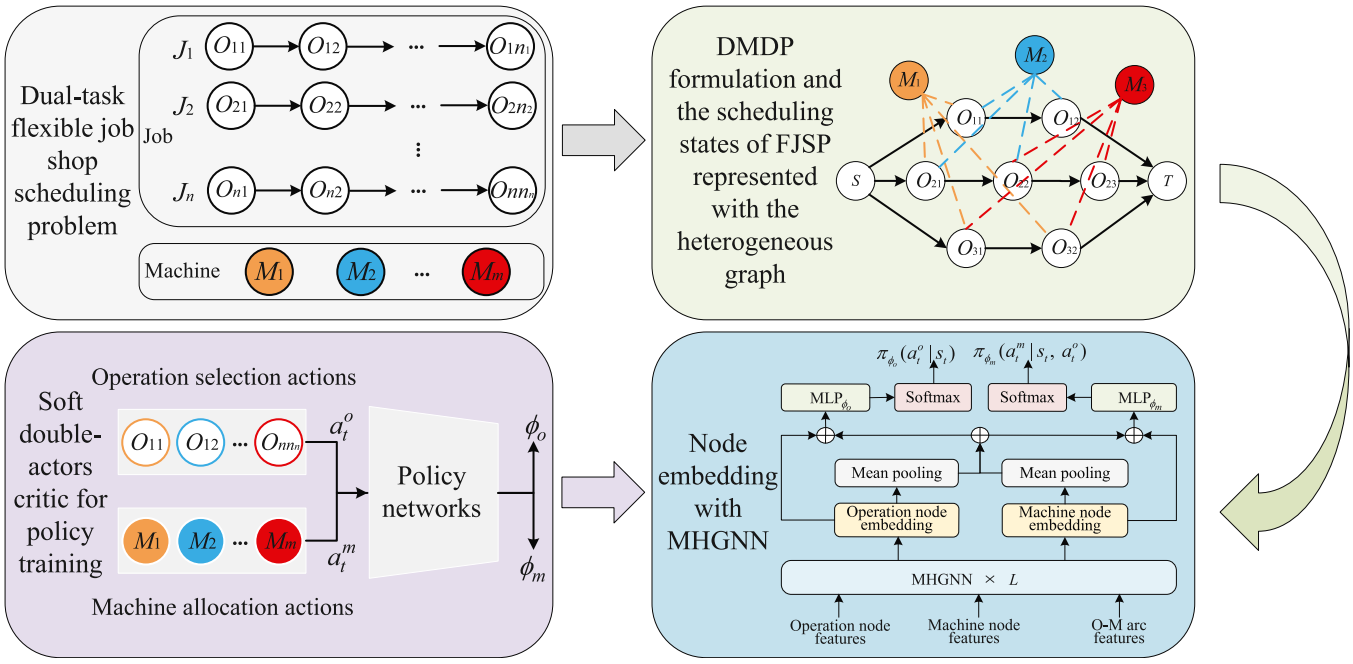


Fig. 3. Overall process of the proposed method for solving FJSP.

4. Proposed method

The overall process of the proposed method for solving FJSP is shown in Fig. 3. Firstly, the task of solving FJSP is decomposed into two subtasks of operation selection and machine allocation, a heterogeneous graph is used to represent the scheduling states of the dual-task FJSP, and the dual-task FJSP is modeled as DMDP. Secondly, the complex scheduling states of the dual-task FJSP are embedded using MHGNN, and a heterogeneous GNN framework is used to encode the operation nodes and machine nodes. The extracted embedded feature information is represented as the operation selection policy and the machine allocation policy. The two policy networks based on MLP including MLP_{ϕ_o} and MLP_{ϕ_m} are used to predict the operation selection policy and machine allocation policy, respectively. Finally, a soft double-actors critic algorithm is used to train the two policy networks and output the operation selection policy and the machine allocation policy.

4.1. DMDP formulation

Owing to the task of solving FJSP being decomposed into two subtasks of operation selection and machine allocation, when this dual-task FJSP is modeled as the MDP, the agent needs to simultaneously control the actions in two dimensions at each time step, which makes the original one-dimensional action space into the two-dimensional action space, *i.e.*, $A = A^o \times A^m$. Therefore, a DMDP is put forward to model the dual-task FJSP, and a heterogeneous graph $HG = (\mathbb{O}, \mathbb{M}, \mathbb{C}, E)$ is used to represent the global scheduling states of the dual-task FJSP, where the definitions of state, action, transition, reward, and policy are as follows.

State: s_t is the scheduling state of the dual-task FJSP at time step t. The operation node $\mathbb{O}_{ij} \in \mathbb{O}$ has three features $\left[SF(O_{ij}), SCT(O_{ij}), |N(\mathbb{O}_{ij})|\right]$. $SF(O_{ij})$ is the scheduling status flag of O_{ij}, where the widely used one-hot encoding method is adopted to set different status flags. It has three different flags [1, 0, 0], [0, 1, 0], and [0, 0, 1], indicating that O_{ij} has been scheduled, O_{ij} is being scheduled, and O_{ij} is not scheduled, respectively. If O_{ij} is not scheduled, $SCT(O_{ij})$ denotes the estimated completion time of O_{ij}: $SCT(O_{i(j-1)}) + T$; otherwise, $SCT(O_{ij})$ denotes the completion time of O_{ij}. Note that if O_{ij} is processed on the machine M_k, then $T = P_{ijk}$; otherwise, $T = \frac{1}{|M_{ij}|}\sum_{M_l \in M_{ij}} P_{ijl}$. $N(\mathbb{O}_{ij})$ is the set of neighboring machine nodes of $\mathbb{O}_{ij}$, *i.e.*, the set of machines that can process O_{ij}. The machine node $\mathbb{M}_k \in \mathbb{M}$ contains two features $\left[CT_t(M_k), \left|N_t(\mathbb{M}_k)\right|\right]$. $CT_t(M_k)$ is the completion time for machine M_k to process the assigned operations to it before the current time step t. $N_t(\mathbb{M}_k)$ is the set of neighboring operation nodes of the machine node $\mathbb{M}_k$ at time step t. The undirected O-M arc $\varepsilon_{ijk} \in E$ connecting the operation node $\mathbb{O}_{ij}$ and the machine node $\mathbb{M}_k$ contains a feature $\left[P_{ijk}\right]$.

Action: The operation selection action $a_t^o \in A_t^o$ and the machine allocation action $a_t^m \in A_t^m$ together form the action $a_t \in A_t$, *i.e.*, $a_t = \left\{a_t^o, a_t^m\right\}$. A_t^o is the eligible operation selection action space. A_t^m is the available machine allocation action space.

Transition: The transition from the current state s_t to the next state s_{t+1} is completed by updating the heterogeneous graph. A new heterogeneous graph is generated at time step t. Specifically, at first a_t^o is executed to select an eligible operation, then a_t^m is executed to allocate an available machine to the operation, and finally the corresponding undirected O-M arc is connected.

Reward: The immediate reward is set to be the difference of the makespan from the current state s_t to the next state s_{t+1}, *i.e.*, $r_t\left(s_t, a_t, s_{t+1}\right) = C_{\max}\left(s_t\right) - C_{\max}\left(s_{t+1}\right)$. The cumulative reward is set to $\sum_{t=0}^{end} r_t\left(s_t, a_t, s_{t+1}\right) = -C_{\max}\left(s_{end}\right)$, where $C_{\max}\left(s_{end}\right) = C_{\max}$.

Policy: The policy $\pi_\phi\left(a_t \middle| s_t\right)$ represents the probability distribution of the action a_t under the current state s_t. Since the whole complete action consists of the operation selection action and the machine allocation action, the policy $\pi_\phi\left(a_t \middle| s_t\right)$ is divided into two sub-policies $\pi_{\phi_o}\left(a_t^o \middle| s_t\right)$ and $\pi_{\phi_m}\left(a_t^m \middle| s_t, a_t^o\right)$. $\pi_{\phi_o}\left(a_t^o \middle| s_t\right)$ is used to train the operation selection action a_t^o. $\pi_{\phi_m}\left(a_t^m \middle| s_t, a_t^o\right)$ is used to train the machine allocate action a_t^m. A soft double-actors critic algorithm is designed to train and optimize the operation selection policy and the machine allocation policy toward maximizing the cumulative reward.

4.2. Heterogeneous GNN framework

4.2.1. Overall design of a heterogeneous GNN framework

The graph isomorphism network (GIN) [26] can well consider node features and information interactions between neighboring nodes during node aggregation. The learned node embedding information can be directly used for node classification, link prediction, etc. However,

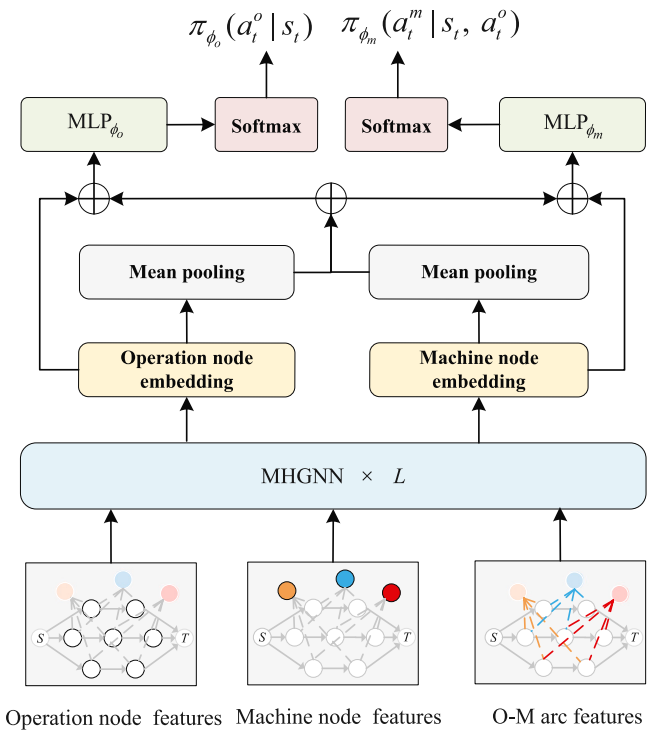


Fig. 4. Overall design of the proposed heterogeneous GNN framework for solving the dual-task FJSP.

GIN is not suitable for dealing with the heterogeneous graph. The heterogeneous GNN [27], as a network model based on the heterogeneous graph, can fully utilize the heterogeneous information of different types of nodes to process the heterogeneous graph, and can enhance the ability of node feature representation by utilizing the heterogeneous information. Therefore, a MHGNN is proposed to embed the complex scheduling states of the dual-task FJSP, and a heterogeneous GNN framework is designed for the dual-task FJSP to efficiently encode the operation nodes and machine nodes.

The overall design of the proposed heterogeneous GNN framework for solving the dual-task FJSP is shown in Fig. 4. Firstly, a heterogeneous graph is used to represent the global scheduling states of the dual-task FJSP. When embedding node information, it is necessary to embed the information of operation nodes and machine nodes separately, including the following steps: (i) the operation node information is embedded into the heterogeneous graph; (ii) the machine node information is embedded into the heterogeneous graph; and (iii) the undirected O-M arcs corresponding to the embedded operation nodes and machine nodes are connected to complete the updating of the heterogeneous graph. The information involved in the operation node embedding and machine node embedding mainly includes the original features of the operation node (*i.e.*, the scheduling status of the operation, the estimated completion time of the operation, and the number of neighboring machine nodes of the operation node), the original features of the machine node (*i.e.*, the completion time for the machine to process the assigned operations to it before the current time step t and the number of neighboring operation nodes of the machine node at time step t), and the original feature of the undirected O-M arc (*i.e.*, the processing time of the operation on the machine). Taking a 3×3 FJSP instance as an example, the entire embedding process of this instance is shown in Fig. 5. Secondly, the heterogeneous GNN framework is used to encode the operation nodes and machine nodes, and the extracted embedded feature information is represented as the operation selection policy and the machine allocation policy. Finally, the two policy networks MLP$_{\phi_o}$ and MLP$_{\phi_m}$ are used to predict the operation selection policy and the machine allocation policy, respectively.

4.2.2. Operation node embedding

In the heterogeneous graph, the feature information of each operation node is related not only to its neighboring operation nodes but also to its neighboring machine nodes. For the heterogeneous graph containing multiple operation nodes and machine nodes, if the feature information of all neighboring operation nodes and neighboring machine nodes is considered, the extracted embedded feature information will be too large, and it will be difficult to extract the effective information. The meta-path aggregated graph neural network (MAGNN) [28] can better embed the node feature information through intra-meta-path aggregation and inter-meta-path aggregation. Therefore, the operation node feature information is processed using MAGNN, and the operation–machine–operation is set as a meta-path with a length of 3. The embedding of an operation node consists of the following three processes: node information transformation, intra-meta-path aggregation, and inter-meta-path aggregation. The embedding process of a target operation node is shown in Fig. 6.

(1) Node information transformation

Let V_m be the set of machine node m and all its neighboring operation nodes. The feature information of the target operation node $v \in V_m$ and the machine node $m \in V_m$ is linearly transformed by

$$h'_v = W_m \times v_t \tag{1}$$

and

$$h'_m = W_m \times m_t, \tag{2}$$

respectively. In Eq. (1), W_m is a weight matrix, v_t is the original feature vector of the target operation node v, and h'_v is the transformed feature vector of the target operation node v. In Eq. (2), m_t is the feature vector of the machine node m, and h'_m is the transformed feature vector of the machine node m.

(2) Intra-meta-path aggregation

All feasible meta-paths $P_m = \{p_1, p_2, \ldots, p_n\}$ containing the machine node m and the target operation node v need to be found in V_m, where n is the number of all feasible meta-paths. There may be multiple available machines for an operation, thus multiple different sets of meta-paths will be generated. Let N_v^p be the set of meta-path-based neighboring operation nodes of the target operation node v. The meta-path formed by the target operation node v and its neighboring operation $u \in N_v^p$ is defined as $p(v, m, u)$. The process of the intra-meta-path aggregation includes the following steps.

Step 1: Transform the meta-path $p(v, m, u)$ into the vector $h_{p(v,m,u)}$ using mean coding:

$$h_{p(v,m,u)} = \text{MEAN}\left(h'_v, h'_m, h'_u\right). \tag{3}$$

Step 2: Perform the weighted aggregation on the vector representation of the meta-path $p(v, m, u)$ by

$$e^p_{vmu} = \text{LeakyRelu}\left(a_p^T \bullet \left[h'_v \parallel h_{p(v,m,u)}\right]\right), \tag{4}$$

where a_p^T denotes the transpose of the attention vector a_p of the meta-path $p(v, m, u)$, and e^p_{vmu} denotes the importance of the meta-path $p(v, m, u)$ to the target operation node v.

Step 3: Use the softmax function to normalize e^p_{vmu}:

$$\alpha^p_{vmu} = \frac{\exp\left(e^p_{vmu}\right)}{\sum_{s \in N_v^p} \exp\left(e^p_{vms}\right)}. \tag{5}$$

Step 4: Use the attention mechanism to perform weighted summation on the vector representations of the meta-path instances related to the target operation node v, and the result is processed using the sigmoid activation function σ:

$$h_v^p = \sigma\left(\sum_{s \in N_v^p} \alpha^p_{vms} \bullet h_{p(v,m,s)}\right). \tag{6}$$

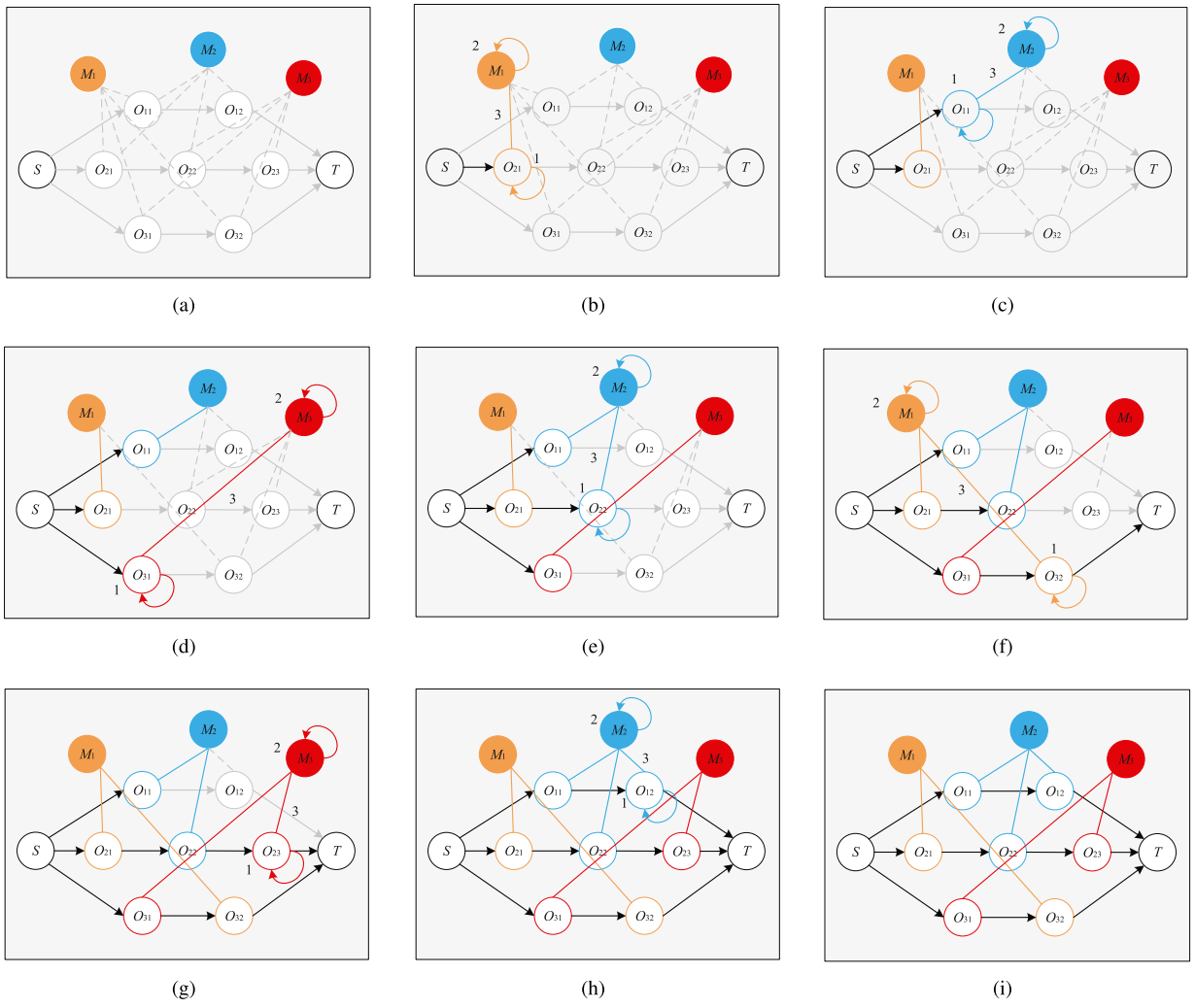


Fig. 5. Embedding process of a FJSP instance.

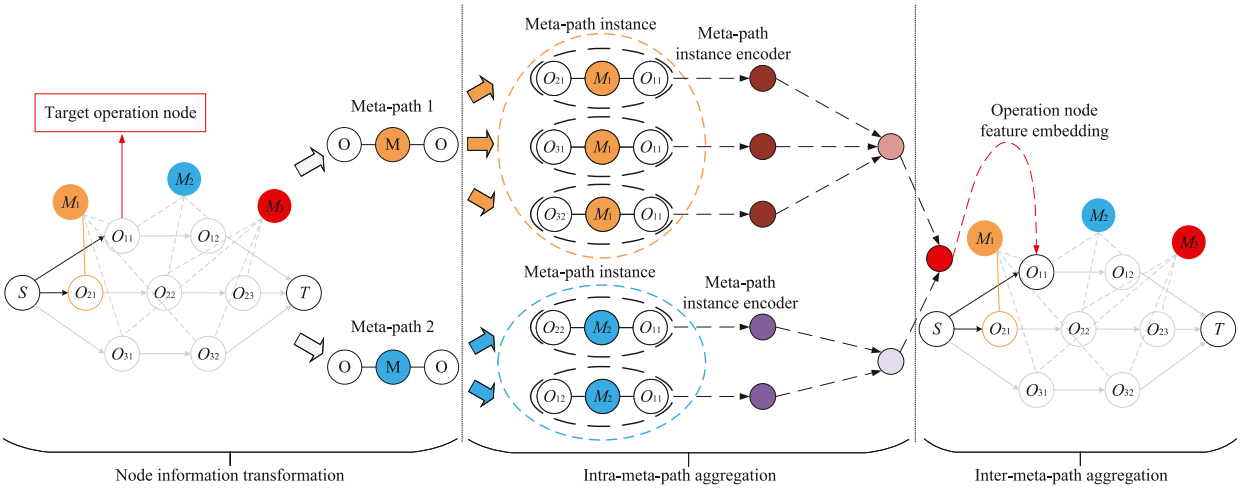


Fig. 6. Embedding process of a target operation node.

(3) Inter-meta-path aggregation

After performing the Intra-meta-path aggregation, for the target operation node v, $|V_m|$ specific vector representations $\{h_v^{p_1}, h_v^{p_2}, \ldots, h_v^{p_n}\}$ will be generated. The process of the inter-meta-path aggregation consists of the following steps.

Step 1: For the meta-path $p_i \in P_m$, the node vectors of all operation nodes in V_m under the specific meta-path are nonlinearly transformed and averaged by

$$S_{p_i} = \frac{1}{|V_m|} \sum_{v \in V_m} \tanh\left(X_m \bullet h_v^{p_i} + b_m\right), \tag{7}$$

where X_m is a weight matrix, and b_m is an offset vector.

Step 2: Use the attention mechanism to compute the attention coefficient e_{p_i}:

$$e_{p_i} = Y_m^T \bullet S_{p_i}, \tag{8}$$

where Y_m^T denotes the transpose of the attention vector Y_m of S_{p_i}.

Step 3: Use softmax function to normalize e_{p_i} to obtain the attention score β_{p_i}:

$$\beta_{p_i} = \frac{\exp\left(e_{p_i}\right)}{\sum_{p_j \in P_m} \exp\left(e_{p_j}\right)}, \tag{9}$$

Step 4: Fuse the vector representations corresponding to these specific meta-paths to obtain the output $h_v^{P_m}$ of the target operation node v:

$$h_v^{P_m} = \sum_{p_i \in P_m} \beta_{p_i} \bullet h_v^{p_i}. \tag{10}$$

Step 5: Calculate the final embedded feature vector h_v of the target operation node v by

$$h_v = \sigma\left(Z_m \bullet h_v^{P_m}\right), \tag{11}$$

where Z_m is a weight matrix.

4.2.3. Machine node embedding

The machine node is connected to its neighboring operation nodes with the undirected O-M arcs in the heterogeneous graph. To facilitate the processing of the feature information of an O-M arc, the original feature vector of the O-M arc is concatenated with the original feature vector of the machine node, *i.e.*, the feature information of the O-M arc is integrated into the machine node. Thus, the feature vector m_t of the machine node m becomes the three-dimensional feature vector. When embedding the machine node m, in addition to considering the feature information of the machine node m itself, the original feature information of its neighboring operation nodes should also be considered. Therefore, three MLPs with the same structure are used to process the feature vector of the machine node m and the original feature vectors of its neighboring operation nodes to obtain the embedded feature vector h_m of the machine node m:

$$h_m = \mathrm{MLP}_\psi\left(\mathrm{ELU}\left(\mathrm{MLP}_{\psi_\mathcal{O}}\left(\sum_{v \in N_t(m)} v_t\right) \parallel \mathrm{MLP}_{\psi_m}\left(m_t\right)\right)\right), \tag{12}$$

where MLP_{ψ_o}, MLP_{ψ_m}, and MLP_ψ all contain two 128-dimensional hidden layers and use ELU as the activation function.

4.2.4. Pooling

The above described embedding process of operation nodes and machine nodes can be regarded as the embedding on a single MHGNN layer. To improve the feature extraction ability, L MHGNN layers with the same structure are stacked and used to embed the operation nodes and machine nodes. The set of embedded feature vectors of all operation nodes and the set of embedded feature vectors of all machine nodes can be obtained in this way. The pooling vectors v_t' and m_t' are obtained by performing mean pooling on $h_v^{(L)}$ and $h_m^{(L)}$ respectively:

$$v_t' = \frac{1}{|\mathcal{O}|} \sum_{v \in \mathcal{O}} h_v^{(L)}, \tag{13}$$

$$m_t' = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} h_m^{(L)}, \tag{14}$$

where $h_v^{(L)}$ denotes the embedded feature vector of the target operation node v on L MHGNN layers, and $h_m^{(L)}$ denotes the embedded feature vector of the machine node m on L MHGNN layers.

4.2.5. Action prediction

The two policy networks MLP_{ϕ_o} and MLP_{ϕ_m} are designed for the operation selection policy and the machine allocation policy, respectively. They have the same network structure including two 128-dimensional hidden layers and the tanh activation function, but they share different parameters. The policy networks map the scheduling states of the dual-task FJSP represented by the heterogeneous graph to the probability distribution, and select actions based on the probability distribution. The whole process of action prediction is as follows. Firstly, the selection probabilities of the operation selection action and the machine allocation action can be calculated by

$$p\left(a_t^o, s_t\right) = \mathrm{MLP}_{\phi_o}\left(h_v^{(L)} \parallel m_t' \parallel v_t'\right) \tag{15}$$

and

$$p\left(a_t^m, s_t\right) = \mathrm{MLP}_{\phi_m}\left(h_m^{(L)} \parallel m_t' \parallel v_t'\right), \tag{16}$$

respectively. Secondly, the selection probabilities are normalized using the softmax function:

$$\pi_{\phi_o}\left(a_t^o \middle| s_t\right) = \frac{\exp\left(p\left(a_t^o, s_t\right)\right)}{\sum_{a_t^{o'} \in A_t^o} \exp\left(p\left(a_t^{o'}, s_t\right)\right)} \tag{17}$$

and

$$\pi_{\phi_m}\left(a_t^m \middle| s_t, a_t^o\right) = \frac{\exp\left(p\left(a_t^m, s_t\right)\right)}{\sum_{a_t^{m'} \in A_t^m} \exp\left(p\left(a_t^{m'}, s_t\right)\right)}. \tag{18}$$

Finally, the different decoding strategies are used to make predictions for the operation selection action a_t^o and the machine allocation action a_t^m. During training, a random sampling decoding strategy is used for policy training to explore more actions. During testing, a greedy decoding strategy is used to test the trained policies for seeking the optimal action.

4.3. Soft double-actors critic

The soft actor–critic (SAC) algorithm [29] is a DRL algorithm based on the idea of maximum entropy. SAC, as an off-policy actor–critic algorithm, differs most from other DRL algorithms in that it optimizes the policy to achieve a higher cumulative return while also maximizing the entropy of the policy to better increase the exploratory space of the actions. Compared with the popular PPO algorithm [30], the SAC algorithm has better convergence and stability and increases the exploration of actions. The original SAC algorithm has only one actor network. It can only use one policy network to learn a single policy and control a single action, *i.e.*, it cannot be applied to control multiple actions. Therefore, the following improvements are made on the basis of the original SAC algorithm to be suitable for solving the dual-task FJSP. The original actor network is transformed into an operation actor network and a machine actor network. After simplifying the original critic network, it consists of a target Q-network and a soft Q-network, each of which contains two 128-dimensional hidden layers. The improved SAC algorithm is named the soft double-actors critic algorithm. The training process of the proposed soft double-actors critic algorithm is shown in Fig. 7.

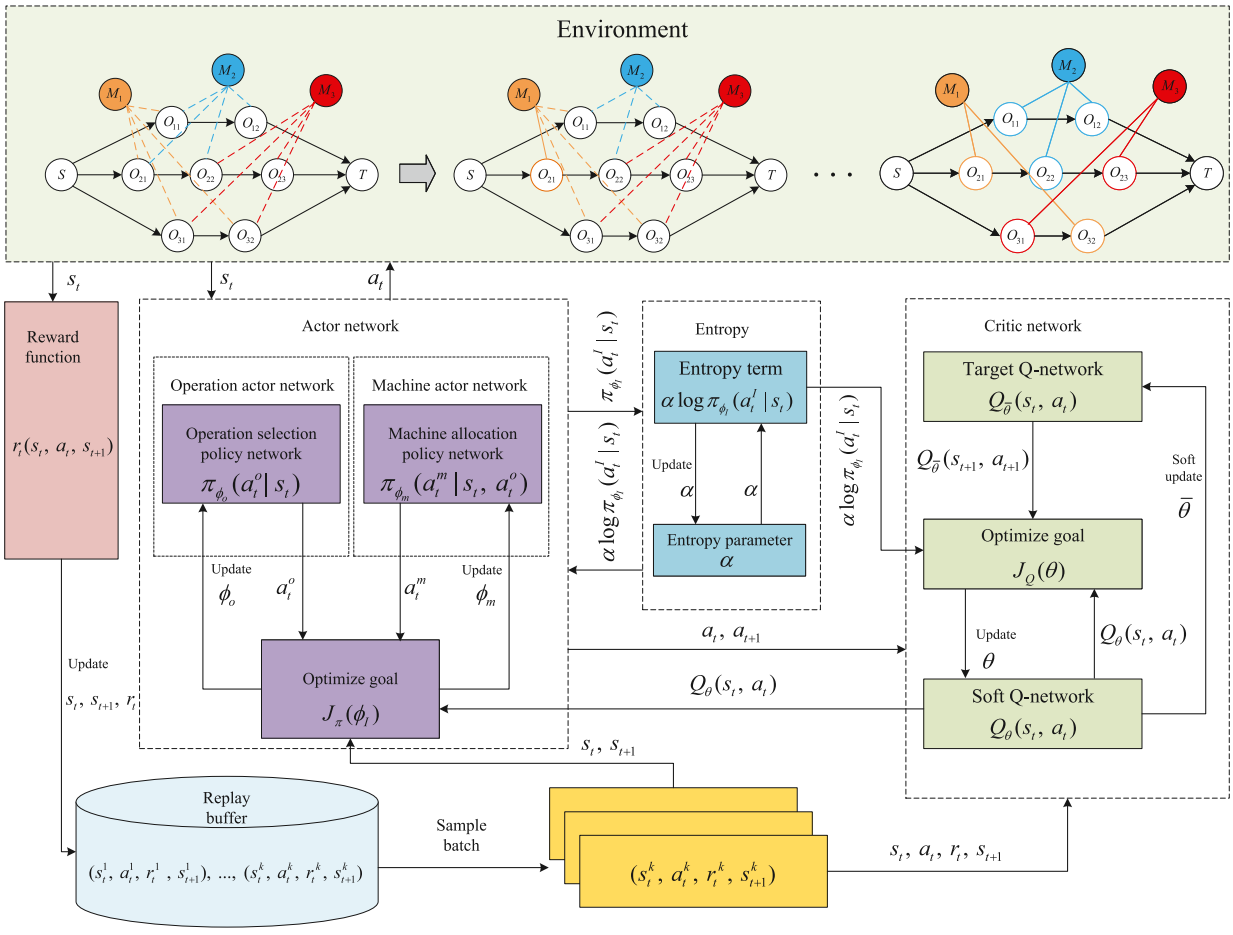


Fig. 7. Training process of the proposed soft double-actors critic algorithm.

4.3.1. Solving the optimal policy

In the proposed soft double-actors critic algorithm, the optimal policy π^* based on maximum entropy is solved by

$$\pi^* = \operatorname{argmax} \sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r_t + \alpha H\left(\pi\left(a_t \middle| s_t\right)\right)], \tag{19}$$

with

$$H\left(\pi\left(a_t \middle| s_t\right)\right) = -\log\left(\pi\left(a_t \middle| s_t\right)\right), \tag{20}$$

where ρ_π denotes the probability of occurrence of the state–action pair (s_t, a_t) under the policy π, $H\left(\pi\left(a_t \middle| s_t\right)\right)$ is the entropy of the policy π under the current state s_t, and α is the weight parameter of the entropy. α is used to weigh the relative importance of entropy and the reward to control the stochasticity of the optimal policy. The entropy parameter α is updated by minimizing the objective function $J(\alpha)$:

$$J(\alpha) = \mathbb{E}_{a_t\sim\pi_t}\left[-\alpha\log\pi_t\left(a_t \middle| s_t\right) - \alpha\bar{\mathcal{H}}\right], \tag{21}$$

where $\bar{\mathcal{H}}$ is a constant that represents the threshold of minimum policy entropy.

4.3.2. Training the actor networks

The two actor networks consist of an operation actor network (*i.e.*, operation selection policy network) and a machine actor network (*i.e.*, machine allocation policy network). These two policy networks are used to train the operation selection policy and the machine allocation policy, respectively. The actor networks are trained by minimizing the objective function $J_\pi\left(\phi_I\right)$:

$$J_\pi\left(\phi_I\right) = \mathbb{E}_{s_t\sim D,\epsilon_t\sim\mathcal{N}}\left[\alpha\log\pi_{\phi_I}\left(a_t^I \middle| s_t\right) - Q_\theta\left(s_t, a_t\right)\right], \tag{22}$$

where $I \in \{o, m\}$, $Q_\theta\left(s_t, a_t\right)$ is the estimated value of the Q-value function, ϕ_I denotes the network parameters of the policy π, D denotes the replay buffer, and ϵ_t is the input noise vector. ϵ_t is sampled from the fixed distribution $\mathcal{N}$. The gradient $\hat{\nabla}_{\phi_I} J_\pi\left(\phi_I\right)$ of $J_\pi\left(\phi_I\right)$ is calculated by

$$\hat{\nabla}_{\phi_I} J_\pi\left(\phi_I\right) = \nabla_{\phi_I}\log\pi_{\phi_I}\left(a_t^I \middle| s_t\right) + \left(\nabla_{a_t^I}\alpha\log\pi_{\phi_I}\left(a_t^I \middle| s_t\right) - \nabla_{a_t^I} Q_\theta\left(s_t, a_t\right)\right)\nabla_{\phi_I} a_t^I. \tag{23}$$

4.3.3. Updating the critic network

The critic network consists of a target Q-network and a soft Q-network. The target Q-network and the soft Q-network are two different Q-value function approximators that have the same network structure and initial parameters. The target Q-network is used to compute the target Q-value. The soft Q-network is used to compute the Q-value of a state–action pair under the current policy. The parameters of the policy networks are updated by calculating the error between the target Q-value and the current estimated Q-value. The critic network is updated by minimizing the loss function $J_Q(\theta)$. $J_Q(\theta)$ can be calculated by

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim D}\left[\frac{1}{2}\left(Q_\theta\left(s_t, a_t\right) - \hat{Q}\left(s_t, a_t\right)\right)^2\right], \tag{24}$$

with

$$\hat{Q}\left(s_t, a_t\right) = r_t + \gamma\mathbb{E}_{s_{t+1}\sim\mathcal{P}}\left[Q_{\bar{\theta}}\left(s_{t+1}, a_{t+1}\right) - \alpha\log\pi_{\phi_I}\left(a_{t+1}^I \middle| s_{t+1}\right)\right], \tag{25}$$

where θ denotes the parameter of the soft Q-network, which is obtained by training the soft Q-network. $\bar{\theta}$ denotes the parameter of the target Q-network, which is obtained by training the target Q-network. $Q_{\bar{\theta}}\left(s_{t+1}, a_{t+1}\right)$ is the target Q-value estimated by the target Q-network.

γ is the discount factor and $0 \leq \gamma \leq 1$. $\mathcal{P}$ denotes the probability of a state transition. The gradient $\hat{\nabla}_\theta J_Q(\theta)$ of $J_Q(\theta)$ is calculated by

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta\left(a_t, s_t\right)\left(Q_\theta\left(s_t, a_t\right) - \hat{Q}\left(s_t, a_t\right)\right). \tag{26}$$

Under the initial state, $\bar{\theta} = \theta$. Under the other state, $\bar{\theta}$ can be softly updated by

$$\bar{\theta} = \zeta\theta + (1-\zeta)\bar{\theta}, \tag{27}$$

where ζ is a weight parameter.

Algorithm 1 Training procedure for the soft double-actors critic algorithm

Input: The maximum number of iterations $\mathcal{J}$ and the parameters α, γ, and ζ.
Output: The trained operation selection policy network and machine allocation policy network.

```
Initialize θ, θ̄, D, s_t, and φ_I;
Get a batch of 𝒦 FJSP instances;
for  = 1 to 𝒥 do
  for  = 1 to 𝒦 do
    while s_t is not terminal do
      Get a_t^I according to π_{φ_I}(a_t^I | s_t), where I ∈ {o, m};
      Get the reward r_t and next state s_{t+1} by performing a_t,
      where a_t = {a_t^o, a_t^m};
      Store (s_t, a_t, r_t, s_{t+1}) into the replay buffer D;
      s_t ← s_{t+1};
    end while
    Get a batch of B samples from D;
    for i = 1 to B do
      Calculate J_Q(θ) by Eq. (24);
      Calculate the gradient of J_Q(θ) by Eq. (26);
      Update θ according to the gradient of J_Q(θ);
      Calculate J_π(φ_I) by Eq. (22);
      Calculate the gradient of J_π(φ_I) by Eq. (23);
      Update φ_I according to the gradient of J_π(φ_I);
      Update α by Eq. (21);
      Update θ̄ by Eq. (27);
    end for
  end for
end for
```

The training procedure for the proposed soft double-actors critic algorithm is shown in Algorithm 1. During each iteration, each FJSP instance will be solved through several time steps. At time step t, the operation selection policy network and the machine allocation policy network are used to select the actions a_t^o and a_t^m, respectively. The complete action a_t is performed to obtain the immediate reward r_t and transition from the current state s_t to the next state s_{t+1}. It takes several state transitions from the initial state to the final state, each state transition will generate a sample (s_t, a_t, r_t, s_{t+1}), and the sample will be stored in the replay buffer D. After a FJSP instance has been solved, a batch of samples are randomly sampled from D. These samples are used to calculate the gradients of the loss function $J_Q(\theta)$ and the objective function $J_\pi\left(\phi_I\right)$ to update the parameter θ of the soft Q-network and the parameters ϕ_I of the policy networks, respectively. Finally, the trained operation selection policy network and machine allocation policy network are obtained through several iterations.

5. Experiments

5.1. Experimental settings

The training set consists of randomly generated FJSP instances. The testing set consists of FJSP instances from the three well-known public benchmarks including Hurink [31], Behnke [32], and Brandimarte [33]. The Hurink dataset contains three sub-datasets, namely Edata, Rdata, and Vdata, where each sub-dataset contains 40 FJSP instances. Each job in each FJSP instance contains the same number of available machines as its operations, where the processing time of each operation ranges from 1 to 100. Moreover, in the Edata sub-dataset, when the total number of machines is ≤ 6, the maximum number of available machines that can be allocated to each operation is 2. When the total number of machines is more than 10, the maximum number of available machines that can be allocated to each operation is 3. In the Rdata sub-dataset, the maximum number of available machines that can be allocated to each operation is 3. In the Vdata sub-dataset, the maximum number of available machines that can be allocated to each operation is four-fifths of the total number of machines. The Behnke dataset contains three sub-datasets, namely, Behnke-m20, Behnke-m40, and Behnke-m60, where each sub-dataset contains 20 FJSP instances. Each job in each FJSP instance contains 5 operations, where the processing time of each operation ranges from 10 to 30. The Brandimarte dataset contains 15 FJSP instances. The number of operations contained in each job in each FJSP instance ranges from 3 to 14. The processing time of each operation ranges from 1 to 30. The number of available machines that can be allocated to each operation ranges from 2 to 6. The experiments are conducted on these three different public benchmarks to verify the efficiency, stability, and generalization of the proposed method. The settings of key parameters for policy training are given in Table 1.

Table 1
Parameter settings for policy training.

Parameter name	Parameter value
Number of MHGNN layers (L)	3
Optimizer	Adam
Learning rate	3×10^{-4}
Maximum number of samples stored in D	1000
Number of samples per batch ($\mathcal{K}$)	16
Entropy parameter (α)	ln 2
Discount factor (γ)	1
Soft update coefficient (ζ)	0.01
Maximum number of iterations ($\mathcal{J}$)	400

To better verify the effectiveness of the proposed method, for FJSP instances of different scales, the proposed method is compared with six commonly used composite scheduling rules including SPT+FIFO, SPT+MOPNR, SPT+MWKR, EET+FIFO, EET+MOPNR, and EET+MWKR. For convenience, the six composite scheduling rules are named Rule 1, Rule 2, Rule 3, Rule 4, Rule 5, and Rule 6, respectively. SPT (*i.e.*, shortest processing time) and EET (*i.e.*, earliest end time) are used as the machine allocation rules. FIFO (*i.e.*, first in first out), MOPNR (*i.e.*, most operation number remaining), and MWKR (*i.e.*, most work remaining) are used as the operation selection rules. Note that when solving the FJSP instances on the three public benchmarks, the scheduling performance is evaluated by calculating the percentage deviation (*i.e.*, gap) between the maximum completion time $C_{\max}$ and the best known solution UB [32]. The gap can be calculated by

$$\text{Gap} = \frac{C_{\max} - \text{UB}}{\text{UB}} \times 100\%. \tag{28}$$

In this experiment, the proposed method for solving FJSP is implemented with PyTorch 1.6 and Python 3.9. The experimental platform consists of an octa-core Intel Core i7-9700K CPU, a 2560-core NVIDIA GeForce RTX 2070 GPU, 64 GB of host memory, 8 GB of GPU memory, and the CentOS 8.1 operating system.

5.2. Convergence analysis for policy training

The six randomly generated FJSP instances of different scales are used for policy training, including 6×6, 10×5, 10×10, 15×15, 20×10, and 30×10. Fig. 8 presents the makespan convergence curves obtained from the policy training on FJSP instances of different scales. As seen in Fig. 8, the fluctuations of the makespan convergence curves become smaller after approximately 50 iterations, which shows that the policy training can quickly converge.

Table 2
Comparison of the solution quality obtained using the proposed method, the four better-performing composite scheduling rules, GIN-PPO, and GAT-PPO on Hurink dataset.

Size	Name	Rule 3		Rule 4		Rule 5		Rule 6		GIN-PPO [22]		GAT-PPO [25]		Proposed		UB
		C_{max}	Gap	C_{max}	Gap	C_{max}	Gap	C_{max}	Gap	C_{max}	Gap	C_{max}	Gap		C_{max}	Gap
15 × 5	1	1016	27.16%	867	8.51%	921	15.27%	868	8.64%	850	6.38%	**803**	**0.50%**	823	3.00%	799
15 × 5	2	1001	33.47%	854	13.87%	826	10.13%	840	12.00%	802	6.93%	**763**	**1.73%**	791	5.74%	750
15 × 5	3	1035	35.29%	856	11.90%	877	14.64%	909	18.82%	866	13.20%	784	2.48%	**768**	**0.39%**	765
15 × 5	4	1221	43.14%	895	4.92%	1005	17.82%	985	15.47%	**849**	**−0.47%**	875	8.83%	**849**	**−0.47%**	853
15 × 5	5	1035	28.73%	871	8.33%	901	12.06%	945	17.54%	850	5.72%	817	1.62%	**816**	**1.49%**	804
20 × 5	6	1478	38.00%	1100	2.71%	1130	5.51%	1139	6.35%	1109	3.55%	**1087**	**1.49%**	1100	2.71%	1071
20 × 5	7	1097	17.20%	986	5.34%	1018	8.76%	1083	15.71%	968	3.42%	**948**	**1.28%**	957	2.24%	936
20 × 5	8	1215	17.05%	1134	9.25%	1077	3.76%	1098	5.78%	1083	4.34%	**1051**	**1.25%**	1057	1.83%	1038
20 × 5	9	1284	20.00%	1181	10.37%	1131	5.70%	1176	9.91%	1112	3.93%	**1094**	**2.24%**	1097	2.52%	1070
20 × 5	10	1362	24.95%	1236	13.39%	1291	18.44%	1269	16.42%	1193	9.45%	1124	3.12%	**1087**	**−0.28%**	1090
20 × 10	11	1411	33.87%	1154	9.49%	1187	12.62%	1140	8.16%	1089	3.32%	**1070**	**1.52%**	1124	6.64%	1054
20 × 10	12	1654	52.44%	1128	3.96%	1231	13.46%	1196	10.23%	1123	3.41%	**1096**	**1.01%**	1121	3.32%	1085
20 × 10	13	1582	47.85%	1152	7.66%	1172	9.53%	1197	11.87%	1106	3.36%	**1086**	**1.50%**	1111	3.83%	1070
20 × 10	14	1589	59.86%	1079	8.55%	1105	11.17%	1124	13.08%	1049	5.43%	**1012**	**1.81%**	1074	8.05%	994
20 × 10	15	1639	53.32%	1150	7.58%	1170	9.45%	1146	7.20%	1117	4.20%	**1086**	**1.59%**	1128	5.52%	1069
30 × 10	16	2326	53.03%	1578	3.82%	1636	7.63%	1610	5.92%	1561	2.56%	**1534**	**0.92%**	1555	2.30%	1520
30 × 10	17	2564	54.64%	1709	3.08%	1738	4.83%	1754	5.79%	1693	1.93%	**1677**	**1.15%**	1689	1.87%	1658
30 × 10	18	2101	40.35%	1553	3.74%	1567	4.68%	1568	4.74%	1531	2.07%	**1511**	**0.94%**	1520	1.54%	1497
30 × 10	19	2033	32.44%	1575	2.61%	1615	5.21%	1607	4.69%	1562	1.63%	**1552**	**1.11%**	1561	1.69%	1535
30 × 10	20	2222	43.45%	1584	2.26%	1614	4.20%	1636	5.62%	1574	1.48%	1569	1.29%	**1565**	**1.03%**	1549
Ave. gap			37.81%		7.07%		9.74%		10.20%		4.30%		**1.87%**		2.73%	

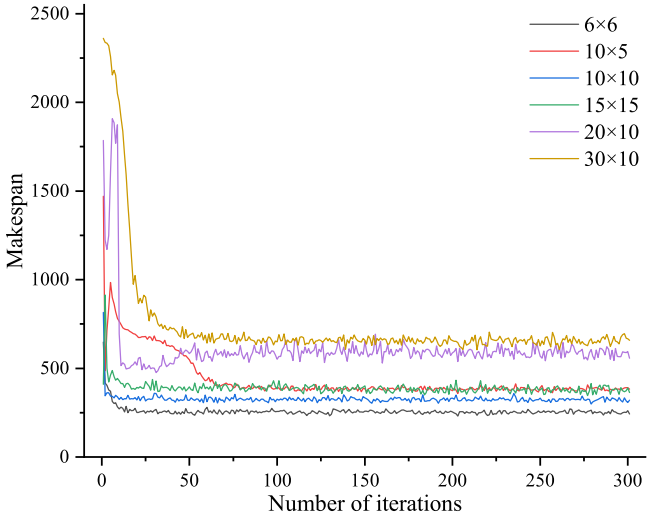


Fig. 8. Makespan convergence curves obtained from the policy training on FJSP instances of different scales.

Table 3
Comparison of the solution time obtained using the proposed method, the four better-performing composite scheduling rules, GIN-PPO, and GAT-PPO on Hurink dataset.

Size	Name	Rule 3 time (s)	Rule 4 time (s)	Rule 5 time (s)	Rule 6 time (s)	GIN-PPO [22] time (s)	GAT-PPO [25] time (s)	Proposed time (s)
15 × 5	1	0.12	0.12	0.14	0.12	0.42	1.19	0.31
15 × 5	2	0.14	0.13	0.13	0.14	0.42	1.18	0.32
15 × 5	3	0.14	0.14	0.15	0.15	0.42	1.17	0.31
15 × 5	4	0.12	0.12	0.13	0.14	0.41	1.19	0.31
15 × 5	5	0.12	0.12	0.13	0.12	0.42	1.19	0.33
20 × 5	6	0.19	0.16	0.17	0.16	0.63	1.67	0.45
20 × 5	7	0.17	0.16	0.17	0.17	0.61	1.69	0.43
20 × 5	8	0.17	0.20	0.19	0.19	0.61	1.69	0.42
20 × 5	9	0.19	0.16	0.19	0.17	0.64	1.67	0.42
20 × 5	10	0.17	0.15	0.17	0.19	0.61	1.69	0.42
20 × 10	11	0.36	0.32	0.36	0.33	1.23	4.84	0.88
20 × 10	12	0.31	0.29	0.35	0.30	1.25	4.85	0.89
20 × 10	13	0.30	0.32	0.33	0.32	1.25	4.83	0.87
20 × 10	14	0.31	0.29	0.33	0.30	1.24	4.86	0.89
20 × 10	15	0.32	0.30	0.35	0.30	1.25	4.83	0.87
30 × 10	16	0.53	0.52	0.55	0.52	1.83	9.91	1.39
30 × 10	17	0.52	0.47	0.54	0.49	1.82	9.90	1.41
30 × 10	18	0.50	0.49	0.52	0.50	1.82	9.91	1.42
30 × 10	19	0.56	0.51	0.58	0.54	1.82	9.90	1.41
30 × 10	20	0.49	0.47	0.52	0.49	1.83	9.89	1.39
Ave. time		0.29	0.27	0.30	0.28	1.03	4.40	0.76

5.3. Experimental results on public benchmarks

5.3.1. Experimental results on Hurink dataset

The ten FJSP instances are selected from each of the two sub-datasets (Rdata and Vdata) of Hurink dataset. Specifically, the five 15 × 5 and five 20 × 5 FJSP instances are selected from Rdata, and the five 20 × 10 and five 30 × 10 FJSP instances are selected from Vdata. In this experiment, the policies trained on the $15\times15, 20\times10$, and 30 × 10 random instances are used to solve the $15\times5, 20\times5$ and 20 × 10, and 30 × 10 benchmark instances, respectively. To ensure fair comparison, the four better-performing composite scheduling rules are selected from the six composite scheduling rules, *i.e.*, Rule 3, Rule 4, Rule 5, and Rule 6. In addition, the two state-of-the-art methods that combine GNN and DRL, including GIN-PPO [22] and GAT-PPO [25], are adopted to solve FJSP. Notably, the best-performing model trained by GIN-PPO and that trained by GAT-PPO on Hurink dataset are adopted to solve FJSP. Tables 2 and 3 present the comparisons of the solution quality and the solution time obtained using the proposed method, the four better-performing composite scheduling rules, GIN-PPO, and GAT-PPO on Hurink dataset, respectively.

As shown in Table 2, in terms of solution quality, the proposed method is significantly superior to these composite scheduling rules on the vast majority of instances, the differences between the proposed method and these composite scheduling rules are small on individual instances. The solution quality of the proposed method is superior to that of GIN-PPO in all instances except for the third instance. The solution quality of the proposed method and that of GAT-PPO are similar on most instances. The C_{max} of the proposed method outperforms UB on a few instances. Fig. 9 presents the relative gaps of the proposed method, the four better-performing composite scheduling rules, GIN-PPO and GAT-PPO to UB on Hurink dataset. As seen in Fig. 9, the average gaps of the best-performing composite scheduling rule (*i.e.*, Rule 4), GIN-PPO, GAT-PPO, and the proposed method are 7.07%, 4.30%, 1.87%, and 2.73%, respectively, which demonstrates that the overall solution quality of the proposed method is far better than that of these composite scheduling rules and GIN-PPO and slightly inferior to that of GAT-PPO. As shown in Table 3, the solution time of the best-performing composite scheduling rule (*i.e.*, Rule 4) ranges from 0.12 to 0.52 s, the solution time of GIN-PPO ranges from 0.41 to 1.83 s, the solution time of GAT-PPO ranges from 1.17 to 9.91 s, and the solution time of the proposed method ranges from 0.31 to 1.42 s. From a comprehensive perspective of solution quality and solution time, the

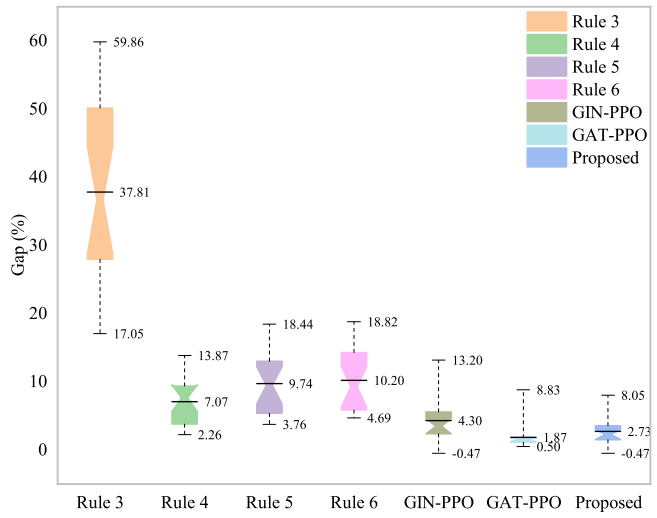


Fig. 9. Relative gaps of the proposed method, the four better-performing composite scheduling rules, GIN-PPO, and GAT-PPO to UB on Hurink dataset.

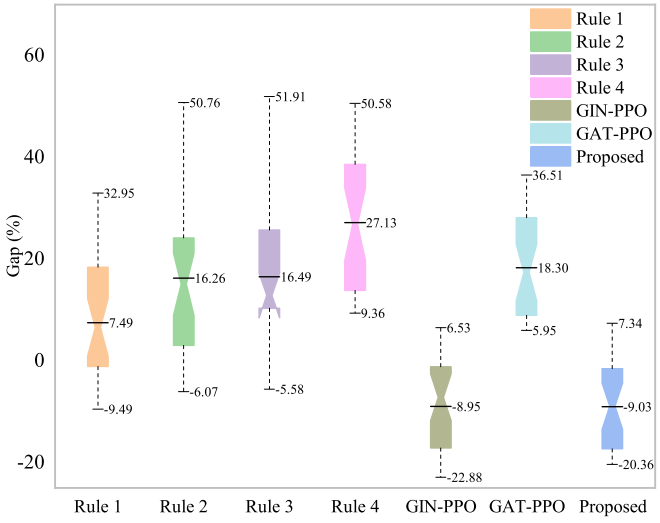


Fig. 10. Relative gaps of the proposed method, the four better-performing composite scheduling rules, GIN-PPO, and GAT-PPO to UB on Behnke dataset.

proposed method is far superior to these composite scheduling rules, superior to GIN-PPO, and slightly better than GAT-PPO.

5.3.2. Experimental results on Behnke dataset

The ten large-scale FJSP instances are selected from each of the three sub-datasets (Behnke-m20, Behnke-m40, and Behnke-m60) of Behnke dataset. In this experiment, the policies trained on the 30 × 10 random instances are used to solve these 30 benchmark instances. To ensure fair comparison, the four better-performing composite scheduling rules are selected from the six composite scheduling rules, *i.e.*, Rule 1, Rule 2, Rule 3, and Rule 4, and the best-performing model trained by GIN-PPO and that trained by GAT-PPO on Behnke dataset are adopted to solve FJSP. Tables 4 and 5 provide the comparisons of the solution quality and the solution time obtained using the proposed method, the four better-performing composite scheduling rules, GIN-PPO, and GAT-PPO on Behnke dataset, respectively.

As shown in Table 4, in terms of solution quality, the proposed method is significantly superior to these composite scheduling rules and GAT-PPO on all instances and superior to GIN-PPO on most instances. In addition, the C_{max} of the proposed method outperforms UB on most large-scale instances. Fig. 10 shows the relative gaps of the proposed method, the four better-performing composite scheduling rules, GIN-PPO, and GAT-PPO to UB on Behnke dataset. As shown in Fig. 10, the average gaps of the best-performing composite scheduling rule (*i.e.*, Rule 1), GIN-PPO, and GAT-PPO are 7.49%, −8.95%, and 18.30% respectively, while the average gap of the proposed method is −9.03%, which shows that the overall solution quality of the proposed method is obviously better than that of these composite scheduling rules and GAT-PPO and slightly better than that of GIN-PPO. As shown in Table 5, the solution time of the best-performing composite scheduling rule (*i.e.*, Rule 1) ranges from 0.59 to 2.13 s, the solution time of GIN-PPO ranges from 1.99 to 9.11 s, the solution time of GAT-PPO ranges from 7.95 to 42.35 s, and the solution time of the proposed method ranges from 1.69 to 8.38 s. From a comprehensive perspective of solution quality and solution time, the proposed method is significantly superior to these composite scheduling rules and GAT-PPO and superior to GIN-PPO for large-scale FJSPs.

Figs. 11(a) and 11(b) present the Gantt charts of the final scheduling results of the best-performing scheduling rule (*i.e.*, Rule 1) and the proposed method on the 17-th instance (100 × 20), respectively. As seen in Figs. 11(a) and 11(b), the idle time in the Gantt chart of the scheduling result of the proposed method is significantly less than that of Rule 1, which demonstrates that the proposed method obtains a better solution quality compared with the best-performing scheduling rule.

5.3.3. Experimental results on Brandimarte dataset

The first ten FJSP instances are selected from Brandimarte dataset. In this experiment, the policies trained on the 6×6, 15×15, 10×10, 20×10, and 30 × 10 random instances are used to solve the 10×6, 15×8 and 15×4, 10 × 10, 20×5 and 20 × 10, and 20 × 15 benchmark instances, respectively. The proposed method is compared with Rule 4, GIN-A3C [20], GIN-PPO [22], GAT-PPO [25], IPSO [34], and SLGA [35] in this experiment. Rule 4 is the best-performing scheduling rule among the six composite scheduling rules on Brandimarte dataset. IPSO is one of the most widely used and best-performing meta-heuristic algorithms for solving FJSP in recent years. GIN-A3C is one of the latest methods to solve FJSP by combining GNN and DRL. SLGA is one of the best-performing methods to solve FJSP by combining meta-heuristic algorithms and RL recently. To ensure fair comparison, the fine-tuned IPSO and SLGA and the best-performing model trained by GIN-A3C, GIN-PPO, and GAT-PPO respectively on Brandimarte dataset are adopted to solve FJSP.

Tables 6 and 7 give the comparisons of the solution quality and the solution time obtained using the proposed method, Rule 4, GIN-A3C, GIN-PPO, GAT-PPO, IPSO, and SLGA on Brandimarte dataset, respectively. Fig. 12 presents the relative gaps of Rule 4, GIN-A3C, GIN-PPO, GAT-PPO, IPSO, SLGA, and the proposed method to UB on Brandimarte dataset. As shown in Table 6, in terms of solution quality, the proposed method outperforms Rule 4 and GIN-A3C on all instances, outperforms GIN-PPO on all instances except for the MK03 instance, outperforms GAT-PPO and IPSO on most instances, and has similar performance as SLGA on some instances. As seen in Fig. 12, the average gaps of Rule 4, GIN-A3C, GIN-PPO, GAT-PPO, IPSO, SLGA, and the proposed method are 29.07%, 23.63%, 14.02%, 18.75%, 8.85%, 6.21%, and 6.15%, respectively, which indicates that the overall solution quality of the proposed method is significantly better than that of Rule 4, GIN-A3C, GIN-PPO, and GAT-PPO, better than that of IPSO, and slightly better than that of SLGA. As shown in Table 7, the solution time of Rule 4 ranges from 0.1 to 0.37 s, the solution time of GIN-A3C ranges from 0.36 to 1.94 s, the solution time of GIN-PPO ranges from 0.43 to 1.56 s, the solution time of GAT-PPO ranges from 0.84 to 7.16 s, the solution time of IPSO ranges from 34.7 to 2675 s, the solution time of SLGA ranges from 27.63 to 1335 s, and the solution time of the proposed method ranges from 0.24 to 1.15 s. The results reveal that

Table 4
Comparison of the solution quality obtained using the proposed method, the four better-performing composite scheduling rules, GIN-PPO, and GAT-PPO on Behnke dataset.

Size	Name	Rule 1		Rule 2		Rule 3		Rule 4		GIN-PPO [22]		GAT-PPO [25]		Proposed		UB
		C_{max}	Gap	C_{max}	Gap	C_{max}	Gap	C_{max}	Gap	C_{max}	Gap	C_{max}	Gap	C_{max}	Gap	
50 × 20	1	313	20.85%	348	34.36%	338	30.50%	356	37.45%	**250**	**−3.47%**	342	32.05%	278	7.34%	259
50 × 20	2	281	11.59%	317	26.29%	342	36.25%	357	42.23%	**247**	**−1.59%**	326	29.88%	**247**	**−1.59%**	251
50 × 20	3	322	27.78%	358	42.06%	366	45.24%	359	42.46%	**249**	**−1.19%**	344	36.51%	261	3.57%	252
50 × 20	4	343	32.95%	361	39.92%	326	26.36%	362	40.31%	**257**	**−0.39%**	337	30.62%	262	1.55%	258
50 × 20	5	328	25.19%	395	50.76%	398	51.91%	361	37.79%	**255**	**−2.67%**	327	24.81%	260	−0.76%	262
50 × 40	6	322	18.38%	322	18.38%	307	12.87%	375	37.87%	271	−0.37%	333	22.43%	**264**	**−2.94%**	272
50 × 40	7	297	14.67%	308	18.92%	319	23.17%	390	50.58%	267	3.09%	324	25.10%	**255**	**−1.54%**	259
50 × 40	8	302	23.27%	295	20.41%	308	25.71%	353	44.08%	261	6.53%	321	31.02%	**253**	**3.27%**	245
50 × 40	9	280	5.66%	326	23.02%	299	12.83%	348	31.32%	**250**	**−5.66%**	325	22.64%	262	−1.13%	265
50 × 40	10	279	10.28%	311	22.92%	289	14.23%	355	40.32%	**249**	**−1.58%**	332	31.23%	252	−0.40%	253
50 × 60	11	262	1.16%	288	11.20%	286	10.42%	359	38.61%	253	−2.32%	319	23.17%	**245**	**−5.41%**	259
50 × 60	12	274	7.45%	288	12.94%	317	24.31%	346	35.69%	242	−5.10%	328	28.63%	**240**	**−5.88%**	255
50 × 60	13	293	14.01%	319	24.12%	295	14.79%	352	36.96%	256	−0.39%	324	26.07%	**244**	**−5.06%**	257
50 × 60	14	320	19.85%	353	32.21%	367	37.45%	368	37.83%	268	0.37%	331	23.97%	**258**	**−3.37%**	267
50 × 60	15	307	19.92%	317	23.83%	344	34.38%	357	39.45%	262	2.34%	328	28.13%	**248**	**−3.13%**	256
100 × 20	16	540	−4.59%	583	3.00%	625	10.42%	619	9.36%	**437**	**−22.79%**	603	6.54%	451	−20.32%	566
100 × 20	17	577	7.85%	683	27.66%	603	12.71%	624	16.64%	**430**	**−19.63%**	598	11.78%	438	−18.13%	535
100 × 20	18	514	−7.39%	630	13.51%	621	11.89%	621	11.89%	**428**	**−22.88%**	598	7.75%	442	−20.36%	555
100 × 20	19	506	−4.89%	585	9.96%	637	19.74%	626	17.67%	**423**	**−20.49%**	591	11.09%	447	−15.98%	532
100 × 20	20	516	−1.15%	594	13.79%	576	10.34%	628	20.31%	**427**	**−18.20%**	586	12.26%	444	−14.94%	522
100 × 40	21	533	0.38%	535	0.75%	520	−2.07%	602	13.37%	**442**	**−16.76%**	564	6.21%	446	−16.01%	531
100 × 40	22	492	−8.21%	559	4.29%	587	9.51%	610	13.81%	444	−17.16%	578	7.84%	**435**	**−18.84%**	536
100 × 40	23	477	−9.49%	495	−6.07%	518	−1.71%	610	15.75%	454	−13.85%	577	9.49%	**434**	**−17.65%**	527
100 × 40	24	526	1.94%	600	16.28%	580	12.40%	631	22.29%	471	−8.72%	562	8.91%	**454**	**−12.02%**	516
100 × 40	25	567	8.83%	564	8.25%	579	11.13%	596	14.40%	460	−11.71%	580	11.32%	**456**	**−12.48%**	521
100 × 60	26	511	−5.02%	529	−1.67%	508	−5.58%	594	10.41%	**439**	**−18.40%**	570	5.95%	442	−17.84%	538
100 × 60	27	535	0.00%	531	−0.75%	614	14.77%	600	12.15%	442	−17.38%	574	7.29%	**433**	**−19.07%**	535
100 × 60	28	489	−7.91%	544	2.45%	522	−1.69%	600	12.99%	442	−16.76%	585	10.17%	**439**	**−17.33%**	531
100 × 60	29	526	−1.15%	529	−0.56%	511	−3.95%	619	16.35%	443	−16.73%	571	7.33%	**441**	**−17.11%**	532
100 × 60	30	548	2.04%	513	−4.47%	518	−3.54%	609	13.41%	458	−14.71%	585	8.94%	**444**	**−17.32%**	537
Ave. gap			7.49%		16.26%		16.49%		27.13%		**8.95%**		18.30%		9.03%	

Table 5
Comparison of the solution time obtained using the proposed method, the four better-performing composite scheduling rules, GIN-PPO, and GAT-PPO on Behnke dataset.

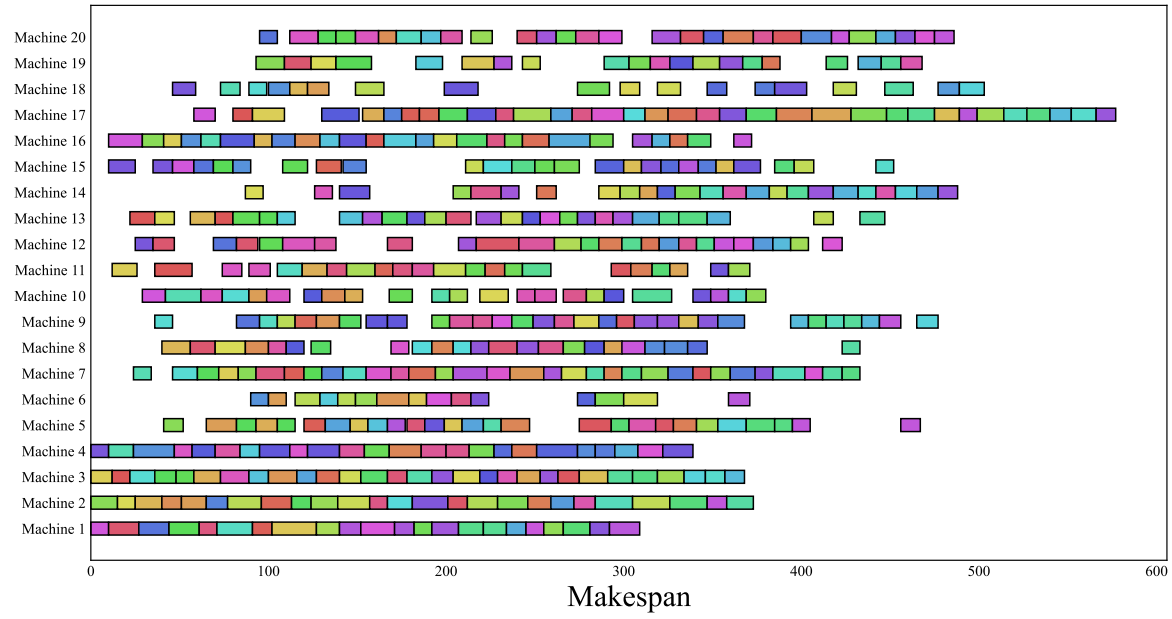
Size	Name	Rule 1 time (s)	Rule 2 time (s)	Rule 3 time (s)	Rule 4 time (s)	GIN-PPO [22] time (s)	GAT-PPO [25] time (s)	Proposed time (s)
50 × 20	1	0.67	0.60	0.56	0.61	2.02	7.98	1.81
50 × 20	2	0.61	0.59	0.54	0.61	2.02	7.98	1.82
50 × 20	3	0.60	0.57	0.54	0.59	2.01	7.95	1.77
50 × 20	4	0.59	0.59	0.52	0.59	2.03	7.97	1.69
50 × 20	5	0.64	0.61	0.58	0.66	1.99	7.98	1.76
50 × 40	6	0.67	0.64	0.61	0.68	2.11	8.97	1.90
50 × 40	7	0.68	0.59	0.57	0.63	2.12	8.99	1.87
50 × 40	8	0.62	0.59	0.58	0.60	2.12	8.99	1.83
50 × 40	9	0.64	0.61	0.59	0.64	2.11	8.99	1.91
50 × 40	10	0.67	0.63	0.60	0.68	2.11	8.99	1.89
50 × 60	11	0.70	0.65	0.63	0.70	2.25	9.95	2.00
50 × 60	12	0.66	0.62	0.58	0.67	2.25	9.96	2.01
50 × 60	13	0.67	0.62	0.61	0.67	2.21	9.94	2.02
50 × 60	14	0.71	0.68	0.64	0.68	2.24	9.97	1.97
50 × 60	15	0.64	0.64	0.58	0.66	2.20	9.98	2.02
100 × 20	16	2.02	1.74	1.52	1.95	8.54	34.25	8.04
100 × 20	17	1.91	1.70	1.47	1.87	8.51	34.35	8.09
100 × 20	18	1.98	1.72	1.48	1.96	8.55	34.29	7.93
100 × 20	19	2.04	1.67	1.47	1.87	8.55	34.31	7.83
100 × 20	20	2.13	1.72	1.54	1.97	8.59	34.34	7.98
100 × 40	21	2.04	1.77	1.67	2.02	8.93	38.38	8.10
100 × 40	22	1.93	1.73	1.58	1.96	8.96	38.42	7.84
100 × 40	23	1.97	1.75	1.61	1.95	8.96	38.35	8.01
100 × 40	24	1.96	1.71	1.56	1.96	8.89	38.34	7.84
100 × 40	25	1.91	1.76	1.64	1.96	8.91	38.41	7.86
100 × 60	26	2.07	1.90	1.71	2.02	9.03	42.35	8.12
100 × 60	27	1.99	1.87	1.66	2.00	9.09	42.31	8.10
100 × 60	28	1.95	1.87	1.66	2.00	9.11	42.30	8.12
100 × 60	29	1.95	1.91	1.70	1.99	9.08	42.30	8.05
100 × 60	30	1.97	1.86	1.63	1.99	9.08	42.32	8.38
Ave. time		1.32	1.20	1.09	1.30	5.49	23.65	4.95

the proposed method is significantly superior to Rule 4, GIN-A3C, GIN-PPO, GAT-PPO, IPSO, and SLGA from a comprehensive perspective of solution quality and solution time.

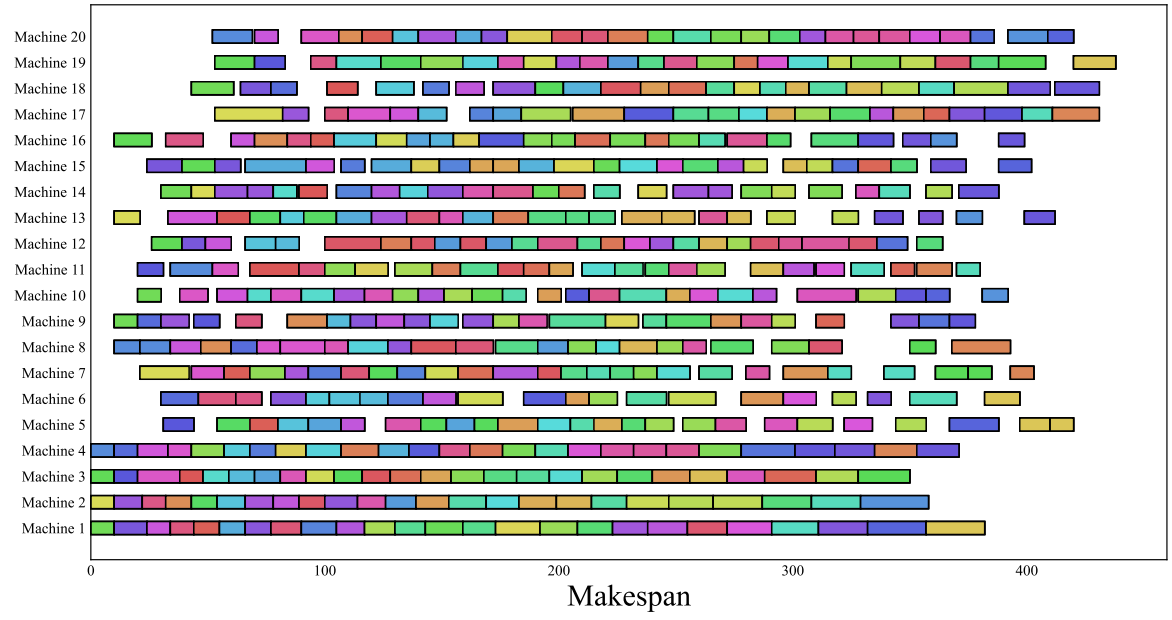
Overall, the efficiency, stability, and generalization of the proposed method have been verified through extensive experiments on three different public benchmarks. Firstly, compared with some advanced methods for solving FJSP, the overall performance of the proposed method is better than that of them in terms of both solution quality and solution time, which verifies the efficiency of the proposed method. Secondly, the performance of the same scheduling rule varies across different public benchmarks, meaning that the stability of the scheduling rules is poor. In contrast, the proposed method performs substantially well on different public benchmarks, meaning that the proposed method has better stability. Finally, the policies trained using the proposed method can be directly used to efficiently solve FJSP instances of different scales, which demonstrates that the proposed method has good generalization.

5.4. Parameter sensitivity analysis

The three most important parameters used for policy training of the proposed method including the learning rate, the soft update coefficient, and the number of MHGNN layers are analyzed, as shown in Fig. 13. The values of these parameters are determined by comprehensively analyzing their influences on makespan and convergence during the policy training process. As shown in Fig. 13(a), when the learning rate is set to 0.01, the makespan fluctuates significantly and it fails to obtain a satisfactory makespan during the policy training process. With the decrease in the learning rate, the makespan gradually stabilizes and a smaller makespan can be obtained, thus the learning rate is chosen as 0.0003. Figs. 13(b) and 13(c) present the makespan obtained under different soft update coefficients and different numbers of MHGNN layers during the policy training process, respectively. It is easy to see that a satisfactory makespan and good convergence can be



(a) Gantt chart of the best-performing composite scheduling rule (i.e., Rule 1)



(b) Gantt chart of the proposed method

Fig. 11. Gantt charts of the final scheduling results obtained with different methods on the 17-th instance (100 × 20).

achieved when the soft update coefficient and the number of MHGNN layers are chosen as 0.01 and 3, respectively.

In addition, how to reasonably determine the values of the other three key parameters, including the number of samples per batch (*i.e.*, batch size), the discount factor, and the initial value of the entropy parameter, is also important for policy training. A larger batch size can improve the convergence speed of policy training, but it is easy to cause out-of-memory. A smaller batch size can improve the generalization effect of the trained policy, but it is easy to cause instability in the training process. Therefore, the number of samples per batch is determined to be 16. Owing to the fact that the agent needs to simultaneously control the operation selection action and the machine allocation action at each time step, the initial value of the entropy parameter is determined by the natural logarithm of the dimension of the action space, *i.e.*, ln 2. The discount factor is determined to be 1 for better calculation of the cumulative reward.

6. Conclusions

A new end-to-end DRL method combined with MHGNN is put forward to efficiently solve FJSP. To better solve complex FJSP, the task of solving FJSP is decomposed into two subtasks of operation selection

Table 6
Comparison of the solution quality obtained using the proposed method, Rule 4, GIN-A3C, GIN-PPO, GAT-PPO, IPSO, and SLGA on Brandimarte dataset.

Size	Name	Rule 4		GIN-A3C [20]		GIN-PPO [22]		GAT-PPO [25]		IPSO [34]		SLGA [35]		Proposed		UB
		C_{max}	Gap	C_{max}	Gap	C_{max}	Gap	C_{max}	Gap	C_{max}	Gap	C_{max}	Gap	C_{max}	Gap	
10 × 6	MK01	47	20.51%	48	23.08%	44	12.82%	42	7.69%	**40**	**2.56%**	**40**	**2.56%**	41	5.13%	39
10 × 6	MK02	39	50.00%	34	30.77%	32	23.08%	36	38.46%	29	11.54%	**27**	**3.85%**	**27**	**3.85%**	26
15 × 8	MK03	216	5.88%	235	15.20%	**204**	**0.00%**	**204**	**0.00%**	**204**	**0.00%**	**204**	**0.00%**	**204**	**0.00%**	204
15 × 8	MK04	76	26.67%	77	28.33%	70	16.67%	67	11.67%	66	10.00%	**60**	**0.00%**	64	6.67%	60
15 × 4	MK05	190	10.47%	192	11.63%	182	5.81%	181	5.23%	175	1.74%	**172**	**0.00%**	173	0.58%	172
10 × 10	MK06	99	70.69%	78	34.48%	78	34.48%	88	51.72%	77	32.76%	**69**	**18.97%**	**69**	**18.97%**	58
20 × 5	MK07	207	48.92%	190	36.69%	157	12.95%	194	39.57%	145	4.32%	**144**	**3.60%**	146	5.04%	139
20 × 10	MK08	539	3.06%	544	4.02%	531	1.53%	**523**	**0.00%**	**523**	**0.00%**	**523**	**0.00%**	527	0.76%	523
20 × 10	MK09	348	13.36%	375	22.15%	331	7.82%	322	4.89%	**320**	**4.23%**	**320**	**4.23%**	331	7.82%	307
20 × 15	MK10	278	41.12%	256	29.95%	247	25.38%	246	24.87%	239	21.32%	254	28.93%	**222**	**12.69%**	197
Ave. gap			29.07%		23.63%		14.02%		18.75%		8.85%		6.21%		**6.15%**	

Table 7
Comparison of the solution time obtained using the proposed method, Rule 4, GIN-A3C, GIN-PPO, GAT-PPO, IPSO, and SLGA on Brandimarte dataset.

Size	Name	Rule 4 time (s)	GIN-A3C [20] time (s)	GIN-PPO [22] time (s)	GAT-PPO [25] time (s)	IPSO [34] time (s)	SLGA [35] time (s)	Proposed time (s)
10 × 6	MK01	0.25	0.83	0.43	0.84	34.70	27.63	0.25
10 × 6	MK02	0.10	0.36	0.44	0.86	50.00	29.11	0.24
15 × 8	MK03	0.22	0.98	0.85	3.08	562.50	112.60	0.67
15 × 8	MK04	0.14	0.83	0.62	1.53	135.00	63.21	0.41
15 × 4	MK05	0.16	0.82	0.65	1.91	131.50	60.35	0.48
10 × 10	MK06	0.20	1.00	0.83	3.13	797.80	72.80	0.64
20 × 5	MK07	0.19	0.62	0.55	1.66	160.60	57.77	0.45
20 × 10	MK08	0.34	1.74	1.31	6.31	944.80	521.70	1.05
20 × 10	MK09	0.35	1.94	1.43	7.00	1762.00	552.50	1.14
20 × 15	MK10	0.37	1.83	1.56	7.16	2675.00	1335.00	1.15
Ave. time		0.23	1.10	0.87	3.35	725.39	283.27	0.65

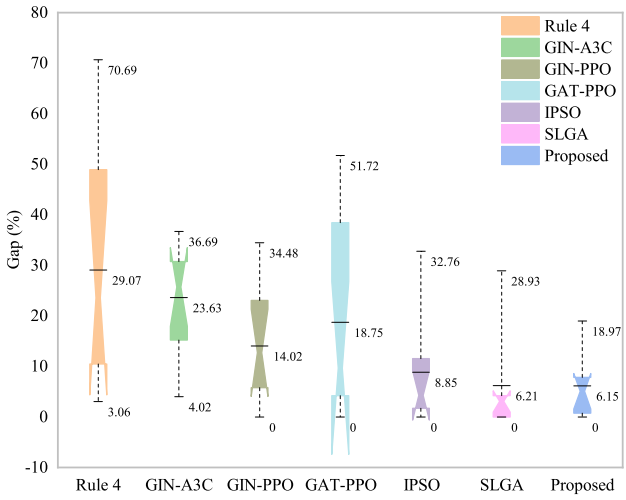


Fig. 12. Relative gaps of Rule 4, GIN-A3C, GIN-PPO, GAT-PPO, IPSO, SLGA, and the proposed method to UB on Brandimarte dataset.

and machine allocation. The heterogeneous graph is adopted to represent the global scheduling states of the dual-task FJSP, and the DMDP is proposed to model the dual-task FJSP, which can more accurately model complex FJSP. By utilizing the advantage of GNN in processing complicated graph structural information, the MHGNN is proposed to effectively obtain the node features of operation nodes and machine nodes and the relationships between them in the heterogeneous graph. The heterogeneous GNN framework is designed to effectively extract the feature information of the embedded operation nodes and machine nodes, which are used to represent the operation selection policy and the machine allocation policy. The soft double-actors critic algorithm is proposed to train high-quality scheduling policies that can be used to efficiently solve FJSP instances of different scales. Compared with the existing methods for solving FJSP, the proposed method improves the modeling capability of complex FJSP, enhances the stability and generalization of scheduling policies, and improves the quality and efficiency of solving FJSP. Extensive experiments are conducted on three different public benchmarks, and the experimental results verify the efficiency, stability, and generalization of the proposed method.

The dynamic events such as machine breakdowns and new job insertions occur frequently in intelligent manufacturing. How to efficiently solve DFJSP is a more challenging problem. The effective solution of DFJSP usually requires real-time decision-making of production scheduling to adapt to dynamically changing tasks and resources, and often requires consideration of multiple optimization objectives, such as minimizing the makespan and maximizing the machine utilization. Therefore, how to extend the proposed method to solve the multi-objective DFJSP will be the focus of future research.

CRediT authorship contribution statement

Lanjun Wan: Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Funding acquisition, Conceptualization. **Long Fu:** Writing – original draft, Visualization, Validation, Software, Methodology, Data curation. **Changyun Li:** Supervision, Funding acquisition. **Keqin Li:** Writing – review & editing, Methodology, Formal analysis.

Declaration of competing interest

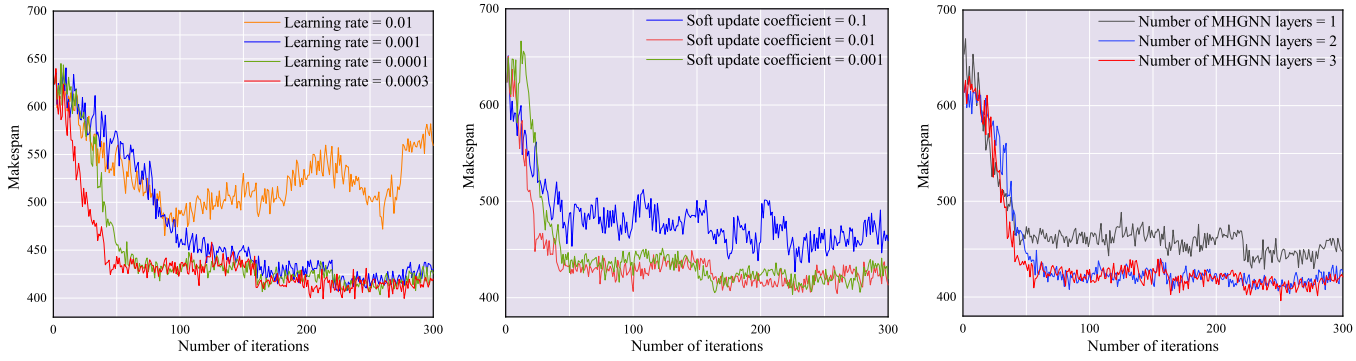
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported by the Scientific Research Foundation of Hunan Provincial Education Department, China [grant number 21B0547]; the Hunan Provincial Natural Science Foundation of China [grant number 2023JJ30217]; the National Natural Science Foundation for Young Scientists of China [grant number 61702177]; and the Research Foundation of Education Bureau of Hunan Province, China [grant number 21A0356].



(a) Learning rate (b) Soft update coefficient (c) Number of MHGNN layers

Fig. 13. Sensitivity analyses of three different key parameters during the policy training process.

References

[1] X. Li, X. Guo, H. Tang, R. Wu, L. Wang, S. Pang, Z. Liu, W. Xu, X. Li, Survey of integrated flexible job shop scheduling problems, Comput. Ind. Eng. (2022) 108786.

[2] S. Dauzère-Pérès, J. Ding, L. Shen, K. Tamssaouet, The flexible job shop scheduling problem: A review, European J. Oper. Res. 314 (2) (2024) 409–432.

[3] Y. Demir, S.K. İşleyen, Evaluation of mathematical models for flexible job-shop scheduling problems, Appl. Math. Model. 37 (3) (2013) 977–988.

[4] S.S. Gran, I. Ismail, T.A. Ajol, A.F.A. Ibrahim, Mixed integer programming model for flexible job-shop scheduling problem (FJSP) to minimize makespan and total machining time, in: 2015 Int. Conf. Comput. Commun. Control. Technol., I4CT, IEEE, 2015, pp. 413–417.

[5] L. Meng, C. Zhang, Y. Ren, B. Zhang, C. Lv, Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem, Comput. Ind. Eng. 142 (2020) 106347.

[6] F. Zhang, Y. Mei, S. Nguyen, M. Zhang, Importance-aware genetic programming for automated scheduling heuristics learning in dynamic flexible job shop scheduling, in: 17th Int. Conf. Parallel Probl. Solving Nat., PPSN, Springer, 2022, pp. 48–62.

[7] Y. Ai, M. Wang, X. Xue, C.-B. Yan, An efficient heuristic algorithm for flexible job-shop scheduling problem with due windows, in: 2022 IEEE 18th Int. Conf. Autom. Sci. Eng., CASE, IEEE, 2022, pp. 142–147.

[8] H. Zhang, G. Xu, R. Pan, H. Ge, A novel heuristic method for the energy-efficient flexible job-shop scheduling problem with sequence-dependent set-up and transportation time, Eng. Optim. 54 (10) (2022) 1646–1667.

[9] D. Yang, M. Wu, D. Li, Y. Xu, X. Zhou, Z. Yang, Dynamic opposite learning enhanced dragonfly algorithm for solving large-scale flexible job shop scheduling problem, Knowl.-Based Syst. 238 (2022) 107815.

[10] K. Sun, D. Zheng, H. Song, Z. Cheng, X. Lang, W. Yuan, J. Wang, Hybrid genetic algorithm with variable neighborhood search for flexible job shop scheduling problem in a machining system, Expert Syst. Appl. 215 (2023) 119359.

[11] L. Wei, J. He, Z. Guo, Z. Hu, A multi-objective migrating birds optimization algorithm based on game theory for dynamic flexible job shop scheduling problem, Expert Syst. Appl. 227 (2023) 120268.

[12] S. Luo, Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning, Appl. Soft Comput. 91 (2020) 106208.

[13] Y. Feng, L. Zhang, Z. Yang, Y. Guo, D. Yang, Flexible job shop scheduling based on deep reinforcement learning, in: 2021 5th Asian Conf. Artif. Intell. Technol., ACAIT, IEEE, 2021, pp. 660–666.

[14] S. Luo, L. Zhang, Y. Fan, Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning, Comput. Ind. Eng. 159 (2021) 107489.

[15] R. Liu, R. Piplani, C. Toro, Deep reinforcement learning for dynamic scheduling of a flexible job shop, Int. J. Prod. Res. 60 (13) (2022) 4049–4069.

[16] M. Panzer, B. Bender, Deep reinforcement learning in production systems: A systematic literature review, Int. J. Prod. Res. 60 (13) (2022) 4316–4341.

[17] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.

[18] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2018.

[19] S. Munikoti, D. Agarwal, L. Das, M. Halappanavar, B. Natarajan, Challenges and opportunities in deep reinforcement learning with graph neural networks: A comprehensive review of algorithms and applications, IEEE Trans. Neural Netw. Learn. Syst. (2023) 1–21, http://dx.doi.org/10.1109/TNNLS.2023.3283523.

[20] Z. Zeng, X. Li, C. Bai, A deep reinforcement learning approach to flexible job shop scheduling, in: 2022 IEEE Int. Conf. Syst. Man. Cybern., SMC, IEEE, 2022, pp. 884–890.

[21] K. Lei, P. Guo, Y. Wang, J. Xiong, W. Zhao, An end-to-end hierarchical reinforcement learning framework for large-scale dynamic flexible job-shop scheduling problem, in: 2022 Int. Jt. Conf. Neural Netw., IJCNN, IEEE, 2022, pp. 1–8.

[22] K. Lei, P. Guo, W. Zhao, Y. Wang, L. Qian, X. Meng, L. Tang, A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem, Expert Syst. Appl. 205 (2022) 117796.

[23] K. Lei, P. Guo, Y. Wang, J. Zhang, X. Meng, L. Qian, Large-scale dynamic scheduling for flexible job-shop with random arrivals of new jobs by hierarchical reinforcement learning, IEEE Trans. Ind. Inform. 20 (1) (2024) 1007–1018.

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

[25] W. Song, X. Chen, Q. Li, Z. Cao, Flexible job-shop scheduling via graph neural network and deep reinforcement learning, IEEE Trans. Ind. Inform. 19 (2) (2023) 1600–1610.

[26] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks? 2019, arXiv preprint arXiv:1810.00826.

[27] C. Zhang, D. Song, C. Huang, A. Swami, N.V. Chawla, Heterogeneous graph neural network, in: Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min., KDD, 2019, pp. 793–803.

[28] X. Fu, J. Zhang, Z. Meng, I. King, MAGNN: Metapath aggregated graph neural network for heterogeneous graph embedding, in: Proc. Web Conf., WWW, 2020, pp. 2331–2341.

[29] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, S. Levine, Soft actor-critic algorithms and applications, 2019, arXiv preprint arXiv:1812.05905.

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017, arXiv preprint arXiv:1707.06347.

[31] J. Hurink, B. Jurisch, M. Thole, Tabu search for the job-shop scheduling problem with multi-purpose machines, OR Spectrum 15 (1994) 205–215.

[32] D. Behnke, M.J. Geiger, Test Instances for the Flexible Job Shop Scheduling Problem with Work Centers, Tech. Rep. RR-12-01-01, Helmut Schmidt Univ., Hamburg, Germany, 2012.

[33] P. Brandimarte, Routing and scheduling in a flexible job shop by tabu search, Ann. Oper. Res. 41 (3) (1993) 157–183.

[34] H. Ding, X. Gu, Improved particle swarm optimization algorithm based novel encoding and decoding schemes for flexible job shop scheduling problem, Comput. Oper. Res. 121 (2020) 104951.

[35] R. Chen, B. Yang, S. Li, S. Wang, A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem, Comput. Ind. Eng. 149 (2020) 106778.