Large-Scale Dynamic Scheduling for Flexible Job-Shop With Random Arrivals of New Jobs by Hierarchical Reinforcement Learning

Kun Lei, Peng Guo, Yi Wang, Jian Zhang, Xiangyin Meng, and Linmao Qian

***Abstract*—As the intelligent manufacturing paradigm evolves, it is urgent to design a near real-time decision-making framework for handling the uncertainty and complexity of production line control. The dynamic flexible job shop scheduling problem (DFJSP) is frequently encountered in the manufacturing industry. However, it is still challenging to obtain high-quality schedules for DFJSP with dynamic job arrivals in real-time, especially facing thousands of operations from a large-scale scene with complex contexts in an assembly plant. This article aims to propose a novel end-to-end hierarchical reinforcement learning framework for solving the large-scale DFJSP in near real-time. In the DFJSP, the processing information of newly arrived jobs is unknown in advance. Besides, two optimization tasks, including job operation selection and job-to-machine assignment, have to be handled, which means multiple actions must be controlled simultaneously. In our framework, a higher-level layer is designed to automatically divide the DFJSP into subproblems, i.e., static FJSPs with different scales. And two lower-level layers are constructed to solve the subproblems. In particular, one layer based on a graph neural network is in charge of sequencing job operations, and another layer based on a multilayer perceptron is used to assign a machine to process the job operations. Numerical experiments, including offline training and online testing, are conducted on several instances with different scales. The results verify the superior performance of the proposed framework compared with existing dynamic scheduling methods, such as well-known dispatching rules and metaheuristics.**

***Index Terms*—Dynamic flexible job shop scheduling problem (DFJSP), graph neural network (GNN), hierarchical reinforcement learning (HRL), Markov decision process (MDP), real-time optimization.**

Manuscript received 25 March 2023; accepted 30 April 2023. Date of publication 3 May 2023; date of current version 11 December 2023. This work was supported in part by the National Key Research and Development Program of China Plan under Grant 2020YFB1712200, and in part by the MOE (Ministry of Education in China) Project of Humanities and Social Sciences under Grant 21YJC630034. Paper no. TII-23-1018. *(Corresponding author: Peng Guo.)*

Kun Lei, Jian Zhang, Xiangyin Meng, and Linmao Qian are with the School of Mechanical Engineering, Southwest Jiaotong University, Chengdu 610031, China (e-mail: kunlei@my.swjtu.edu.cn; zhangjian@swjtu.edu.cn; xymeng@swjtu.edu.cn; linmao@swjtu.edu.cn).

Peng Guo is with the School of Mechanical Engineering, Southwest Jiaotong University, Chengdu 610031, China, and also with the Technology and Equipment of Rail Transit Operation and Maintenance Key Laboratory of Sichuan Province, Southwest Jiaotong University, Chengdu 610031, China (e-mail: pengguo318@swjtu.edu.cn).

Yi Wang is with the Department of Mathematics, Auburn University at Montgomery, Montgomery, AL 36124-4023 USA (e-mail: ywang2@aum.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2023.3272661.

Digital Object Identifier 10.1109/TII.2023.3272661

I. Introduction

THE dynamic flexible job shop scheduling problem (DFJSP) acts as one of the most commonly encountered production scheduling problems in the modern manufacturing environment, including the semiconductor production process and automobile assembly line [1]. Generally, the DFJSP consists of two key elements, jobs, and machines. In today's complicated manufacturing systems, the uncertainty of jobs' arrival on the shop floor disturbs the planned schedule and even deteriorates the efficiency of a production line. Therefore, online rescheduling methods need to be exploited to solve this kind of scheduling problem.

In DFJSP, management decision-making focuses on dynamically rescheduling jobs to available machines with the minimization of the overall production cost, e.g., the overall makespan. DFJSP is a variant of a job-shop scheduling problem (JSSP) and a flexible job-shop scheduling problem (FJSP). All of them are NP-hard combinatorial optimization problems [2]. There are two peculiarities of the DFJSP. The first one is that new jobs are unknown a priori and dynamically arriving. It means that the rescheduling framework has to make real-time decisions instead of offline ones. The second peculiarity of DFJSP is to control multiple actions of job operation selection and job-to-machine assignment simultaneously. Therefore, it is impractical to find optimal solutions based on exact methods. In the past decades, most DFJSP research has been performed around metaheuristic algorithms [3]. But they are time-consuming and infeasible for real-time optimization. Recently, many learning-based methods have been applied to efficiently solve the combinatorial optimization problem, such as the traveling salesman problem (TSP) and vehicle routing problem (VRP) [4]. The successful applications of learning-based methods motivate us to explore their potential in resolving large-scale DFJSP in real time. Next, we briefly review two research areas related to the underlying study: previous studies related to DFJSP and reinforcement learning (RL) -based methods in solving scheduling problems.

Previous studies related to DFJSP: Previous methods for DFJSP can be divided into three categories as follows:

1) completely reactive scheduling;
2) predictive reactive scheduling;
3) robust proactive scheduling [5].

In completely reactive scheduling methods, no firm schedule is given in advance, and the scheduling decisions are generated locally in real-time when dynamic events occur. In this way, the dynamic scheduling problem is partitioned into a series of static subproblems, each of which is solved for quick response in the case of a dynamic event. The method is suitable for a production environment with frequent dynamic events and high uncertainty due to its practicability and real-time nature. The most commonly reactive method is dispatching rules and their extensions [6], [7], [8]. However, the reactive method cannot guarantee the overall optimization of a dynamic problem because it myopically optimizes subproblems that are not independent of each other in a long-term perspective.

The second category, the predictive-reactive method, generates a schedule in advance to optimize the shop performance, ignoring the possibility of disruptions on the shop floor. Thus, the predictive schedule would be modified (rescheduled) to achieve feasibility and effectiveness when a disruption appears during the execution. The key issues of this method are whether to perform rescheduling at a particular disruption and what rescheduling approach will be used. To this end, the predictive-reactive method can be further divided into two types: 1) periodic rescheduling; and 2) event-driven rescheduling. Some operational research-based (OR) heuristic and metaheuristic methods [9], [10] are designed to reschedule. However, the periodic and event-driven rescheduling strategies are not suitable for some real-world production environments with high uncertainty. The robust scheduling method [11], [12] aims to build a predictive schedule to minimize the effects of disruptions so that the actual execution deviates from the predictive schedule and is avoided as much as possible. However, this method usually reduces machine utilization, and thus leads to low production efficiency.

Traditional algorithms suffer from the dilemma between efficiency and quality. Existing exact methods and metaheuristic algorithms cannot generate a solution within an allowable time period for this type of dynamic scheduling problem [2]. On the other hand, dispatching rules produce a solution in near real-time at the cost of quality due to its simple mathematical operation involved in the decision-making process [48]. Recently, genetic programming (GP) has gradually become a popular method to explore effective dispatching rules when facing dynamic events using comprehensive shop floor information. Compared with traditional manually-made priority rules, GP designs rules based on automatically iterative, population-based, and randomized evolution. For example, [34] focused on feature selection to evolve scheduling heuristics and improve the performance for DFJSP automatically. To improve the training efficiency and effectiveness of the GP, [36] proposed a surrogate-assisted evolutionary multitask algorithm that contained the phenotypic characterization for measuring the behaviors of scheduling rules and building a surrogate for each task accordingly. The authors in [37] designed a tree-based GP in which they proposed a recombinative mechanism to provide guidance for recognizing effective and adaptive recombination for parents to produce offspring. The authors in [38] proposed a multitree GP with an archive to evolve the routing and sequencing rules for DFJSP. There is also research that aims to heuristic generation based on GP [39].

RL-based methods in solving scheduling problems: Although the dispatching rule-based methods can generate a feasible solution in real-time, efficient dispatching rules commonly require substantial domain expertise and trial-and-error [13]. Recently, lots of RL-based methods [13], [16], [17], [18], [19] are presented to improve the effectiveness in solving combinatorial optimization problems. Those methods can get high-quality solutions within a competitive computational time period compared with previously surveyed state-of-the-art exact or heuristic methods. However, RL-based methods mainly focus on routing problems such as TSP, VRP, and JSSP. In Table I, we classify the recent research on DJSP and DFJSP as two main categories based on [27]: 1) integrated dispatching rules into DRL methods; and 2) end-to-end learning methods. The main disparity between these two types of methods is that the end-to-end DRL agent directly outputs the solution by extracting the information features of the instance input without designing and adapting traditional optimization research methods. In detail, [43] provided a decentralized Q-learning algorithm to choose efficient scheduling rules on each rescheduling point for JSP with random job arrival. For DFJSP, [44] designed a Q-learning algorithm to choose and apply the most suitable machine selection rule and dispatching rule for a partially flexible job shop scheduling problem with new job insertions. The authors in [14] proposed a double deep Q-networks (DDQN) to choose the most proper rule from six designed composite dispatching rule pools to solve the DFJSP, with the objective of minimizing total tardiness. The authors in [45] proposed a two-hierarchy deep Q-network for the multiobjective dynamic flexible job shop scheduling problem with new job insertions. The authors in [46] proposed a soft ϵ-greedy behavior policy in the double deep Q-networks architecture to improve the performance of rescheduling. The authors in [47] used two deep Q-learning networks and a real-time processing framework to process each dynamic event and generate a complete scheduling scheme. Then, the schedule will be further optimized by the proposed local search algorithm. The authors in [42] proposed a multiagent proximal policy optimization algorithm containing three agents in which the objective agent periodically determines the temporary objectives to be optimized, and then the job agent and machine agent choose a job selection rule and machine assignment rule for a DFJSP. The authors in [48] used the double deep Q-network algorithm to train the scheduling agents for a flexible job shop with constant job arrivals, in which specialized state and action representations are proposed to handle the variable specification of the problem in dynamic scheduling. Although, integrating dispatching rules into DRL methods can learn to choose efficient dispatching rules and solve the DFJSP in near real-time, the quality of the scheduling solutions depends heavily on a designed dispatching rule, which needs complex domain knowledge [13], [16]. *To the best of our knowledge,*

TABLE I
EXISTING RESEARCH ON RL-BASED ALGORITHMS FOR DYNAMIC SCHEDULING PROBLEM

Type	Work	Problem	Dynamic events	Objective	Algorithm
Integrated dispatching rules into DRL	Wang et al. 2020 [43]	DJSP	Uncertain assembly times	Total earliness penalty, Completion time cost	Dual Q-learning
	Bouazza et al. 2017 [44]	DFJSP	Random job arrival	Makespan, Total weighted completion time	Q-learning
	Luo et al. 2020 [14]	DFJSP	Random job arrival	Total tardiness	DDQN
	Luo et al. 2021 [45]	DFJSP	Random job arrival	Total tardiness, Machine utilization rate	DDQN
	Chang et al. 2022 [46]	DFJSP	Random job arrival	Penalties for earliness and tardiness	DDQN
	Wang et al. 2022 [47]	DFJSP	Multiple events	Makespan, Average machine utilization rate	DQN
	Liu et al. 2022 [48]	DFJSP	Random job arrival	Total tightness	DDQN
	Luo et al. 2022 [42]	DFJSP	Random job arrival	Total tardiness, Average machine utilization rate	PPO
End-to-end methods	Ours	DFJSP	Random job arrival	Makespan	HRL (DDQN, Multi-PPO)

TABLE II
THE DEFINITION OF NOTATION

Notations	Definitions
$\mathcal{M}$	The machine set $\mathcal{M} = \{M_1, \ldots, M_m\}$, there are m machines indexed by M_m
$\mathcal{J}$	Jobs set $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$, there are n machines indexed by J_n
$\mathcal{O}_i$	Each job has n_i operations is presented by $\mathcal{O}_i = \{O_{i1}, O_{i2}, \ldots, O_{in_i}\}$
p_{ijk}	The processing time for operation j of job i on machine k
$\mathcal{M}_{ij}$	A subset of machines for operation O_{ij} is presented by $\mathcal{M}_{ij} \subseteq \mathcal{M}$
G	Disjunctive graph $G = (\mathcal{O}, \mathcal{C}, \mathcal{D})$
$\mathcal{O}$	$\mathcal{O} = \{O_{ij} \vert \forall i, j\} \cup \{$"0", "1"$\}$ is the set of all operations where "0" and "1" denote dummy starting and ending nodes
$\mathcal{C}$	A set of conjunctive arcs which represent the precedence constraints between consecutive operations from the same job
$\mathcal{D}$	A set of disjunction (undirected) arcs, in which each arc connects a pair of operations that can be processed by the same machine
$\mathcal{D}_t^o$	$\mathcal{D}_t^o \subseteq \mathcal{D}$ is the disjunctive arcs which have been assigned directions
$\hat{G}_t$	$\hat{G}_t = (\mathcal{O}_{new} \cup \mathcal{O}_{remain}, \mathcal{C} \cup \mathcal{D}_t^o)$ is an approximation of G_t
$C_{\max}$	The makespan
s_t^*, a_t^*, r_t^*	The state, action, reward of higher-level agent
$I(s_t^*)$	A binary indicator that equals to 1 if the action is to release
time_{avg}	The average processing time of each operation between the timestep t and $t+1$
$\mathcal{J}_{\text{new}}$	The static subproblem consists of a set of released new jobs
$\mathcal{J}_{\text{remain}}$	The static subproblem consists of a set of remaining jobs
$\mathcal{O}_{\text{new}}$	The set of operations of new released jobs
$\mathcal{O}_{\text{remain}}$	The set of operations of remaining uncompleted jobs
$s_t^{\triangledown,o}, a_t^{\triangledown,o}, r_t^{\triangledown,o}$	The state, action, reward of the job operation selection agent
$s_t^{\triangledown,m}, a_t^{\triangledown,m}, r_t^{\triangledown,m}$	The state, action, reward of the machine assignment agent
$LB_t(O_{ij})$	The lower bound of the completion time of O_{ij} at timestep t
$I_t(O_{ij})$	A binary indicator which equals to 1 if O_{ij} is scheduled.
$T_t(k)$	The completion time for machine k before timestep t
$\triangle T$	The period of rescheduling
U	The load factor of job shop
μ_a	The average processing time of all operations
μ_o	The average operation number of each job

there is no research to solve the DFJSP via an end-to-end learning-based method. Moreover, the aforementioned works motivate us to introduce an end-to-end learning-based method to solve the large-scale DFJSP in near real-time.

This article proposes a novel end-to-end hierarchical reinforcement learning (HRL) framework to solve the large-scale DFJSP. Taking into account the persistent impact of the random arriving jobs on the overall optimization of DFJSP, we designed a higher-level agent to dynamically and automatically decide whether to reschedule after a new job arrives or to wait to receive more future jobs. Once the higher-level agent determines to reschedule, the DFJSP is divided into a series of static FJSPs subproblems of different scales. We then employ two lower-level agents to cope with the static FJSP cooperatively. In particular, one policy based on a graph neural network is trained to schedule a job operation, and the second policy is trained to assign a machine to process the job operation. To verify the effectiveness of our framework, a number of experiments, including offline training and online testing, are conducted on several large-scale instances with different scales. In this way, the hierarchical framework can learn to realize near real-time decision-making at each rescheduling decision point. Our main contributions are listed as follows.

1) We proposed a novel end-to-end hierarchical reinforcement learning framework to solve the large-scale DFJSP in near real-time efficiently. The definitions of state, action, and reward in each Markov decision process (MDP) of each layer in the HRL framework are given for addressing DFJSP.
2) In the proposed HRL framework, we designed a DDQN for the higher-level agent. Moreover, we used a multipointer network (MPN) architecture to parameterize the policies for the two lower-level agents. Meanwhile, a multiproximal policy optimization (multi-PPO) algorithm is proposed to train the architecture.
3) The experimental results show that the proposed HRL framework outperforms previous traditional rescheduling and learning-based approaches. Furthermore, owing to the scalability of the designed model, it can be

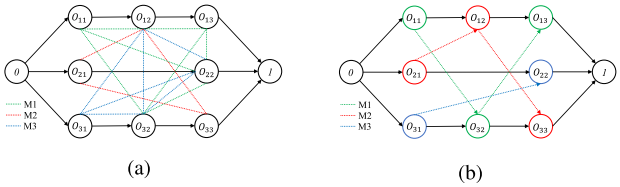


Fig. 1. Disjunctive graph representation of FJSP. (a) Disjunctive graph for an FJSP instance. (b) Example of a feasible solution.

deployed to solve the scheduling problems with different scales without re-training. Also, it can be generalized to the scheduling problems with different data distributions from the training data.

The rest of this article is organized as follows. Section II describes the background information on the concepts and technologies relevant to our study. Section III details the HRL framework. Section IV gives the experimental results and discussions. Finally, Section V concludes this article.

II. Problem Formulation

In this section, we formally introduce the definition of the DFJSP and the disjunctive graph representation of the static FJSP. The commonly used notation in this article is shown in Table II.

Dynamic flexible job-shop scheduling problem: The DFJSP with random new job arrivals can be defined as follows. There are n successive new jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ that dynamically arrive, and m machines in the set $\mathcal{M} = \{M_1, \ldots, M_m\}$ are available to process them with minimal production cost. Job J_i consists of a specific sequence of n_i consecutive operations $\mathcal{O}_i = \{O_{i1}, O_{i2}, \ldots, O_{in_i}\}$ with precedence constraints. Operation O_{ij} of job J_i 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 $k \in \mathcal{M}_{ij}$. In our case, the optimization goal is to minimize the makespan $C_{\max}$ described by:

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

where C_{in_i} denotes the completion time of the last operation of job i.

Disjunctive graph of static FJSP: Generally, an FJSP can be defined as a disjunctive graph $G = (\mathcal{O}, \mathcal{C}, \mathcal{D})$. In the graph, $\mathcal{O} = \{O_{ij} | \forall i, j\} \cup \{\text{“0”}, \text{“1”}\}$ is the set of all operations, where “0″ and “1″ denote dummy starting and ending nodes, as illustrated in Fig. 1. Moreover, $\mathcal{C}$ is a set of conjunctive arcs which represent the precedence constraints between consecutive operations from the same job. $\mathcal{D}$ is a set of disjunction (undirected) arcs, in which each arc connects a pair of operations that can be processed by the same machine. Fig. 1(a) represents a static FJSP instance, in which black arrows are conjunctive arcs, and colored lines are disjunctive arcs that correspond to eligible machine cliques in different colors. Furthermore, Fig. 1(b) is a complete solution, in which disjunctive arcs are allocated in directions.

A DFJSP typically faces the following three difficulties.

1) The scale of the problem is typically very large in modern production scenarios in which tens of thousands of products (jobs) are waiting for processing on numerous machines. For example, In line with our discussions with production managers of a Chinese engineering equipment manufacturing enterprise, nearly 5000 assembly operations are involved in the QS650 type railway ballast cleaner assembly plant. Even for a static FJSP, finding a high-quality solution relying on OR methods (e.g., a mixed integer programming (MIP) model using modern solvers such as Gurobi) or metaheuristic algorithms in a reasonable running time is often impractical due to its NP-hardness [20].
2) The new jobs dynamically online arrive. In this case, the scheduling methods with the offline style can not work for dealing with dynamic events. If all jobs of one day are known ahead, it is a static FJSP without uncertainty and can be optimized by exact or metaheuristic methods in advance, even for large-scale instances. However, knowing all jobs in advance is almost impossible in the real world. It is necessary to obtain a high-quality schedule in a very short computational time for responding the dynamic events. In this case, these methods cannot be applicable.
3) The optimization goal of the DFJSP problem is the overall production cost, i.e., makespan $C_{\max}$. However, the existing rescheduling methods summarized above cannot optimize the problem globally. For example, the completely reactive scheduling method myopically reschedules to optimize each subproblem when a new job arrives, ignoring that the divided subproblems are not independent of each other. While the periodic rescheduling is triggered when the manufacturing system receives a preprogrammed number of new jobs. It is too simple when facing the DFJSP to obtain a globally optimal schedule.

III. Method

Our method is initially motivated by those challenges listed above. We next present a novel end-to-end hierarchical reinforcement learning framework containing a higher-level agent and two lower-level agents. In this framework, the higher- and lower-level agents collaborate to solve the large-scale DFJSP. In detail, the higher-level agent maintains a job *buffer* without a fixed capacity to cache newly arrived jobs (the new job arriving sequentially which can be seen in arrows 1–3 in Fig. 2). Furthermore, the higher-level agent *determines whether to release the cached jobs (described by arrows 6–7 in Fig. 2) to the lower-level agents or to wait for more jobs when a new job arrives (described by arrow 4–5 in Fig. 2)*. Although caching newly arrived jobs could delay the scheduling of earlier arrived jobs, collecting more arrived jobs provides the possibility to explore global optimization for rescheduling, i.e., more effectively scheduling jobs to machines. On the other hand, waiting for more jobs will increase machine idle time. Once the higher-level agent decides to release the cached jobs to the lower-level agents, a new static FJSP is obtained and optimized. The job buffer is then cleared to cache the next arriving jobs. In this way, a complex DFJSP is divided into a series of static FJSPs. We next formulate this

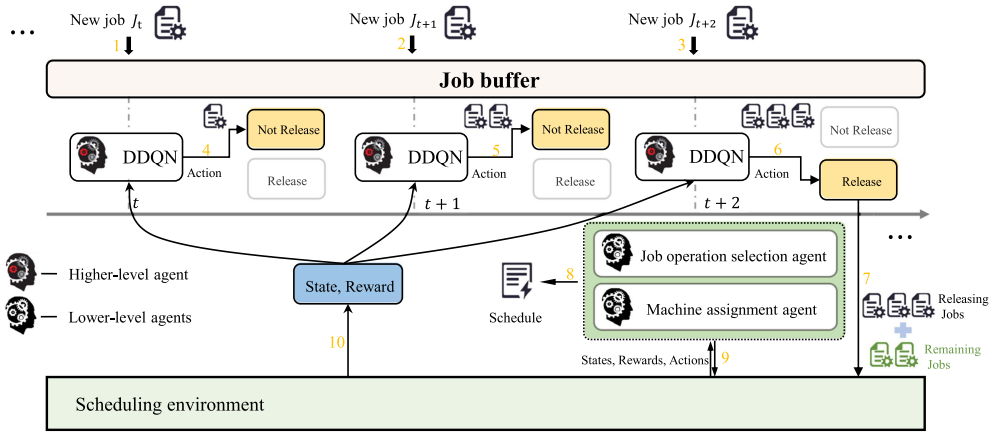


Fig. 2. End-to-end hierarchical reinforcement learning framework for DFJSP.

procedure as an MDP and define the state, action, and reward consequently.

A static subproblem includes jobs with uncompleted job operations prior to rescheduling and the newly released jobs. Two tasks, including job operation selection and job-to-machine assignments, must be solved in a static subproblem. Unfortunately, in the standard RL setting, an RL agent interacts with the environment in the following way: The agent in a state makes an action based on a policy at each timestep, then obtains a reward and transfers to the next state. In this case, the RL agent selects an action from a single action space. However, two actions must be controlled simultaneously at each timestep in the static FJSP. To this end, *two lower-level agents are constructed to dispatch a job operation and assign a compatible machine for processing it, respectively*.

Overall, the higher-level agent eyes global optimization from a long-term perspective. While the lower-level agents optimize a static subproblem and work together with the higher-level agent to optimize the DFJSP. The details of the framework are given in the following two subsections. In the following sections, x^* and x^∇ denote the variable x located in high-level and lower-level agents, respectively. We first introduce the workflow of the higher-level agent.

A. Higher-Level Agent

Assume that n new jobs will randomly arrive on the shop floor. At each timestep t ($t \in \{1, \ldots, n\}$), the higher-level agent automatically decides whether to release the cached jobs or to wait for caching more jobs. As shown in Fig. 2, at timesteps t and $t+1$, the higher-level agent decides to wait to collect more new jobs, i.e., the action 'not release' (arrows 4–5 in Fig. 2) is taken. Then, at timestep $t+2$, the higher-level agent releases the cached three jobs to the lower-level agents (illustrated by arrows 6–7 in Fig. 2). The static subproblem consists of the uncompleted (remaining) jobs J_{remain} before rescheduling and newly released jobs $\{J_t, J_{t+1}, J_{t+2}\}$. In this way, the DFJSP is divided into a series of static FJSPs. We formulate this procedure as an MDP below.

1) The MDP Definition for Higher-Level Agent: State: State s_t^* is a 2-D feature that contains a number n_i^{new} of the cached jobs and the variance value of completion time of all machines at the timestep. Those feature values are normalized and concatenated together.

Action: The action a_t^* is whether to release the cached jobs, which is denoted by a binary indicator $I(s_t^*)$ that equals to 1 if the action is to release.

Reward: The reward r_t^* is calculated as follows. We first compute the average processing time of each operation between the timestep t and $t+1$, i.e., time_{avg}. Then, the immediate reward of the timestep t is set as $-\text{time}_{\text{avg}}$ (the aim of RL agent is maximizing the reward). The total reward is the sum of all immediate rewards of each timestep. In this way, the higher-level agent is encouraged to optimize the overall DFJSP from a long-term view.

2) Double Deep Q-Network Agent: We use the DDQN [21] as the higher-level agent, in which a greedy policy is evaluated by an online Q-network while its value is estimated by a target network $\widehat{Q}$. We use the multilayer perceptron (MLP) to parameterize the online value function $Q(s, a; \phi_j)$ and the target value function $\widehat{Q}(s, a, \phi_j^-)$ where ϕ_j and ϕ_j^- are the parameters of the Q-network and the $\widehat{Q}$ network in updating step j. The agent first interactions with the dynamically scheduling environment to store the state-action-reward transition tuples $(s_t^*, a_t^*, r_t^*, s_{t+1}^*)$ into the replay buffer D. Then, it uniformly samples the transitions $(s, a, r, s') \sim U(D)$ to learn a greedy policy. And the online Q-network model at step j updates by the following loss function based on temporal difference error:

$$Y(\phi_j) = \mathbb{E}_{(s,a,r,s')\sim U(D)}[(r + \gamma \widehat{Q}(s', \underset{a'}{\text{argmax}} Q(s', a'; \phi_j); \phi_j^-) - Q(s, a; \phi_j))] \quad (2)$$

where γ is the discount factor, and ϕ_j and ϕ_j^- are the parameters of the Q-network and the $\widehat{Q}$ network at updating step j. After every C step, we copy the parameters ϕ_j to ϕ_j^-. Overall, the higher-level agent interact with the scheduling environment to receive the state and immediate reward (arrow 10 in Fig. 2), and make the action.

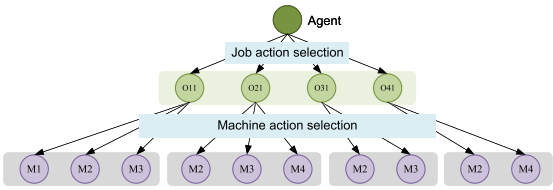


Fig. 3. Workflow of two lower-level agents.

B. Lower-Level Agents

The procedure of the two lower-level agents is illustrated in Fig. 3, in which one agent (named job agent) decides job operation selection and another (machine agent) takes charge of assigning a machine to a job operation for processing. In this way, the complex static FJSP is split into two tasks and optimized by the two lower-level agents in cooperation. The static subproblem consists of a set of released new jobs $\mathcal{J}_{\text{new}}$ and the set of remaining jobs $\mathcal{J}_{\text{remain}}$, as shown in Fig. 2. The set of operations of the static subproblem is denoted by $\{\mathcal{O}_{\text{new}} \cup \mathcal{O}_{\text{remain}}\}$. At each timestep $t \in \{1, \ldots, |\mathcal{O}_{\text{new}} \cup \mathcal{O}_{\text{remain}}|\}$, the job agent first selects a job operation, and then the machine agent assigns the job operation to a machine to process it. Overall, the lower-level agent interacts with the scheduling environment to receive the state and immediate reward (arrow 9 in Fig. 2), and make the job selection and machine assignment actions. The MDP formulation of each task is described as follows. In the following sections, $y_t^{\nabla,o}$ and $y_t^{\nabla,m}$ denotes the variable y located in job agent and machine agent, respectively.

1) The MDP Definition for the Job Operation Selection Task:

State: The state $s_t^{\nabla,o}$ at timestep t is a disjunctive graph $G_t = (\mathcal{O}_{\text{new}} \cup \mathcal{O}_{\text{remain}}, \mathcal{C} \cup \mathcal{D}_t^o, \mathcal{D})$, where $\mathcal{D}_t^o \subseteq \mathcal{D}$ is the disjunctive arcs which have been assigned directions (i.e., the actions denoted in the disjunctive graph of static FJSP) till timestep t, and $\mathcal{D}_t \subseteq \mathcal{D}$ is the set of remaining disjunction arcs. Each node of set $\mathcal{C}$ consists of two features denoted by $[LB_t(O_{ij}), I_t(O_{ij})], i \in \{1, \ldots, n\}, j \in \{1, \ldots, n_i\}$. Here, $LB_t(O_{ij})$ is the lower bound of the completion time of O_{ij} at timestep t. The value of $LB_t(O_{ij})$ equals to either the completion time for a scheduled operation O_{ij} or an estimated lower bound that equals to $LB_t(O_{i,j-1}) + \min p_{ijk}, (k \in \mathcal{M}_{ij})$. Besides, $I_t(O_{ij})$ is a binary indicator that equals 1 if O_{ij} is scheduled.

Action: The job operation action design followed by previous DRL approaches [22], [28], [32] for FJSP. In detail, the action $a_t^{\nabla,o}$ is an eligible operation (i.e., its immediate predecessor is completed). Here, the initial action space equals to $J_{\text{new}} \cup J_{\text{remain}}$ (i.e., the new arrival jobs and the remaining uncompleted jobs at each rescheduling point), and then it depends on the first idle job operation at each timestep.

Reward: The objective of the lower-level agents is to minimize the makespan of the static subproblem. Therefore, the two lower-level agents share a joint reward. The shared reward is described below.

2) The MDP Definition for the Machine Assignment Task:

State: The state $s_t^{\nabla,m}$ at timestep t consists of all machine states where each state contains a 2-D feature $[T_t(k), p_{ijk}](k \in \{1, \ldots, m\})$, where $T_t(k)$ is the completion time for machine k before timestep t. Besides, p_{ijk} is the processing time of operation O_{ij} on k if machine k is compatible for the operation, otherwise p_{ijk} is set to the average processing time $\frac{1}{|\mathcal{M}_{ij}|}\sum_k p_{ijk}(k \in \mathcal{M}_{ij})$.

Action: The action $a_t^{\nabla,m}$ is a compatible machine for the operation selected by the job operation agent at timestep t. And the action space depends on the number of compatible machines for the operation (in FJSP, each operation of a job can be processed by various machines).

Reward: To minimize the makespan of the static subproblem, we define the immediate reward as the partial solution between two states at timestep t and $t+1$, i.e., the interval time. The joint reward is defined by $r_t^{\nabla} = -(H_{t+1} - H_t)$ where $H_t = \max_{ij}\{LB_t(O_{ij})\}$.

3) Parameterizing Policies: We design two policy networks (named multipointer network (MPN)) for the two lower-level agents. Both policy networks are built with encoder–decoder architecture, but they have different network structures. We introduce the two encoder–decoder architectures as follows.

Job encoder: The disjunctive graph is used as the state in the MDP for the job agent, which provides a global view of the scheduling states containing numerical and structural information, such as the precedence constraints, job processing sequence on each machine, the compatible machine set for each operation, and the processing time of a compatible machine for each operation. For effective scheduling performance, those state information needs to be embedded in a disjunctive graph. This motivates us to encode the complex graph state by using a graph neural network (GNN). We use a graph isomorphism network (GIN) as the job encoder, which is a popular GNN variant. At timestep t, the inputs of the job encoder are the state $s_t^{\nabla,o}$ defined by disjunctive graph $G_t = (\mathcal{O}_{\text{new}} \cup \mathcal{O}_{\text{remain}}, \mathcal{C} \cup \mathcal{D}_t^o, \mathcal{D})$. The outputs are the node embeddings. Each layer of GIN is formulated as follows:

$$\boldsymbol{h}_{v,t}^{(l)} = \text{MLP}_{\theta_l}^{(l)}\left(\left(1+\epsilon^{(l)}\right)\cdot \boldsymbol{h}_{v,t}^{(l-1)} + \sum_{u\in\mathcal{N}(v)} \boldsymbol{h}_{u,t}^{(l-1)}\right) \quad (3)$$

where $\boldsymbol{h}_{v,t}^{(l)}$ is the embedding of node v from the l-layer GIN where $l \in \{1, \ldots, L\}$ and $\boldsymbol{h}_{v,t}^{(0)}$ refers to its raw features of input, $\text{MLP}_{\theta_l}^{(l)}$ is an MLP with parameter θ_l, $\epsilon^{(l)}$ is a learnable parameter, and $\mathcal{N}(v)$ is a neighborhood set of node v.

GIN is initially presented for the undirected graph. However, the mixed disjunctive graph G_t contains undirected arcs and directed arcs. It should be preprocessed before GIN extracting. One strategy is to replace the undirected arc with two directed arcs connecting the same nodes in opposite directions, i.e., a fully directed graph. But this strategy makes a graph too dense to be efficiently processed by GIN [13]. We adopt an "adding arc scheme" referring to [13] in which undirected arcs are ignored in the initial state to alleviate the computational complexity, as shown in Fig. 4. In the figure, Arc (O_{22}, O_{13}) is added in the graph state after the action tuple $a_7 = (O_{13}, M_3)$ is selected. In this way, the graph $G_t = (\mathcal{O}_{\text{new}} \cup \mathcal{O}_{\text{remain}}, \mathcal{C} \cup \mathcal{D}_t^o, \mathcal{D})$ is approximated by $\hat{G}_t = (\mathcal{O}_{\text{new}} \cup \mathcal{O}_{\text{remain}}, \mathcal{C} \cup \mathcal{D}_t^o)$. The outputs of the

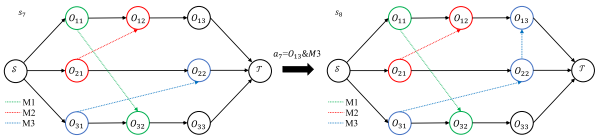


Fig. 4. Adding arc scheme for the disjunctive graph of FJSP.

encoder are the final node embedding $h_{v,t}^{(L)}$ and a graph pooling vector $\boldsymbol{h}_{\mathcal{G}}^{t} = 1/|\mathcal{O}_{\text{new}} \cup \mathcal{O}_{\text{remain}}| \sum_{v \in \{\mathcal{O}_{\text{new}} \cup \mathcal{O}_{\text{remain}}\}} h_{v,t}^{(L)}$.

Machine encoder: There is no graph structure on the state information in the MDP for the machine agent. Therefore, we use an MLP layer to encode the raw features of each machine. The outputs of the machine encoder are the embedding of each node (machine) $\boldsymbol{h}_k^t (k \in \{1, \ldots, m\})$, and the pooling vector $\boldsymbol{u}^t$.

Decoders: To select a job operation action $a_t^{\nabla,o}$ and a machine action $a_t^{\nabla,m}$ based on $s_t^{\nabla,o}$ and $s_t^{\nabla,m}$, we design two decoders to further process the outputs of the two encoders. Both the two decoders are constructed by MLP layers and have the same network structure, but they do not share the same parameters. In the decoding stage, the MLP layers output a job operation action score $c_{t,v}^{o}, v \in \{1, \ldots, |\mathcal{O}_{\text{new}} \cup \mathcal{O}_{\text{remain}}|\}$ and a machine action score $c_{t,k}^{m}, k \in \{1, \ldots, |\mathcal{M}|\}$ computed by (4) and (5).

$$c_{t,v}^{o} = \text{MLP}_{\theta_{\pi_o}} \left(\boldsymbol{h}_{v,t}^{(L)} || \boldsymbol{h}_{\mathcal{G}}^{t} || \boldsymbol{u}_t \right), \tag{4}$$

$$c_{t,k}^{m} = \text{MLP}_{\theta_{\pi_m}} \left(\boldsymbol{h}_{k}^{t} || \boldsymbol{h}_{\mathcal{G}}^{t} || \boldsymbol{u}_t \right) \tag{5}$$

where $||$ is the concatenation operation. And the selected operations are masked by setting the score value to $-\infty$. Then, a softmax function is used to compute a distribution over the scores, respectively. Finally, either a sampling strategy or a greedy decoding strategy selects the job operation action and the machine action based on the computed distribution.

Note: The previous research [22], [23] on scheduling problems is bounded by the instance scale (number of jobs, number of operations of each job, and number of machines). Because the size of the nodes used in their neural networks is determined by the scale of the input instances. However, the scales of the static subproblems are varied. In contrast, the proposed MPN architecture can be used on different scale instances without retraining, which is verified in computational results.

4) Training Algorithm for the Lower-Level Agents: We use the proximal policy optimization (PPO) to train the MPN, which is a policy gradient algorithm with an actor–critic style. The standard PPO only contains one actor, in which one policy $\pi(a|s)$ is learned to take action at each timestep. However, for the lower-level agents, two joint policies could be learned to control the job operation action and machine action. Based on the proposed MPN architecture, we extend the PPO algorithm to learn two joint policies $\pi_{\theta_o}(a_t^{\nabla,o}|s_t^{\nabla,o})$ and $\pi_{\theta_m}(a_t^{\nabla,m}|s_t^{\nabla,m})$ to solve the static subproblem with two tasks. Compared to one actor used in the PPO, the proposed multi-PPO contains two actors and they share one joint state value function $v_\phi(s_t^{\nabla,o}, s_t^{\nabla,m})$, i.e., the critic. In detail, the advantage function is given as follows:

$$\hat{A}_t = \sum_{t'=t}^{T} \gamma^{t'} r_{t'}^{l} - v_\phi(s_t^{\nabla,o}, s_t^{\nabla,m}) \tag{6}$$

TABLE III
PARAMETER SETTINGS OF THE SIMULATOR

Parameter	Value
Total number of machines (m)	$\{10, 20\}$
Number of initial jobs at the beginning (n)	$\{10\}$
Number of new arrived jobs (n')	$\{20, 100, 200, 1000\}$
Number of compatible machines of each operation	Unif $[1, m]$
Number of operations in each job	Unif $[m - 5, m + 5]$
Processing time of an operation on a compatible machine	Unif [0, 99]
The load factor of job shop, U	$\{1, 2, 4\}$
The rescheduling period (interval time), $\triangle T$	$\{\triangle t_{\text{avg}} \times n'/10\}$

where γ is a discount factor and T denotes the end of an episode). Each actor i is updated by the clip objective computed by

$$L_{\text{CLIP}}^{i}(\theta_i) = \hat{\mathbb{E}}_t[\min\{\delta_t^i(\theta_i)\,\hat{A}_t,$$

$$\text{clip}(\delta_t^i(\theta_i), 1 - \epsilon, 1 + \epsilon)\hat{A}_t\}], i \in \{o, m\} \tag{7}$$

where the probability ratio $\delta_t^i(\theta_i)$ between the updated and the prior-update job operation/machine policy is defined as $\frac{\pi_{\theta_i}(a_t^{\nabla,i}|s_t^{\nabla,i})}{\pi_{\theta_i^{old}}(a_t^{\nabla,i}|s_t^{\nabla,i})}$ and ϵ is a clipping parameter. Besides, the critic network is trained by minimizing the mean squared error (MSE) objective computed by

$$L_{MSE}(\phi) = \hat{\mathbb{E}}_t \left[\text{MSE} \left(\sum_{t'=t}^{T} \gamma^{t'} r_{t'}^{l}, v_\phi(s_t^{\nabla,o}, s_t^{\nabla,m}) \right) \right]. \tag{8}$$

IV. COMPUTATIONAL EXPERIMENT

Dataset: The designed dynamic simulator is abstracted from the actual production situation faced by CRCC High-Tech Equipment Corporation Limited (CRCCE). CRCCE is a subsidiary of China Railway Construction Corporation Limited (CRCC) and will receive production orders constantly each week. However, there are no publicly available instances for DFJSP in such dynamic manufacturing systems to train and evaluate the proposed HRL model. Thus, we design a simulator to generate the training and testing instances referring to the similar parameter settings of [14] and [42]. We assume that several jobs exist on the shop floor in the beginning. Then, new jobs arrive modeled by a Poisson process, so the interarrival time between two consecutively arrived jobs follows an exponential distribution. The average arrival time is modeled by $\triangle t_{\text{avg}} = \frac{\mu_a \mu_o}{Um}$, where μ_a is the average processing time of all operations, μ_o is the average operation number of each job, and U is a load factor of the job shop. We generate the datasets by the parameters listed in Table III. We construct the model by PyTorch and use Python 3.7 to implement the code. Besides, We used the hardware with Intel Xeon E5 V3 2600 CPU and a single Nvidia Tesla V100 GPU.

Baselines: We name a baseline method in the format of "higher-level method + lower-level method" for better readability. "$\triangle T$ rescheduling" is a periodic rescheduling method, which means the dynamic problem is divided into subproblems with a fixed interval $\triangle T$ determined by the specific scale. They are as follows.

1) $\triangle T$ **rescheduling + dispatching rules:** Dispatching rules [8] are widely used in the manufacturing industry, which contains job sequencing and machine assignment dispatching rules for FJSP. Four job sequencing and two

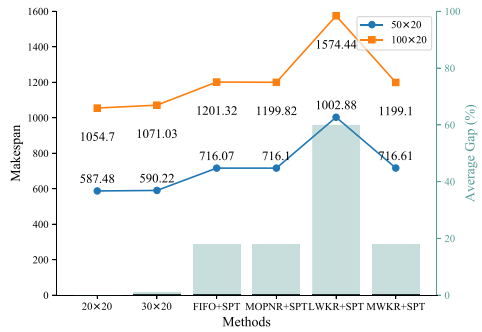


Fig. 5. Generalization verification of lower-level agents.

machine assignment dispatching rules are selected based on their good performance as shown in [8], and they are combined into eight compound dispatching rules. The four job sequencing dispatching rules include First in First Out (FIFO), Most Operation Number Remaining (MOPNR), Least Work Remaining (LWKR), and Most Work Remaining (MWKR), and the two machine assignment dispatching rules include Shortest Processing Time (SPT) and Earliest End Time (EET).

2) **Learning to choose dispatching rules (L2D):** L2D methods are widely studied in recent years. Due to the difference in the objective function and dynamic event settings, it is hard to compare the experimental results directly from the literature. Thus, we implement this baseline based on [14] and [46], and we formulate the DFJSP as an MDP as follows.

a) State: The state combines $s_t^{\nabla,o}$, i.e., the disjunctive graph $G_t = (\mathcal{O}_{\text{new}} \cup \mathcal{O}_{\text{remain}}, \mathcal{C} \cup \mathcal{D}_t^o, \mathcal{D})$, and $s_t^{\nabla,m}$ which contains the machine information. These two states give comprehensive shop floor information which can be extracted by the agent to make decisions.

b) Action: The action space consists of the composite dispatching rules with top-k performance (the performance of some dispatching rules is significantly lower than others, here k = 3).

c) Reward: The reward equals the reward of the higher-level agent which aims to optimize the overall objective of DFJSP.

d) Agent: We also use the DDQN as the job selection agent, and the details are given in Section III-A.

3) $\triangle T$ **rescheduling + ACO:** Ant colony optimization (ACO) [24] is one of the most representative metaheuristic algorithms for solving DFJSP.

4) **Optimal:** For the static FJSP, we obtain their solutions via MIP solved by the Gurobi solver with a time limit of 3600 s. Due to the NP-hardness, Gurobi can only solve optimally small-scale FJSPs within an acceptable time period.

We conduct a series of experiments on DFJSP and FJSP benchmarks with different scales to evaluate the proposed HRL. Specifically, we mainly aim to answer the following questions.

a) How does the proposed HRL algorithm compare with previous the baseline methods?

b) Can both higher-level and lower-level agents generalize to large-scale DFJSP and static FJSP instances?

c) Does the higher-level agent lean to split the DFJSP from a long-term view?

d) Are lower-level agents needed to solve the static FJSP efficiently?

e) Can lower-level agents generalize to public benchmarks? (these benchmarks typically have a different distribution from the training dataset that is generated by the uniform distribution.)

A. Main Results—How Does the Proposed HRL Algorithm Compare With Previous the Baseline Methods?

The average objective, Gap percentage, and running time of all methods on 128 instances for each scale are listed in Table IV, in which the Gap percentage is computed by the minimal objective of all methods (the smaller, the better). In the ACO column, "-" denotes that the ACO cannot get a solution within 10 h (we run the ACO for one time and 200 iterations to get the results). That denotes that this type of metaheuristic algorithm could not solve the DFJSP in real-time. *Our method completely outperforms all baseline methods on all scales*. The L2D approach does outperform all single dispatching rules, but its performance is limited by the manually-made rules. Moreover, the running time of our method is very close to dispatching rules and far smaller than the ACO. This performance guarantees that our method can respond to the dynamic event in near real-time.

B. Ablation Studies

1) Can Both Higher-Level and Lower-Level Agents Generalize to Large-Scale DFJSP and Static FJSP Instances?: Yes. To verify the scalability/generalization ability of the higher-level agent, we evaluate the model trained by 100 dynamically arrived jobs on larger-scale instances, i.e., 200 and 1000 dynamically arrived jobs. As shown in Table V, although the generalization ability to scale 1000 is slightly deteriorated compared to scale 200 (seeing the performance gap between our method and dispatching rules), our method also outperforms the baseline methods in both two scales. It verifies that our method *can generalize to larger-scale instances without retraining.*

To verify the generalization ability of the lower-level agents, we evaluate the models trained by 20×20 and 30×20 instances on the 50×20 and 100×20 instances (hence, there is no need to use a higher-level agent). For the static FJSP, each scale instance is named by $|\mathcal{J}| \times |\mathcal{M}|$.), respectively. As shown in Fig. 5, the model trained by 30×20 slightly outperforms the one trained by 20×20. Both trained models can get much better results than all dispatching rules. It strongly manifests that *our framework can generalize to unseen instances with different scales.*

Overall, owing to the scalability of the higher- and lower-level agents, once the proposed HRL architecture is well-trained, it can be deployed to solve any scales of DFJSPs or static FJSPs.

2) Does the higher-level agent lean to split the DFJSP from a long-term view?: Yes. To verify the ability of our higher-level agent to split the DFJSP into a series of static FJSPs from

TABLE IV
MAIN RESULTS OF ALL METHODS ON TEST DATASETS

Size		HRL	Meta-heuristic	L2D	Dispatching rules							
		Ours	ACO		FIFO+SPT	MWKR+SPT	MOPNR+SPT	FIFO+EET	MWKR+EET	LWKR+SPT	MOPNR+EET	LWKR+EET
20-10-1	Obj.	**1286.86**	1395.23	1454.31	1506.71	1557.27	1736.26	2375.35	3478.81	3664.94	6861.59	7620.78
	Gap	**0.00%**	8.48%	13.06%	17.07%	21.00%	34.91%	84.56%	170.30%	184.77%	433.15%	492.14%
	Time (s)	3.92	487.5	1.28	0.93	0.96	1.02	0.95	0.95	**0.89**	0.98	1.03
100-10-1	Obj.	**5211.35**	5485.54	5536.69	5756.8	6648.51	6181.89	9808.56	11847.09	12681.01	24254.14	24604.53
	Gap	**0.00%**	5.26%	6.24%	10.47%	27.59%	18.63%	88.23%	127.35%	143.35%	365.44%	372.17%
	Time (s)	16.89	3402.35	8.15	7.75	7.57	7.23	7.79	6.98	7.22	**6.85**	7.55
100-20-2	Obj.	**2802.32**	–	3112.56	3182.43	3587.523	3447.31	8760.18	11400.81	14732.66	45477.84	42167.01
	Gap	**0.00%**		11.06%	13.56%	28.01%	23.01%	212.60%	306.83%	425.73%	1522.86%	1404.71%
	Time (s)	38.23		21.68	19.28	19.67	18.93	**18.56**	19.74	19.38	19.22	20.05
20-10-2	Obj.	**918.32**	1025.38	1078.31	1155.46	1229.55	1318.11	1898.77	2740.22	3712.33	5178.43	7278.04
	Gap	**0.00%**	11.66%	17.43%	25.87%	33.94%	43.59%	106.84%	198.50%	304.39%	464.10%	692.82%
	Time (s)	4.08	456.33	1.21	1.1	0.95	**0.93**	1.03	0.95	0.94	**0.93**	0.94
100-10-2	Obj.	**2934.25**	3218.09	3384.19	3466.69	4084.56	4109.43	6792.14	9288.82	11635.56	23599.87	21482.51
	Gap	**0.00%**	9.68%	15.33%	18.16%	39.21%	40.06%	131.50%	216.59%	296.58%	704.36%	632.19%
	Time (s)	16.35	3456.48	8.38	7.79	7.16	7.15	7.97	7.15	7.19	7.26	**7.13**
100-20-4	Obj.	**1838.3**	–	2284.02	2391.86	3237.07	2897.36	7806.9	10885.06	14580.31	45762	40704.61
	Gap	**0.00%**		24.26%	30.11%	76.09%	57.61%	324.68%	492.12%	693.14%	2389.36%	2114.25%
	Time (s)	37.45		21.22	19.24	**18.62**	19.52	20.39	19.24	18.93	20.09	19.06

Note: We name the instance size in the format of "number of dynamic jobs n' - number of machines m - job shop load factor U".

TABLE V
GENERALIZATION PERFORMANCE TESTING OF OUR METHOD

Size		Ours	L2D	FIFO+SPT	MOPNR+SPT	MWKR+SPT	FIFO+EET	MWKR+EET	LWKR+SPT	LWKR+EET	MOPNR+EET
200-10-2	Obj.	**5296.25**	6115.36	6305.00	7091.07	7753.23	14799.19	18206.84	21275.45	42843.84	45994.58
	Gap	**0.00%**	15.46%	19.05%	33.89%	46.40%	179.44%	243.78%	301.73%	708.98%	768.48%
	Time (s)	**38.56**	26.42	24.31	**22.10**	21.89	25.73	22.32	22.36	22.32	22.91
200-20-4	Obj.	**3089.68**	3789.25	3910.02	4766.13	5213.91	13568.95	18732.04	27892.50	78655.56	92011.70
	Gap	**0.00%**	22.64%	26.55%	54.25%	68.75%	339.17%	506.27%	802.76%	2445.75%	2878.03%
	Time (s)	**79.83**	67.58	58.84	60.38	**57.51**	64.52	61.49	61.40	59.03	63.38
1000-20-4	Obj.	**13018.52**	13976.39	14219.61	16979.21	18935.89	61606.16	81650.11	377840.66	476172.61	
	Gap	**0.00%**	7.35%	9.22%	30.42%	45.45%	373.21%	527.18%	926.55%	2802.33%	3557.65%
	Time (s)	**493.25**	433.57	389.34	396.52	365.81	414.04	361.84	**341.76**	342.21	385.53

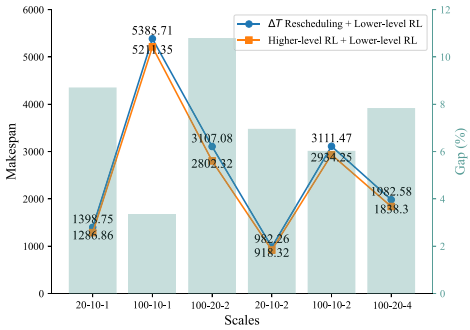


Fig. 6. Effectiveness of the higher-level agent.

TABLE VI
EFFECTIVENESS OF LOWER-LEVEL AGENTS ON STATIC FJSP

Size		MIP	Dispatching Rules		Ours
			Best	Worst	
20 × 10	Obj.	391.41	566.32	1815.82	**454.85**
	Gap	0.00%	44.69%	363.92%	**16.21%**
	Time (s)	3600	**0.21**	0.23	0.34
20 × 20	Obj.	322.54	430.79	2762.01	**361.75**
	Gap	0.00%	33.56%	756.33%	**12.16%**
	Time (s)	3600	0.46	**0.44**	1.08
30 × 20	Obj.	-	525.08	3462.27	**433.42**
	Gap		21.15%	698.83%	**0.00%**
	Time (s)		0.86	**0.82**	1.97

Note: "Best" and "Worst" denote the best and worst results of eight dispatching rules for each scale instance.

a long-term perspective, we compare the "$\triangle T$ rescheduling" with our higher-level agent via evaluating different scale DFJSP instances. We conduct the comparison experiment with three different scales and job-shop load factors. As shown in Fig. 6, the higher-level agent can get better results in all problem scales. *Moreover, the gap values between the two methods denoted by the green bars in the figure show that the solution quality of our higher-level agent is increased steadily by* $3.34\% - 10.80\%$ *over "$\triangle T$ rescheduling"*. These results demonstrate that the higher-level agent can divide the DFJSP from a long-term perspective for global optimization on different scales. Namely, the higher-level agent is needed to achieve global optimization for the DFJSP.

3) Are lower-level agents needed to solve the static FJSP efficiently?: Yes. To answer this, we evaluate the performance of the lower-level agents alone on the static FJSP. In Table VI, we reported the average objective makespan, running time, and Gap from the Gurobi solver (the smaller, the better) for 128 randomly generated instances solved by the proposed framework, dispatching rules, and Gurobi, respectively. As we can see, the running time used by our proposed algorithm is much less than that of Gurobi, which reaches the presetting time limit (3600 s) in solving the instances of scales 20×10 and 20×20. Gurobi cannot get a solution in solving the instance of scale 30×20 within 3600 (denoted by "-" in the "MIP" column). Although the running time of dispatching rules is slightly better than ours, it is immaterial because the solution quality of our method is significantly better.

The superiority of the higher-level agent over traditional "$\triangle T$ rescheduling" and the lower-level agents over all dispatching rules demonstrate that the two-level architecture is needed to

TABLE VII
RESULTS ON PUBLIC BENCHMARKS

Instance	Dispatching rules	Meta-heuristics		Ours	UB
	Best	ACO	2SGA [25]		
10 × 5	613.4 (20.98%)	**537.4 (6.04%)**	**510.4 (0.67%)**	547.8 (7.85%)	507.0
15 × 5	842.8 (6.14%)	**821.2 (3.42%)**	**795.0 (0.13%)**	820.0 (3.27%)	794.0
20 × 5	1090.0 (4.73%)	**1068.2 (2.59%)**	**1041.2 (0.04%)**	1060.2 (1.86%)	1040.8
10 × 10	716.4 (5.38%)	**694.8 (2.28%)**	**680.2 (0.06%)**	681.8 (0.29%)	679.8
15 × 10	894.4 (15.08%)	**852.0 (9.65%)**	**796.6 (2.50%)**	860.8 (10.75%)	777.2
20 × 10	1119.8 (6.22%)	**1126.4 (6.87%)**	**1067.2 (1.23%)**	1096.8 (4.04%)	1054.2
30 × 10	1611.2 (3.80%)	**1597.2 (2.89%)**	**1557.2 (0.36%)**	1584.2 (2.06%)	1552.2
15 × 15	1002.4 (5.43%)	**993.0 (4.44%)**	**953.0 (0.23%)**	981.6 (3.23%)	950.8
Ave. Gap	8.54%	**4.77%**	**0.80%**	4.20%	0.00%

Note: "UB" column is the best result from the literature. We run the ACO for one time and 200 iterations to get the results.

achieve better near real-time optimization compared with all baseline methods.

4) Can Lower-Level Agents Generalize to Public Benchmarks?: Yes. To verify the generalization performance of the proposed framework from random instance training to real-world instance testing, we perform experiments on the 40 Hurink-Vdata instances, which can be sorted into eight groups via their scales, each with five instances. We trained our model in three groups (10×5, 10×10, and 15×15), and the remaining five groups are used for generalization testing. We reported the average value and objective gap for each group, as shown in Table VII. Besides, our framework's performance is compared with the state-of-the-art metaheuristic algorithms, a two-stage genetic algorithm (2SGA) [25].

Reference [25] has conducted *a total of 1600 experiment runs for each instance, and each experimental run consumes one to 30 min* depending on the problem scale and the genetic algorithm (GA) parameter selected, respectively. However, the running time of our method is *0.07 to 0.41 s for different problem scales.* Although the solutions of 2SGA surpass ours in gap value, the running time of our method is much less than theirs. Also, our method is slightly better than the ACO in terms of the solution quality. Our method is a good tradeoff between quality and running time in a dynamic manufacturing environment.

5) Further Discussion: Scheduling of the contemporary manufacturing factory greatly requires the capacity of the scheduling system to cope with unpredictable dynamic events in real time. Compared with previous traditional rescheduling and learning-based methods, our HRL framework can achieve better performance in terms of solution quality and running time. More contributions of the proposed framework except for the improvement of the model performance could be useful insights for practitioners and managers:

a) *Generalization ability:* Dynamic scheduling suffers from the problem instances with everchanging scale, which cannot be handled by the static architecture of existing machine learning algorithms [48]. Namely, the majority of current research makes the assumption of the scheduling instances with a fixed size. However, it is extremely not suitable for the real scheduling scenario in which the scale of the scheduling problem is usually different from each other per day. Owing to the scalability of all level agents in the proposed HRL framework, once the model is well-trained, it can be deployed to solve any scales of DFJSPs or static FJSPs. The factory will be beneficial (saving cost and reducing the waiting time) without retraining when new scheduling problems with different scales arrive.

b) *Robust performance:* Most recent research has combined the heuristic dispatching rules and DRL, which usually adopts a DRL agent to select the manually-designed dispatching rules to confront different scheduling scenarios and optimization objectives. This method is suitable for some dynamic scheduling problems taking advantage of its low complexity in running time. However, the design of effective dispatching rules is a tedious task because of: 1) requiring a myriad of specialized knowledge [31]; 2) delicate and limited performance for different FJSPs [32]. Moreover, some studies adopt the DRL technology to improve the performance of the metaheuristic algorithms for FJSP like GA [33]. Those methods acquire higher solution quality but are time consuming and have high complexity in running time as metaheuristic algorithms that are not applicable for some real-time scheduling scenarios with dynamics and uncertainty [14]. By contrast, our approach solves the DFJSPs in an end-to-end manner by extracting the information features of the instance input without designing and adopting traditional optimization research methods. The managers can also benefit from its low running-time complexity which is similar to the dispatching rules, and its strong generalization ability from randomly generated instances training to the public benchmarks testing.

c) *Efficient applications:* Our architecture also has shown the potential to be applied to handle other types of dynamic events like machine breakdown, in which it is just needed to mask the broken machines via the machine assignment policy distribution. Moreover, it can be extended to other types of dynamic scheduling problems (e.g., dynamic flow-shop and open-shop problems) because those problems can also be represented by disjunctive graphs.

V. CONCLUSION

This article presented a novel end-to-end hierarchical reinforcement learning framework to solve a large-scale DFJSP in near real-time effectively. In the framework, a higher-level DDQN agent aimed to optimize the overall objective in a long-term perspective by dynamically dividing the DFJSP into a series of static FJSPs. Then, two lower-level agents based on GNN learned two policies to coordinate the two static FJSP tasks. Benefiting from the design of networks, both the higher-level and lower-level agents could generalize to larger-scale instances without retraining. By leveraging the cooperation of the higher-level and lower-level agents, our method could find globally better solutions. Numerical experiments on several large-scale instances will verify the superiority of our framework to the existing dynamic scheduling methods, such as well-known dispatching rules and existing heuristics.

In the future, we plan to expand our framework to cope with multiple objectives conflicting with each other such as tardiness, production cost, and energy consumption [26]. Moreover, another valuable direction is to extend our method to other types of dynamic scheduling problems and combinatorial optimization problems, such as those resembling dynamic vehicle routing problems. Another potential direction is applying rough set theory [49] into the DRL architecture to help the agent make decisions, which is a powerful technique to capture characteristics in knowledge discovery under the dynamic system, system identifies, and decision-making. For example, the authors in [50] recently proposed a kind of singular rough sets scheduling algorithm that makes use of the attribute value form and upper and lower approximately characteristic, and applied it to solve the job shop problem in a dynamic system. Thus, it is a promising research topic for DFJSP in the future.

REFERENCES

[1] K. Z. Gao, P. N. Suganthan, T. J. Chua, C. S. Chong, T. X. Cai, and Q. K. Pan, "A two-stage artificial bee colony algorithm scheduling flexible job-shop scheduling problem with new job insertion," *Expert Syst. Appl.*, vol. 42, no. 21, pp. 7652–7663, 2015.

[2] J. Q. Li, Q. K. Pan, and Y. C. Liang, "An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems," *Comput. Ind. Eng.*, vol. 59, no. 4, pp. 647–662, 2010.

[3] X. Li and L. Gao, "A hybrid genetic algorithm with variable neighborhood search for dynamic IPPS," in *Effective Methods for Integrated Process Planning and Scheduling*, vol. 2. Berlin, Germany: Springer, 2020, pp. 429–453.

[4] Q. Wang and C. Tang, "Deep reinforcement learning for transportation network combinatorial optimization: A survey," *Knowl.-Based Syst.*, vol. 233, 2021, Art. no. 107526.

[5] S. V. Mehta, "Predictable scheduling of a single machine subject to breakdowns," *Int. J. Comput. Integr. Manuf.*, vol. 12, no. 1, pp. 15–38, 1999.

[6] C. Rajendran and O. Holthaus, "A comparative study of dispatching rules in dynamic flowshops and jobshops," *Eur. J. Oper. Res.*, vol. 116, no. 1, pp. 156–170, 1999.

[7] V. Subramaniam, G. Lee, T. Ramesh, G. Hong, and Y. Wong, "Machine selection rules in a dynamic job shop," *Int. J. Adv. Manuf. Technol.*, vol. 16, no. 12, pp. 902–908, 2000.

[8] H.-H. Doh, J.-M. Yu, J.-S. Kim, D.-H. Lee, and S.-H. Nam, "A priority scheduling approach for flexible job shops with multiple process plans," *Int. J. Prod. Res.*, vol. 51, no. 12, pp. 3748–3764, 2013.

[9] R. T. Nelson, C. A. Holloway, and R. Mei-Lun Wong, "Centralized scheduling and priority implementation heuristics for a dynamic job shop model," *AIIE Trans.*, vol. 9, no. 1, pp. 95–102, 1977.

[10] N. Kundakcı and O. Kulak, "Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem," *Comput. Ind. Eng.*, vol. 96, pp. 31–51, 2016.

[11] R. Buddala and S. S. Mahapatra, "Two-stage teaching-learning-based optimization method for flexible job-shop scheduling under machine breakdown," *Int. J. Adv. Manuf. Technol.*, vol. 100, no. 5, pp. 1419–1432, 2019.

[12] N. Al-Hinai and T. Y. ElMekkawy, "Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm," *Int. J. Prod. Econ.*, vol. 132, no. 2, pp. 279–291, 2011.

[13] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and X. Chi, "Learning to dispatch for job shop scheduling via deep reinforcement learning," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, 2020, vol. 33, pp. 1621–1632.

[14] S. Luo, "Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning," *Appl. Soft Comput.*, vol. 91, 2020, Art. no. 106208.

[15] M. Zhao, X. Li, L. Gao, L. Wang, and M. Xiao, "An improved Q-learning based rescheduling method for flexible job-shops with machine failures," in *Proc. IEEE 15th Int. Conf. Automat. Sci. Eng.*, 2019, pp. 331–337.

[16] Y. Ma et al., "A hierarchical reinforcement learning based optimization framework for large-scale dynamic pickup and delivery problems," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, 2021, vol. 34, pp. 23609–23620.

[17] K. Lei, P. Guo, Y. Wang, X. Wu, and W. Zhao, "Solve routing problems with a residual edge-graph attention neural network," *Neurocomputing*, vol. 508, pp. 79–98, 2022.

[18] Z. Pan, L. Wang, J. Wang, and J. Lu, "Deep reinforcement learning based optimization algorithm for permutation flow-shop scheduling," *IEEE Trans. Emerg. Topics Comput. Intell.*, early access, Nov. 1, 2021, doi: 10.1109/TETCI.2021.3098354.

[19] C. Lin, D. Deng, Y. Chih, and H. Chiu, "Smart manufacturing scheduling with edge computing using multiclass deep Q network," *IEEE Trans. Ind. Informat.*, vol. 15, no. 7, pp. 4276–4284, Jul. 2019.

[20] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: Freeman, 1979.

[21] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[22] I.-B. Park, J. Huh, J. Kim, and J. Park, "A reinforcement learning approach to robust scheduling of semiconductor manufacturing facilities," *IEEE Trans. Automat. Sci. Eng.*, vol. 17, no. 3, pp. 1420–1431, Jul. 2020.

[23] X. Chen and Y. Tian, "Learning to perform local rewriting for combinatorial optimization," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, 2019, Art. no. 564.

[24] S. Zhang and T. N. Wong, "Flexible job-shop scheduling/rescheduling in dynamic environment: A hybrid MAS/ACO approach," *Int. J. Prod. Res.*, vol. 55, no. 11, pp. 3173–3196, 2017.

[25] F. M. Defersha and D. Rooyani, "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," *Comput. Ind. Eng.*, vol. 147, 2020, Art. no. 106605.

[26] L. He, R. Chiong, W. Li, S. Dhakal, Y. Cao, and Y. Zhang, "Multiobjective optimization of energy-efficient job-shop scheduling with dynamic reference point-based fuzzy relative entropy," *IEEE Trans. Ind. Informat.*, vol. 18, no. 1 pp. 600–610, Jan. 2022.

[27] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour D'Horizon," *Eur. J. Oper. Res.*, vol. 290, no. 2, pp. 405–421, 2021.

[28] 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.

[29] C. W. Parsonson, A. Laterre, and T. D. Barrett, "Reinforcement learning for branch-and-bound optimisation using retrospective trajectories," 2022, *arXiv:2205.14345*.

[30] Z. Wang et al., "Learning cut selection for mixed-integer linear programming via hierarchical sequence model," in *Proc. 11th Int. Conf. Learn. Representations*, 2023.

[31] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," 2019, *arXiv:1906.01227*.

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

[33] R. Chen, B. Yang, S. Li, and S. Wang, "A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem," *Comput. Ind. Eng.*, vol. 149, 2020, Art. no. 106778.

[34] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Evolving scheduling heuristics via genetic programming with feature selection in dynamic flexible job-shop scheduling," *IEEE Trans. Cybern.*, vol. 51, no. 4, pp. 1797–1811, Apr. 2021.

[35] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Genetic programming with adaptive search based on the frequency of features for dynamic flexible job shop scheduling," in *Evolutionary Computation in Combinatorial Optimization*, L. Paquete and C. Zarges, Eds. Cham, Switzerland:Springer Int. Publ., 2020, pp. 214–230.

[36] F. Zhang, Y. Mei, S. Nguyen, M. Zhang, and K. C. Tan, "Surrogate-assisted evolutionary multitask genetic programming for dynamic flexible job shop scheduling," *IEEE Trans. Evol. Comput.*, vol. 25, no. 4, pp. 651–665, Aug. 2021.

[37] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Correlation coefficient-based recombinative guidance for genetic programming hyperheuristics in dynamic flexible job shop scheduling," *IEEE Trans. Evol. Comput.*, vol. 25, no. 3, pp. 552–566, Jun. 2021.

[38] M. Xu, F. Zhang, Y. Mei, and M. Zhang, "Genetic programming with archive for dynamic flexible job shop scheduling," in *Proc. IEEE Congr. Evol. Comput.*, 2021, pp. 2117–2124.
[39] F. Zhang, Y. Mei, S. Nguyen, K. C. Tan, and M. Zhang, "Multitask genetic programming-based generative hyperheuristics: A case study in dynamic scheduling," *IEEE Trans. Cybern.*, vol. 52, no. 10, pp. 10515–10528, Oct. 2022.
[40] A. Esteso, D. Peidro, J. Mula, and M. Díaz-Madroñero, "Reinforcement learning applied to production planning and control," *Int. J. Prod. Res.*, pp. 1–18, 2022 doi: 10.1080/00207543.2022.2104180.
[41] F. N. Shimim and B. M. Whitaker, "A reinforcement learning approach to the dynamic job scheduling problem," in *Proc. IEEE Green Energy Smart Syst. Syst.*, 2022, pp. 1–6, doi: 10.1109/IGESSC55810.2022.9955328.
[42] 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. Automat. Sci. Eng.*, vol. 19, no. 4, pp. 3020–3038, Oct. 2022.
[43] H. Wang, B. R. Sarker, J. Li, and J. Li, "Adaptive scheduling for assembly job shop with uncertain assembly times based on dual Q-learning," *Int. J. Prod. Res.*, vol. 59, no. 19, pp. 5867–5883, 2021.
[44] W. Bouazza, Y. Sallez, and B. Beldjilali, "A distributed approach solving partially flexible job-shop scheduling problem with a Q-learning effect," *IFAC-PapersOnline*, vol. 50, no. 1, pp. 15890–15895, 2017.
[45] S. Luo, L. Zhang, and Y. Fan, "Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning," *Comput. Ind. Eng.*, vol. 159, 2021, Art. no. 107489.
[46] 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, 2022, Art. no. 760.
[47] H. Wang et al., "Multi-objective reinforcement learning framework for dynamic flexible job shop scheduling problem with uncertain events," *Appl. Soft Comput.*, vol. 131, 2022, Art. no. 109717.
[48] 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.
[49] Z. Pawlak, "Rough sets," *Int. J. Comput. Inf. Sci.*, vol. 11, pp. 341–356, 1982.
[50] Y. Hu, Y. Fu, L. Jia, and Q. Li, "An algorithm of job shop rolling scheduling based on singular rough sets," in *Proc. 8th ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw., Parallel/Distrib. Comput.*, 2007, vol. 2, pp. 221–225.

Kun Lei received the master's degree in mechanical engineering from Southwest Jiaotong University, Chengdu, China, under the supervision of Dr. Peng Guo and Dr. Yi Wang.

He is currently working as an Algorithm Engineer with Shanghai Qizhi Institute, Shanghai, China. Prior to this, he was a Research Assistant with Westlake University, Hangzhou, China. He has authored or coauthored several articles in top conferences and journals, including ICLR, ESWA, and *Neurocomputing*. His research interests include deep reinforcement learning techniques and their applications.

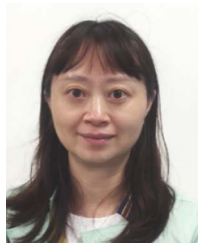
Peng Guo received the Ph.D. degree in mechanical engineering from Southwest Jiaotong University, Chengdu, China, in 2014.

He is currently an Associate Professor with the School of Mechanical Engineering, Southwest Jiaotong University. From 2016 to 2017, he was a Visiting Scholar with the Faculty of Economics and Business Administration, University of Jena, Jena, Germany. From 2019 to 2020, he was a Researcher with Université Gustave Eiffel, Lille, France. His research interests include manufacturing operations management and logistics system modeling.

Dr. Guo was the recipient of the Omega Best Paper Award being the first author in 2019.

Yi Wang received the first Ph.D. degree in mechanical engineering from Southwest Jiaotong University, Chengdu, China, in 1997, and the second Ph.D. degree in mathematics from West Virginia University, WV, USA, in 2003.

He is currently a Professor in mathematics with Auburn University Montgomery (AUM), Montgomery, USA. His research interests include machine learning, mathematical optimization, and reliability analysis.



Jian Zhang received the Ph.D. degree in mechanical design and theory from Southwest Jiaotong University, Chengdu, China, in 2015.

She is currently a Professor with the School of Mechanical Engineering, Southwest Jiaotong University. Her research interests include intelligent manufacturing.

Xiangyin Meng received the Ph.D. degree in mechanical design and theory from Southwest Jiaotong University, Chengdu, China, in 2012.

He is currently an Associate Professor with the School of Mechanical Engineering, Southwest Jiaotong University. His research interests include mechatronics and edge intelligence.

Linmao Qian received the Ph.D. degree in mechanical design and theory from Tsinghua University, Beijing, China, in 1999.

He is currently a Professor with the School of Mechanical Engineering, Southwest Jiaotong University, Chengdu, China. His research interests include intelligent manufacturing and nanotribology.