Expert Systems With Applications 205 (2022) 117796

Contents lists available at ScienceDirect

Expert Systems With Applications

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

A multi-action deep reinforcement learning framework for flexible Job-shop scheduling problem

Kun Lei [a], Peng Guo [a,b,*], Wenchao Zhao [a], Yi Wang [c], Linmao Qian [a], Xiangyin Meng [a], Liansheng Tang [d]

[a] School of Mechanical Engineering, Southwest Jiaotong University, Chengdu 610031 China
[b] Technology and Equipment of Rail Transit Operation and Maintenance Key Laboratory of Sichuan Province, Chengdu 610031 China
[c] Department of Mathematics, Auburn University at Montgomery, Montgomery, AL 36124-4023 USA
[d] School of Economics and Management, Ningbo University of Technology, Ningbo 315211 China

ARTICLE INFO

Keywords:
Flexible job-shop scheduling problem
Multi-action deep reinforcement learning
Graph neural network
Markov decision process
Multi-proximal policy optimization

ABSTRACT

This paper presents an end-to-end deep reinforcement framework to automatically learn a policy for solving a flexible Job-shop scheduling problem (FJSP) using a graph neural network. In the FJSP environment, the reinforcement agent needs to schedule an operation belonging to a job on an eligible machine among a set of compatible machines at each timestep. This means that an agent needs to control multiple actions simultaneously. Such a problem with multi-actions is formulated as a multiple Markov decision process (MMDP). For solving the MMDPs, we propose a multi-pointer graph networks (MPGN) architecture and a training algorithm called multi-Proximal Policy Optimization (multi-PPO) to learn two sub-policies, including a job operation action policy and a machine action policy to assign a job operation to a machine. The MPGN architecture consists of two encoder-decoder components, which define the job operation action policy and the machine action policy for predicting probability distributions over different operations and machines, respectively. We introduce a disjunctive graph representation of FJSP and use a graph neural network to embed the local state encountered during scheduling. The computational experiment results show that the agent can learn a high-quality dispatching policy and outperforms handcrafted heuristic dispatching rules in solution quality and *meta*-heuristic algorithm in running time. Moreover, the results achieved on random and benchmark instances demonstrate that the learned policies have a good generalization performance on real-world instances and significantly larger scale instances with up to 2000 operations.

1. Introduction

Flexible Job-shop scheduling problem (FJSP), playing an essential role in the modern manufacturing industry, is widely used in various manufacturing processes, such as semiconductor manufacturing, automotive, and textile manufacturing (Brucker & Schlie, 1990; Garey, Johnson, & Sethi, 1976; Jain & Meeran, 1999; Kacem, Hammadi, & Borne, 2002). It is a generalized job-shop scheduling problem (JSP), which is a classical NP-hard combinatorial optimization problem in computer science and operations research. In FJSP, a job consists of a specific sequence of consecutive operations, of which each operation is assigned to an eligible machine among a set of compatible machines for optimizing one or multiple objectives, such as makespan, mean completion time, maximum flow time, total tardiness et al. (Chaudhry & Khan, 2016; Xie, et al., 2019). Compared with a JSP problem, an FJSP is more complicated and flexible since the operations belonging to a job can be assigned to one or more compatible machines with different processing times.

Currently, existing methods for solving NP-hard combinatorial optimization problems can be summarized in two categories: exact and approximate methods. The exact methods such as mathematical programming search for optimal solutions in the entire solution space, but those methods are challenging to solve large-sized scheduling problems in a reasonable time owing to their NP-hardness (Li, Pan, & Liang, 2010). Due to the intractability of FJSP instances, more and more approximate methods, including heuristics, *meta*-heuristics, and machine learning techniques, are developed to solve instances of real-world problems. Generally, an approximate method can achieve a good trade-

* Corresponding author at: School of Mechanical Engineering, Southwest Jiaotong University, Chengdu 610031 China.
E-mail address: pengguo318@swjtu.edu.cn (P. Guo).

https://doi.org/10.1016/j.eswa.2022.117796
Received 5 February 2022; Received in revised form 12 May 2022; Accepted 5 June 2022
Available online 8 June 2022

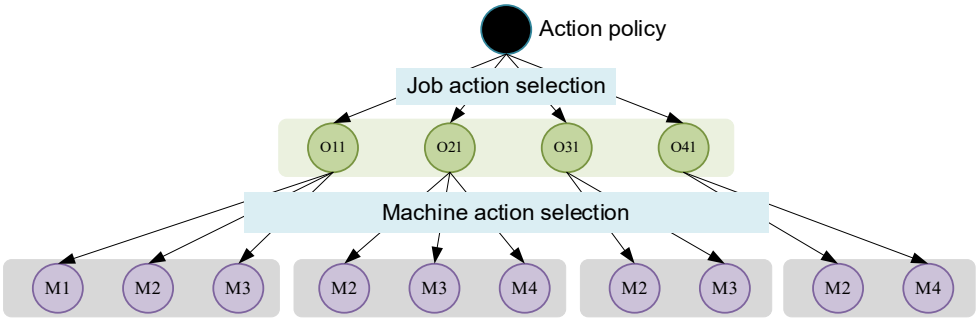


Fig. 1. Illustration of the multi-action space for FJSP.

off between the computational effort and the quality of resulted schedules. In particular, the swarm intelligence (SI) and evolutionary algorithms (EAs), such as genetic algorithm (GA), particle swarm optimization (PSO), ant colony optimization (ACO), artificial bee colony, et al., have shown strength in solving FJSP instances.

Despite the SI and EAs can solve FJSP within a reasonable amount of time compared with exact mathematic optimization methods, these methods are not competent in real-time scheduling environments, as they may still suffer unpredictably extremely long computing time to obtain a satisfactory solution when a significant number of iterations is necessitated by an underlying algorithm. Dispatching rules, as delegates of heuristic methods, are widely used in real-time scheduling systems, such as considering the disruption of dynamic events. A dispatching rule typically possesses lower computational complexity and is easier to implement than mathematical programming and *meta*-heuristics. Generally, the dispatching rules used in solving FJSP can be divided into two basic categories: the job selection rules and the machine selection rules. These rules are designed and combined to minimize scheduling objectives such as mean flow time, mean tardiness, and maximum tardiness. However, efficient dispatching rules commonly require substantial domain expertise knowledge and trial-and-error (Zhang, et al., 2020), and fail to guarantee a local optimum (Luo, 2020).

Recently, deep reinforcement learning (DRL) algorithms have cast a silver lining to provide a scalable method to solve scheduling problems with common characteristics. Several learning-based works (Bengio, Lodi, & Prouvost, 2021) focus on other types of combinatorial optimization problems, such as traveling salesman problems (TSPs) and vehicle routing problems (VRPs), but it is not known that DRL has been used to study a complicated scheduling problem such as FJSPs. This gap in knowledge motivates us to exploit a learning-based approach for FJSP.

Generally, an RL agent interacts with the environment according to the following behavior: an agent first receives a state s_t and selects an action a_t based on the state at each timestep, then obtains a reward r_t and transfers to the next state s_{t+1}. In the setup of RL, the action a_t is selected from action space $\mathscr{A}$. However, in this paper, a hierarchical multi-action space of FJSP is constructed with a job operation action space and a machine action space, which means that the general setup of RL cannot be applied to FJSP. For solving the FJSP, this RL setting involves with a job operation action space and machine action space. In particular, the two action spaces have a hierarchical structure instead of a parallel structure. At each timestep, the RL agent selects an operation action from its eligible operation action space and then chooses a machine action for the selected operation action from its compatible machine action space. The compatible machine action space is related to the selected operation action. Fig. 1 shows an example of the hierarchically structured action space of FJSP. It contains four job operation actions shown in green, and each job operation action has a compatible machine action space in purple.

In this paper, we proposed a novel end-to-end model-free DRL architecture on FJSP and demonstrated that it yields superior performance in terms of solution quality and efficiency. The proposed model-free DRL architecture can be directly applied to arbitrary FJSP scenarios without modeling the environment in advance. That is to say, the transition probability distribution (and the reward function) associated with the Markov decision process (MDP) is not explicitly defined when invoking the environment. Meanwhile, based on the advantages of our design of policy networks, our architecture is not bounded by the instance size (numbers of job/operation and machine), unlike previous work (Park, et al., 2020). The main contributions of this article are listed as follows.

- We introduce a Markov Decision Process formulation for a multi-action space of FJSP. The definitions of states, actions, and rewards in the introduced multiple Markov decision processes (MMDPs) are given for addressing FJSP. Moreover, the disjunctive graph representation of FJSP is introduced, and a graph neural network is used to embed a local state.
- A multi-pointer graph network (MPGN) architecture is designed to solve the proposed MMDPs. An MPGN consisting of two encoder-decoder components are used to define the job operation action policy and the machine action policy for predicting a probability distribution over different operations or machines, respectively. As a result, an MPGN can be trained by small-scale problems and generalized to large-scale ones.
- In the training phase, an algorithm called multi-Proximal Policy Optimization (multi-PPO) is proposed to learn two sub-policies, including a job operation action policy and a machine action policy for handling FJSP instances. The computational experiments on benchmarks demonstrate the generalization performance of the proposed framework.

The rest of the paper is organized as follows. Section 2 summarizes the relevant literature. Section 3 describes the background information on the concepts and technologies relevant to our study. Section 4 explains our methodologies, including the MMDPs formulation, the parameterized two sub-policies, and the multi-PPO algorithm. Section 5 gives the experimental results and discussions. Finally, conclusions and prospects are exposed in Section 6. Moreover, Appendix A introduces the details of the comparison dispatch rules. Appendix B analyzes the sensitivities of the hyperparameters. Appendix C showcases the convergence curves of our method for problem instances of different scales.

2. Literature review

Research on the solution approaches to solve flexible job shop scheduling problems has been reviewed thoroughly in (Chaudhry & Khan, 2016; Xie, et al., 2019). This section briefly reviews two research areas related to the underlying study: previous studies related to FJSP and deep reinforcement learning techniques in solving scheduling problems and other combinatorial optimization problems.

Table 1
Survey of End-to-end DRL methods and DRL for dispatching rules and *meta*-heuristic algorithms in solving scheduling problems.

Framework	Author	Reinforcement learning	Neural network	Agent	Problem
End-to-end DRL methods	(Waschneck, et al., 2018)	Value-Based, Deep Q-Leaning	DNN	Multi-Agent	Job-shop scheduling
	(X. Chen & Tian, 2019)	Actor-Critic	LSTM	Single-Agent	Job-shop scheduling
	(Solozabal, Ceberio, & Takáč, 2020)	Policy-Based	LSTM	Single-Agent	Job-shop scheduling
	(Liu, Chang, & Tseng, 2020)	Actor-Critic	CNN	Single-Agent	Job-shop scheduling
	(Hameed & Schwung, 2020)	Value-Based, DDPG	GNN	Multi-Agent	Job-shop scheduling
	(Han & Yang, 2020)	Value-Based, Deep Q-Learning	CNN	Single-Agent	Job-shop scheduling
	(C. Zhang, et al., 2020)	Policy-Based, PPO	GNN	Single-Agent	Job-shop scheduling
	(L. Wang, et al., 2021)	Policy-Based, PPO	DNN	Single-Agent	Job-shop scheduling
	Ours	**Policy-Based, Multi-PPO**	**GNN**	**Single-Agent**	**Flexible job-shop scheduling**
Integrated dispatching rules or *meta*-heuristic algorithms into DRL	(Shahrabi, Adibi, & Mahootchi, 2017)	Value-Based, Q-Leaning	–	Single-Agent	Job-shop scheduling
	(Lopes Silva, et al., 2019)	Value-Based, Q-Leaning	–	Multi-Agent	Parallel machine scheduling
	(Zhao, et al., 2019)	Value-Based, Q-Leaning	–	Single-Agent	Flexible job-shop scheduling
	(Chen, et al., 2020)	Value-Based, Q-Leaning	–	Single-Agent	Flexible job-shop scheduling
	(Zhou, Zhang, & Horn, 2020)	Value-Based, Q-Leaning	DNN	Single-Agent	Job-shop scheduling
	(Luo, 2020)	Value-Based, Deep Q-Learning	DNN	Single-Agent	Flexible job-shop scheduling

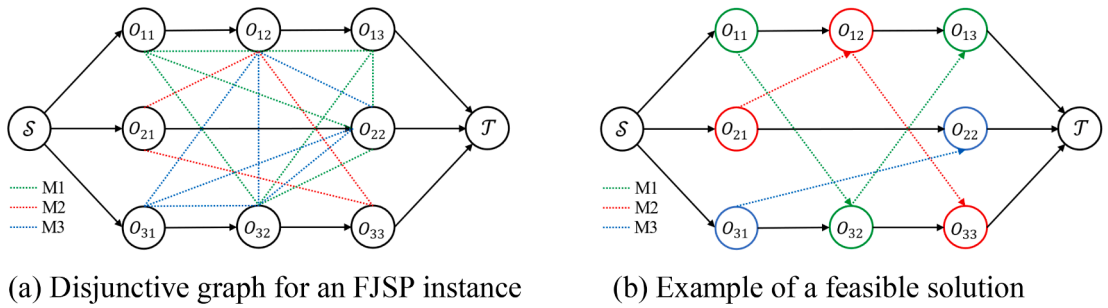
2.1. Solving FJSP via mathematical programming and heuristics

Over the past decades, there have been many methods, including exact and approximate algorithms for solving this class of FJSPs. Exact methods based on mathematical programming have been applied to find optimal solutions for small-sized and medium-sized FJSP instances. Still, the proposed methods of the literature have failed in solving large instances of the problem. (Brucker & Schlie, 1990) presented a polynomial running time algorithm to solve an FJSP with two jobs. (Fattahi, et al., 2007) proposed a sequence-position variable-based model for formulating FJSP and tested the performance of their model on 20 small-sized and medium-sized instances with LINGO. (Özgüven, Özbakır, & Yavuz, 2010) developed mixed-integer linear programming (MILP) model for solving FJSP instances using precedence variables and used their model to solve the same 20 instances with a CPLEX solver. (Demir & Kürşat İşleyen, 2013) evaluated the performances of five mathematical models in terms of makespan and CPU time and demonstrated that the MILP model with precedence variables has the least computation time. Moreover, (Naderi & Roshanaei, 2021) developed an exact method based on logic-based Benders decomposition for FJSP with a popular objective function, makespan. They claimed that the performance of their method outperforms a constraint programming model and a mixed-integer programming model in terms of optimality gaps, robustness, and number of found optima. Despite the success of various exact methods, they are inapplicable in solving large-sized FJSP instances due to their NP-hardness.

Approximate methods such as swarm intelligence (SI) and evolutionary algorithms (EA) are employed to solve scheduling problems in recent years, including genetic algorithm (GA) (Pezzella, Morganti, & Ciaschetti, 2008), (Defersha & Rooyani, 2020), particle swarm optimization (PSO) (Zhang, et al., 2009), ant colony optimization (ACO) (Xing, et al., 2010), tabu search (TS) (Li, et al., 2010), artificial bee colony (ABC) (Li, Pan, & Gao, 2011), evolutionary algorithm (EA) (Chiang & Lin, 2013), neighborhood search (NS) (Yazdani, Amiri, & Zandieh, 2010), scatter search (SS) (González, Vela, & Varela, 2015), biogeography-based optimization (BBO) (Rahmati & Zandieh, 2012), discrete harmony search (DHS) (Gao, et al., 2016), metaheuristics (Baykasoglu, 2002), and hybrid metaheuristics (Bożejko, Uchroński, & Wodecki, 2010). Although the EA and SI methods have less computation time than an exact algorithm, they are incompetent in real-time scheduling environments. For solving a scheduling problem more practically, a series of dispatching rules are designed. (Doh, et al., 2013) suggested a heuristic approach that combines machine assignment rules and job sequencing rules for solving FJSP with multiple process plans. They tested the performance of a scheme combining 3 machine assignment rules and 12 job selection rules. Recently, genetic programming (GP), a hyper-heuristic method, has been effectively applied to represent and evolve scheduling heuristics for FJSP and dynamic flexible job-shop scheduling problem (DFJSP) (Zhang, Mei, & Zhang, 2019). *In general, the design of efficient dispatching rules is non-trivial, which usually requires domain knowledge and a long development time period.*

2.2. Solving combinational optimal problems via DRL

Recently, more and more works applied the DRL technologies to combinatorics optimal problems and have made some incredible achievements. Most of the DRL applications focus on routing problems such as TSP and VRP (Kool, van Hoof, & Welling, 2019; Lei, et al., 2021; Lu, Zhang, & Yang, 2019; Nazari, et al., 2018; Vinyals, Fortunato, & Jaitly, 2015; Wu, et al., 2019), graph optimal problems (Dai, et al., 2017; Li, Chen, & Koltun, 2018), radio access network problems (Zhu, et al., 2021), and the satisfiability problems (Amizadeh, Matusevych, & Weimer, 2018; Selsam, et al., 2018). However, scheduling problems, especially for FJSP, are scarcely studied. The existing works of DRL for scheduling problems can be classified into two basic categories: 1) integrating the DRL technologies into the traditional optimization research methods, such as the heuristic dispatching rules, *meta*-heuristic algorithms; 2) the end-to-end DRL architectures in which the DRL agent directly outputs the solution by extracting the information features of the instance input without designing and adopting traditional



(a) Disjunctive graph for an FJSP instance (b) Example of a feasible solution

Fig. 2. Disjunctive graph representation of FJSP.

optimization research methods. Table 1 summarizes the classified two categories labeled "end-to-end DRL methods" and "DRL for dispatching rules and *meta*-heuristic algorithms", respectively.

Some works based on DRL for solving scheduling problems are summarized as follows. (Wang, et al., 2021) proposed a DRL approach for dynamic Job-shop scheduling in intelligent manufacturing, and they claimed that their method outperforms heuristic rules and *meta*-heuristic algorithms. (Chen & Tian, 2019) used DRL to train a local search policy to rewrite an existing solution for solving a job scheduling problem, in which the states are represented by a Directed Acyclic Graph (DAG) that describes the dependency among the schedules of different jobs. In (Waschneck, et al., 2018), cooperative agents based on Deep Q-Network (DQN) are designed for production scheduling, in which each DQN agent optimizes the dispatching rules at one work center and monitors the actions of other agents optimizing a global reward. (Oren, et al., 2021) proposed a Deep Q-Learning (DQL) approach for solving a parallel machine job scheduling problem, in which states are represented as graphs. (Lin, et al., 2019) proposed a DQN-based approach to solving a JSP, in which a policy is learned to select a priority dispatching rule for each machine. (Hu, et al., 2020) proposed a DQN with a Petri-net convolution layer to solve a dynamic scheduling problem. (Zhang, et al., 2020) suggested an end-to-end DRL approach to automatically learn priority dispatching rules for the JSP. They claim that their method has strong performance against the best existing dispatching rules.

Research using DRL on FJSP is relatively rare. (Luo, 2020) developed a DQN agent to select the most suitable one from the proposed six composite dispatching rules at each rescheduling point for a dynamic flexible Job-shop scheduling problem (DFJSP). However, the design of effective dispatching rules is a tedious task because of: (a) requiring a myriad of specialized knowledge and trial-and-error; (b) limited performance for different FJSPs. *In contrast, the proposed end-to-end method learns a policy to solve an FJSP instance based on extracting informative knowledge from low-level raw features of jobs and machines without manually designed dispatching rules.* Moreover, some studies adopt the DRL technology to improve the performance of the *meta*-heuristic algorithms for FJSP, like genetic algorithm (GA). (Chen, et al., 2020) proposed a reinforcement learning method to dynamically adjust the key parameters of GA during the iteration process. They claimed their method significantly outperforms its competitors, such as standard GA in solving FJSP. Those methods acquire higher solution quality but are time-consuming and have high complexity in running time that are not applicable for some real-time scheduling scenarios with dynamics and uncertainty (Luo, 2020). (Park, et al., 2020) proposed a multi-agent reinforcement learning approach for solving FJSP with setup times, in which each agent represents a machine and shares a neural network. However, the trained policies in that method cannot deal with instances with different job sizes. *In contrast, the proposed method can directly train a policy to solve variable problem sizes, including large-sized problems.*

2.3. Summary

In the above-related work, most traditional methods, including exact methods based on mathematical programming and metaheuristics, cannot apply to large FJSP instances or real-time FJSP instances due to their time complexity. Some researchers have used DRL to solve combinatorial optimization problems and achieved good results, but FJSP has received less attention. Some DRL-based methods in solving FJSP are designed to select composite dispatching rules instead of directly finding scheduling solutions, whose performance depends on the design of dispatching rules. *To the best of our knowledge, there is no research to solve the FJSP via multiple action end-to-end DRL framework without predetermined dispatching rules.* In this paper, we adopt the MMDPs to formulate the FJSP and propose an end-to-end DRL-based framework to solve FJSP. The framework can train a policy to solve unseen instances with variable sizes directly.

3. Background of FJSP and RL

Flexible Job-shop Scheduling problem. A standard Flexible Job-shop Scheduling problem can be stated as follows. A set $\mathcal{J} = \{J_1, \cdots, J_n\}$ of n jobs and a set $\mathcal{M} = \{M_1, \cdots, M_m\}$ of m machines are given. Each job J_i consists of a specific sequence of n_i consecutive operations $\mathcal{O}_i = \{O_{i1}, O_{i2}, \cdots, O_{in_i}\}$ with precedence constraints. An operation of a job denoted by O_{ij} can be processed on a subset of eligible machines $\mathcal{M}_{ij} \subseteq \mathcal{M}$. Furthermore, let p_{ijk} denote the processing time of operation O_{ij} on machine M_k. Each operation cannot be interrupted once started, and one machine can only perform one operation at a time. Besides, all machines and jobs are available at the start of the time horizon. In this work, the optimization goal of FJSP is to assign operations to compatible machines and determine a sequence of operations on a machine for minimizing the *makespan* C_{max} described by Equation (1), where C_{in_i} denotes the completion time of job i. Let $n \times m$ denote the size of an FJSP instance.

$$C_{max} = \max\{C_{in_i}\}, i \in \{1, \cdots, n\} \tag{1}$$

Reinforcement learning. The standard reinforcement learning based on a Markov decision process (MDP) is defined by a tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where an agent interacts with the environment at each discrete timestep. At timestep t, the agent receives a state s_t from state space $\mathcal{S}$ and takes an action a_t from action space $\mathcal{A}$. Then, the agent obtains a reward r_t from the environment and then enters the next state s_{t+1} based on s_t and a_t according to transition probability distribution P. Moreover, $\pi(a|s)$ is defined as the probability of selecting an action at a given state s. The goal of reinforcement learning is to learn a policy π which can maximize the expected total return $R(\pi) = \mathbb{E}_\pi[\sum_t \gamma^t r_t]$ where $\gamma \in [0, 1]$ is a discount factor.

Disjunctive graph for Flexible Job-shop Scheduling problem. In general, an FJSP problem can be defined by a disjunctive graph $G = (\mathcal{O}, \mathcal{C}, \mathcal{D})$. Here, $\mathcal{O} = \{O_{ij} | \forall i.j\} \cup \{\mathcal{S}, \mathcal{T}\}$ is a set of all operations where $\mathcal{S}$ and $\mathcal{T}$ are the dummy starting operation and a dummy ending operation, as illustrated in Fig. 2 (a) and (b). Besides, $\mathcal{C}$ is a set of conjunctive arcs representing the precedence constraints between consecutive operations from the same job. $\mathcal{D}$ is a set of disjunction (undirected) arcs, in

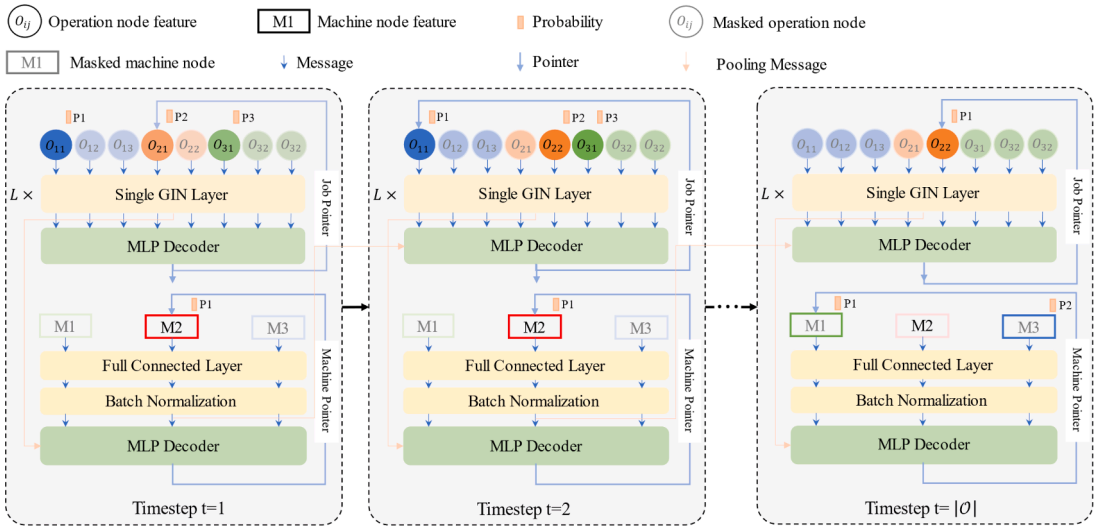


Fig. 3. The MPGN architecture for the FJSP.

which each arc connects a pair of operations that can be processed by the same machine. Fig. 2 (a) and (b) show a 3 × 3 FJSP instance's disjunctive graph and a feasible solution, respectively. In Fig. 2 (a), black arrows represent conjunctive arcs and colored lines represent disjunctive arcs which correspond to eligible machine cliques in different colors. In Fig. 2 (b), all disjunctive arcs are allocated in directions that denote a processing sequence of operations on a machine.

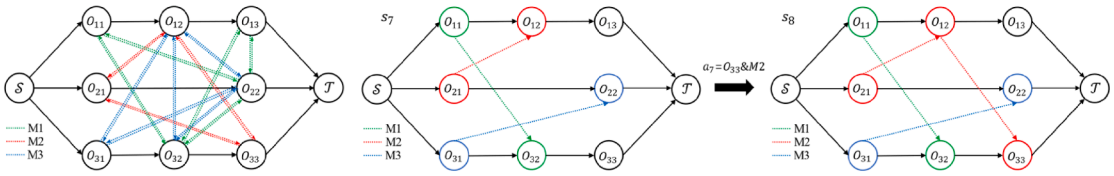
4. Proposed methods

In this section, we formally introduce our proposed framework. We first conceptualize a flexible job-shop scheduling process as a multi-action reinforcement learning task and further define such a task as a multiple Markov decision process. Then, we propose a novel multi-pointer graph network (MPGN) based on GNN, which is suitable for multi-action combination optimization problems such as FJSP, or train scheduling/rescheduling problems (Corman, et al., 2014; Lange & Werner, 2018). Moreover, we shall design a multi-proximal policy optimization algorithm (multi-PPO) based on actor-critic architecture for training the proposed MPGN.

4.1. Problem setting

FJSP is a sequential decision-making task. $|\mathscr{O}|$ timesteps of consecutive decisions are needed for solving an FJSP instance. At each timestep, the agent outputs a priority index for each eligible operation and selects the one via sampling or greedy decoding strategy to dispatch. Then, the agent computes a priority index for each compatible machine and then selects the one via sampling or greedy decoding strategy for the dispatched operation. In each timestep, the agent needs to control multiple actions at the same time. This multi-action (Wang & Yu, 2016) (also called multi-task) reinforcement learning problem is modeled as an MDP defined by a tuple $(\mathscr{S}, \mathscr{A}, P, r, \gamma)$. The action set $\mathscr{A} = \mathscr{A}_o \times \mathscr{A}_m$ is two-dimensional, in which $\mathscr{A}_o$ is the eligible operation sub-action set and $\mathscr{A}_m$ is the compatible machine sub-action set. This MDP coupled with a multi-action space is referred to as multiple MDPs (MMDPs) (Wang & Yu, 2016). We define the MMDPs over the states, actions, and rewards in the proposed framework as follows.

1. **States**: The global state s_t at timestep t is composed of local states s_t^o and s_t^m. Local state s_t^o is a disjunctive graph $G_t = (\mathscr{O}, \mathscr{C} \cup \mathscr{D}_t^o, \mathscr{D}_t)$ where $\mathscr{D}_t^o \subseteq \mathscr{D}$ is the disjunctive arcs which have been assigned directions (i.e., the actions denoted in the disjunctive graph of FJSP) till timestep t, and $\mathscr{D}_t \subseteq \mathscr{D}$ is the set of remaining disjunction arcs. Each node $O_{ij} \in \mathscr{O}$ contains two features $\left[LB_t(O_{ij}), I_t(O_{ij})\right], i \in \{1, \cdots, n\}, j \in \{1, \cdots, n_i\}$, where $LB_t(O_{ij})$ is either the completion time for a scheduled operation O_{ij} or the estimated minimum completion time $LB_t(O_{i,j-1}) + \min(p_{ijk}, k \in \mathscr{M}_{ij})$ for an unscheduled operation O_{ij}, and $I_t(O_{ij})$ is a one dimension feature that equals to 1 if O_{ij} is scheduled, or 0 otherwise. The other local state s_t^m is consisted of all machine states where each state of the machine contains two-dimension features $\left[T_t(M_k), p_{ijk}\right], k \in \{1, \cdots, m\}$ where $T_t(M_k)$ is the completion time for machine M_k before timestep t. Furthermore, p_{ijk} is the processing time of operation O_{ij} on machine M_k if machine M_k is compatible for the operation, or the average processing time $\frac{1}{|K|}\sum_k p_{ijk}, k \in K$ where K is a set of compatible machines otherwise.
2. **Actions**: The actions at timestep t are composed of job operation action $a_o \in \mathscr{A}_o$ and machine action $a_m \in \mathscr{A}_m$. $\mathscr{A}_o$ is the set of eligible operations and $\mathscr{A}_m$ is the set of compatible machines for action a_o. The job operation action space $\mathscr{A}_o$ depends on the intersection of unfinished jobs, and the machine action space $\mathscr{A}_m$ depends on the number of compatible machines for a selected operation.
3. **Transition**: At timestep t, the agent samples an operation a_o and select a machine for the operation. Then, the directions of disjunctive arcs are updated based on the current job operation action and machine action. A new disjunctive graph is generated as a new local state s_{t+1}^o. An example is shown in Fig. 4 (b). Then, the machine completion time $T_{t+1}(M_k)$ and the processing time p_{ijk} of a_o are updated as a new local state s_{t+1}^m.
4. **Reward**: The goal is to learn a policy to optimally schedule such that the objective of makespan is minimized. To this end, we first calculate the gap between the partial solutions between two continuous timesteps t and $t+1$, i.e., $d_t = C(s_{t+1})\text{-}C(s_t)$ where $C(s_t) = \max_{ij}\{LB_t(O_{ij})\}$ is defined as the makespan of $LB_t(O_{ij})$. Then, we designate the negative value of the difference d_t as the immediate reward at each timestep, i.e., $r(s_t, a_o, a_m) = -d_t$ (the goal of the RL agent is maximizing the cumulative reward). The cumulative reward is the negative value of the terminal makespan when all operations are scheduled.
5. **Policy**: the policy $\pi_\Theta(a_o, a_m|s)$ compose of two stochastic sub-policies $\pi_{\theta_o}(a_o|s)$ and $\pi_{\theta_m}(a_m|s, a_o)$, where θ_o and θ_m are the parameters for the job operation policy and machine policy, respectively, and $\Theta = [\theta_o, \theta_m]$ is the entire set of all parameters. Policy $\pi_\Theta(a_o, a_m|s)$ selects a job operation action a_o and a machine action a_m according to a



(a) Fully directed disjunctive graph (b) Adding arc scheme for FJSP

Fig. 4. Fully directed disjunctive graph and adding arc scheme for FJSP.

probability distribution in job operation action space $\mathscr{A}_o$ and machine action space $\mathscr{A}_m$, respectively.

4.2. Parameterizing two sub-policies

To handle FJSP with a multi-action space, we proposed a multi-pointer graph network (MPGN) for parameterizing two sub-policies in charge of respective action selection. The MPGN architecture shown in Fig. 3 consists of two encoder-decoder components, which define the job operation action policy and the machine action policy, respectively. We next describe in detail the constructions of the two encoder-decoder components.

(1) Job operation encoder (graph embedding)

The disjunctive graph in the above MMDPs provides a complete view of the scheduling states containing numerical and structural information, such as the precedence constraints, processing order on each machine, compatible machine set for each operation, and the processing time of a compatible machine for each operation. It is crucial to extract all state information embedded in the disjunctive graph to achieve effective scheduling performance. It motivates us to embed the complex graph state by exploiting a graph neural network (GNN).

We used the Graph Isomorphism Network (GIN) (Xu, et al., 2018) to encode the disjunctive graph, i.e., local state s_t^o. GIN is a popular GNN variant, which has provably strong discrimination power and powerful isomorphism test (Xu, et al., 2018). At timestep t, local state s_t^o is defined by a disjunctive graph $G_t = (\mathscr{O}, \mathscr{C} \cup \mathscr{D}_t^o, \mathscr{D}_t)$, in which each node is encoded via a L-layer GIN. Each layer of the GIN is expressed as follows, where $\boldsymbol{h}_{v,t}^{(l)}$ is the embedding of node v from the l-th layer GIN and $\boldsymbol{h}_{v,t}^{(0)}$ is the primary features of input, $MLP_{\theta_l}^{(l)}$ is a multi-layer perceptron (MLP) for the l-th layer and θ_l are the parameters of this layer, $\epsilon^{(l)}$ is a learnable parameter, where $l \in \{1, \cdots, L\}$ and $v \in \{1, \cdots, |\mathscr{O}|\}$, and $\mathscr{N}(v)$ is a neighborhood set of node v.

$$\boldsymbol{h}_{v,t}^{(l)} = MLP_{\theta_l}^{(l)}\left(\left(1+\epsilon^{(l)}\right)\bullet \boldsymbol{h}_{v,t}^{(l-1)} + \sum_{u\in\mathscr{N}(v)} \boldsymbol{h}_{u,t}^{(l-1)}\right), l\in\{1,\cdots,L\}; v,t\in\{1,\cdots,|\mathscr{O}|\}, \tag{2}$$

The disjunctive graph G_t is a mixed graph with undirected arcs and directed arcs. A pre-process strategy is needed to perform the transition of the G_t before inputting it in GIN. In detail, an undirected arc is replaced with two directed arcs connecting the same nodes in opposite directions, as shown in Fig. 4 (a). However, this strategy makes a graph too dense to be efficiently processed by GIN (Xu, et al., 2018). Similar to (Zhang, et al., 2020), we use an "adding arc scheme" which ignores undirected disjunctive arcs in the initial state to reduce computational complexity, as shown in Fig. 4 (b).

In Fig. 4 (b), arc (O_{12}, O_{33}) is added to the graph following the actions of scheduling a job and assigning a machine at a timestep, i.e., arc (O_{12}, O_{33}) for action $a_7 = (O_{33}, M2)$. In this way, let $\overline{G}_t = (\mathscr{O}, \mathscr{C} \cup \mathscr{D}_t^o)$ approximate $G_t = (\mathscr{O}, \mathscr{C} \cup \mathscr{D}_t^o, \mathscr{D}_t)$. The neighborhood set $\mathscr{N}(v)$ of node v is connected by conjunctive (directed) arcs. The encoder outputs final embeddings $\boldsymbol{h}_{v,t}^{(L)}$ of the nodes and a graph pooling vector $\boldsymbol{h}_{\mathscr{G}}^t$ for the entire graph, i.e., $\boldsymbol{h}_{\mathscr{G}}^t$ is the mean of the final node embeddings:$\boldsymbol{h}_{\mathscr{G}}^t = 1/|\mathscr{O}|\sum_{v\in\mathscr{O}} \boldsymbol{h}_{v,t}^{(L)}$.

(2) Machine encoder (node embedding)

There is no graph structure in the machine's state information in which each node represents the primary features of each machine and a node does not connect to other nodes with a directed or undirected arc. Therefore, we adopt a full connected layer to encode the local state s^m. At timestep t, the primary features of each machine node at local state s_t^m is defined as a two-dimensional vector $\left[T_t(M_k), p_{ijk}\right]$, $k \in \{1, \cdots, m\}$ as mentioned earlier. The outputs at each node (machine) are embedding vector $\boldsymbol{h}_k^t$ and the pooling vector $\boldsymbol{u}^t$.

(3) Decoders (action selection)

The completed schedule is sequentially constructed by decoding at each timestep $t \in \{1, \cdots, |O|\}$, where $|O|$ is the total number of operations in an FJSP instance. At each timestep t, the job decoder selects a job operation action a_t^o and the machine decoder selects a machine action a_t^m to form a complete action $a_t = (a_t^o, a_t^m)$ based on the embeddings of local state s_t^o and s_t^m from the encoders.

The two decoders are based on MLP layers and have the same network structure but do not share their parameters. In decoding, each decoder computes a probability distribution over either the job operation action space or the machine action space, respectively. First, the two decoders are employed to output a job operation action score $c_{t,v}^o$ and a machine action score $c_{t,k}^m$, computed as, where $\bullet \,||\, \bullet$ is the concatenation operation.

$$c_{t,v}^o = MLP_{\theta_{\pi_o}}\left(\boldsymbol{h}_{v,t}^{(L)} || \boldsymbol{h}_{\mathscr{G}}^t || \boldsymbol{u}^t\right), v \in \{1, \cdots, |\mathscr{O}|\}, \tag{3}$$

$$c_{t,k}^m = MLP_{\theta_{\pi_m}}\left(\boldsymbol{h}_k^t || \boldsymbol{h}_{\mathscr{G}}^t || \boldsymbol{u}^t\right), k \in \{1, \cdots, |\mathscr{M}|\}, \tag{4}$$

Then, we used a masking scheme similar to (Nazari, et al., 2018) to avoid 1) dispatching the operations that have already been scheduled or those violating precedence constraints and 2) selecting the machines that cannot process an operation. In detail, the operations which are already scheduled or violate the precedence constraint are masked by setting their score $c_{t,v}^o$ and $c_{t,k}^m$ to $-\infty$. Similarly, we mask incompatible machines for a selected operation at timestep t.

Finally, we normalize the job operation action scores and the machine action scores by a *Softmax* function as follows.

$$p_i\left(a_t^o\right) = \frac{e^{c_{t,i}^o}}{\sum_v e^{c_{t,v}^o}}, \tag{5}$$

$$p_j\left(a_t^m\right) = \frac{e^{c_{t,j}^m}}{\sum_k e^{c_{t,k}^m}}. \tag{6}$$

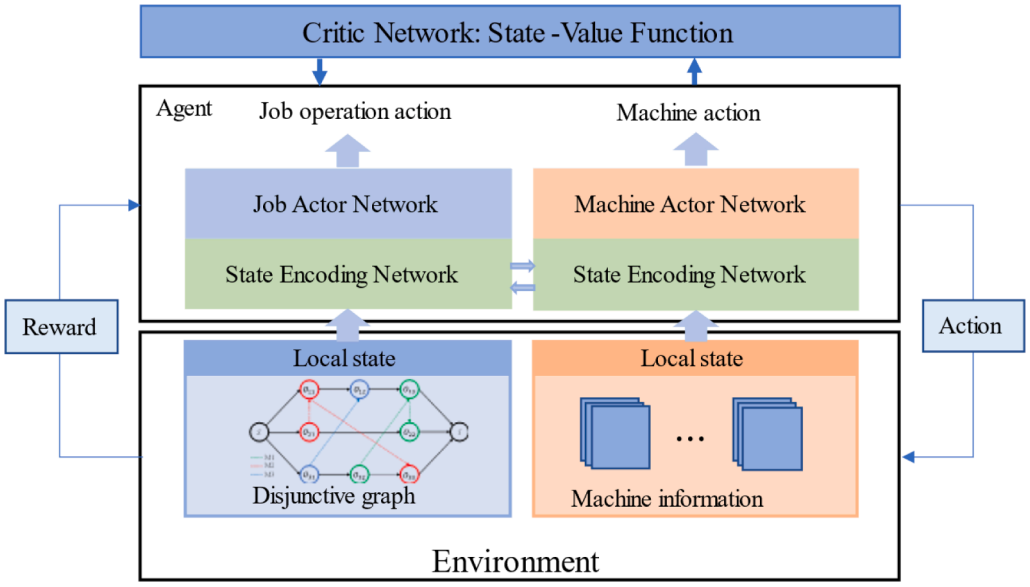


Fig. 5. Multiple actor-critic architecture for a multi-action space scheduling problem.

Finally, either a sampling strategy or a greedy decoding strategy to predict a_t^o and a_t^m based on $p_i(a_t^o)$ and $p_j(a_t^m)$ are applied.

4.3. Multi-Proximal policy optimization

To cope with this kind of multi-action reinforcement problem, we proposed a multi-Proximal Policy Optimization (multi-PPO) algorithm that takes a multiple actor-critic architecture and adopts PPO as its policy optimization method for learning the two sub-policies. The PPO algorithm is a state-of-the-art policy gradient approach with an actor-critic style, which is widely used to deal with both discrete and continuous control tasks (Schulman, et al., 2017). However, the PPO algorithm cannot be directly used to handle a multi-action task since it generally contains one actor to learn one policy that can only control a single action at each timestep. By contrast, the proposed multi-PPO architecture includes two actor networks (job operation and machine encoder-decoders described in Section 4.2 as the two actor networks, respectively).

In detail, the job operation actor makes job operation action-selection, and the machine actor makes machine action-selection, i.e., the job operation actor learns a stochastic policy $\pi_{\theta_o}(a_o|s)$ to select a job operation action a_o and the machine actor learns a stochastic policy $\pi_{\theta_m}(a_m|s,a_o)$ to choose a machine action for the selected action a_o at each timestep. In this way, the complete action is denoted by a tuple (a_o, a_m).

In the multi-PPO architecture, a single critic network with parameters ϕ is employed and works as an estimator to learn an approximate $\widehat{v}_\phi(s_t)$ of the state-value function. In the proposed multiple actor-critic architecture, $\widehat{v}_\phi(s_t)$ is used to calculate the variance-reduced advantage function estimator $\widehat{A}$. We ran the two actors in the environment to collect experience tuples for computing estimator $\widehat{A}_t$ formulated by Equation (7), where γ is a discount factor and T is much less than the length of an episode ($|\mathscr{O}|$ in FJSP). The training process of the proposed multi-PPO is described as follow.

$$\widehat{A}_t = \sum_{t'=t}^{T} \gamma^{t'} r_{t'} - \widehat{v}_\phi(s_t), \tag{7}$$

To generate a policy π_{θ_o} for discrete job operation actions and a policy π_{θ_m} for discrete machine actions, the two actor networks of the multi-PPO output the probability distributions over the job operation and machine action spaces, respectively. Then, a job operation and a machine action are selected by decoding strategy (random sampling or greedy decoding introduced in Section 5) over the generated probability distributions, respectively.

At each iteration of the training process, two actors of the multi-PPO running in the environment collect training experience tuples and then the two sub-policies π_{θ_o} and π_{θ_m} are updated via the collected samples (the training process as shown in Fig. 5). Specially, the two sub-policies are optimized by maximizing their respective clipped surrogate and entropy objectives. The updating processes are given as the following steps.

a) Training job operation and machine actor networks. The clipped surrogate objective $L^{\mathscr{k}}_{\mathrm{CLIP}}(\theta_{\mathscr{k}})$ and entropy objective $L^{\mathscr{k}}_{E}(\theta_{\mathscr{k}})$ for each actor $\mathscr{k}(\mathscr{k} \in \{o,m\})$ are calculated by the following equations, respectively, where the probability ratio $\delta_t^{\mathscr{k}}(\theta_{\mathscr{k}})$ between the updated and the prior-update job operation/machine policy is defined as $\frac{\pi_{\mathscr{k}}(a_t^{\mathscr{k}}|s_t)}{\pi_{\theta_{\mathscr{k}}old}(a_t^{\mathscr{k}}|s_t)}$ and ϵ is a clipping parameter. The job operation/machine actor network is trained to maximize $L(\theta_{\mathscr{k}}) = c_p L^{\mathscr{k}}_{\mathrm{CLIP}}(\theta_{\mathscr{k}}) + c_e L^{\mathscr{k}}_{E}(\theta_{\mathscr{k}})$, where c_p and c_e are clip and entropy coefficient hyperparameters (In this paper, $c_p = 1$ and $c_e = 0.01$ following (Schulman, et al., 2017)).

$$L^{\mathscr{k}}_{\mathrm{CLIP}}(\theta_{\mathscr{k}}) = \widehat{\mathbb{E}}_t\left[\min\left\{\delta_t^{\mathscr{k}}(\theta_{\mathscr{k}})\widehat{A}_t, \mathrm{clip}\left(\delta_t^{\mathscr{k}}(\theta_{\mathscr{k}}), 1-\epsilon, 1+\epsilon\right)\widehat{A}_t\right\}\right], \tag{8}$$

$$L^{\mathscr{k}}_{E}(\theta_{\mathscr{k}}) = \widehat{\mathbb{E}}_t\left[Entropy\left(\pi_{\theta_{\mathscr{k}}}\left(a_t^{\mathscr{k}}|s_t\right)\right)\right], \tag{9}$$

In the multi-PPO algorithm, we remark that the probability ratio $\delta_t^o(\theta_o)$ only considers the policy π_{θ_o} and the probability ratio $\delta_t^m(\theta_m)$ only considers the policy π_{θ_m}, which means that the two sub-policies π_{θ_o} and π_{θ_m} are viewed as two independent distributions.

b) Updating critic network. The critic network is updated to minimize the Mean Squared Error (MSE) objective $L_{MSE}(\phi)$ computed by: where r_t is the reward at timestep t.

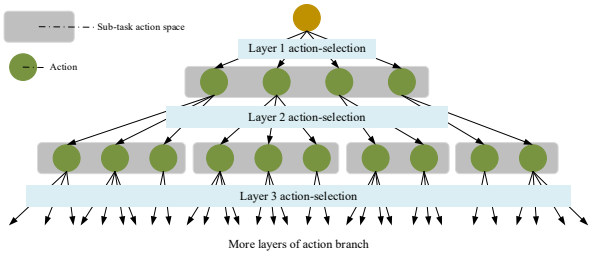


Fig. 6. A hierarchical action spaces structure.

$$L_{\mathrm{MSE}}(\phi) = \widehat{\mathbb{E}}_t\left[\mathrm{MSE}\left(r_t, \widehat{v}_\phi(s_t)\right)\right], \tag{10}$$

Algorithm 1 multi-PPO for FJSP

1: Input: number of training steps E_t, PPO update steps E_s, batch size B; training actor networks $\pi_{\theta_\mathscr{M}}(\mathscr{M} \in \{o,m\})$, and behavior actor networks $\pi_{\theta_\mathscr{M}^{old}}(\mathscr{M} \in \{o,m\})$ with trainable parameters $\theta_\mathscr{M}$ and $\theta_\mathscr{M}^{old}(\mathscr{M} \in \{o,m\})$; critic network v_ϕ with trainable parameters ϕ; policy loss coefficient c_p; value function loss coefficient c_v; entropy loss coefficient c_e; clipping parameter ϵ.
2: Initialization: Initialize $\theta_\mathscr{M}$ and ϕ; $\theta_\mathscr{M} \rightarrow \theta_\mathscr{M}^{old}(\mathscr{M} \in \{o,m\})$;
3: for $e = 1, \cdots, E_t$ do
4: Sampling B FJSP instances from a uniform distribution;
5: for $b = 1, \cdots, B$ do
6: for $t = 0, 1, 2, \cdots$ do
7: Sample $a_{b,t}^\mathscr{M}$ based on $\pi_{\theta_{\mathscr{M}old}}\left(a_{b,t}^\mathscr{M}|s_{b,t}\right)$ $(\mathscr{M} \in \{o,m\})$;
8: Observe reward $r_{b,t}$ and next state $s_{b,t+1}$;
9: $\widehat{A}_{b,t} = \sum_{t'=t}^{T} \gamma^{t'} r_{b,t'} - \widehat{v}_\phi(s_{b,t})$; $\delta_{b,t}^\mathscr{M}(\theta_\mathscr{M}) = \dfrac{\pi_{\theta_\mathscr{M}}\left(a_{b,t}^\mathscr{M}|s_{b,t}\right)}{\pi_{\theta_{\mathscr{M}old}}\left(a_{b,t}^\mathscr{M}|s_{b,t}\right)}$ $(\mathscr{M} \in \{o,m\})$;
10: if $s_{b,t}$ is terminal then
11: break;
12: end
13: end for
14: $L_{\mathrm{CLIP}}^{b,\mathscr{M}}(\theta_\mathscr{M}) = \widehat{\mathbb{E}}_t\left[\min\{\delta_{b,t}^\mathscr{M}(\theta_\mathscr{M})\widehat{A}_{b,t}, \mathrm{clip}\left(\delta_{b,t}^\mathscr{M}(\theta_\mathscr{M}), 1-\epsilon, 1+\epsilon\right)\widehat{A}_{b,t}\}\right]$, $(\mathscr{M} \in \{o,m\})$;
15: $L_E^{b,\mathscr{M}}(\theta_\mathscr{M}) = \widehat{\mathbb{E}}_t\left[\mathrm{Entropy}\left(\pi_{\theta_\mathscr{M}}\left(a_{b,t}^\mathscr{M}|s_{b,t}\right)\right)\right], (\mathscr{M} \in \{o,m\})$;
16: Aggregate Job/machine actor Loss: $L(\theta_\mathscr{M}) = c_p L_{\mathrm{CLIP}}^{b,\mathscr{M}}(\theta_\mathscr{M}) + c_e L_E^{b,\mathscr{M}}(\theta_\mathscr{M})$, $(\mathscr{M} \in \{o,m\})$;
17: Aggregate critic Loss: $L_{\mathrm{MSE}}(\phi) = \widehat{\mathbb{E}}_t[\mathrm{MSE}(r_t, \widehat{v}_\phi(s_t))]$;
18: end for
19: for *PPO step* $= 1 \cdots, E_s$ do
20: Update $\theta_\mathscr{M}(\mathscr{M} \in \{o,m\})$ and ϕ by a gradient method w.r.t. $L(\theta_\mathscr{M})$ and $L_{\mathrm{MSE}}(\phi)$;
21: end for
22: $\theta_\mathscr{M}^{old} \leftarrow \theta_\mathscr{M}(\mathscr{M} \in \{o,m\})$;
23: end for
24: Output: Trained parameter sets $\theta_\mathscr{M}$ $(\mathscr{M} \in \{o,m\})$ of the job/machine actor;

The core of the multi-PPO algorithm in pseudo-code is shown in Algorithm 1. Given a mini-batch size B, FJSPs generates B instances from a uniform distribution for training the two actor networks. Lines 4–13 present the scheduling processes in which the two behavior actors collect experience tuples until all operations are scheduled. Then, the two training actors interact with the environment to calculate the probability ratios through the collected experience tuples. Lines 14–19 calculate the aggregate job operation actor, machine actor, and critic losses. Based on the calculated losses, the multi-PPO performs updates (E_s times in Algorithm 1) on the two actor and critic networks, respectively. After that, the updated parameters of the two training actors are duplicated to the two behavior actors, as shown in line 24. Once the whole training process (lines 4–24) is implemented E_t times, the multi-PPO outputs the parameters of the two actor networks to perform online testing without retraining.

The proposed multi-PPO architecture can be extended to solve a combinational optimization problem with a hierarchical action space, which can be described as a tree structure. Fig. 6 shows an example of a problem with three sub-tasks and constructs a hierarchical action space structure with three layers of action selection. In the figure, each grey area demonstrates the sub-task action space for an action-selection layer and each node in the grey area is a discrete action that corresponds to a sub-task action-selection space (the grey areas) of the next layer.

The multi-PPO architecture includes multiple actors and one critic for the optimization problem with a general hierarchical action space. Specially, one actor network is exclusively in charge of a single sub-task action selection. And one critic network is used for FJSP that takes responsibility for leading the parameter update of all actors. Throughout the training phase, the multi-PPO runs all actors in the environment to collect experience tuples and updates the parameters of all actors to learn separate policies.

5. Computational experiment

In this section, we show our computational results. To evaluate the performance of the proposed framework, we performed it on both random instances and publicly available FJSP benchmarks.

Dataset: the proposed framework is evaluated on variously sized FJSP instances. Training set with 12,800 FJSP instances, validation set with 128 FJSP instances, and testing set with 128 FJSP instances are all randomly generated. For example, Table 2 shows a $n \times m = 3 \times 3$ FJSP instance with processing times for each operation, in which each job contains 3 operations and '-' denotes that a machine cannot process an operation. During the training phase, a lot of time is needed to train the MPGN model for large-scale FJSP instances. In this work, the MPGN is trained and validated using small- and middle-scale randomly generated instances, i.e., $6 \times 6, 10 \times 10, 15 \times 15, 20 \times 10, 20 \times 20$, and 30×20 instances, and is directly tested on much larger instances using 50×20 and 100×20 randomly generated instances to reduce the training time and to test generalization of the proposed framework. Furthermore, we also tested the performance of the proposed framework on the well-known public FJSP benchmarks, including Hurink's instances (Hurink, Jurisch, & Thole, 1994) and Behnke's instances (Behnke & Geiger, 2012).

Hyperparameters: We trained the MPGN using fixed hyperparameters except for batch size for different problem scales. We conducted a sensitivity analysis of several key hyperparameters in Appendix B. According to the analysis, we found that the hyperparameters' values are suitable for this paper. Each batch includes 32, 16, 16, 16, 8, and 4 instances for $6 \times 6, 10 \times 10, 15 \times 15, 20 \times 10, 20 \times 20$, and 30×20 scales, respectively. The training data, validation data, and testing data are generated on the fly. We used $L = 2$ GIN layers (Eq. (2)) which are shared by π_{θ_o} and $\widehat{v}_\phi$. In each GIN layer, $MLP_{\theta_l}^{(l)}$ includes two hidden layers with hidden dimension 128. The job operation action selection decoder $MLP_{\theta_{\pi o}}$, machine action selection decoder $MLP_{\theta_{\pi m}}$, and state-

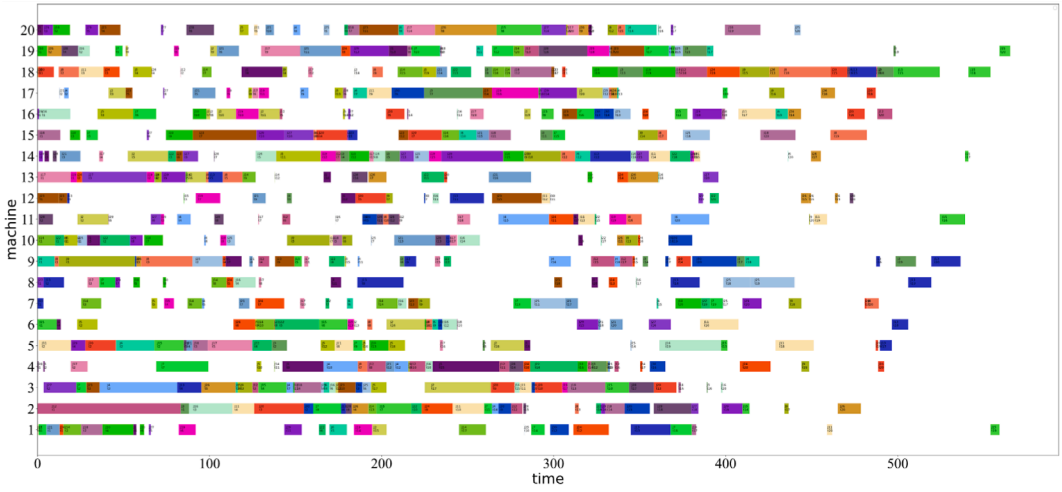
Table 2
An example 3×3 FJSP instance.

p_{ijk}	Job 1			Job 2			Job 3		
	O_{11}	O_{12}	O_{13}	O_{21}	O_{22}	O_{23}	O_{31}	O_{32}	O_{33}
Machine 1	–	–	56.4	–	66.1	–	–	69.5	37.8
Machine 2	45.3	22.5	–	35.8	–	65.4	–	–	–
Machine 3	–	9.8	–	–	78.7	26.3	34.9	54.4	–

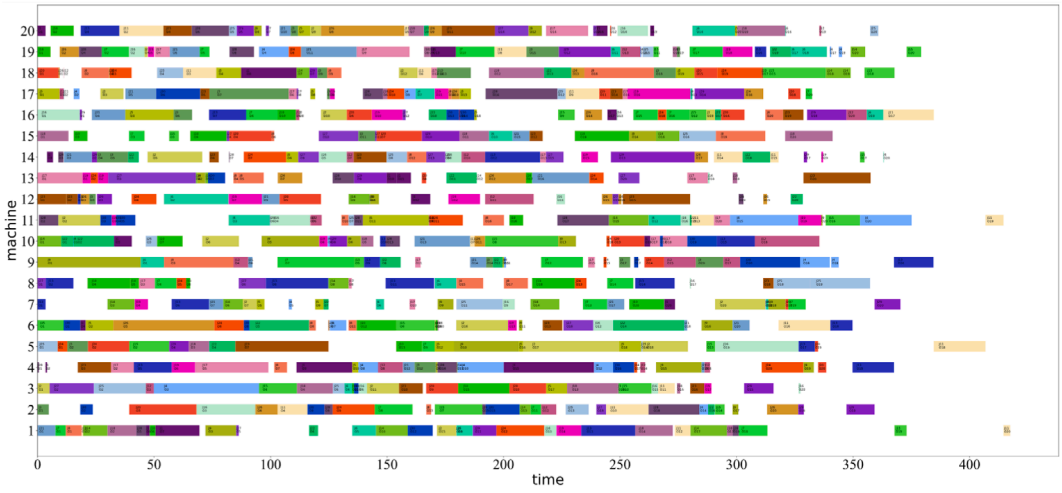
Table 3
Results of all methods on randomly generated instances.

Size		MIP	FIFO + SPT	MOPNR + SPT	LWKR + SPT	MWKR + SPT	FIFO + EET	MOPNR + EET	LWKR + EET	MWKR + EET	Ours
6 × 6	Obj.	227.86	328.45	329.28	397.02	331.58	418.62	438.59	474.21	613.07	**272.32**
	Gap	0.00%	44.15%	44.51%	74.24%	45.52%	83.72%	92.48%	108.11%	169.06%	**19.51%**
	Time (s)	0.73	0.028	0.032	0.028	0.027	0.026	**0.025**	**0.025**	0.026	0.041
10 × 10	Obj.	255.88	385.82	377.60	495.94	382.43	661.25	711.71	821.53	1173.02	**320.45**
	Gap	0.00%	50.78%	47.57%	93.82%	49.46%	158.42%	178.14%	221.06%	358.43%	**25.23%**
	Time (s)	2962	0.084	0.092	0.085	0.087	**0.077**	0.086	0.079	0.080	0.14
15 × 15	Obj.	287.23	413.00	412.91	567.56	409.99	966.56	1046.48	1259.30	1957.39	**347.99**
	Gap	0.00%	43.79%	43.76%	97.60%	42.74%	236.51%	264.34%	338.43%	581.47%	**21.15%**
	Time (s)	3600	0.24	0.26	0.23	0.24	**0.21**	0.24	0.25	0.25	0.39
20 × 10	Obj.	391.41	566.32	569.36	733.16	567.37	1063.45	1107.93	1210.85	1815.82	**454.85**
	Gap	0.00%	44.69%	45.46%	87.31%	44.96%	171.70%	183.06%	209.36%	363.92%	**16.21%**
	Time (s)	3600	0.21	0.24	0.22	0.21	**0.19**	0.23	0.23	0.23	0.34
20 × 20	Obj.	322.54	434.48	430.79	609.96	430.72	1262.36	1384.12	1709.83	2762.01	**361.75**
	Gap	0.00%	34.71%	33.56%	89.11%	33.54%	291.38%	329.13%	430.11%	756.33%	**12.16%**
	Time (s)	3600	0.42	0.46	0.45	0.44	**0.38**	0.41	0.42	0.44	1.08
30 × 20	Obj.	–	528.51	525.08	741.08	522.93	1633.91	1732.54	2087.02	3462.27	**433.42**
	Gap		21.94%	21.15%	70.98%	20.65%	276.98%	299.74%	381.52%	698.83%	**0.00%**
	Time (s)		0.78	0.86	0.83	0.83	**0.69**	0.78	0.81	0.82	1.97

value prediction network MLP_ϕ all have 2 hidden layers with hidden dimension 128. In the multi-PPO algorithm, we set the coefficient for clipping, policy loss, value function, and entropy to 0.2, 2, 1, and 0.01 following the PPO method, respectively. The proposed framework is running by the well-known Adam optimizer with a learning rate $lr = 1 \times 10^{-3}$ during the training process. The model is constructed by PyTorch, and the code is implemented using Python 3.7. We used the hardware with Intel Xeon E5 V3 2600 CPU and a single Nvidia Tesla V100 GPU. All



(a) The Gantt chart of the best dispatching rule



(b) The Gantt chart of our method

Fig. 7. The Gantt charts of our method and the best dispatching rule for 30 × 20 FJSP instance.

Table 4
Results of all methods on randomly generated instances.

Size		FIFO + SPT	MOPNR + SPT	LWKR + SPT	MWKR + SPT	FIFO + EET	MOPNR + EET	LWKR + EET	MWKR + EET	Ours (20 × 20)	Ours (30 × 20)
50 × 20	Obj.	716.07	716.10	1002.88	716.61	2567.54	2631.44	2889.44	4829.83	590.22	**587.48**
	Gap	21.89%	21.89%	70.71%	21.98%	337.04%	347.92%	391.84%	722.13%	0.47%	**0.00%**
	Time (s)	1.34	1.48	1.39	1.41	**1.10**	1.32	1.35	1.36	4.12	4.12
100 × 20	Obj.	1201.32	1199.82	1574.44	1199.10	5004.78	5041.13	5184.85	7840.83	1071.03	**1054.70**
	Gap	13.90%	13.76%	49.28%	13.69%	374.52%	377.97%	221.06%	643.42%	1.55%	**0.00%**
	Time (s)	6.66	7.40	6.83	7.03	**4.62**	6.35	6.70	6.99	18.34	18.34

test data and the code can be found at https://github.com/pengguo318/FJSPDRL.

Baseline methods: For each problem at a particular scale, we compared the performance of the proposed method with the well-known dispatching rules in the literature. For FJSP, a job sequencing dispatching rule and a machine assignment dispatching rule are needed to complete a schedule solution. There are hundreds of dispatching rules for the job-shop scheduling problem and FJSP in literature with a wide range of performance (Zhang, et al., 2020). However, it is nearly impossible to evaluate hundreds of dispatching rules. Reference (Doh, et al., 2013) tested 36 rule combinations with 12 rules for job sequencing and three rules for machine selection. Based on the performance of the 36 combined rules, we chose the top-ranked four job sequencing rules and two machine assignment dispatching rules and combined them as eight compound dispatching rules as the baseline in our paper, i.e., four job sequencing dispatching rules including *First in First Out* (FIFO), *Most Operation Number Remaining* (MOPNR), *Least Work Remaining* (LWKR), and *Most Work Remaining* (MWKR), and two machine assignment dispatching rules including *Shortest Processing Time* (SPT) and *Earliest End Time* (EET). The eight compound dispatching rules are used as the benchmark for testing the performance of the proposed end-to-end DRL framework. The comparing methods are all implemented using Python 3.7. Interested readers can refer to Appendix A for more details. For the randomly generated FJSP instances, we obtain their solutions according to the MIP solved by Gurobi solver with a time limit of 3600 s. We calculated the Gap of the solutions of all methods compared to those obtained by MIP. Furthermore, for the public benchmark FJSP instances, we calculated the Gap value via the state-of-the-art lower bound from literature.

Decoding strategy: We use two decoding strategies (i.e., sampling and greedy decoding) for training and testing in the proposed method.

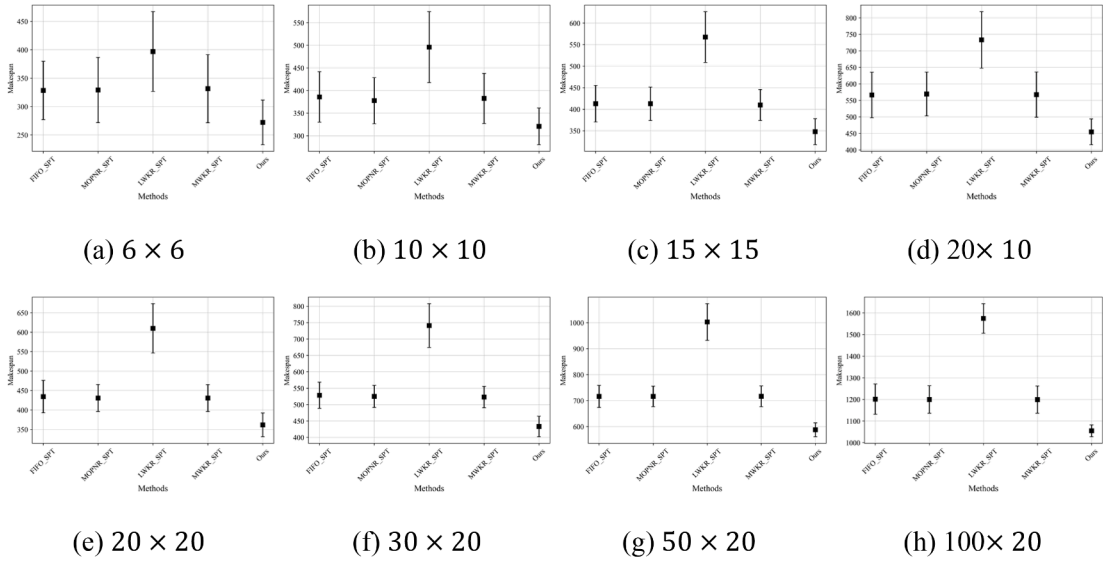
- Stochastic sampling: In each decoding timestep, the random policy samples the node to be selected according to the probability distribution to construct an effective solution.
- Greedy decoding: Different from the stochastic sampling, the node with the highest probability is chosen greedily in each decoding timestep.

During the training process, stochastic sampling is helpful to explore the environment to obtain a better model performance. On the other hand, we used greedy decoding for testing, which selects the action with the highest probability.

5.1. Computational results of randomly generated instances

We first evaluated the proposed framework on randomly generated small- and medium-scale instances (i.e., $6 \times 6, 10 \times 10, 15 \times 15, 20 \times 20$, and 30×20 FJSP instances) for training and testing. Fig. 10 in Appendix C shows the convergence curves of problem instances with different scales. We reported the average objective value *makespan*, running times, and Gap to the Gurobi solver (the smaller, the better) for 128 randomly generated instances of the proposed and baseline methods, summarized in Table 3. The best results, i.e., the lowest computing time, *makespan*, and Gap value are highlighted in bold for each problem.

It can be seen that the trained MPGN model consistently outperforms all baseline dispatching rules for all FJSP scales. Moreover, the performance of the proposed method is more stable than the baseline dispatching rules when the problem scales increase. The baseline



(a) 6 × 6 (b) 10 × 10 (c) 15 × 15 (d) 20× 10

(e) 20 × 20 (f) 30 × 20 (g) 50 × 20 (h) 100× 20

Fig. 8. Means plot and 95% confidence level Tukey's HSD intervals for solution values of 128 instances of different methods and scales.

Table 5
Results of the ranking of the top three dispatching rules in performance, 2SGA and our method on Hurink' instances.

Instance		Dispatching rules			Meta-heuristics (Defersha & Rooyani, 2020)		DRL	
		FIFO + EET	MWKR + EET	MOPNR + EET	RGA	2SGA	Ours	UB
10 × 5	Vdata_la1	640 (12.28%)	644 (12.98%)	672 (17.89%)	577 (1.16%)	**572 (0.48%)**	610 (7.02%)	570*
	Vdata_la2	624 (17.96%)	613 (15.88%)	653 (23.44%)	535 (1.18%)	**532 (0.63%)**	555 (4.91%)	529*
	Vdata_la3	572 (19.92%)	566 (18.66%)	558 (16.98%)	485 (1.62%)	**481 (0.92%)**	532 (11.53%)	477*
	Vdata_la4	649 (29.28%)	593 (18.13%)	605 (20.52%)	510 (1.49%)	**506 (0.70%)**	530 (5.58%)	502*
	Vdata_la5	582 (27.35%)	577 (26.26%)	522 (14.22%)	468 (2.28%)	**463 (1.24%)**	507 (10.94%)	457*
15 × 5	Vdata_la6	873 (9.26%)	911 (14.02%)	880 (10.14%)	804 (0.60%)	**801 (0.21%)**	820 (2.63%)	799*
	Vdata_la7	784 (4.67%)	782 (4.41%)	842 (12.42%)	756 (0.88%)	**751 (0.26%)**	757 (1.07%)	749*
	Vdata_la8	827 (8.1%)	886 (15.82%)	853 (11.5%)	768 (0.38%)	**766 (0.19%)**	782 (2.22%)	765*
	Vdata_la9	884 (3.63%)	926 (8.56%)	888 (4.1%)	858 (0.59%)	**854 (0.18%)**	879 (3.05%)	853*
	Vdata_la10	846 (5.22%)	899 (11.82%)	846 (5.22%)	808 (0.44%)	**806 (0.19%)**	862 (7.21%)	804*
20 × 5	Vdata_la11	1142 (6.63%)	1142 (6.63%)	1113 (3.92%)	1074 (0.32%)	**1072 (0.09%)**	1101 (2.80%)	1071*
	Vdata_la12	985 (5.24%)	1012 (8.12%)	1008 (7.69%)	939 (0.27%)	**937 (0.08%)**	950 (1.50%)	936*
	Vdata_la13	1099 (5.88%)	1108 (6.74%)	1085 (4.53%)	1041 (0.32%)	**1039 (0.08%)**	1053 (1.45%)	1038*
	Vdata_la14	1111 (3.83%)	1172 (9.53%)	1142 (6.73%)	1073 (0.26%)	**1071 (0.09%)**	1086 (1.50%)	1070*
	Vdata_la15	1113 (2.2%)	1138 (4.5%)	1126 (3.4%)	1092 (0.30%)	**1090 (0.09%)**	1111 (2.02%)	1089*
10 × 10	Vdata_la16	774 (7.95%)	754 (5.16%)	724 (0.98%)	718 (0.10%)	**717* (0.00%)**	**717*(0.00%)**	717*
	Vdata_la17	662 (2.48%)	660 (2.17%)	739 (14.4%)	**646* (0.00%)**	**646* (0.00%)**	647 (0.15%)	646*
	Vdata_la18	670 (1.06%)	728 (9.8%)	686 (3.47%)	**663* (0.00%)**	**663* (0.00%)**	**663* (0.00%)**	663*
	Vdata_la19	668 (8.27%)	688 (11.51%)	765 (23.99%)	644 (4.18%)	**620 (0.42%)**	626 (1.46%)	617*
	Vdata_la20	808 (6.88%)	768 (1.59%)	774 (2.38%)	**756* (0.00%)**	**756* (0.00%)**	**756*(0.00%)**	756*
15 × 10	Vdata_la21	929 (15.55%)	935 (16.29%)	997 (24.0%)	856 (6.13%)	**826 (3.20%)**	887 (10.32%)	804
	Vdata_la22	860 (16.85%)	846 (14.95%)	877 (19.16%)	781 (5.77%)	**751 (3.29%)**	793 (7.74%)	736
	Vdata_la23	878 (7.73%)	905 (11.04%)	961 (17.91%)	858 (5.11%)	**834 (3.09%)**	858 (5.28%)	815
	Vdata_la24	868 (12.0%)	898 (15.87%)	935 (20.65%)	823 (5.81%)	**796 (3.03%)**	883 (13.94%)	775
	Vdata_la25	937 (23.94%)	930 (23.02%)	914 (20.9%)	804 (6.02%)	**778 (3.54%)**	883 (16.80%)	756
20 × 10	Vdata_la26	1097 (4.08%)	1159 (9.96%)	1164 (10.44%)	1077 (2.13%)	**1065 (1.19%)**	1089 (3.32%)	1054
	Vdata_la27	1195 (10.24%)	1192 (9.96%)	1219 (12.45%)	1106 (1.90%)	**1099 (1.40%)**	1123 (3.60%)	1084
	Vdata_la28	1117 (4.39%)	1212 (13.27%)	1190 (11.21%)	1093 (2.07%)	**1084 (1.37%)**	1106 (3.36%)	1070
	Vdata_la29	1063 (6.94%)	1131 (13.78%)	1140 (14.69%)	1019 (2.47%)	**1007 (1.14%)**	1049 (5.53%)	994
	Vdata_la30	1127 (5.43%)	1182 (10.57%)	1164 (8.89%)	1095 (2.38%)	**1083 (1.42%)**	1117 (4.49%)	1069
30 × 10	Vdata_la31	1577 (3.75%)	1604 (5.53%)	1632 (7.37%)	1528 (0.54%)	**1526 (0.39%)**	1561 (2.70%)	1520*
	Vdata_la32	1717 (3.56%)	1774 (7.0%)	1807 (8.99%)	1666 (0.48%)	**1666 (0.53%)**	1693 (2.11%)	1658*
	Vdata_la33	1556 (3.94%)	1603 (7.08%)	1599 (6.81%)	1505 (0.54%)	**1504 (0.49%)**	1531 (2.27%)	1497*
	Vdata_la34	1620 (5.4%)	1596 (3.84%)	1611 (4.81%)	1543 (0.49%)	**1541 (0.38%)**	1562 (1.63%)	1537*
	Vdata_la35	1586 (2.39%)	1665 (7.49%)	1637 (5.68%)	1556 (0.47%)	**1556 (0.44%)**	1574 (1.61%)	1549*
15 × 15	Vdata_la36	1029 (8.54%)	1024 (8.02%)	1113 (17.41%)	991 (4.31%)	**949 (0.08%)**	985 (3.90%)	948*
	Vdata_la37	1047 (6.19%)	1059 (7.4%)	1165 (18.15%)	1045 (5.68%)	**993 (0.67%)**	1028 (4.26%)	986*
	Vdata_la38	973 (3.18%)	991 (5.09%)	1057 (12.09%)	957 (1.51%)	**943* (0.00%)**	948 (0.53%)	943*
	Vdata_la39	990 (7.38%)	1024 (11.06%)	1064 (15.4%)	979 (5.82%)	**928 (0.60%)**	979 (6.18%)	922*
	Vdata_la40	973 (1.88%)	1004 (5.13%)	1040 (8.9%)	972 (1.76%)	**955* (0.00%)**	968 (1.36%)	955*
Ave. Gap		8.54%	10.47%	11.85%	2.11%	**0.80%**	4.20%	0.00%

Note: the 'UB (Dauzère-Pérès & Paulli, 1997; Jurisch, 1992; Mastrolilli & Gambardella, 2000)' column is the best result from literature, and '*' denotes the result is optimal.

dispatching rules deteriorate as the scale increases. Although the baseline dispatching rules outperform our method slightly in computing time, it is immaterial when compared to the considerable solution quality improvement. The proposed method is more efficient than the Gurobi solver that takes 3600 s to solve each of the 20 × 20, and 30 × 20 FJSP instances. For a clear view, the Gantt charts of our method and the best dispatching rule solving for 30 × 20 FJSP instance are given in Fig. 7. By comparing the two Gantt charts, it is clear that the idle time between operations on each machine of the best dispatching rule is much more than ours.

Moreover, we also evaluated the generalization performance from small-scale training to large-scale testing. We directly test on the 50 × 20 and 100 × 20 FJSP instances using the models trained by 20 × 20, and 30 × 20 FJSP instances, respectively. The results of all methods are listed in Table 4, in which the Gap value is calculated by the smallest average objective of our method and the baseline methods due to the Gurobi solver cannot solve the large-scale FJSP instances in a reasonable time. We can observe that both MPGN models trained by 20 × 20, and 30 × 20 FJSP instances, respectively, outperform the baseline dispatching rules for all scales when testing in large-scale ones. Remarkably, the results of the MPGN trained by 30 × 20 instances are slightly better than the one trained by 20 × 20 instances. The performance demonstrates that the proposed method can distill insightful information from small-scale instances and generalize to large-scale ones.

Based on the performance of the proposed method on small-, medium-, and large-scale FJSP instances, we concluded that the proposed method could train a high-quality dispatching policy via randomly generated training data and has a good generalization performance from training on small-scale FJSP instances to testing on large-scale ones.

For further analyzing the computational results, the one-way analysis of variance (ANOVA) test is considered. Prima facie, we removed "FIFO_EET", "MOPNR_EET", "LWKR_EET", "MWKR_EET" from the statistical analysis since they are obviously worse than the other ones. Fig. 8 shows the means plot and 95% confidence level Tukey's Honestly Significant Difference (HSD) intervals for solution values of 128 instances of different methods and scales. If the Tukey HSD intervals of the two methods overlap, the performances of the paired methods are not statistically different. It can be seen from the figure that our method only has a slightly overlapping Tukey's HSD intervals with those of other methods for the instances of sizes 6 × 6 and 10 × 10. This fact clearly indicates that our method is statistically better than the other ones.

5.2. Computational results of benchmark instances

We performed experiments on Hurink's and Behnke's instances to evaluate the generalization performance from random instance training to real-world instance testing. For Hurink's instances, we put them into eight groups, each with five instances according to their scales. We

Table 6
Results of the other four dispatching rules and our method on Hurink' instances.

Instance		MWKR + SPT	MWKR + EET	MWKR + EET	MOPNR + EET	MWKR + SPT	Ours	UB
10 × 5	Vdata_la1	835 (46.49%)	847 (48.6%)	919 (61.23%)	922 (61.75%)	839 (47.19%)	**610 (7.02%)**	570*
	Vdata_la2	853 (61.25%)	835 (57.84%)	702 (32.7%)	984 (86.01%)	814 (53.88%)	**555 (4.91%)**	529*
	Vdata_la3	656 (37.53%)	676 (41.72%)	637 (33.54%)	738 (54.72%)	751 (57.44%)	**532 11.53%)**	477*
	Vdata_la4	681 (35.66%)	760 (51.39%)	794 (58.17%)	808 (60.96%)	828 (64.94%)	**530 (5.58%)**	502*
	Vdata_la5	689 (50.77%)	788 (72.43%)	696 (52.3%)	716 (56.67%)	708 (54.92%)	**507 (10.94%)**	457*
15 × 5	Vdata_la6	1196 (49.69%)	1177 (47.31%)	1046 (30.91%)	1390 (73.97%)	1316 (64.71%)	**820 (2.63%)**	799*
	Vdata_la7	893 (19.23%)	1025 (36.85%)	1064 (42.06%)	1121 (49.67%)	1195 (59.55%)	**757 (1.07%)**	749*
	Vdata_la8	1055 (37.91%)	1185 (54.9%)	1138 (48.76%)	1259 (64.58%)	1156 (51.11%)	**782 (2.22%)**	765*
	Vdata_la9	1077 (26.26%)	1147 (34.47%)	1110 (30.13%)	1292 (51.47%)	1065 (24.85%)	**879 (3.05%)**	853*
	Vdata_la10	1101 (36.94%)	1231 (53.11%)	949 (18.03%)	1357 (68.78%)	1248 (55.22%)	**862 (7.21%)**	804*
20 × 5	Vdata_la11	1429 (33.43%)	1569 (46.5%)	1301 (21.48%)	1477 (37.91%)	1522 (42.11%)	**1101 (2.80%)**	1071*
	Vdata_la12	1235 (31.94%)	1099 (17.41%)	1159 (23.82%)	1249 (33.44%)	1244 (32.91%)	**950 (1.50%)**	936*
	Vdata_la13	1249 (20.33%)	1289 (24.18%)	1354 (30.44%)	1524 (46.82%)	1363 (31.31%)	**1053 (1.45%)**	1038*
	Vdata_la14	1258 (17.57%)	1532 (43.18%)	1478 (38.13%)	1613 (50.75%)	1488 (39.07%)	**1086 (1.50%)**	1070*
	Vdata_la15	1387 (27.36%)	1429 (31.22%)	1486 (36.46%)	1715 (57.48%)	1376 (26.35%)	**1111 (2.02%)**	1089*
10 × 10	Vdata_la16	1139 (58.86%)	1067 (48.81%)	1204 (67.92%)	1241 (73.08%)	1367 (90.66%)	**717*(0.00%)**	717*
	Vdata_la17	970 (50.15%)	912 (41.18%)	1044 (61.61%)	1379 (113.47%)	1305 (102.01%)	**647 (0.15%)**	646*
	Vdata_la18	1132 (70.74%)	1053 (58.82%)	983 (48.27%)	1204 (81.6%)	1280 (93.06%)	**663* (0.00%)**	663*
	Vdata_la19	978 (58.51%)	999 (61.91%)	1437 (132.9%)	1188 (92.54%)	1252 (102.92%)	**626 (1.46%)**	617*
	Vdata_la20	1029 (36.11%)	1049 (38.76%)	1022 (35.19%)	1246 (64.81%)	1596 (111.11%)	**756*(0.00%)**	756*
15 × 10	Vdata_la21	1403 (74.5%)	1283 (59.58%)	1292 (60.7%)	1709 (112.56%)	1941 (141.42%)	**887 (10.32%)**	804
	Vdata_la22	1200 (63.04%)	1350 (83.42%)	1146 (55.71%)	1527 (107.47%)	1572 (113.59%)	**793 (7.74%)**	736
	Vdata_la23	1287 (57.91%)	1349 (65.52%)	1271 (55.95%)	1682 (106.38%)	1770 (117.18%)	**858 (5.28%)**	815
	Vdata_la24	1326 (71.1%)	1378 (77.81%)	1284 (65.68%)	2096 (170.45%)	1684 (117.29%)	**883 (13.94%)**	775
	Vdata_la25	1370 (81.22%)	1212 (60.32%)	1393 (84.26%)	1979 (161.77%)	1547 (104.63%)	**883 (16.80%)**	756
20 × 10	Vdata_la26	1611 (52.85%)	1551 (47.15%)	1393 (32.16%)	2022 (91.84%)	2087 (98.01%)	**1089 (3.32%)**	1054
	Vdata_la27	1754 (61.81%)	1630 (50.37%)	1677 (54.7%)	2401 (121.49%)	2384 (119.93%)	**1123 (3.41%)**	1084
	Vdata_la28	1596 (49.16%)	1500 (40.19%)	1693 (58.22%)	2198 (105.42%)	2366 (121.12%)	**1106 (3.36%)**	1070
	Vdata_la29	1643 (65.29%)	1533 (54.23%)	1423 (43.16%)	2240 (125.35%)	2038 (105.03%)	**1049 (5.43%)**	994
	Vdata_la30	1759 (64.55%)	1545 (44.53%)	1508 (41.07%)	2267 (112.07%)	1808 (69.13%)	**1117 (4.20%)**	1069
30 × 10	Vdata_la31	2081 (36.91%)	2141 (40.86%)	2553 (67.96%)	2525 (66.12%)	2402 (58.03%)	**1561 (2.56%)**	1520*
	Vdata_la32	2277 (37.33%)	2444 (47.41%)	2454 (48.01%)	2736 (65.02%)	2839 (71.23%)	**1693 (1.93%)**	1658*
	Vdata_la33	2305 (53.97%)	2010 (34.27%)	2198 (46.83%)	2561 (71.08%)	2587 (72.81%)	**1531 (2.07%)**	1497*
	Vdata_la34	2248 (46.26%)	2064 (34.29%)	2242 (45.87%)	2477 (61.16%)	2717 (76.77%)	**1562 (1.63%)**	1537*
	Vdata_la35	2098 (35.44%)	2010 (29.76%)	2174 (40.35%)	2629 (69.72%)	2653 (71.27%)	**1574 (1.48%)**	1549*
15 × 15	Vdata_la36	1600 (68.78%)	1512 (59.49%)	1851 (95.25%)	2078 (119.2%)	2049 (116.14%)	**985 (3.90%)**	948*
	Vdata_la37	1700 (72.41%)	1710 (73.43%)	1498 (51.93%)	2178 (120.89%)	1917 (94.42%)	**1028 (4.26%)**	986*
	Vdata_la38	1466 (55.46%)	1430 (51.64%)	1449 (53.66%)	2235 (137.01%)	1784 (89.18%)	**948 (0.53%)**	943*
	Vdata_la39	1679 (82.1%)	1628 (76.57%)	1713 (85.79%)	1944 (110.85%)	2079 (125.49%)	**979 (6.18%)**	922*
	Vdata_la40	1584 (65.86%)	1596 (67.12%)	1495 (56.54%)	1925 (101.57%)	2367 (147.85%)	**968 (1.36%)**	955*
Ave. Gap		50.07%	50.21%	51.20%	80.90%	85.45%	**4.20%**	0.00%

Note: the 'UB (Dauzère-Pérès & Paulli, 1997; Jurisch, 1992; Mastrolilli & Gambardella, 2000)' column is the best result from literature, and '*' denotes the result is optimal.

trained policies for solving three groups, i.e., 10 × 5, 10 × 10 , and 15 × 15, and other groups are solved by the trained three policies for testing generalization performance.Moreover, we directly compared the results of the proposed method with the state-of-the-art *meta*-heuristic algorithms, including a regular genetic algorithm (RGA) and a two-stage genetic algorithm (2SGA) (Defersha & Rooyani, 2020). The average results of the existing *meta*-heuristics of the reference (Defersha & Rooyani, 2020), the top three dispatching rules and our method are listed in Table 5. Besides, the results of the remaining five dispatching rules and our method are also listed in Table 6.

The results provided in Tables 5 and 6 consist of the solution value and the gap to 'UB'(the best result found by the existing algorithms in literature). It can be found that our method significantly outperforms all the dispatching rules when training on small-scale instances and generalizing on larger-scale ones, i.e., 10 × 5 instance training for 10 × 5 and 15 × 5 one testing, 10 × 10 instance training for 10 × 10, 15 × 10, 20 × 10, 30 × 10 one testing, and 15 × 15 instance training for 15 × 15 one testing.

Although the results of RGA and 2SGA surpass ours in gap value (4.20% vs. 2.11% vs. 0.80%), the running time of the proposed method is much less than the RGA and 2SGA. The two *meta*-heuristics proposed by reference (Defersha & Rooyani, 2020) have conducted a total of 1600 experiment runs for each instance and each experimental run consumes 1 to 30 min depending on the problem size and the GA parameter selected, respectively. In contrast, the running time of the proposed method is 0.07 s to 0.41 s for different problem sizes. The proposed method is a good trade-off between result quality and computational time for a real-time environment. Overall, compared with the *meta*-heuristic algorithms, the parallelization of both models and data can be used in neural networks. There is no need to update the model's parameters in the testing phase, so a larger batch size can be used if memory allows. Moreover, the success of *meta*-heuristic algorithms such as GA, PSO, ABC et al., comes from repeated iterations and multiple runs for an instance. In contrast, our model is well trained in the proposed framework, and it can solve various problem instances with different sizes. That means our model can be trained with small-sized instances and used to solve larger ones. Furthermore, the results of the trained model by the randomly generated data on benchmark instances demonstrate that the proposed method has a good generalization performance from random instance training to real-world instance testing.

Furthermore, we tested the larger Behnke's instances with up to 100 × 60 scale using the proposed method, which is categorized into nine groups according to their scales. We trained policies with 20 × 20, 20 × 40, and 20 × 60 instances. These trained policies are used to solve other instances in those three groups, i.e., 20 × 20, 20 × 40, and 20 × 60, respectively. Furthermore, the instances in other groups of larger scales are solved by the trained three policies for testing generalization performance. The results of the ranking of the top four dispatching rules in performance and our method are listed in Table 7, and the results of other dispatching rules can be found in Table 8. Unfortunately, there are

Table 7
Results of the ranking of the top four dispatching rules in performance and our method on Behnke's instances.

Instance		FIFO + SPT	MOPNR + SPT	LWKR + SPT	MWKR + SPT	Ours	UB
20 × 20	1	185 (41.22%)	221 (68.70%)	211 (61.07%)	214 (63.36%)	**143 (9.16%)**	131
	2	159 (22.31%)	201 (54.62%)	199 (53.08%)	202 (55.38%)	**142 (9.23%)**	130
	3	187 (46.09%)	220 (71.88%)	239 (86.72%)	220 (71.88%)	**139 (8.59%)**	128
	4	189 (46.51%)	205 (58.91%)	198 (53.49%)	233 (80.62%)	**144 (11.63%)**	129
	5	206 (54.89%)	197 (48.12%)	221 (66.17%)	200 (50.38%)	**146 (9.77%)**	133
50 × 20	6	317 (22.39%)	367 (41.70%)	345 (33.2%)	347 (33.98%)	**250 (-3.47%)**	259
	7	310 (23.51%)	339 (35.06%)	340 (35.46%)	355 (41.43%)	**247 (-1.59%)**	251
	8	309 (22.62%)	369 (46.43%)	339 (34.52%)	351 (39.29%)	**249 (-1.19%)**	252
	9	330 (27.91%)	379 (46.9%)	338 (31.01%)	360 (39.53%)	**257 (-0.39%)**	258
	10	325 (24.05%)	375 (43.13%)	387 (47.71%)	378 (44.27%)	**255 (-2.67%)**	262
100 × 20	11	564 (-0.35%)	609 (7.6%)	592 (4.59%)	635 (12.19%)	**437 (–22.79%)**	566
	12	553 (3.36%)	657 (22.8%)	647 (20.93%)	654 (22.24%)	**430 (-19.63%)**	535
	13	495 (-10.81%)	595 (7.21%)	563 (1.44%)	601 (8.29%)	**428 (–22.88%)**	555
	14	519 (-2.44%)	595 (11.84%)	570 (7.14%)	657 (23.5%)	**423 (-20.49%)**	532
	15	520 (-0.38%)	620 (18.77%)	574 (9.96%)	567 (8.62%)	**427 (-18.20%)**	522
20 × 40	16	153 (25.41%)	170 (39.34%)	174 (42.62%)	170 (39.34%)	**129 (5.74%)**	122
	17	180 (36.36%)	179 (35.61%)	201 (52.27%)	186 (40.91%)	**140 (6.06%)**	132
	18	157 (27.64%)	174 (41.46%)	211 (71.54%)	177 (43.9%)	**133 (8.13%)**	123
	19	140 (12.00%)	158 (26.4%)	154 (23.2%)	179 (43.2%)	**138 (10.40%)**	125
	20	201 (58.27%)	205 (61.42%)	190 (49.61%)	204 (60.63%)	**142 (11.81%)**	127
50 × 40	21	298 (9.56%)	336 (23.53%)	347 (27.57%)	316 (16.18%)	**271 (-0.37%)**	272
	22	298 (15.06%)	315 (21.62%)	316 (22.01%)	296 (14.29%)	**267 (3.09%)**	259
	23	273 (11.43%)	309 (26.12%)	309 (26.12%)	310 (26.53%)	**261 (6.53%)**	245
	24	271 (2.26%)	305 (15.09%)	313 (18.11%)	333 (25.66%)	**250 (-5.66%)**	265
	25	263 (3.95%)	264 (4.35%)	284 (12.25%)	329 (30.04%)	**249 (-1.58%)**	253
100 × 40	26	537 (1.13%)	530 (-0.19%)	556 (4.71%)	523 (-1.51%)	**442 (-16.76%)**	531
	27	533 (-0.56%)	540 (0.75%)	532 (-0.75%)	562 (4.85%)	**444 (-17.16%)**	536
	28	457 (-13.28%)	503 (-4.55%)	547 (3.8%)	513 (-2.66%)	**454 (-13.85%)**	527
	29	551 (6.78%)	571 (10.66%)	578 (12.02%)	586 (13.57%)	**471 (-8.72%)**	516
	30	531 (1.92%)	573 (9.98%)	582 (11.71%)	653 (25.34%)	**460 (-11.71%)**	521
20 × 60	31	185 (49.19%/)	193 (55.65%)	183 (47.58%)	168 (35.48%)	**145 (16.94%)**	124
	32	156 (23.81%)	**142 (12.7%)**	169 (34.13%)	162 (28.57%)	144 (14.29%)	126
	33	167 (24.63%)	180 (34.33%)	182 (35.82%)	158 (17.91%)	**141 (5.22%)**	134
	34	161 (33.06%)	187 (54.55%)	184 (52.07%)	192 (58.68%)	**134 (10.74%)**	121
	35	186 (41.98%)	168 (28.24%)	195 (48.85%)	171 (30.53%)	**147 (12.21%)**	131
50 × 60	36	290 (11.97%)	302 (16.6%)	281 (8.49%)	291 (12.36%)	**253 (-2.32%)**	259
	37	269 (5.49%)	281 (10.2%)	302 (18.43%)	321 (25.88%)	**242 (-5.1%)**	255
	38	280 (8.95%)	326 (26.85%)	305 (18.68%)	322 (25.29%)	**256 (-0.39%)**	257
	39	303 (13.48%)	339 (26.97%)	333 (24.72%)	329 (23.22%)	**268 (0.37%)**	267
	40	284 (10.94%)	327 (27.73%)	310 (21.09%)	334 (30.47%)	**262 (2.34%)**	256
100 × 60	41	493 (-8.36%_	514 (-4.46%)	518 (-3.72%)	523 (-2.79%)	**439 (-18.4%)**	538
	42	518 (-3.18%)	541 (1.12%)	549 (2.62%)	575 (7.48%)	**442 (-17.38%)**	535
	43	465 (-12.43%)	538 (1.32%)	556 (4.71%)	540 (1.69%)	**442 (-16.76%)**	531
	44	470 (-11.65%)	513 (-3.57%)	566 (6.39%)	470 (-11.65%)	**443 (-16.73%)**	532
	45	521 (-2.98%)	534 (-0.56%)	506 (-5.77%)	494 (-8.01%)	**458 (-14.71%)**	537
Ave. Gap		15.64%	26.29%	27.50%	27.79%	**−2.64%**	0.00%

Note: the 'UB (Behnke & Geiger, 2012)' column is the best result from literature, and '*' denotes the result is optimal.

no more public computational results for our comparison except for (Behnke & Geiger, 2012). In Tables 7 and 8, we listed the results generated by constraint programming with 60 min running time from (Behnke & Geiger, 2012) in column 'UB'. It can be seen that our method still outperforms all dispatching rules and is better than the 'UB' according to the average gap.

Moreover, we noted that the performance of the dispatching rules is unstable on both Hurink's instances and Behnke's instances. For example, the ranking of top four dispatching rules in performance are "FIFO + EET", "MWKR + EET", "MOPNR + EET", and "MWKR + SPT" on Hurink's instances, while those are "FIFO + SPT", "MOPNR + SPT", "LWKR + SPT", and "MWKR + SPT" on Behnke's instances. In contrast, our method has stable performance for both two benchmark groups. Moreover, the running time of our method has the big advantage compared with the existing *meta*-heuristics and constraint programming, and has the almost perfect scale with that of the dispatching rules.

5.3. Further discussion

The proposed end-to-end DRL framework can achieve better performance in terms of solution quality and running time on FJSP compared with other existing algorithms, such as dispatching rules and *meta*-heuristic algorithms. Meanwhile, our method also has superiority compared with other DRL-based methods. Moreover, our architecture has shown the potential (low complexity in running time) to be applied in more complicated FJSP scenarios with dynamics and uncertainty and extend to other types of scheduling problems (e.g., flow-shop and open-shop) because those problems can also be represented by disjunctive graphs.

Some works have integrated the DRL technologies into the traditional optimization research methods, such as the heuristic dispatching rules and *meta*-heuristic algorithms. In contrast, the proposed end-to-end DRL architecture in which the DRL agent directly outputs the solution by extracting the information features of the instance input without designing and adopting traditional optimization research methods. In this way, the proposed framework has a similar complexity in running time compared to the dispatching rules and has robust generalization and strong performance.

Based on the advantages of our design of policy networks, our architecture is not bounded by the instance size (numbers of job/operation and machine), unlike previous DRL-based work (Park, et al., 2020). Moreover, the success of most *meta*-heuristic algorithms such as GA

Table 8
Results of the other four dispatching rules and our method on Behnke's instances.

Instance		FIFO + EET	MOPNR + EET	LWKR + EET	MWKR + EET	Ours	UB
20 × 20	1	197 (50.38%)	251 (91.6%)	246 (87.79 %)	257 (96.18%)	**143 (9.16%)**	131
	2	204 (56.92%)	260 (100%)	253 (94.62%)	247 (90%)	**142 (9.23%)**	130
	3	193 (50.78%)	259 (102.34%)	253 (97.66%)	255 (99.22%)	**139 (8.59%)**	128
	4	191 (48.06%)	259 (100.78%)	252 (95.35%)	256 (98.45%)	**144 (11.63%)**	129
	5	210 (57.89%)	264 (98.5%)	243 (82.71%)	257 (93.23%)	**146 (9.77%)**	133
50 × 20	6	355 (37.07%)	492 (89.96%)	478 (84.56%)	519 (100.39%)	**250 (-3.47%)**	259
	7	357 (42.23%)	473 (88.45%)	520 (107.17%)	531 (111.55%)	**247 (-1.59%)**	251
	8	367 (45.63%)	496 (96.83%)	514 (103.97%)	538 (113.49%)	**249 (-1.19%)**	252
	9	360 (39.53%)	456 (76.74%)	511 (98.06%)	512 (98.45%)	**257 (-0.39%)**	258
	10	371 (41.6%)	465 (77.48%)	484 (84.73%)	503 (91.98%)	**255 (-2.67%)**	262
100 × 20	11	630 (11.31%)	782 (38.16%)	891 (57.42%)	914 (61.48%)	**437 (–22.79%)**	566
	12	610 (14.02%)	844 (57.76%)	918 (71.59%)	903 (68.79%)	**430 (-19.63%)**	535
	13	608 (9.55%)	828 (49.19%)	880 (58.56%)	893 (60.9%)	**428 (–22.88%)**	555
	14	629 (18.23%)	789 (48.31%)	902 (69.55%)	866 (62.78%)	**423 (-20.49%)**	532
	15	612 (17.24%)	821 (57.28%)	893 (71.07%)	910 (74.33%)	**427 (-18.20%)**	522
20 × 40	16	189 (54.92%)	223 (82.79%)	190 (55.74%)	207 (69.67%)	**129 (5.74%)**	122
	17	202 (53.03%)	238 (80.3%)	217 (64.39%)	228 (72.73%)	**140 (6.06%)**	132
	18	201 (63.41%)	222 (80.49%)	215 (74.8%)	218 (77.24%)	**133 (8.13%)**	123
	19	198 (58.4%)	217 (73.6%)	224 (79.2%)	227 (81.6%)	**138 (10.40%)**	125
	20	189 (48.82%)	208 (63.78%)	239 (88.19%)	228 (79.53%)	**142 (11.81%)**	127
50 × 40	21	368 (35.29%)	427 (56.99%)	435 (59.93%)	441 (62.13%)	**271 (-0.37%)**	272
	22	370 (42.86%)	416 (60.62%)	411 (58.69%)	434 (67.57%)	**267 (3.09%)**	259
	23	347 (41.63%)	398 (62.45%)	415 (69.39%)	430 (75.51%)	**261 (6.53%)**	245
	24	362 (36.6%)	404 (52.45%)	415 (56.6%)	402 (51.7%)	**250 (-5.66%)**	265
	25	386 (52.57%)	385 (52.17%)	423 (67.19%)	396 (56.52%)	**249 (-1.58%)**	253
100 × 40	26	604 (13.75%)	699 (31.64%)	730 (37.48%)	737 (38.79%)	**442 (-16.76%)**	531
	27	601 (12.13%)	695 (29.66%)	730 (36.19%)	747 (39.37%)	**444 (-17.16%)**	536
	28	573 (8.73%)	693 (31.5%)	746 (41.56%)	723 (37.19%)	**454 (-13.85%)**	527
	29	631 (22.29%)	663 (28.49%)	710 (37.6%)	725 (40.5%)	**471 (-8.72%)**	516
	30	606 (16.31%)	693 (33.01%)	710 (36.28%)	748 (43.57%)	**460 (-11.71%)**	521
20 × 60	31	184 (48.39%)	219 (76.61%)	207 (66.94%)	217 (75%)	**145 (16.94%)**	124
	32	208 (65.08%)	222 (76.19%)	203 (61.11%)	229 (81.75%)	**144 (14.29%)**	126
	33	232 (73.13%)	226 (68.66%)	218 (62.69%)	244 (82.09%)	**141 (5.22%)**	134
	34	189 (56.2%)	204 (68.6%)	215 (77.69%)	230 (90.08%)	**134 (10.74%)**	121
	35	210 (60.31%)	225 (71.76%)	227 (73.28%)	233 (77.86%)	**147 (12.21%)**	131
50 × 60	36	361 (39.38%)	396 (52.9%)	407 (57.14%)	420 (62.16%)	**253 (-2.32%)**	259
	37	361 (41.57%)	399 (56.47%)	426 (67.06%)	418 (63.92%)	**242 (-5.1%)**	255
	38	351 (36.58%)	393 (52.92%)	435 (69.26%)	424 (64.98%)	**256 (-0.39%)**	257
	39	358 (34.08%)	430 (61.05%)	414 (55.06%)	432 (61.8%)	**268 (0.37%)**	267
	40	356 (39.06%)	390 (52.34%)	417 (62.89%)	447 (74.61%)	**262 (2.34%)**	256
100 × 60	41	599 (11.34%)	667 (23.98%)	727 (35.13%)	742 (37.92%)	**439 (-18.4%)**	538
	42	615 (14.95%)	688 (28.6%)	721 (34.77%)	726 (35.7%)	**442 (-17.38%)**	535
	43	617 (16.2%)	684 (28.81%)	723 (36.16%)	699 (31.64%)	**442 (-16.76%)**	531
	44	616 (15.79%)	688 (29.32%)	685 (28.76%)	722 (35.71%)	**443 (-16.73%)**	532
	45	608 (13.22%)	683 (27.19%)	685 (27.56%)	711 (32.4%)	**458 (-14.71%)**	537
Ave. Gap		37.03%	61.53%	65.41%	69.39%	**−2.64%**	0.00%

Note: the 'UB (Behnke & Geiger, 2012)' column is the best result from literature, and '*' denotes the result is optimal.

comes from repeated iterations and multiple runs for an instance. In contrast, once the proposed model is well trained in the proposed framework, it can solve various problem instances with different sizes through single running.

Compared with exact algorithms and heuristic algorithms, the parallelization of both models and data can be used in neural networks. There is no need to update the model's parameters in the testing phase, so a larger batch size can be used if memory allows. Furthermore, the proposed framework can train model with different problem sizes using the same hyperparameters. Besides, Fig. 9 in Appendix B shows that the proposed framework is not sensitive to different hyper-parameters. Therefore, it can be used directly without hyper-parameter optimization when extending our framework to other graph combinatorics optimization problems.

6. Conclusion

In this paper, an end-to-end DRL-based framework is introduced to learn a dispatching policy for minimizing the *makespan* of FJSP. We proposed a multi-MDPs formulation for FJSP and introduced novel representations of states, actions, and rewards to address the complex scheduling problem. We introduced the disjunctive graph representation of FJSP to represent local states and used the graph neural network to embed local states. Then we designed a neural network to learn two policies, a job operation action policy and a machine action policy. The two policies are used to predict a probability distribution over job operations and machines, respectively. Moreover, the trained policy network is able to deal with unseen instances with different scales in terms of jobs, operations, and eligible machines. Meanwhile, a multi-Proximal Policy Optimization algorithm is designed to train the neural network. To evaluate the generalization of the proposed framework, the policy network is trained by randomly generated instances to test on the publicly available benchmarks and trained by small-scale instances to test on large-scale ones. Furthermore, the experiments on random instances and public benchmarks show that the proposed method significantly outperforms the existing dispatching rules.

A valuable direction for future investigation is the development of the proposed framework to address the multiple objective FJSP (MO-FJSP). Evolutionary algorithms like the genetic algorithm (GA) have been studied for treating multiple objectives conflicting with each other such as makespan, mean completion time, total workload (Gao, et al., 2007), and a sequence dependence & setup time (Chou, Chien, & Gen,

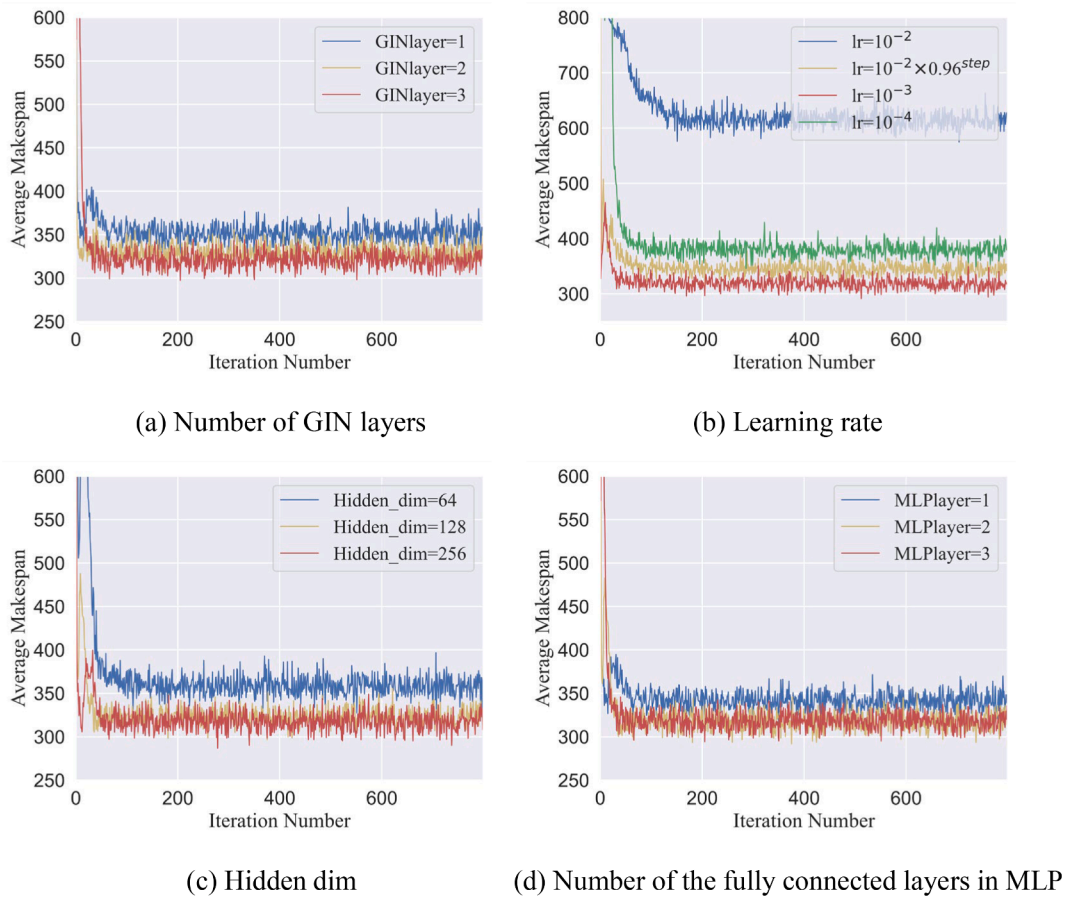


Fig. 9. Sensitivity analyses of hyper-parameters in our training process.

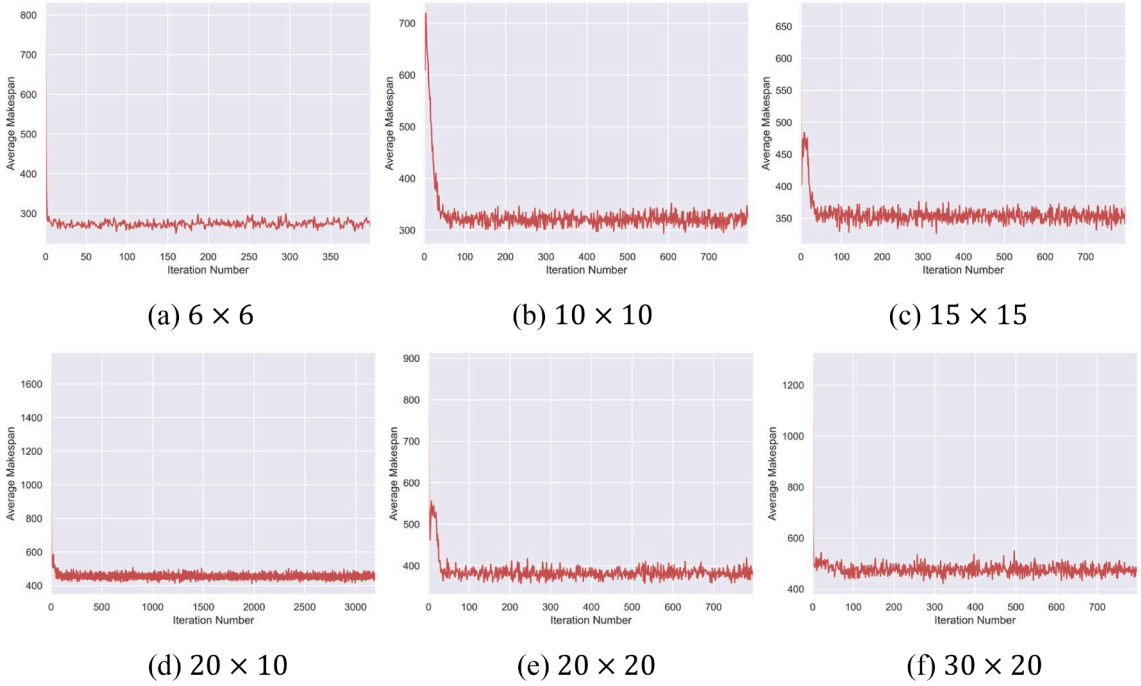


Fig. 10. The convergence curves of average *makespan* for each iteration in the training phase.

2014). While EAs have shown success in multi-objective optimization, the running time is often too long (Gao, et al., 2022). It is interesting to expand the proposed framework to multi-objective optimization by redefining the reward and the raw feature of the state in the proposed multi-MDPs. Moreover, the proposed framework has shown great potential (low complexity in running time) to be applied in real-time FJSP scenarios with dynamics and uncertainty, such as random job arrivals and unpredictable machine breakdowns.

CRediT authorship contribution statement

Kun Lei: Conceptualization, Methodology, Writing – original draft. **Peng Guo:** Conceptualization, Methodology, Supervision, Writing – review & editing. **Wenchao Zhao:** Visualization. **Yi Wang:** Writing – review & editing. **Linmao Qian:** Investigation, Funding acquisition. **Xiangyin Meng:** Data curation, Funding acquisition. **Liansheng Tang:** 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.

Acknowledgements

The research of this paper is supported by the National Key Research and Development Program of China (grant number 2020YFB1712200), the MOE (Ministry of Education in China) Project of Humanities and Social Sciences (No. 21YJC630034) and the University Major Research Program of Humanities and Social Sciences of Zhejiang Province (grant number 2018QN060).

Appendix A. . The details of the baseline dispatching rules

In this section, we formally introduce the baseline dispatching rules for FJSP. An FJSP can be divided into two sub-problems, a job sequencing problem and a machine selection problem. Hence an FJSP needs two types of dispatching rules. A job dispatching rule is used to select a job operation from all unscheduled job operations to be scheduled. Moreover, a machine selection rule is used to select a machine after an operation of a job is scheduled. This paper adopts well-known four job sequencing rules and two machine selection rules combined into eight compound dispatching rules as the baseline methods. Before introducing the two types of dispatching rules, their notations are summarized in Table 9.

Job sequencing rules:

(a) FIFO (First in First Out): select the earliest arriving job in the queue of a machine.
(b) MOPNR (Most Operation Number Remaining): select a job that has the greatest number of remaining operations to be done. The priority index for selecting the operation can be calculated via:$z_{ij}= n_i - b_{ij} + 1, \forall i \in \mathscr{J}, j \in \mathscr{O}_i$.
(c) LWKR (Least Work Remaining): select a job that has the least total processing time remaining to be done. The priority index for determining the operation can be calculated via:$z_{ij} = \sum_j^{n_i} \min\left(p_{ijk}\right), k \in \mathscr{M}_{ij}, \forall i \in \mathscr{J}, j \in \mathscr{O}_i$.
(d) MWKR (Most Work Remaining): select a job that has the most total processing time remaining to be done. The priority index for determining the operation can be calculated via:$z_{ij} = \sum_j^{n_i} \max\left(p_{ijk}\right), k \in \mathscr{M}_{ij}, \forall i \in \mathscr{J}, j \in \mathscr{O}_i$.

Table 9
The notations for dispatching rules.

z_{ij}	the priority index of operation O_{ij} ($\forall i \in \mathscr{J}, j \in \mathscr{O}_i$).
n_i	the number of operations for job J_i.
b_{ij}	the number of scheduled operations when handling operation O_{ij}.
$\mathscr{M}_{ij}$	A set of machines that can process operation O_{ij}.
p_{ijk}	the processing time of operation O_{ij} on machine k ($\forall i \in \mathscr{J}, j \in \mathscr{O}_i, k \in \mathscr{M}_{ij}$).

Machine selection rules:

(a) SPT (Shortest Processing Time): select a machine with the shortest processing time for the operation of a job. The priority index for determining the operation can be calculated via:$z_{ij} = p_{ijk}, k \in \mathscr{M}_{ij, \forall i \in \mathscr{J}\, j \in \mathscr{O}_i}$.
(b) EET (Earliest End Time): select a machine that is idle at the earliest time.

Appendix B. . The sensitivity analysis of several key hyperparameters

In this section, we conducted a sensitivity analysis of hyperparameters in the training process. We mainly analyzed four key hyperparameters on how they affect the quality of the proposed framework in terms of solution performance and training time, including the learning rate, hidden dim, number of GIN layers, and number of the fully connected layer in MLP. First, we re-trained the model by setting the number of GIN layers or fully connected layers in MLP to [1, 2, 3] and the hidden dim as [64, 128, 256]. For the learning rate, we used constant levels of 10^{-4}, 10^{-3}, 10^{-2}, and a decay one $10^{-2} \times 0.96^{step}$ where *step* equals the iteration number divided by ten and rounded up.

Fig. 9 shows the convergence curves of the training process on the four hyperparameters. It can be found that the results tend to get better with an increased number of GIN layers, but the training time is 0.74 h, 0.91 h, and 1.21 h for layers 1, 2, and 3, respectively. We found that GIN layer equals 2 is a good trade-off between the quality of the solutions and training time. For the same reasons, we use hidden dim with 128 and MLP layer with 2 in the training process.

It can be seen that learning rate 10^{-2} without decay is much worse than the other two in terms of the quality of the solution. Learning rate 10^{-3} can get the best performance of the proposed model. Therefore, we decided to use a learning rate of 10^{-3} to train the proposed model for all problem sizes.

Appendix C. . The convergence curves of average *makespan* for problems of different scales.

This section showed the training convergence curves of average *makespan* for all problem scales, including $6 \times 6, 10 \times 10, 15 \times 15, 20 \times 10, 20 \times 20$, and 30×20. In each curve, an iteration represents the average *makespan* of a batch. The training time for each scale problem is 0.25 h, 0.91 h, 2.8 h, 2.3 h, 7.8 h, 19.5 h.

References

S. Amizadeh S. Matusevych M. Weimer Learning to solve circuit-SAT: An unsupervised differentiable approach In International Conference on Learning Representations 2018 URL.

Baykasoglu, A. (2002). Linguistic-based meta-heuristic optimization model for flexible job shop scheduling. *International Journal of Production Research, 40*, 4523–4543.Doi: 10.1080/00207540210147043.

Behnke, D., & Geiger, M. (2012). Test instances for the flexible job shop scheduling problem with work centers. URL: *Research Report*, 1–31 http://edoc.sub.uni-hamburg.de/hsu/volltexte/2012/2982/.

Bengio, Y., Lodi, A., & Prouvost, A. (2021). Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research, 290*, 405–421.Doi: 10.1016/j.ejor.2020.07.063.

Bożejko, W., Uchroński, M., & Wodecki, M. (2010). Parallel hybrid metaheuristics for the flexible job shop problem. *Computers & Industrial Engineering, 59*, 323–333.Doi: 10.1016/j.cie.2010.05.004.

Brucker, P., & Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing, 45*, 369–375.https://doi.org/10.1007/BF02238804.

Chaudhry, I. A., & Khan, A. A. (2016). A research survey: Review of flexible job shop scheduling techniques. *International Transactions in Operational Research, 23*, 551–591.Doi: 10.1111/itor.12199.

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

X. Chen Y. Tian Learning to perform local rewriting for combinatorial optimization In Proceedings of the 33rd International Conference on Neural Information Processing Systems 2019 6281 6292 URL.

Chiang, T. C., & Lin, H. J. (2013). A simple and effective evolutionary algorithm for multiobjective flexible job shop scheduling. *International Journal of Production Economics, 141*, 87–98.Doi: 10.1016/j.ijpe.2012.03.034.

Chou, C., Chien, C., & Gen, M. (2014). A Multiobjective Hybrid Genetic Algorithm for TFT-LCD Module Assembly Scheduling. *IEEE Transactions on Automation Science and Engineering, 11*, 692–705.https://doi.org/10.1109/TASE.2014.2316193.

Corman, F., D'Ariano, A., Pacciarelli, D., & Pranzo, M. (2014). Dispatching and coordination in multi-area railway traffic management. *Computers & Operations Research, 44*, 146–160.Doi: 10.1016/j.cor.2013.11.011.

H. Dai E. Khalil Y. Zhang B. Dilkina L. Song Learning Combinatorial Optimization Algorithms over Graphs In 31st Conference on Neural Information Processing Systems 2017 URL:https://proceedings.neurips.cc/paper/2017/file/d9896106ca98d3d05b8cbdf4fd8b13a1-Paper.pdf.

Dauzère-Pérès, S., & Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research, 70*, 281–306.https://doi.org/10.1023/A:1018930406487.

Defersha, F., & Rooyani, D. (2020). An efficient two-stage genetic algorithm for a flexible job-shop scheduling problem with sequence dependent attached/detached setup, machine release date and lag-time. *Computers & Industrial Engineering, 147*, 106605.Doi: 10.1016/j.cie.2020.106605.

Demir, Y., & Kürşat İşleyen, S. (2013). Evaluation of mathematical models for flexible job-shop scheduling problems. *Applied Mathematical Modelling, 37*, 977–988.Doi: 10.1016/j.apm.2012.03.020.

Doh, H., Yu, J., Kim, J., Lee, D., & Nam, S. (2013). A priority scheduling approach for flexible job shops with multiple process plans. *International Journal of Production Research, 51*, 3748–3764.https://doi.org/10.1080/00207543.2013.765074.

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

Gao, J., Gen, M., Sun, L., & Zhao, X. (2007). A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job shop scheduling problems. *Computers & Industrial Engineering, 53*, 149–162.Doi: 10.1016/j.cie.2007.04.010.

Gao, K., Suganthan, P., Pan, Q., Chua, T., Cai, T., & Chong, C. (2016). Discrete harmony search algorithm for flexible job shop scheduling problem with multiple objectives. *Journal of Intelligent Manufacturing, 27*, 363–374.https://doi.org/10.1007/s10845-014-0869-8.

Gao, L.-y., Wang, R., Liu, C., & Jia, Z.-h. (2022). Multi-objective Pointer Network for Combinatorial Optimization. *arXiv preprint arXiv:.11860*. https://doi.org/10.48550/arXiv.2204.11860.

Garey, M., Johnson, D., & Sethi, R. (1976). The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research, 1*, 117–129. https://doi.org/10.1287/moor.1.2.117

González, M. A., Vela, C. R., & Varela, R. (2015). Scatter search with path relinking for the flexible job shop scheduling problem. *European Journal of Operational Research, 245*, 35–45.Doi: 10.1016/j.ejor.2015.02.052.

M. Hameed A. Schwung Reinforcement Learning on Job Shop Scheduling Problems Using Graph Networks. arXiv e-prints 2020 arXiv:2009.03836.

Han, B., & Yang, J. (2020). Research on Adaptive Job Shop Scheduling Problems Based on Dueling Double DQN. *IEEE. Access, 8*, 186474–186495.https://doi.org/10.1109/ACCESS.2020.3029868.

Hu, L., Liu, Z., Hu, W., Wang, Y., Tan, J., & Wu, F. (2020). Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinforcement learning with graph convolutional network. *Journal of Manufacturing Systems, 55*, 1–14.Doi: 10.1016/j.jmsy.2020.02.004.

Hurink, J., Jurisch, B., & Thole, M. (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spektrum, 15*, 205–215.https://doi.org/10.1007/BF01719451.

Jain, A. S., & Meeran, S. (1999). Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research, 113*, 390–434.Doi: 10.1016/S0377-2217(98)00113-1.

Jurisch, B. (1992). *Scheduling jobs in shops with multi-purpose machines*. Germany: University of Osnabrück.

Kacem, I., Hammadi, S., & Borne, P. (2002). Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 32*, 1–13.https://doi.org/10.1109/TSMCC.2002.1009117.

W. Kool H. van Hoof M. Welling Attention, Learn to Solve Routing Problems! In International Conference on Learning Representations 2019 URL.

Lange, J., & Werner, F. (2018). Approaches to modeling train scheduling problems as job-shop problems with blocking constraints. *Journal of Scheduling, 21*, 191–207. https://doi.org/10.1007/s10951-017-0526-0.

Lei, K., Guo, P., Wang, Y., Wu, X., & Zhao, W. (2021). Solve routing problems with a residual edge-graph attention neural network. *arXiv e-prints*, arXiv:2105.02730. https://doi.org/10.48550/arXiv.2105.02730.

Li, J., Pan, Q., & Gao, K. (2011). Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems. *The International Journal of Advanced Manufacturing Technology, 55*, 1159–1169.https://doi.org/10.1007/s00170-010-3140-2.

Li, J., Pan, Q., & Liang, Y. (2010). An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems. *In Computers & Industrial Engineering, 59*, 647–662. https://doi.org/10.1016/j.cie.2010.07.014

Li, Z., Chen, Q., & Koltun, V. (2018). Combinatorial optimization with graph convolutional networks and guided tree search. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 31, 537-546. URL: https://proceedings.neurips.cc/paper/2018/file/8d3bba7425e7c98c50f52ca1b52d3735-Paper.pdf.

Lin, C., Deng, D., Chih, Y., & Chiu, H. (2019). Smart Manufacturing Scheduling With Edge Computing Using Multiclass Deep Q Network. *IEEE Transactions on Industrial Informatics, 15*, 4276–4284.https://doi.org/10.1109/TII.2019.2908210.

Liu, C. L., Chang, C. C., & Tseng, C. J. (2020). Actor-Critic Deep Reinforcement Learning for Solving Job Shop Scheduling Problems. *IEEE. Access, 8*, 71752–71762.https://doi.org/10.1109/ACCESS.2020.2987820.

Lopes Silva, M. A., de Souza, S. R., Freitas Souza, M. J., & Bazzan, A. L. C. (2019). A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems. *Expert Systems with Applications, 131*, 148–171.Doi: 10.1016/j.eswa.2019.04.056.

H. Lu X. Zhang S. Yang A learning-based iterative method for solving vehicle routing problems In International Conference on Learning Representations 2019 URL.

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

Mastrolilli, M., & Gambardella, L. (2000). *Effective neighbourhood functions for the flexible job shop problem., 3*, 3–20.Doi: 10.1002/(SICI)1099-1425(200001/02)3:1<3::AID-JOS32>3.0.CO;2-Y.

Naderi, B., & Roshanaei, V. (2021). Critical-Path-Search Logic-Based Benders Decomposition Approaches for Flexible Job Shop Scheduling. *INFORMS Journal on Optimization.https://doi.org/10.1287/ijoo.2021.0056., 4*, 1–28.

Nazari, M., Oroojlooy, A., Takáč, M., & Snyder, L. (2018). Reinforcement learning for solving the vehicle routing problem. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems. 31*, 9861-9871. URL: https://proceedings.neurips.cc/paper/2018/file/9fb4651c05b2ed70fba5afe0b039a550-Paper.pdf.

Oren, J., Ross, C., Lefarov, M., Richter, F., Taitler, A., Feldman, Z., … Daniel, C. (2021). SOLO: Search Online, Learn Offline for Combinatorial Optimization Problems. *In Proceedings of the International Symposium on Combinatorial Search, 12*, 97–105.

Özgüven, C., Özbakır, L., & Yavuz, Y. (2010). Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling, 34*, 1539–1548.Doi: 10.1016/j.apm.2009.09.002.

Park, I., Huh, J., Kim, J., & Park, J. (2020). A Reinforcement Learning Approach to Robust Scheduling of Semiconductor Manufacturing Facilities. *IEEE Transactions on Automation Science and Engineering, 17*, 1420–1431.https://doi.org/10.1109/TASE.2019.2956762.

Pezzella, F., Morganti, G., & Ciaschetti, G. (2008). A genetic algorithm for the Flexible Job-shop Scheduling Problem. *Computers & Operations Research, 35*, 3202–3212.Doi: 10.1016/j.cor.2007.02.014.

Rahmati, S., & Zandieh, M. (2012). A new biogeography-based optimization (BBO) algorithm for the flexible job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology, 58*, 1115–1129. https://doi.org/10.1007/s00170-011-3437-9

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

D. Selsam M. Lamm B. Benedikt P. Liang L. de Moura D.L. Dill Learning a SAT Solver from Single-Bit Supervision In International Conference on Learning Representations 2018 URL.

Shahrabi, J., Adibi, M. A., & Mahootchi, M. (2017). A reinforcement learning approach to parameter estimation in dynamic job shop scheduling. *Computers & Industrial Engineering, 110*, 75–82.Doi: 10.1016/j.cie.2017.05.026.

Solozabal, R., Ceberio, J., & Takáč, M. (2020). Constrained combinatorial optimization with reinforcement learning. *arXiv e-prints*, arXiv:2006.11984. https://doi.org/10.48550/arXiv.2006.11984.

Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer Networks. *In Advances in Neural Information Processing Systems, 28*. URL: https://proceedings.neurips.cc/paper/2015/file/29921001f2f04bd3baee84a12e98098f-Paper.pdf.

Wang, H., & Yu, Y. (2016). Exploring Multi-action Relationship in Reinforcement Learning. In R. Booth & M.-L. Zhang (Eds.), *PRICAI 2016: Trends in Artificial Intelligence*, 574-587. https://doi.org/10.1007/978-3-319-42911-3_48.

Wang, L., Hu, X., Wang, Y., Xu, S., Ma, S., Yang, K., … Wang, W. (2021). Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning. *Computer Networks, 190*, 107969.Doi: 10.1016/j.comnet.2021.107969.

Waschneck, B., Reichstaller, A., Belzner, L., Altenmüller, T., Bauernhansl, T., Knapp, A., & Kyek, A. (2018). Optimization of global production scheduling with deep reinforcement learning. *Procedia CIRP, 72*, 1264–1269.Doi: 10.1016/j.procir.2018.03.212.

Wu, Y., Song, W., Cao, Z., Zhang, J., & Lim, A. (2019). Learning Improvement Heuristics for Solving Routing Problems. *IEEE Transactions on Neural Networks and Learning Systems, 1–13*. https://doi.org/10.1109/TNNLS.2021.3068828

Xie, J., Gao, L., Peng, K., Li, X., & Li, H. (2019). Review on flexible job shop scheduling. *IET Collaborative Intelligent Manufacturing, 1*, 67–77. https://doi.org/10.1049/iet-cim.2018.0009

Xing, L. N., Chen, Y. W., Wang, P., Zhao, Q. S., & Xiong, J. (2010). A Knowledge-Based Ant Colony Optimization for Flexible Job Shop Scheduling Problems. *Applied Soft Computing, 10*, 888–896.Doi: 10.1016/j.asoc.2009.10.006.

K. Xu W. Hu J. Leskovec S. Jegelka How Powerful are Graph Neural Networks? In International Conference on Learning Representations 2018 URL.

Yazdani, M., Amiri, M., & Zandieh, M. (2010). Flexible job-shop scheduling with parallel variable neighborhood search algorithm. *Expert Systems with Applications, 37*, 678–687.https://doi.org/10.1016/j.eswa.2009.06.007.

Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P. S., & Chi, X. J. (2020). Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning. URL *In Advances in Neural Information Processing Systems, 33*, 1621–1632 https://proceedings.neurips.cc/paper/2020/file/11958dfee29b6709f48a9ba0387a2431-Paper.pdf.

Zhang, F., Mei, Y., & Zhang, M. (2019). A New Representation in Genetic Programming for Evolving Dispatching Rules for Dynamic Flexible Job Shop Scheduling. *Evolutionary Computation in Combinatorial Optimization, 33–49*. https://doi.org/10.1007/978-3-030-16711-0_3

Zhang, G., Shao, X., Li, P., & Gao, L. (2009). An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. *Computers & Industrial Engineering, 56*, 1309–1318.Doi: 10.1016/j.cie.2008.07.021.

Zhao, M., Li, X., Gao, L., Wang, L., & Xiao, M. (2019). An improved Q-learning based rescheduling method for flexible job-shops with machine failures. In *In 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)* (pp. 331–337). https://doi.org/10.1109/COASE.2019.8843100

Zhou, L., Zhang, L., & Horn, B. (2020). Deep reinforcement learning-based dynamic scheduling in smart manufacturing. *Procedia CIRP, 93*, 383–388.Doi: 10.1016/j.procir.2020.05.163.

Zhu, P., Zhang, J., Xiao, Y., Cui, J., Bai, L., & Ji, Y. (2021). Deep reinforcement learning-based radio function deployment for secure and resource-efficient NG-RAN slicing. *Engineering Applications of Artificial Intelligence, 106*, 104490.Doi: 10.1016/j.engappai.2021.104490.