A Hierarchical Multi-Action Deep Reinforcement Learning Method for Dynamic Distributed Job-Shop Scheduling Problem With Job Arrivals

Jiang-Ping Huang, Liang Gao, *Senior Member, IEEE*, and Xin-Yu Li, *Member, IEEE*

***Abstract*— The Distributed Job-shop Scheduling Problem (DJSP) is a significant issue in both academic and industrial fields. In real-world production, uncertain disturbances such as job arrivals are inevitable. In the paper, the DJSP with job arrivals is addressed with a Multi-action Deep Reinforcement Learning (MDRL) method. Firstly, a multi-action Markov Decision Process (MDP) is formulated, where a hierarchical multi-action space combining operation set and factory set is proposed. The reward function is related to the machine idle time. Additionally, the state transition is also elaborately designed, which includes four typical cases based on job arrival times. Then, a scheduling policy with two decision networks is proposed, where the Graph Neural Network (GNN) is applied to extract the intrinsic information of the scheduling scheme. A Proximal Policy Optimization (PPO) with two actor-critic frameworks is designed to train the model to achieve intelligent decision-making with hierarchical action selections. Extensive experiments are conducted based on 1350 instances. The comparison among 17 composite rules, 3 closely-rated DRL methods, and 2 metaheuristics has proven the outperformance of the proposed MDRL. The application of the MDRL in an automotive engine manufacturing company has demonstrated its engineering value in the industrial field.**

***Note to Practitioners*—The DJSP with job arrivals is a common challenge faced by equipment manufacturers, specifically in the electronic device manufacturing industry. These manufacturers are located in different areas and have varying facility configurations and operation trajectories. To address this challenge, a machine learning-based method can be applied for scheduling daily production tasks. This method divides the DJSP into two subproblems, namely job assigning and job sequencing, and uses two decision networks based on DRL to solve them. To address the uncertainty caused by job arrivals, the rescheduling process and the state update mechanism are carefully designed. A GNN is used for feature extraction at each decision point, and it feeds the decision networks with the extracted features to make the optimal selection. The proposed method has the ability of self-learning and self-adapting, and its effectiveness has been proven through experiments on 1350 test instances. Its practical application has been demonstrated in the production scenarios of an automotive engine manufacturing company. In the future, the method can be adopted to solve more complex distributed manufacturing problems that have constraints such as transportation costs and machine breakdowns.**

***Index Terms*— Deep reinforcement learning, dynamic scheduling, distributed scheduling, job-shop scheduling problem.**

I. Introduction

THE increasing demand for personalized and customized manufacturing requires enterprises to be more flexible and responsive, where the facilities should be equipped with diverse operation trajectories and product configurations. The production in this scenario can be referred to Job-shop Scheduling Problem (JSP). As the global economy thrives, the distributed manufacturing paradigm has been a hotspot. Compared with the centralized manufacturing model, its responsiveness and flexibility to urgent production demands are promising. As an important branch of distributed manufacturing, the Distributed Job-shop Scheduling Problem (DJSP), has received much attention [1], [2]. In the real-life manufacturing system, unexpected disturbances always occur, which has an impact on the effectiveness of the scheduling scheme. Especially, the custom manufacturing provides more flexibility for customers to place and cancel orders, which brings more frequent job arrivals. Thus, the dynamic DJSP with job arrivals is a valuable practical issue that deserves intensive study. The Apple mobile phone company is one practical case of the dynamic DJSP. Its OEM companies are located worldwide, such as China, Japan, and Singapore. The production tasks for these companies are delivered daily, which causes uncertain job arrivals. Thus, it is critical to adjust the scheduling scheme in real time and achieve efficient production in uncertain environments.

Various methods have been developed for scheduling problems in different scenarios. The three most popular methods are exact methods [3], Priority Dispatch Rules (PDRs) [4], and metaheuristics [5], [6]. Exact methods are good at finding

Manuscript received 31 October 2023; revised 22 January 2024 and 2 March 2024; accepted 16 March 2024. Date of publication 1 April 2024; date of current version 7 February 2025. This article was recommended for publication by Associate Editor Z. Pei and Editor J. Li upon evaluation of the reviewers' comments. This work was supported in part by the National Science Foundation of China under Grant 51825502, Grant U21B2029, and Grant 52188102; and in part by the Key Research and Development Program of Hubei Province under Grant 2021AAB001. *(Corresponding author: Xin-Yu Li.)*

The authors are with the State Key Laboratory of Intelligent Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: d202180294@hust.edu.cn; gaoliang@mail.hust.edu.cn; lixinyu@mail.hust.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TASE.2024.3380644, provided by the authors.

Digital Object Identifier 10.1109/TASE.2024.3380644

the optimal solutions for small-scale problems but are limited when it comes to large-scale or dynamic problems. PDRs are easy to implement and can respond quickly to uncertain disturbances. However, they are not generalized enough, and no single PDR can perform well in all scenarios [7]. Metaheuristics are the most widely studied methods for scheduling problems due to their strong ability to search for optimal solutions. But they are time-consuming due to their iterative evolutionary trajectory. Some metaheuristics have also been proposed for dynamic shop scheduling problems, but most of them produce a strongly robust schedule by considering the impact of dynamic events ahead [8]. Alternatively, they initially generate a complete schedule and repair it with a heuristic or a PDR when the disturbance occurs [9]. However, these methods are not suitable for the dynamic DJSP studied in the paper.

The Deep Reinforcement Learning (DRL) method has demonstrated significant potential in solving various scheduling problems, which is due to its self-learning and self-regulating [10], [11], [12], [13], [14]. DRL can learn an optimal scheduling policy by exploring historical data. It also shows pretty good performance in real-time scheduling for dynamic problems due to its fast responsiveness [15]. One such problem is the studied dynamic DJSP, which consists of two closely coupled subproblems, namely, job assigning and job sequencing. For each arrived job, a factory must be identified and used to store it. Immediately after that, the sequence in which the job will be processed in the factory must be determined. Moreover, due to the uncertain disturbances caused by random job arrivals, the scheduling scheme must be adjusted in real time. To ensure the continuity of the problem-solving and the real-time responsiveness of the production system, a hierarchical Multi-action DRL (MDRL) is designed. This method can take both job assigning and job sequencing into consideration simultaneously.

The main contributions of the paper are presented as follows. (1) A multi-action Markov Decision Process (MDP) is formulated, where a hierarchical multi-action space combining operation set and factory set is proposed, and the state transition is elaborately classified into four cases according to job arrival times; (2) A Graph Neural Network (GNN) is utilized to uncover the intrinsic features hidden in the dynamic disjunctive graph that is specially designed for the DJSP with job arrivals; (3) A Proximal Policy Optimization (PPO) with two actor-critic frameworks is proposed to ensure intelligent decision-making by employing the hierarchical action selections at each decision point.

The paper is structured in the following manner. Section II presents the closely related work. Section III describes the studied problem. The details of the proposed method are presented in Section IV. In Section V, the comparative experiments are presented, and the practical test is conducted with the cases from an automotive engine manufacturing company. Section VI provides a conclusion and outlines plans for further research.

II. Related Work

With the distributed manufacturing paradigm widely advocated, DJSP has received much attention. An adaptive Genetic Algorithm (GA) with genetic search and without the optimal crossover rate is proposed for DJSP [16]. A GA that combines with a Gantt chart is designed for DJSP to minimize the maximum completion time (makespan, *i.e.*, C_{max}), job tardiness, or manufacturing cost [17]. For DJSP with the minimization of makespan, a mixed integrated linear programming model and a Simulated Annealing (SA) are developed [18]. Besides, DJSP is also modeled as a set of fuzzy constraint satisfaction problems, and solved by an agent-based fuzzy constraint-directed negotiation method [19]. Chaouch et al. [2] design a hybrid ant-based algorithm with a dynamic assignment rule for DJSP. The energy-efficient DJSP is solved by Jiang et al. [20] via a multi-objective evolutionary algorithm with decomposition. Sahman [1] solves DJSP with a discrete spotted hyena optimizer. The existing methods are almost for the static DJSP, and take no consideration of the unexpected disturbances, thus are hard to be directly applied to the studied dynamic DJSP.

The process of practical production is complex and variable, and unexpected disturbances are common [21], such as random job arrivals [22], order cancellation [23], and others. To adapt to these sudden changes, dynamic scheduling has been a critical technique that can reduce the impact of such disturbances. Dynamic scheduling techniques mainly include PDRs, metaheuristics, and DRL methods. PDRs can be divided into single PDRs [24] and composite PDRs [25], and are the simplest methods for dynamic scheduling problems. Metaheuristics are another important dynamic scheduling algorithm. By considering energy efficiency, Luo et al. [26] solve the flexible flow shop scheduling problem with job arrivals by a GPU-based parallel GA. With an improved Particle Swarm Optimization (PSO) algorithm, the dynamic JSP with random job arrivals is addressed [22]. Yan and Yu [27] improve a multi-objective differential evolution algorithm to solve the multi-objective dynamic flexible JSP (FJSP) to optimize the energy cost, the scheduling stability, and the makespan. For dynamic JSP with unconditional job arrivals, Ali et al. [28] propose an evolutionary GA where the virtual crossover operators are designed. Even though some PDRs and metaheuristics have been proposed for dynamic shop scheduling problems, almost all of them are for problems with centralized manufacturing environments, and hardly consider the distributed characteristics. Despite the simplicity of PDRs, they are not always effective in finding optimal solutions and are often integrated with other algorithms. Additionally, their performance depends on the application scenarios, and it is almost impossible to find a PDR that works well in all scenarios. Although metaheuristics can generate high-quality solutions, it often comes at a significant time cost, and their operators are generally designed for specific problems and are not directly applicable to other related problems.

DRL methods have been widely used in the scheduling field [29], especially for dynamic scheduling problems [30]. Zhou et al. [31] propose a smart scheduler based on DRL to handle uncertain jobs and random events such as urgent or simultaneous orders and machine failure in smart manufacturing factories. Johnson et al. [32] apply a multi-agent RL system based on Double DQN (DDQN) for FJSP to minimize makespan. Liu et al. [11] use a DDQN method

to solve FJSP with job arrivals. Grumbach et al. [33] study the robust-stable scheduling in dynamic flow shops and propose a DRL method implemented with OpenAI frameworks. Wang et al. [34] solve the dynamic FJSP with two DQNs and a real-time processing framework. Yan et al. [35] propose a double-layer Q-learning algorithm for an optimization problem integrated with FJSP and preventive maintenance, and solve it by considering worker and machine resources. In response to the production process of aerospace structural parts, a dual-system RL method for FJSP with frequent emergency insertion orders is proposed [36]. For a stochastic parallel machine scheduling problem with heterogeneous jobs, an RL-based scheduling framework is proposed to minimize the weighted tardiness [37]. A hierarchical RL method based on GNN is designed for a large-scale dynamic FJSP with random job arrivals [38]. To minimize the total tardiness of the FJSP with random job arrivals, Zhao et al. [39] propose a DRL method with an attention-based policy network. DRL has been proven an excellent solution for solving dynamic problems. It has been successfully applied in the scheduling field, as it can learn from historical data and extract real-time features from the environment, leading to an optimal decision. Despite its advantages, DRL methods are rarely used in the dynamic distributed shop scheduling problems. Inspired by this fact, the paper proposes a DRL-based method that considers the distributed and dynamic characteristics of the problem simultaneously.

III. Problem Description

In this paper, the DJSP with job arrivals is studied. DJSP consists of f identical factories, $\{1, 2, \ldots, k, \ldots, f\}$, each of which is an independent job shop with m machines. There is a total of n jobs that need to be machined following their own process routes, denoted by $\{1, 2, \ldots, j, \ldots, n\}$. n is the sum of n_i and n_a, where n_i represents the number of the jobs that arrive at the production system at the initial moment, while n_a is the number of the jobs with uncertain arrival times. The arrival time of job j is denoted by a_j. Each job comprises m operations, $\{1, 2, \ldots, i, \ldots, m\}$, and each operation must be processed on a unique machine in one factory. The processing time of operation i of job j (operation o_{ji}) is p_{ji}, and it does not depend on the factory in which the job is. It is assumed that the completion time of operation o_{ji} is CT_{ji}, and the makespan of factory k is CT_k which refers to the maximum completion time among all operations in factory k, *i.e.*,

$$CT_k = max\left(CT_{ji}^k\right). \tag{1}$$

where CT_{ji}^k is the completion time of operation o_{ji} that is machined in factory k. The makespan is the maximum completion time among factories, *i.e.*,

$$CT_{max} = max(CT_k). \tag{2}$$

The objective is to minimize the makespan. For the dynamic DJSP, the arrival times, processing times, and process routes of the jobs that will enter the system are not known in advance. Each job can only become available in the system after it arrives. At any time, any machine can only handle one operation, and each job can undergo one operation. It is necessary to determine which factory each job is assigned to and the sequence in which it will be machined in the factory. It is recommended to consider rescheduling the established production scheme in response to uncertain job arrivals in order to avoid inefficiencies. Thus, the new job arrival is set as the rescheduling event to trigger the rescheduling process. For the established scheduling scheme, only operations that have a completion time greater than the arrival time of the new job need to be rescheduled. The operations that have a completion time equal to or less than the arrival time no longer require rescheduled.

For the dynamic DJSP, three challenges need to be considered:

a) The information regarding jobs that have not arrived yet cannot be accessed until they actually arrive. This requires the proposed DRL-based model be highly flexible and adaptable. On one hand, the model should be able to adjust to any changes in the scale of the problem; On the other hand, it is impractical to train models for different problem sizes separately, so the model's generalizability must be ensured.

b) The coupling between the two subproblems, namely job assigning and job sequencing, poses additional challenges to decision-making. On one hand, the sequencing of jobs is influenced by the factory they are assigned to, and on the other hand, job sequencing also affects job assigning. The complex rescheduling process (*i.e.*, state transition presented in Section IV) for DJSP arises due to the distributed characteristics, which require each job's operations to be machined in the same factory. This process is heavily dependent on job arrival times.

c) For the dynamic problem, the ability of the proposed method to extract the global features is crucial. Comprehensive information from the scheduling environment can help the agent make wiser decisions.

IV. Multi-Action DRL for DJSP With Job Arrivals

In this section, a hierarchical MDRL method is proposed to tackle the challenges associated with the dynamic DJSP. First, a novel disjunctive graph representation for the dynamic DJSP is proposed. It takes into account the uncertainty of the unarrived jobs and enhances the flexibility of the solution representation. Then, the dynamic DJSP is formulated as a multi-action MDP, with the job candidates and factory candidates as the primary-subordinate action spaces, where each job candidate has its own factory set. This approach allows the coupling subproblems be solved sequentially, facilitating faster decisions and the rescheduling process. Next, a multi-action policy based on GNN is presented, which extracts the local and global features to provide a comprehensive observation for the agent. Finally, a PPO with two actor-critic frameworks is designed to train the multi-action scheduling policy, allowing for independent and orderly operation selection and factory selection.

A. Multi-Action Markov Decision Process Formulation

The sequential decision problems can be modeled as an MDP, $M(S, A, T, R, \Upsilon)$. S is the state space that stores the

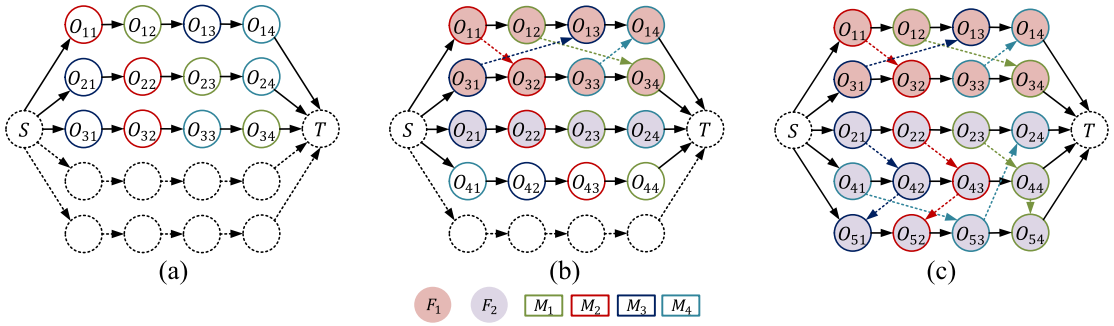


Fig. 1. Stitched disjunctive graph model.

observed environment at each decision point. A is the action space that the agent can choose from at each decision point. T is the state transaction function which is used to shift the current state to the next state according to the action selection. R is the reward function that shows the scores the agent has obtained by interacting with the environment. Υ is the discounter factor which is a decimal between 0 and 1. It is used to estimate the effect of the earlier actions on the current state. The difference between the multi-action RL [40] and the traditional RL is the definition of the action space. In the traditional RL, the action space is defined as a one-dimensional set. However, in the multi-action RL, the action space is a multi-dimensional set denoted as $A = A_1 \times A_2 \times \ldots A_q \times \ldots \times A_Q$, where Q represents the number of sub-actions in the problem, and A_q is a sub-action set. The details of the multi-action MDP of the dynamic DJSP are presented as follows.

1) State Representation: The disjunctive graph has been one of the most important and widely used solution representation for JSP since it was proposed. It is a directed graph, denoted by $G = (V, C \cup D)$, where V is a set of $mn + 2$ vertices corresponding to the mn job tasks and two dummy vertices marking the start and end of the process. The set C consists of a group of conjunctive arcs that indicate the precedence constraints among the job tasks. At the start of the process, the set D contains undirected disjunctive edges that connect the job tasks executed on the same machine without any specific order. As the scheduling process moves froward, the undirected arcs are directed, indicating that the tasks processed on the same machine are sequenced progressively.

The DJSP is an extension of JSP, and a new disjunctive graph representation is designed for DJSP. In this graph, the placed factory information is carried by the vertices, and it can be represented as $G = (V, (C_1 \cup D_1) \cup (C_2 \cup D_2) \cup \ldots \cup (C_k \cup D_k) \cup \ldots \cup (C_f \cup D_f))$. Due to the dynamic characteristics of the studied problem, the proposed disjunctive graph representation changes dynamically as the jobs arrive. In Fig. 1, an instance with 2 factories, 3 initial jobs, and 2 jobs that will arrive is presented. Each job consists of 4 operations. The initial information of the instance is shown in Fig. 1(a), which includes the process routes of the arrived jobs and the empty nodes of the unarrived jobs. From Fig. 1(a), it can be found that operations o_{11}, o_{12}, o_{13}, and o_{14} should be sequentially processed on machines M_2, M_1, M_3, and M_4. Fig. 1(b) shows a partial schedule that assigns jobs 1 and 3 to factory F_1, while job 2 is assigned to factory F_2, job 4 has just arrived and is waiting to be allocated, and job 5 has not yet arrived. The operations in factory F_1 that are colored in pink have been scheduled with directed disjunctive edges (for example, o_{11} and o_{32} are directed with a red dotted arrow, which means they will be sequentially processed on machine M_2 of factory F_1). On the other hand, the operations in factory F_2 which are colored in purple, have not been scheduled. Fig. 1(c) presents a complete solution where jobs 1 and 3, highlighted in pink, are assigned to factory F_1, and jobs 2, 4, and 5, highlighted in purple, are assigned to factory F_2. The process sequence of operations on the same machine in each factory is directed by the dotted arrow of the same color.

At decision point t, the state s_t comprises all the information in the disjunctive graph $G_t = (V_t, (C_{1t} \cup D_{1t}) \cup (C_{2t} \cup D_{2t}) \cup \ldots \cup (C_{kt} \cup D_{kt}) \cup \ldots \cup (C_{ft} \cup D_{ft}))$. This graph includes a set of vertices V_t that contains all the static features of the jobs that have arrived. Additionally, each factory k has a set of precedence constraints C_{kt} and a set of directed disjunctive edges D_{kt}. Besides the static information such as the processing time and the process routes, vertex v corresponding to operation o_{ji} also records two variable information $C_{LB}(o_{ji})$ and $b(o_{ji})$, which is treated as the node feature. $C_{LB}(o_{ji})$ is the estimated lower bounder of the completion time of operation o_{ji}, and it is calculated by

$$C_{LB}(o_{ji}) = \begin{cases} C_{LB}(o_{j(i-1)}) + p_{ji}, & i > 1 \\ a_j + p_{ji}, & i = 1. \end{cases} \tag{3}$$

$b(o_{ji})$ is used to mark whether operation o_{ji} has been scheduled or not, if o_{ji} has been scheduled, $b(o_{ji})$ is valued as 1, other it is valued as 0.

$$b(o_{ji}) = \begin{cases} 1, & if\ o_{ji}\ is\ scheduled \\ 0, & otherwise. \end{cases} \tag{4}$$

2) Hierarchical Multi-Action Space: The traditional DRL method generally selects an action from the action space by observing the environment and executes the action to transfer to the next state. When dealing with the DJSP with job arrivals, the agent needs to select an operation before assigning it to a factory. To accomplish this, a hierarchical multi-action space $A = A_{op} \times A_{fac}$ that combines the operation action space A_{op} with the factory action space A_{fac} is proposed.

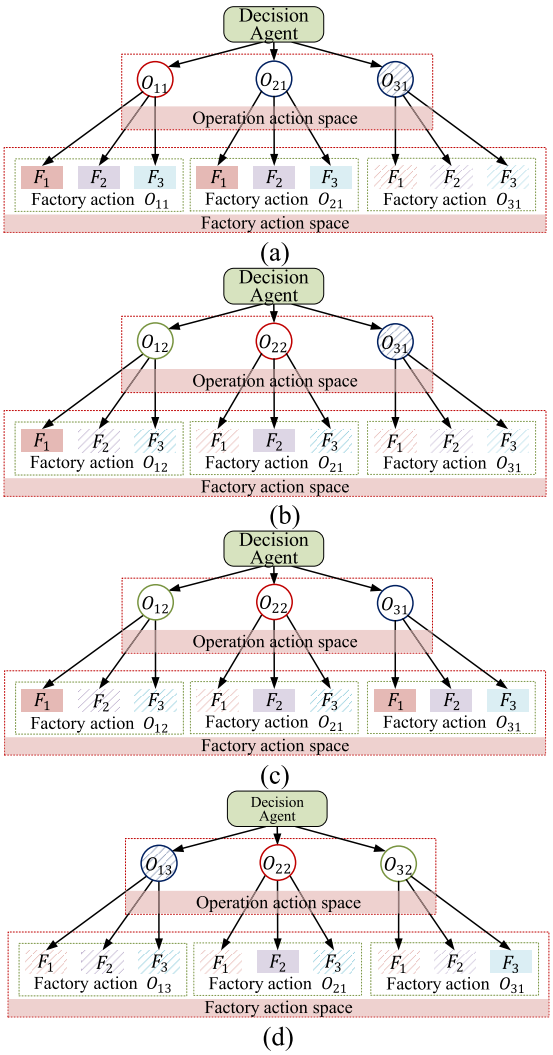


Fig. 2. Illustration of the multi-action space for dynamic DJSP with job arrivals.

A_{op} contains the eligible operations and the ineligible operations, and A_{fac} contains the eligible factories and the ineligible factories. When a job arrives, its eligible operation is the first operation if it has not been scheduled. However, if a job has already been scheduled, its eligible operation refers to the operation whose previous operation has been scheduled. Jobs that have not arrived yet or have already been completed have no eligible operation. DJSP requires that all operations of one job be machined in the same factory. Therefore, for an unassigned operation, its eligible factories are the factories where it can be assigned. If the operation is the first operation of a job, then all factories are eligible for it. Otherwise, its eligible factory can only be the factory where the job's first operation has been assigned. Take an instance $3 \times (2+1) \times 3$ as an example. In this case, there are 3 factories, 2 initial jobs, and 1 job that will arrive. Each job has 3 operations. In Fig. 2, the operations and the factories that are ineligible are masked with a slash. Fig. 2(a) illustrates that at the initial time, the operation space consists of $\{o_{11}, o_{21}, o_{31}\}$, and the factory space for each operation is $\{F_1, F_2, F_3\}$. However, job 3 has not yet arrived in the system, so operation o_{31} is masked and its factory action space is also masked. In Fig. 2(b), it is assumed that operation o_{11} has been assigned to factory F_1 and operation o_{21} has been assigned to factory F_2. As a result, the operation action space turns to $\{o_{12}, o_{22}, o_{31}\}$. In the factory space of o_{12}, factories F_2 and F_3 are masked, and in the factory space of o_{22}, factories F_1 and F_3 are masked. However, since job 3 has not yet arrived, o_{31} and its factory action space remain masked. In Fig. 2(c), it can be seen that job 3 has entered the system, and its first operation o_{31} is now eligible. At the same time, factories F_1, F_2, and F_3, which are associated with o_{31}, are also eligible. In Fig. 2(d), it can be observed that the last operation of job 1 (o_{13}) has been completed, and it is now an ineligible operation, along with its associated factories. Furthermore, operation o_{31} has been assigned to and sequenced in factory F_3, making factories F_1 and F_2 ineligible, and o_{32} is now an eligible operation.

3) Reward: For the RL methods, the reward an agent gets from the environment is crucial, it drives the agent to improve its performance. In this paper, the objective is to minimize makespan. To achieve this, a reward function based on Machine-Idle-Time (MIT) is designed. It is assumed that the start processing time of the selected operation is st_t^a and the completion time of the previous operation on the same machine is fc_t. The reward function is defined as the gap between st_t^a and fc_t and is calculated by

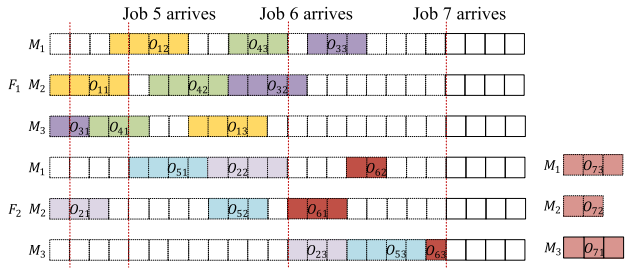
$$r_t = -\left(st_t^a - fc_t\right) \tag{5}$$

When the selected action is the first operation on the current machine, fc_t is valued at 0, *i.e.*, $r_t = -st_t^a$. The MIT-based reward ensures that the cumulative reward and machine idle time are inversely proportional. A greater cumulative reward means a smaller machine idle time, which helps minimize the makespan.

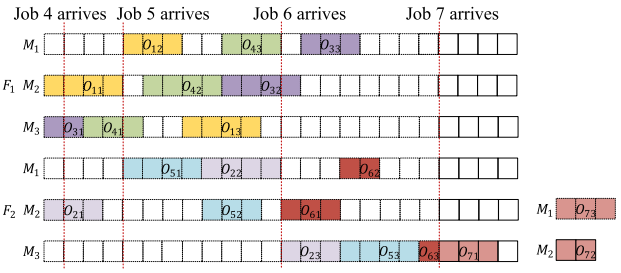
4) State Transition: In production system, unexpected disturbances can significantly impact the efficiency and stability of scheduling methods, leading to an unavoidable effect on the environment's state. When facing uncertain disturbances, it's important to consider when and how to react [41]. An arrival-time-based rescheduling strategy enables real-time response. At decision point t, it is necessary to check whether any new jobs have arrived in the system or not. If there is at least one job waiting in the system, then the scheduled operations whose completion time is greater than the earliest arrival time among the waiting jobs should be rescheduled. To facilitate understanding, the operations that have been scheduled until the current time are marked with dashed lines in Fig. 3, 4, and 5 to indicate that there is no need to reschedule them.

In the case of the studied DJSP with job arrivals, the rescheduling brings a more complex state transition process. At any decision point t, the state transition should consider four cases based on the hierarchical action space. To make it easier to understand, four symbols are defined in Table I.

Case 1: $\boldsymbol{\mathcal{J}_{a\&u} = \emptyset}$, *i.e.*, no job that has arrived has not been scheduled. In this case, if there is at least an eligible operation available in the operation action space, the agent will select a suitable operation and determine the earliest start processing time. However, if there are no eligible operations available, the agent will wait until a new job arrives.

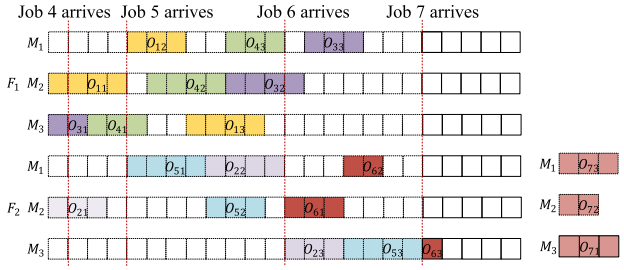


(a) Current state s_t and the information of the arrived job

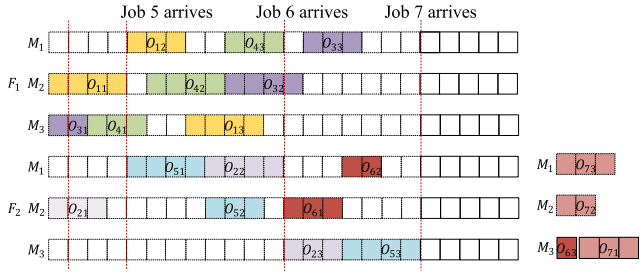


(b) Next state s_{t+1} based on the selected action

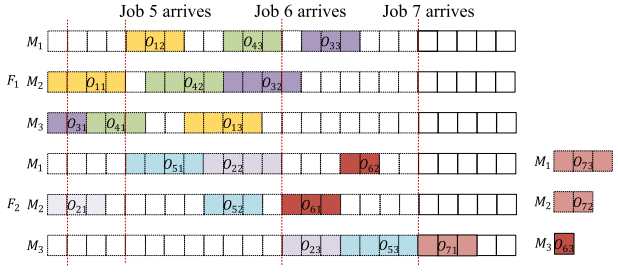
Fig. 3. Illustration of the state transition in Case 2.



(a) Current state s_t and the information of the arrived job.



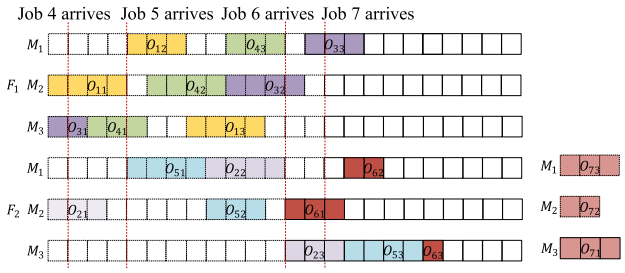
(b) Current state s_t updated by the new job arrival.



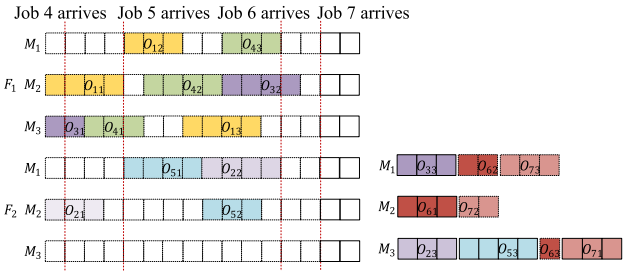
(c) Next state s_{t+1} based on the selected action.

Fig. 4. Illustration of the state transition in Case 3.

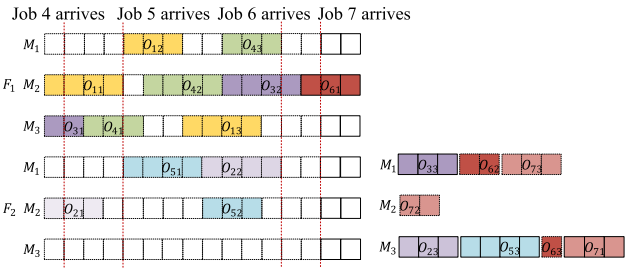
Case 2: $\mathcal{J}_{a\&u} \neq \emptyset$ **and** $\boldsymbol{min}(\mathcal{A}_{a\&u}) \geq \boldsymbol{max}(\mathcal{C}_s)$, *i.e.*, at least one job has arrived but not been scheduled. Additionally, the earliest arrival time in set $\mathcal{A}_{a\&u}$ is not less than the maximum completion time of all the operations that have already been scheduled. An example is shown in Fig. 3(a), where $\mathcal{J}_{a\&u} = \{7\}$. The arrival time of job 7 equals the completion time of operation o_{63}.



(a) Current state s_t and the information of the arrived job.



(b) Current state s_t updated by the new job arrival.



(c) Next state s_{t+1} based on the selected action

Fig. 5. Illustration of the state transition in Case 4.

TABLE I
SYMBOLS FOR CLASSIFYING THE STATE TRANSACTION

Symbol	Definition
$\mathcal{J}_{a\&u}$	Set of jobs that have arrived, but have not yet been scheduled
$\mathcal{A}_{a\&u}$	Set of the arrival times of the jobs in set $\mathcal{J}_{a\&u}$
$\mathcal{C}_s$	Set of the completion times of the operations that have been scheduled
$\mathcal{CF}_s$	Set of the completion times refers to the completion times of the first operations of each job in the scheduling scheme

In this case, it is not necessary to consider the rescheduling. The agent must choose an eligible operation from the operation action space and ensure that the operation is processed as soon as possible. If the selected operation is the first operation of the arrived job in set $\mathcal{J}_{a\&u}$, a factory should be selected to assign the operation to before sequencing. In comparison to the current state s_t illustrated in Fig. 3(a), the next state s_{t+1}

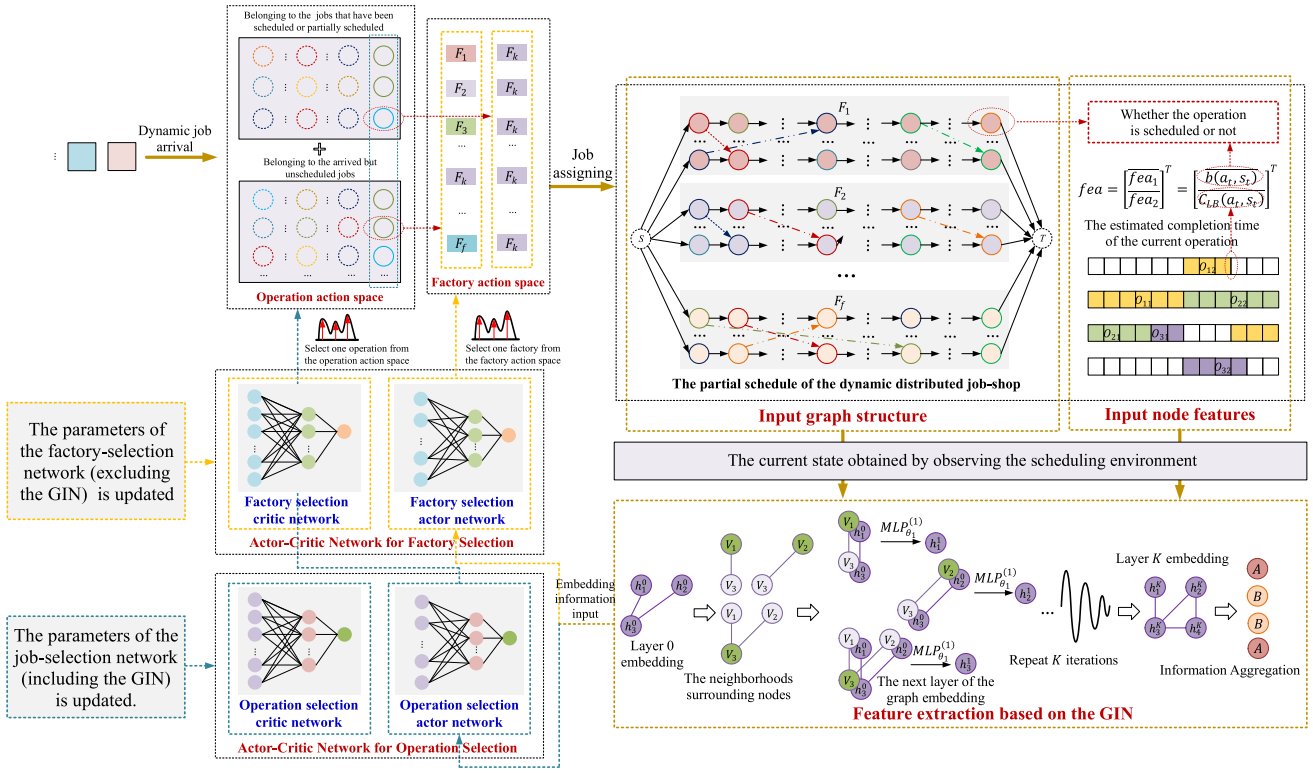


Fig. 6. The framework of GIN-based multi-action scheduling policy.

demonstrated in Fig. 3(b) is updated based on the action selections. In the updated state, job 7 has been allocated to factory F_2, and operation o_{71} has been scheduled on machine M_3.

Case 3: $\mathcal{J}_{a\&u} \neq \emptyset$ **and** $\boldsymbol{min}(\mathcal{A}_{a\&u}) \geq \boldsymbol{max}(\mathcal{CF}_s)$**and** $\boldsymbol{min}(\mathcal{A}_{a\&u}) \leq \boldsymbol{max}(\mathcal{C}_s)$, *i.e.*, at least one job has arrived but not been scheduled, the earliest arrival time in set $\mathcal{A}_{a\&u}$ is not less than the maximum completion times in set $\mathcal{CF}_s$, and it is also not greater than the maximum completion time among all operations that have already been scheduled. For instance, in Fig. 4(a), $\mathcal{J}_{a\&u} = \{7\}$, and the arrival time of job 7 is between the completion times of operation o_{53} and operation o_{63}.

In this case, with a partial schedule in which the completion time of operations o_{63} is greater than the arrival time of job 7. Therefore, the rescheduled operation is o_{63}. The current state s_t and the eligible operation actions are updated as shown in Fig. 4(b), where all operations whose completion time is greater than job 7's arrival time are returned to the unscheduled operation set. In this way, the candidates for the machines in different factories may be changed. For example, due to the return of operation o_{63}, the candidate set for machine M_3 in factory F_2 is now shifted from $\{o_{71}\}$ to $\{o_{63}, o_{71}\}$. Next, the agent should choose an operation from the eligible operation actions and set the earliest start processing time for it. In Fig. 4(c), it is assumed that operation o_{71} is selected and assigned to factory F_2, and thus the next state s_{t+1} is updated.

Case 4: $\mathcal{J}_{a\&u} \neq \emptyset$ **and** $\boldsymbol{min}(\mathcal{A}_{a\&u}) < \boldsymbol{max}(\mathcal{CF}_s)$, *i.e.*, at least one job has arrived but not been scheduled. Additionally, the smallest arrival time in set $\mathcal{A}_{a\&u}$ is less than the maximum completion time in set $\mathcal{CF}_s$. As in Fig. 5(a), the arrival time of job 7 is less than the completion time of operation o_{61}.

In this case, any job whose first operation's completion time is greater than the smallest arrival time of the newly arrived jobs must be rescheduled. Additionally, the current state, eligible operations, and eligible factories must be updated accordingly. As presented in Fig. 5(b), the operations that should be rescheduled are o_{23}, o_{33}, o_{53}, o_{61}, o_{62}, and o_{63}. The eligible operations are also updated and highlighted in bold: $\{o_{23}, o_{33}, o_{53}, o_{61}, o_{71}\}$. If operation o_{61} is selected at this decision point, the next state is updated as shown in Fig. 5(c).

B. GNN-Based Policy

GNN is an important neural network that addresses the graph-structured data and explores the inner connections among edges and nodes. It has been widely used in various fields, including predicting the industrial equipment's remaining useful life [42] and decentralized wireless resource allocations [43]. The Graph Isomorphism Network (GIN) [44], as the simplest version of GNN, is applied in the proposed MDRL method to extract features hidden in the disjunctive graph of the DJSP with job arrivals.

In this section, the scheduling policy is presented in detail. As shown in Fig. 6, first, the agent observes the scheduling environment and feeds the partial disjunctive graph and the node features into the GIN-based feature extraction network. Second, the GIN outputs the disjunctive graph's embedding information, which combines the global information and the local information. Third, the job-selection network calculates the probability distribution over the job candidates and

chooses the action with the maximum probability during testing. Fourth, based on the selected job operation, the factory-selection network chooses the most suitable factory with the maximum probability during testing over the eligible factory candidates. Finally, the chosen operation is assigned to the chosen factory, and is set at the position where it can be processed earliest.

1) GIN-Based Feature Extraction: The proposed disjunctive graph representation provides a comprehensive overview of the scheduling scheme. It allows for obtaining information about the job sequence on each machine, the precedence constraints, the operation processing time, and the machines' available time. By performing feature extraction, valuable information can be obtained, which can aid in achieving effective scheduling performance for both the job-selection network and the factory-selection network. The simple and powerful GIN [44] is adopted in the paper, and the embedding layer in GIN is provided as

$$h_v^{(\kappa)} = MLP_{\theta_\kappa}^{(\kappa)}\left(\left(1+\epsilon^{(\kappa)}\right)\cdot h_v^{(\kappa-1)} + \sum\nolimits_{u\in N(v)} h_u^{(\kappa-1)}\right) \quad (6)$$

where $h_v^{(\kappa)}$ is the embedding representation of vertex v at the κ^{th} iteration, and $h_v^{(0)}$ represents the raw input information from vertex v. $MLP_{\theta_k}^{(\kappa)}$ is a Multi-Layer Perceptron (MLP) with parameters θ_κ at the κ^{th} iteration. ϵ is a learnable parameter. $N(v)$ is the set of neighbor vertices of vertex v. After iterating K times, the disjunctive graph's global embedding representation is obtained with an average pooling function

$$h_g = \sum\nolimits_{v\in V} h_v^{\mathrm{K}} \Big/ nm \quad (7)$$

where nm is the number of vertices. For the GIN in the proposed MDRL method, the MLP has two hidden layers, each layer has 64 neurons. As presented in Section IV-A, any vertex v has two features, and thus the dimension of the input layer is two.

2) Action Selection: With the features extracted by GIN, two actor networks ($\pi_{op}\left(a_{op,t}|s_t\right)$ and $\pi_{fac}\left(a_{fac,t}|s_t\right)$) are proposed to score the operation candidates and the factory candidates determined by the selected operation, where $a_{op,t}$ donates one operation action in the operation action space A_{op} at decision point t, and $a_{fac,t}$ represents one factory action in the factory action space A_{fac} at decision point t. Both actors are composed of a MLP that has one hidden layer with 32 neurons. At decision point t, the job-selection network $\pi_{op}\left(a_{op,t}|s_t\right)$ takes the features provided by the GIN as input, and produces a probability distribution over all the operations in the operation action space by

$$p\left(a_{op,t}\right) = \frac{exp\left(MLP_{\theta_\pi}\left(\left[h_{a_{op,t}}^{(\mathrm{K})}, h_g(s_t)\right]\right)\right)}{\sum_{a_{op,t}\in A_{op,t}} exp\left(MLP_{\theta_\pi}\left(\left[h_{a_{op,t}}^{(\mathrm{K})}, h_g(s_t)\right]\right)\right)}. \quad (8)$$

then the operation with the maximum probability is selected. It's worth noting that the MLP generates scores for all operations in the action space, but for ineligible operations, these scores are masked as an infinitesimal, resulting in a probability value of zero. This makes it impossible for such operations to be selected. Once an operation is selected, the factory where the operation will be assigned must also be selected. For the DJSP, once a job has started processing in one factory, it cannot be transferred to another factory. Therefore, it is necessary to determine if the selected operation is the first operation of a job. If it is not, then the eligible factory is the one where the job has been assigned. However, if it is the first operation, then all factories are eligible. The factory-selection network $\pi_{fac}\left(a_{fac,t}|s_t\right)$ provides a probability distribution over all the factory candidates, with the features extracted by GIN reshaped to match the size of factory candidates. Like the job-selection network, the probability values of the ineligible factories are masked as zeros, and the factory with the maximum probability is selected.

C. Proximal Policy Optimization With Two Actor-Critic Frameworks

The PPO [45] is an important version of the policy gradient method that follows an actor-critic style. As explained in Section IV-B, there are two actor networks proposed to select operations and factories, respectively, both sharing the same network architecture. During the training stage, two critic networks are also presented to evaluate the performance of the actors, and they have a different network architecture with the actors' networks. Their input layers have 64 neurons, whereas the dimension of the actors' network is 128. The job-selection network and factory-selection network share the same GIN architecture and the data output by the GIN. The two actor networks take the local information $h_{a_{op,t}}^{(K)}$ and the global information $h_g(s_t)$ as input. The critic networks evaluate how good the action taken at decision point t by taking the global information $h_g(s_t)$ as input.

During the training stage, the job-selection actor network and critic network work together to optimize their parameters and those of the GIN. The factory-selection actor network and critic network, on the other hand, take the features extracted by GIN as input, and then update their own parameters. Two different loss functions are used in the process. For the job-selection network, at decision point t, the loss is calculated as

$$L_t^{CLIP+VF+S}(\theta) = c_p L_t^{CLIP}(\theta) - c_v L_t^{VF}(\phi) + c_e S[\pi_\theta](S_t) \quad (9)$$

where $L_t^{CLIP}(\theta)$ is the policy surrogate, $L_t^{VF}(\phi)$ is value function value, and $S[\pi_\theta](S_t)$ is the entropy bonus. c_p, c_v, and c_e are all loss coefficients, and their values are set as the original paper [45]. For the factory-selection network, the $MSELoss$ function is used to evaluate the difference between the outputs from the actor and the critic.

V. Experiment With Discussion

In this section, the comparative tests are conducted to assess the effectiveness, generalization, and stability of the proposed agent, and they are carried out against the PDRs, related DRL methods, and metaheuristics. In addition, a case study is presented to demonstrate the practical value of the MDRL method. To ensure the rationality of experimental results, 1350 test instances are generated by referring to paper [46]. The time interval between two consecutive job

arrivals follows Exponential Distribution with parameter E_{ave}. Due to the space limitations, the settings of the instance configurations, training details, and detailed results from the comparative experiments and case study are all presented in the Supplementary Material. All experiments are conducted on Intel(R) Xeon(R) W-3365 CPU @ 2.70GHz and NVIDIA GeForce RTX 3090 Ti.

A. Comparative Experiments

In this part, extensive comparative experiments are presented to validate the performance of the proposed MDRL. The three most dominant scheduling methods, namely, PDRs and DRL methods, as well as with metaheuristics are compared. To evaluate the difference among the results obtained from these methods, the Relative Percentage Increase (RPI) is used, which is calculated by

$$RPI = \frac{Method_{sol} - Best_{sol}}{Best_{sol}} \times 100\% \tag{10}$$

where $Method_{sol}$ is the result obtained from a specific method, and $Best_{sol}$ is the best result obtained for a given instance. In addition to RPI, the win rate is also used as a performance metric for the algorithms. It is the percentage of instance configurations for which a particular method produces the minimum average makespan. This measure helps to assess the adaptability of the algorithms to different production configurations. The details of the experiments are presented as follows.

1) Comparison With the PDRs: The comparison between the proposed MDRL and 17 PDRs is presented. Since no existing PDRs have been specially designed for DJSP, five assigning rules and four sequencing rules are selected to create 17 Composite Rules (CRs) for DJSP as the compared PDRs. The rescheduling trigger mechanism used in these CRs is the same as that of the proposed MDRL. The details of the assigning rules, sequencing rules and 17 CRs are presented in the Supplementary Material, as space limitations prevent a comprehensive discussion in the main manuscript.

The experiment results of various test configurations are presented in Fig. 7, which shows the box plots and win rates of compared methods. As shown in Fig. 7(a.1), when there are 2 factories, the MDRL has an RPI near 0.00, with CR17 following with the second mean and median. However, the difference between CR17 and the MDRL is significant. In the win rate pie chart (Fig. 7(a.2)), when excluding the MDRL, CR17 is the winner among the compared algorithms, with 12 smallest average makespans. CR5 follows with 10 smallest average makespans, and CR13 ranks third with 8 smallest average makespans. However, when the proposed MDRL is included, the situation changes dramatically. As Fig. 7(a.3) shows, the MDRL obtains the best 37 averages (82.22%, 37 out of 45). Fig. 7(b) shows a similar conclusion when there are 3 factories. The MDRL still has the smallest average RPI, and the compared PDRs are far worse than the MDRL. In terms of the win rates shown in Fig. 7(b.2) and(b.3), when excluding the MDRL, CR1 is the winner with the 12 smallest average makespans. But when the MDRL is included, about 84.44% (38 out of 45) of the best results of the test cases are obtained by the MDRL. Finally, Fig. 7(c) indicates that the MDRL performs well under configurations with 4 factories, with the smallest average RPI and the maximum win rate (82.22%, 37 out 45) all being obtained by the MDRL.

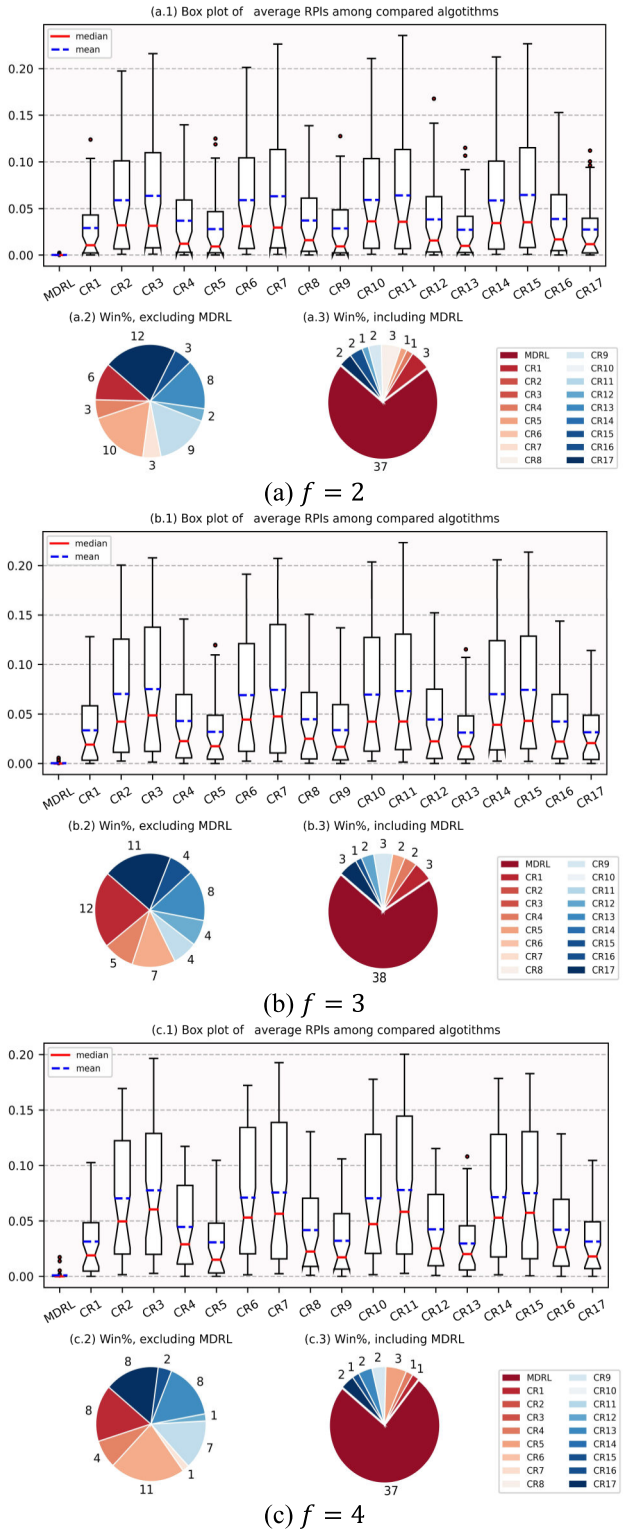
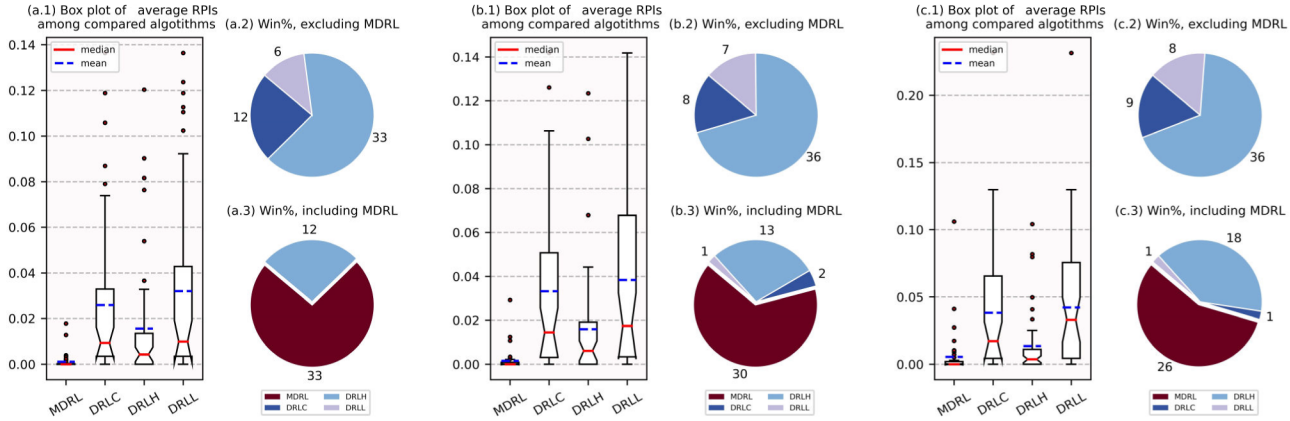


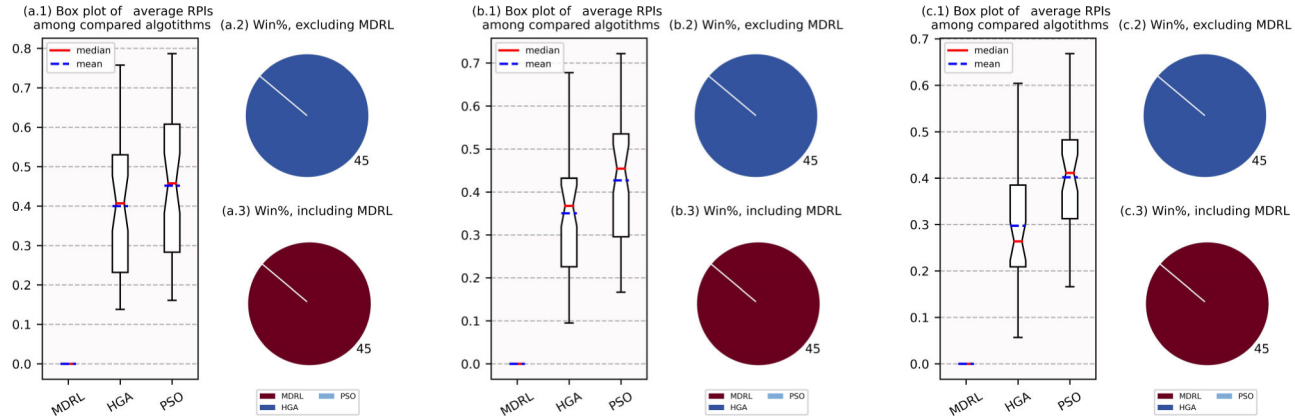
Fig. 7. Comparison among MDRL and PDRs.

2) Comparison With the DRL Methods: To further prove the effectiveness of the proposed MDRL, the other DRL methods intended for similar problems are compared. The



(a) $f = 2$ (b) $f = 3$ (c) $f = 4$

Fig. 8. Comparison among MDRL and DRLs.



(a) $f = 2$ (b) $f = 3$ (c) $f = 4$

Fig. 9. Comparison among MDRL and metaheuristics.

first compared DRL [46] (named DRLL) is designed for the FJSP with new job insertions; The second compared DRL [47] (named DRLC) is applied to solve the dynamic FJSP with random job arrivals; The third DRL [48] (named DRLH) is intended for solving the adaptive JSP.

The results of the experiment are presented in Fig. 8, which includes box plots and pie charts showing the win rate. As seen in Fig. 8(a.1), (b.1), and (c.1), the proposed MDRL has the lowest means and medians, followed by DRLH. DRLC and DRLL have the third and fourth ranks, respectively.

When there are 2 factories, pie charts in Fig. 8(a.2) and (a.3) indicate that the DRLH has the most of the smallest average makespans excluding MDRL. However, when the MDRL is included, the smallest averages are overwhelmingly generated by the MDRL (77.33%, 33 out of 45), while the remaining 12 smallest averages are produced by the DRLH. Referring to the pie charts in Fig. 8(b), it can be observed that the win rate of the DRLH (80%, 36 out of 45) drops significantly when the MDRL is included (28.89%, 13 out of 45). On the other hand, the win rate of the MDRL is 66.67% (30 out of 45). Similarly, the pie charts in Fig. 8(c) also confirm the same conclusion as seen in Fig. 8(a) and (b).

3) Comparison With the Metaheuristics: Due to the absence of metaheuristics specifically designed for the DJSP with job arrivals, two closely related metaheuristics, namely PSO [22] and Hybrid Genetic Algorithm (HGA) [28], are used as the compared algorithms. Both of these algorithms are proposed for the dynamic JSP while considering job arrivals. To adapt to the DJSP, all the arrived but unfinished jobs are sorted in ascending order based on their total processing times and are assigned to factories in turn. Both compared algorithms are run five repeats, and the averages of the five runs are set as the compared values.

The experimental results are presented in Fig. 9, which includes box plots and win rate pie charts. By analyzing the win rate pie charts in Fig. 9 (a.2), (b.2), and (c.2), it is evident that HGA outperforms PSO by having the smallest average makespan. However, when considering the MDRL, as shown in Fig. 9 (a.3), (b.3), and (c.3), it can be observed that MDRL obtains the smallest average makespans. Based on the overall performance, the three algorithms can be ranked as MDRL, HGA, and PSO.

B. Case Study

1) Scenario Description: To validate the engineering value of the proposed MDRL, it is tested by the cases from an automotive engine manufacturing company [49]. The company has two identical shopfloors that produce automobile

engine cooling fan parts including fan bracket (P_1), adapter flange (P_2), bracket adapter plate (P_3), and fan hub (P_4). Each part consists of five operations, each of which needs to be processed by a specific machine. Both shopfloors have five machines including miller (M_1), driller (M_2), machining center (M_3), lathe (M_4), and CNC lathe (M_5), and can process any part independently. In daily production, parts orders are released with high flexibility. This means that the above scenario can be simulated as the DJSP with job arrivals. The machining information of the four parts is obtained from practical production, and the average time interval between job arrivals (E_{ave}) is five minutes. As reference [49] does, the setup time is considered, and thus all the processing times are multiplied by 10 in the testing. The value of E_{ave} should be turned to 50. In this part, three scenarios with different scales are tested, and all the test data are publicly available.[1] The scheduling schemes for the following scenarios that are generated by the proposed MDRL are shown in detail in the Supplementary Material.

Scenario 1 (The Production with Small Scale [49]): 25 parts need to be machined in the automotive engine manufacturing company, but five of them are already in the shopfloors. The remaining 20 parts are released randomly. By using the proposed MDRL, a production scheme is obtained, and its makespan is 1598, which is 13.53% more productive than the value of 1848 from reference [49]. Additionally, the makespans of the two shopfloors are 1598 for Shopfloor 1 and 1422 for Shopfloor 2. The difference of 11.01% between them is not significant.

Scenario 2 (The Production with Medium Scale): The case presented in [49] is with 25 parts, but the scale is not large enough. Therefore, a case with 45 parts is introduced. The MDRL produces a scheduling scheme for the new case, which is shown in the Supplementary Material. With the scheduling scheme, the makespan of Shopfloor 1 is 2947 and that of Shopfloor 2 is 3082. The difference between the two shopfloor makespans is only 4.38%, which indicates that the proposed method works well for load balance between the shopfloors.

Scenario 3 (The Production with Large Scale): To demonstrate the adaptivity of the MDRL, a test case involving 110 parts is used. The MDRL generates a sophisticated scheduling scheme which results in a makespan of 6309 for Shopfloor 1 and 6521 for Shopfloor 2. Even for such a large case, the load difference between the two shopfloors is only 3.25%, which is still a small amount.

The practical application of the MDRL can demonstrate its engineering value. Moreover, by comparing the scheduling schemes obtained for the three scenarios with different scales, it can be observed that the MDRL method's ability to balance the load between the shopfloors does not weaken as the production scale increases, but instead becomes even stronger. This further proves the effectiveness of the proposed method.

C. Discussion

Comparative experiments are conducted in this section to compare the proposed MDRL, PDRs, DRL methods, and metaheuristics, which proves the effectiveness of the MDRL. The case study based on practical production further validates the utility of the MDRL and shows its capacity for load balance. The following reasons are summarized for the outperformance. First, the proposed multi-action MDP is fully adapted to the DJSP with job arrivals, where the state transition divides the rescheduling process into four cases according to job arrival times. These four cases determine how the agent reacts to job arrivals and facilitate better solutions. Additionally, the application of GIN is beneficial for the feature extraction of the dynamic disjunctive graph at each decision point, which provides the agent with a comprehensive perspective of the environment and helps make intelligent decisions. In contrast, the composite PDRs lack a global perspective. They assign a value to each waiting job according to one single feature, such as waiting time or processing time, and select the job with minimum or maximum "value" for next processing. Despite their simplicity, they are less adaptable, as their decision-making relies excessively on one certain feature. It is hard to find a composite PDR that can perform well in all scenarios. Last, the multi-action policy makes it possible to assign jobs and sequence them simultaneously, which facilitates problem-solving. Most importantly, the proposed method can learn from the scheduling environment and adapt to different scenarios. In summary, the designed agent can have a comprehensive observation of the scheduling environment and can respond to uncertain job arrivals with intelligent decisions. Thus, the proposed MDRL can outperform.

VI. Conclusion and Future Research

A novel method called hierarchical multi-action DRL has been proposed for the DJSP with job arrivals. This method takes into account the distributed and dynamic characteristics of this problem by designing a multi-action MDP with a two-dimensional set of actions. The first dimension consists of the operations, while the second dimension represents a factory set that is established purposefully for each operation. The GIN extracts environment features from the disjunctive graph, and two decision policies are used for operation and factory selection. To handle the uncertain disturbance from job arrivals, the state transition is classified into four typical cases based on job arrival times. The state can be updated according to the four cases, leading to a more rational response to the disturbance caused by job arrivals, which facilitates better solution generation. The MDRL method has been tested on 1350 different instances, and the results show its effectiveness when compared to PDRs, DRL methods, and metaheuristics. Additionally, a case study from the practical industry further proves the applicability of the MDRL.

As the globalized economy flourishes, there is a greater need for distributed manufacturing to help reduce production and transportation costs. Thus, scheduling job tasks effectively among these factories while considering transportation costs can be challenging. In addition, as a common dynamic event in the industry, the machine breakdown should also be studied. To make the most of production resources and increase productivity, further research should be conducted while taking into account these constraints.

[1] https://github.com/Annie-Summer/MDRL_Dataset

REFERENCES

[1] M. A. Sahman, "A discrete spotted hyena optimizer for solving distributed job shop scheduling problems," *Appl. Soft Comput.*, vol. 106, Jul. 2021, Art. no. 107349.

[2] I. Chaouch, O. B. Driss, and K. Ghedira, "A novel dynamic assignment rule for the distributed job shop scheduling problem using a hybrid ant-based algorithm," *Int. J. Speech Technol.*, vol. 49, no. 5, pp. 1903–1924, May 2019.

[3] Q. H. Liu, X. Y. Li, L. Gao, and J. X. Fan, "A multi-MILP model collaborative optimization method for integrated process planning and scheduling problem," *IEEE Trans. Eng. Manag.*, vol. 71, pp. 4574–4586, 2022.

[4] M. Durasevic and D. Jakobovic, "A survey of dispatching rules for the dynamic unrelated machines environment," *Exp. Syst. Appl.*, vol. 113, pp. 555–569, Dec. 2018.

[5] J.-P. Huang, Q.-K. Pan, Z.-H. Miao, and L. Gao, "Effective constructive heuristics and discrete bee colony optimization for distributed flowshop with setup times," *Eng. Appl. Artif. Intell.*, vol. 97, Jan. 2021, Art. no. 104016.

[6] D. Lei and B. Su, "A multi-class teaching–learning-based optimization for multi-objective distributed hybrid flow shop scheduling," *Knowl.-Based Syst.*, vol. 263, Mar. 2023, Art. no. 110252.

[7] H. Xiong, H. Fan, G. Jiang, and G. Li, "A simulation-based study of dispatching rules in a dynamic job shop scheduling problem with batch release and extended technical precedence constraints," *Eur. J. Oper. Res.*, vol. 257, no. 1, pp. 13–24, Feb. 2017.

[8] J. Duan and J. Wang, "Robust scheduling for flexible machining job shop subject to machine breakdowns and new job arrivals considering system reusability and task recurrence," *Exp. Syst. Appl.*, vol. 203, Oct. 2022, Art. no. 117489.

[9] X. Y. Li and L. Gao, "A hybrid genetic algorithm and Tabu search for multi-objective dynamic JSP," in *Effective Methods for Integrated Process Planning and Scheduling* (Engineering Applications of Computational Methods), X. Y. Li and L. Gao, Eds. Berlin, Germany: Springer, 2020, pp. 377–403.

[10] R. Wang, G. Wang, J. Sun, F. Deng, and J. Chen, "Flexible job shop scheduling via dual attention network-based reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 3, pp. 3091–3102, Mar. 2024.

[11] R. Liu, R. Piplani, and C. Toro, "Deep reinforcement learning for dynamic scheduling of a flexible job shop," *Int. J. Prod. Res.*, vol. 60, no. 13, pp. 4049–4069, 2022.

[12] K. Lei et al., "A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem," *Exp. Syst. Appl.*, vol. 205, Nov. 2022, Art. no. 117796.

[13] R. Chen, W. Li, and H. Yang, "A deep reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for the job-shop scheduling problem," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1322–1331, Feb. 2023.

[14] W. Song, X. Chen, Q. Li, and Z. Cao, "Flexible job-shop scheduling via graph neural network and deep reinforcement learning," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1600–1610, Feb. 2023.

[15] S. Luo, L. Zhang, and Y. Fan, "Real-time scheduling for dynamic partial-no-wait multiobjective flexible job shop by deep reinforcement learning," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 4, pp. 3020–3038, Oct. 2022.

[16] F. Chan, S. Chung, and P. Chan, "An adaptive genetic algorithm with dominated genes for distributed scheduling problems," *Exp. Syst. Appl.*, vol. 29, no. 2, pp. 364–371, Aug. 2005.

[17] H. Z. Jia, J. Y. H. Fuh, A. Y. C. Nee, and Y. F. Zhang, "Integration of genetic algorithm and Gantt chart for job shop scheduling in distributed manufacturing systems," *Comput. Ind. Eng.*, vol. 53, no. 2, pp. 313–320, Sep. 2007.

[18] B. Naderi and A. Azab, "An improved model and novel simulated annealing for distributed job shop problems," *Int. J. Adv. Manuf. Technol.*, vol. 81, nos. 1–4, pp. 693–703, Oct. 2015.

[19] C.-Y. Hsu, B.-R. Kao, V. L. Ho, and K. R. Lai, "Agent-based fuzzy constraint-directed negotiation mechanism for distributed job shop scheduling," *Eng. Appl. Artif. Intell.*, vol. 53, pp. 140–154, Aug. 2016.

[20] E.-D. Jiang, L. Wang, and Z.-P. Peng, "Solving energy-efficient distributed job shop scheduling via multi-objective evolutionary algorithm with decomposition," *Swarm Evol. Comput.*, vol. 58, Nov. 2020, Art. no. 100745.

[21] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *J. Scheduling*, vol. 12, no. 4, pp. 417–431, Aug. 2009.

[22] Z. Wang, J. Zhang, and S. Yang, "An improved particle swarm optimization algorithm for dynamic job shop scheduling problems with random job arrivals," *Swarm Evol. Comput.*, vol. 51, Dec. 2019, Art. no. 100594.

[23] S. A. Fahmy, S. Balakrishnan, and T. Y. ElMekkawy, "A generic deadlock-free reactive scheduling approach," *Int. J. Prod. Res.*, vol. 47, no. 20, pp. 5657–5676, Oct. 2009.

[24] S. Shady, T. Kaihara, N. Fujii, and D. Kokuryo, "A novel feature selection for evolving compact dispatching rules using genetic programming for dynamic job shop scheduling," *Int. J. Prod. Res.*, vol. 60, no. 13, pp. 4025–4048, Jul. 2022.

[25] O. Holthaus and C. Rajendran, "Efficient jobshop dispatching rules: Further developments," *Prod. Planning Control*, vol. 11, no. 2, pp. 171–178, Jan. 2000.

[26] J. Luo, S. Fujimura, D. El Baz, and B. Plazolles, "GPU based parallel genetic algorithm for solving an energy efficient dynamic flexible flow shop scheduling problem," *J. Parallel Distrib. Comput.*, vol. 133, pp. 244–257, Nov. 2019.

[27] W. Yan and D. Yu, "Optimal scheduling and decision making method for dynamic flexible job shop," *J. Syst. Simul.*, vol. 32, no. 11, pp. 2073–2083, Nov. 2020.

[28] K. B. Ali, A. J. Telmoudi, and S. Gattoufi, "Improved genetic algorithm approach based on new virtual crossover operators for dynamic job shop scheduling," *IEEE Access*, vol. 8, pp. 213318–213329, 2020.

[29] I.-B. Park and J. Park, "Scalable scheduling of semiconductor packaging facilities using deep reinforcement learning," *IEEE Trans. Cybern.*, vol. 53, no. 6, pp. 3518–3531, Jun. 2023.

[30] C. L. Liu and T. H. Huang, "Dynamic job-shop scheduling problems using graph neural network and deep reinforcement learning," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 53, no. 11, pp. 6836–6848, Nov. 2023.

[31] T. Zhou et al., "Reinforcement learning for online optimization of job-shop scheduling in a smart manufacturing factory," *Adv. Mech. Eng.*, vol. 14, no. 3, Mar. 2022, Art. no. 168781322210861.

[32] D. Johnson, G. Chen, and Y. Lu, "Multi-agent reinforcement learning for real-time dynamic production scheduling in a robot assembly cell," *IEEE Robot. Autom. Lett.*, vol. 7, no. 3, pp. 7684–7691, Jul. 2022.

[33] F. Grumbach, A. Müller, P. Reusch, and S. Trojahn, "Robust-stable scheduling in dynamic flow shops based on deep reinforcement learning," *J. Intell. Manuf.*, vol. 35, no. 2, pp. 667–686, Dec. 2022.

[34] H. Wang, J. Cheng, C. Liu, Y. Zhang, S. Hu, and L. Chen, "Multi-objective reinforcement learning framework for dynamic flexible job shop scheduling problem with uncertain events," *Appl. Soft Comput.*, vol. 131, Dec. 2022, Art. no. 109717.

[35] Q. Yan, H. Wang, and F. Wu, "Digital twin-enabled dynamic scheduling with preventive maintenance using a double-layer Q-learning algorithm," *Comput. Oper. Res.*, vol. 144, Aug. 2022, Art. no. 105823.

[36] L. I. U. Yahui, X. W. Shen, G. U. Xinghai, P. Tao, B. A. O. Jinsong, and Z. Dan, "A dual-system reinforcement learning method for flexible job shop dynamic scheduling," *J. Shanghai Jiaotong Univ.*, vol. 56, no. 9, pp. 1262–1275, Sep. 2022.

[37] J. Julaiti, S. Oh, D. Das, and S. Kumara, "Stochastic parallel machine scheduling using reinforcement learning," *J. Adv. Manuf. Process.*, vol. 4, no. 4, Oct. 2022, Art. no. e10119.

[38] K. Lei, P. Guo, Y. Wang, J. Zhang, X. Y. Meng, and L. M. Qian, "Large-scale dynamic scheduling for flexible job-shop with random arrivals of new jobs by hierarchical reinforcement learning," *IEEE Trans. Ind. Informat.*, vol. 20, no. 1, pp. 1007–1018, May 2023.

[39] L. Zhao, J. Fan, C. Zhang, W. Shen, and J. Zhuang, "A DRL-based reactive scheduling policy for flexible job shops with random job arrivals," *IEEE Trans. Autom. Sci. Eng.*, early access, May 12, 2023, doi: 10.1109/TASE.2023.3271666.

[40] H. Wang and Y. Yu, "Exploring multi-action relationship in reinforcement learning," in *PRICAI 2016: Trends in Artificial Intelligence* (Lecture Notes in Computer Science), R. Booth and M. L. Zhang, Eds. Cham, Switzerland: Springer, 2016, pp. 574–587.

[41] I. Sabuncuoglu and M. Bayiz, "Analysis of reactive scheduling problems in a job shop environment," *Eur. J. Oper. Res.*, vol. 126, no. 3, pp. 567–586, Nov. 2000.

[42] Z. Kong, X. Jin, Z. Xu, and B. Zhang, "Spatio-temporal fusion attention: A novel approach for remaining useful life prediction based on graph neural network," *IEEE Trans. Instrum. Meas.*, vol. 71, pp. 1–12, 2022.

[43] Z. Wang, M. Eisen, and A. Ribeiro, "Learning decentralized wireless resource allocations with graph neural networks," *IEEE Trans. Signal Process.*, vol. 70, pp. 1850–1863, 2022.

[44] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" 2018, *arXiv:1810.00826*.
[45] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
[46] S. Luo, "Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning," *Appl. Soft Comput.*, vol. 91, Jun. 2020, Art. no. 106208.
[47] J. Chang, D. Yu, Y. Hu, W. He, and H. Yu, "Deep reinforcement learning for dynamic flexible job shop scheduling with random job arrival," *Processes*, vol. 10, no. 4, p. 760, Apr. 2022.
[48] B.-A. Han and J.-J. Yang, "Research on adaptive job shop scheduling problems based on dueling double DQN," *IEEE Access*, vol. 8, pp. 186474–186495, 2020.
[49] J.-P. Huang, L. Gao, X.-Y. Li, and C.-J. Zhang, "A cooperative hierarchical deep reinforcement learning based multi-agent method for distributed job shop scheduling problem with random job arrivals," *Comput. Ind. Eng.*, vol. 185, Nov. 2023, Art. no. 109650.

Jiang-Ping Huang received the M.S. degree from Shanghai University, China, in 2021. She is currently pursuing the Ph.D. degree with the Huazhong University of Science and Technology, China. Her current research interests include intelligent optimization, reinforcement learning, and scheduling.

Liang Gao (Senior Member, IEEE) received the B.Sc. degree from Xidian University, China, in 1996, and the Ph.D. degree from the Huazhong University of Science and Technology (HUST), China, in 2002. He is currently a Full Professor with HUST. He has authored over 250 articles. He serves as the Editor-in-Chief for *IET Collaborative Intelligent Manufacturing* and an Associate Editor for *Swarm and Evolutionary Computation* and *Journal of Industrial and Production Engineering*.

Xin-Yu Li (Member, IEEE) received the Ph.D. degree in industrial engineering from the Huazhong University of Science and Technology (HUST), China, in 2009. He is currently a Professor with the Department of Industrial and Manufacturing Systems Engineering, State Key Laboratory of Digital Manufacturing Equipment and Technology, School of Mechanical Science and Engineering, HUST. He has published more than 90 refereed articles. His research interests include intelligent algorithm, big data, and machine learning.