Knowledge Guided DRL for Intelligent Reconfiguration and Scheduling in Customized and Personalized Manufacturing Workshop

Shulin Lan, Member, IEEE, Yinfei Jiang*, Chen Yang*, Member, IEEE, Lihui Wang,
George Q. Huang, Fellow, IEEE, Weiming Shen, Fellow, IEEE, and Liehuang Zhu, Senior Member, IEEE

Abstract*—To meet personalized user demands, customized and personalized production has become an effective manufacturing paradigm. However, wired network connections inhibit flexible production line reconfiguration and current DRL methods cannot converge and obtain eligible scheduling results for customized and personalized production (CPP) due to the high-dimensional solution space and the negligence of significant machine reconfiguration time. To address this challenge, we first propose a wireless manufacturing system framework to support ultra-flexible reconfiguration and resource scheduling. Next, we build a reconfiguration oriented scheduling model to reflect the significant impact of reconfiguration time. Then, we design a Knowledge Guided Deep Reinforcement Learning (KGDRL) algorithm to effectively solve the CPP scheduling problem facing the dimension explosion problem. The knowledge guidance incorporates reconfiguration time and machine workload to significantly reduce the feasible action space, enabling the rapid convergence of KGDRL. The experiment results show that our approach provides a robust and scalable solution and obtains shorter total makespan of whole production during scheduling.

***Keywords*—customized and personalized production, reconfigurable manufacturing system, deep reinforcement learning, knowledge guidance, resource scheduling**

I. Introduction

Customized and personalized products (CPPs), which best meet individual needs, have gradually become the main business model for many companies and a key factor in their competitiveness [1]. Traditional discrete manufacturing systems typically rely on rigid production lines, where each machine is assigned to a fixed processing stage and production jobs are executed sequentially. However, such systems often struggle to meet the growing demand for flexibility in small-batch, multi-variety production. To address these challenges, lean manufacturing workshops increasingly adopt reconfiguration scheduling as an effective solution. This approach optimizes the use of limited cost and space resources by enabling a single production system to flexibly handle a wide variety of jobs under changing production conditions. To further illustrate how reconfiguration improves production efficiency in such settings, we take sand core manufacturing in the foundry industry as a representative example, as shown in Fig. 1. A typical production process includes three stages: material handling, dip coating, and polishing. Since the handling operation is relatively short while the dip coating process involves longer waiting times, machine idleness and task queuing often occur at the coating stage. Without reconfiguration, some machines (e.g., M2 and M5) would remain idle while multiple jobs are waiting to be processed. By enabling reconfiguration, idle machines from the handling and polishing stages can be reassigned to the coating stage, effectively reducing machine idleness and improving processing efficiency.

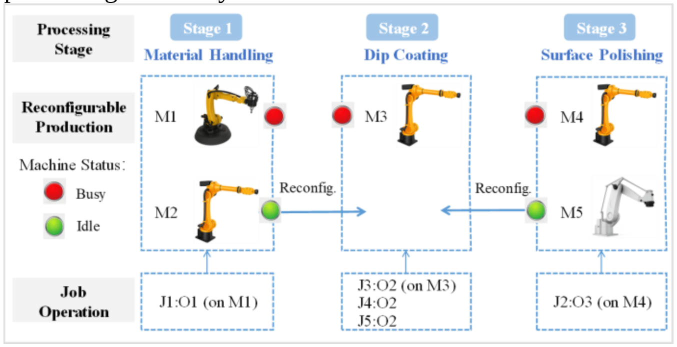

Fig. 1. Reconfiguration decision in sand core CPP production

However, reconfigurable production introduces new challenges. Machines must be able to transition smoothly between different types of operations to meet the diverse requirements of various job types. Before performing a specific operation, each machine must undergo a reconfiguration process, which typically involves tool

This work was supported in part by the National Key R&D Program of China under Grant 2021YFB1715700; in part by the National Natural Science Foundation of China under Grant 62103046, 72201266 and 72192843. (Corresponding author: Chen Yang, Yinfei Jiang)

Shulin Lan, and Yinfei Jiang is with the School of Economics and Management, University of the Chinese Academy of Sciences, Beijing, China. (e-mail: lanshulin@ucas.ac.cn; jiangyinfei22@mails.ucas.ac.cn).
Chen Yang, and Liehuang Zhu are with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing, China. (e-mail: yangchen666@bit.edu.cn; liehuangz@bit.edu.cn).
Lihui Wang is with the Department of Production Engineering, KTH Royal Institute of Technology, Stockholm 10044, Sweden (e-mail: lihuiw@kth.se).
George Q. Huang is with the Industrial and Systems Engineering at The Hong Kong Polytechnic University, Hong Kong (e-mail: gq.huang@polyu.edu.hk).
Weiming Shen is with the State Key Lab of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: wshen@ieee.org).

switching, material preparation, and parameter adjustments. These steps incur a non-negligible reconfiguration time, which is often overlooked in existing models. Moreover, reconfiguration time is highly dynamic and stage-dependent. For example, reconfiguration from handling to dip coating is relatively fast due to shared tools and materials, while transitioning from coating to polishing takes significantly longer. Even within the same coating stage, variations in product color resulting from multi-variety production can lead to considerable differences in reconfiguration time. This variability in reconfiguration time introduces heterogeneity and uncertainty into the scheduling environment, posing considerable challenges for the deployment of efficient learning-based scheduling algorithms. Therefore, an effective production scheduling strategy is essential to account for the impact of reconfiguration and ensure the overall efficiency of the CPP production process.

Cloud Manufacturing systems (CMfg) have long been considered a practical framework to traditional production models due to their features such as low-latency connectivity, full-shop collaboration, and real-time data sharing. However, current CMfg cannot efficiently handle a group of CPP jobs submitted by individual customers due to the following reasons: **1) Wired network connections in existing manufacturing systems limit machine mobility and hinder flexible reconfiguration of production lines.** State-of-the-art wired manufacturing networks primarily focus on flexibility design to enhance manufacturing efficiency [2-4]. Despite these advancements, such systems remain unable to support CPP scheduling due to limitations in machine mobility. Without sufficient space and function flexibility, idle machines could not be dynamically reconfigured to busy stages to accelerate the processing. **2) Existing scheduling models neglect the variety of reconfiguration time during the scheduling process.** These models are established to address challenges such as reducing energy costs [5] and managing workforce constraints [6]. Reconfiguration time can vary significantly across different machines, potentially leading to long processing time and, as a result, lower production efficiency in the workshop. **3) Current scheduling methods fail to obtain convergent solutions under the dynamic nature and dimensional complexity of the CPP problem.** While machine learning algorithms and heuristic approaches can produce near-optimal results for most traditional flexible job-shop scheduling problems, they often struggle in more complex scenarios. These methods primarily address heterogeneous job types, such as batch processing [7], job splitting [8], and random job arrivals [9]. However, they largely overlook the reconfiguration status of machines, which is critical for effective CPP scheduling. This oversight, combined with the explosion of computational dimensions introduced by machine reconfigurability, makes it challenging to develop robust and efficient scheduling strategies.

As a result, this study aims to propose an intelligent reconfigurable resource scheduling method for ultra-flexible wireless CPP workshops in discrete manufacturing with multi-variety, small-batch production. The main contributions of this paper are summarized as follows:

1) An edge-computing based ultra-flexible CMfg framework is proposed to support smart adaptive reconfiguration and resource scheduling, according to CPP job requirements. Idle machines can adjust their scheduling status and move to busy production lines and process the corresponding CPP jobs with the support of wireless connections, enabling them to meet the diverse demands of CPP jobs.

2) A reconfiguration-oriented scheduling model is established to quantify the impact of reconfiguration time on the scheduling process. The model incorporates CPP jobs, reconfigurable production machines, different processing speed, varying reconfiguration time, and optimization objectives to reflect how reconfiguration influences production. Once a machine is reconfigured, these factors directly affect subsequent processing operations and their corresponding processing time.

3) A Knowledge-Guided Deep Reinforcement Learning (KGDRL) algorithm is developed to address the CPP scheduling problem. By integrating prior knowledge of reconfiguration time and machine workload, the proposed approach effectively narrows the feasible action space, enhances decision-making efficiency, and ensures the selection of optimal actions in high-dimensional environments at each time step. The knowledge-guided strategy allows the agent to learn a robust and generalizable scheduling policy across diverse CPP reconfiguration scheduling scenarios.

The remaining paper is organized as follows. Section II reviews related articles and summarizes essential research gaps. Section III describes CMfg framework and the model of CPP. Section IV designs KGDRL details to solve CPP. Section V shows the experiments of the proposed method. The last section concludes this research and points out future work.

II. Related Works

A. Flexible and Reconfigurable Manufacturing Systems

The flexible manufacturing system showcased remarkable capabilities throughout the entire production lifecycle, including processing, scheduling and logistics [10-12]. Wang et al. [13] proposed a smart flexible manufacturing system based on digital-twin and developed the corresponding applications. Li et al. [14] analyzed the potential of flexible manufacturing systems and Industry 4.0 for sustainable and smart manufacturing. However, flexible manufacturing systems face significant challenges in dynamic operation scheduling during CPP scheduling due to the limited machine mobility.

The reconfigurable manufacturing system (RMS) can rapidly change its hardware, and software components to adjust its production functionality [15]. Unlike traditional flexible job shop scheduling problems, reconfigurable job shop scheduling must explicitly account for machine reconfiguration time. Zhang et al. [16] introduced a distributed manufacturing scheduling system for mass customized production. Hu et al. [17] proposed a collaborative system to solve the large-scale dynamic scheduling problem. Production line reconfiguration is increasingly recognized as an effective approach to cope with production dynamics and operational uncertainties in modern manufacturing systems [18-19]. A reconfigurable job shop consists of multiple reconfigurable systems or other flexible manufacturing units, forming a production area that serves as a basic unit for enabling personalized and customized manufacturing. It emphasizes coordination across the entire production process and the optimization of overall system performance.

These research work has provided valuable insights into the scheduling problems of reconfigurable manufacturing systems, but most studies do not consider machine reconfiguration time as a variable, and therefore fail to account for the impact of reconfiguration during CPP scheduling.

B. Intelligent Production Scheduling

Intelligent scheduling methods have been widely adopted in manufacturing systems, playing a crucial role in improving production efficiency, optimizing resource allocation, and addressing uncertainties in production processes. Chen et al. [20] proposed an advanced Grey Wolf algorithm for solving hybrid scheduling problems. Mahmoodi et al. [21] proposed a data-driven resource allocation method to deal with the smart manufacturing resources. Khou et al. [22] used a particle swarm optimization algorithm to improving both economic efficiency and environmental sustainability in microgrids that incorporate electric vehicles. Although existing studies have achieved high-quality FJSP solutions using heuristic algorithms, these methods are highly sensitive to initial strategies. As a result, their efficiency and solution quality cannot be reliably guaranteed in dynamic reconfiguration scenarios.

As research continues to improve the accuracy of modeling the dynamic characteristics of reconfigurable job shops, the solution space for their scheduling optimization problems has grown exponentially [23]. In this context, deep reinforcement learning (DRL) has emerged as an effective tool for addressing these challenges, leveraging its strong adaptability, long-term optimization capabilities, and ability to handle complex environments. Some scholars have used deep Q-networks to help agents learn the features of high-dimensional spaces [24,25], taking advantage of their strong generalization ability and adaptability to solve high-dimensional scheduling problems. Lu [26] applied DRL algorithm to studying the scheduling of cloud manufacturing resources. Zhang [27] proposed DRL algorithms to tackle the issue of multi-AGV environments during the scheduling process.

Existing DRL algorithms mainly focus on optimizing traditional scheduling problems, ignoring reconfigurable characteristic of CPP scheduling. As a result, the training results often fail to converge and it becomes difficult to effectively select the appropriate processing actions in high-dimension spaces.

III. Problem Formulation

A. System Framework

This work proposed an edge-cloud based manufacturing system framework for CPP production. As shown in Fig. 2. The framework is composed of the following components.

1) CPP CMfg Platform

The CMfg platform can accept the CPP jobs submitted by distributed individuals or groups. The jobs contain important CPP parameters. It also provides other related services, such as production planning, job assignment, data acquisition, visibility and traceability services. CPP jobs are decomposed into a lot of operations that can be processed at different stages of the workshop. The platform also provides visibility and traceability services for customers to track manufacturing progress. It can integrate and connect a group of smart factories but here we only consider the case of one smart factory (the flow shop).

2) Manufacturing Node

The edge manufacturing node (EMN) is an edge-computing based manufacturing service node that can accept personalized production jobs distributed from the CMfg platform, provide data processing and storage services for the shopfloor things and manage the production processes in the workshop. EMN collects the real-time status of the manufacturing things and makes smart decisions using these data and intelligent scheduling models and algorithms. Based on the status of things and jobs, EMN can reconfigure the resources and schedule the jobs to the resources optimally.

3) Wireless Smart Factory

All the elements in the factory are connected using wireless communication technologies such as 5G/6G. Without long electrical cables between machines, the wireless connection between machines allows free re-arrangement of the production lines [28] and support the continuously improvement of resource scheduling [29]. The stages are settled as the largest set of the processes required by different CPP jobs. A CPP job may only flow across some of the stages, thus easily leading to the unbalanced workload in different stages. Therefore, making smart decisions on the reconfiguration and scheduling of production resources for CPP jobs is crucial.

B. Reconfigurable Jobshop Optimization Model

The Reconfigurable CPP Scheduling Problem (RCSP) involves scheduling a set of CPP jobs on reconfigurable machines. Suppose there has n CPP jobs $J=\{j \mid j=1,2,...n\}$ and r machines $M=\{m \mid m=1,2,...r\}$. For each job j, it has k operations $O=\{i \mid i=1,2,...k\}$. The objective function is to minimize the makespan of the entire scheduling process $\max_{j\in 1...n}\{T_j\}$. Operation scheduling matrix σ_j is the optimization variable, denoting the operation results of job j. σ_j is $n\times r$ matrix where the value of matrix element β_{ir}^{j} are :

$$\beta_{im}^{j}=\begin{cases}1 & \text{if operation } i \text{ of job } j \text{ is to be excuted in machine } m\\ 0 & \text{otherwise}\end{cases} \quad (1)$$

We use X to represent the processing sequence of operations, and its specific definition is as follows:

$$X_{ip}^{j}=\begin{cases}1 & \text{if operation } p \text{ is next step of job } j \text{ operation } i\\ 0 & \text{otherwise}\end{cases} \quad (2)$$

The optimization model, which aims to minimize the makespan, is presented as follows. Detailed definitions of the variables are provided in Table I.

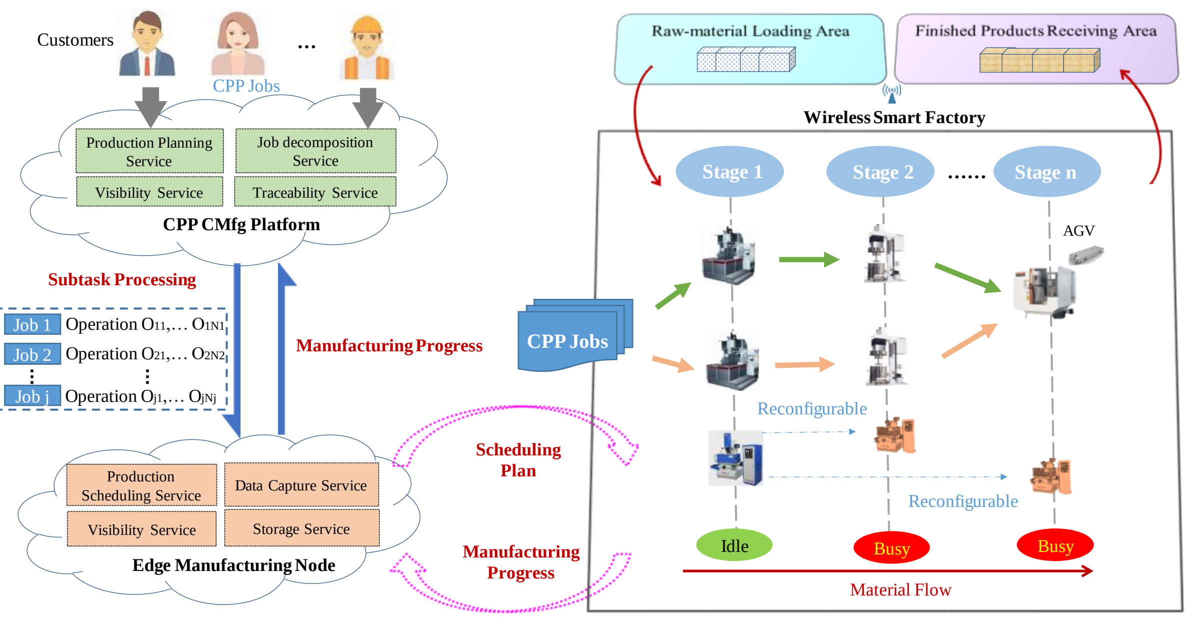

Fig. 2. An edge-cloud and wireless connection based CMfg framework for CPP production

$$\min \max_{j\in 1\ldots n}\{T_j\} \tag{3}$$

$$s.t.\ \ OPTY_{ij} \in \bigcup_{m=1}^{r} \beta_{im}^{j} \times MATY_r \tag{4}$$

$$Ast_m \geq \sum_{m=1}^{r} \beta_{im}^{j} \times Tt_{0\ \mathrm{m}} \tag{5}$$

$$\mathrm{X}_{ip}^{j} \times (Sot_{pj} - Eot_{ij} - \sum_{u=1}^{r}\sum_{v=1}^{r} \beta_{iu}^{j}\beta_{pv}^{j} Tt_{uv}) \geq 0 \tag{6}$$

$$\mathrm{Eot}_{ij} - Sot_{ij} = \sum_{m=1}^{r} \beta_{im}^{j} \times q_{ij} \times Pt_m(OPTY_{ij}) \times Sp_m\left(OPTY_{ij}\right) \tag{7}$$

$$T_j = Eot_{ij} + \sum_{m=1}^{r} \beta_{im}^{j} \times Tt_{m\ \mathrm{r+1}} \tag{8}$$

$$\beta_{im}^{j} \times \beta_{pm}^{l} \times \left(Sot_{pl} - Eot_{ij}\right)(Sot_{pl} - Eot_{ij} - \sum_{u=1}^{r}\sum_{v=1}^{r} \beta_{iu}^{j} \times \beta_{pv}^{l} \times Rt_{uv}) \geq 0 \tag{9}$$

$$\beta_{im}^{j} \times (Sot_{ij} - Ast_m) \geq 0 \tag{10}$$

$$\beta_{im}^{j} \times (Eot_{ij} - Aet_m) \leq 0 \tag{11}$$

$$j \in 1\ldots n,\ i \in 1\ldots k,\ m \in 1\ldots r \tag{12}$$

$$l \in 1\ldots n,\ p \in 1\ldots k, u \in 1\ldots r, v \in 1\ldots r \tag{13}$$

Constraint (4) ensures that each operation is assigned to an accessible machine. Constraint (5) ensures that each operation starts only after the raw material transfer time, where index 0 denotes the raw material storage point. Constraint (6) requires that the previous process and the transfer of semi-finished products are completed before the next process starts. Constraint (7) specifies the processing time for each operation. Constraint (8) denotes that the completion time of each job is when the product is transferred to the finished product storage area (index r+1). Constraint (9) ensures that the start time of the next process is after the end time of the previous process plus the machine reconfiguration time. Constraints (10) and (11) ensure that the processing time falls within the machine's available working window, meaning that the available start time is no earlier than the actual start time (10), and the available end time is no later than the actual end time (11). Notifications (12) and (13) define the range of all constraints.

TABLE I. VARIABLE EXPLANATION

variable	**explanation**
$OPTY_{ij}$	processing stage type of operation *i* in job *j*
q_{ij}	operation workload of operation *i* in job *j*
Pt_{m}	processing time of machine *m* per workload
Rt_{uv}	reconfiguration time within machine *u* to machine *v*
Tt_{uv}	transfer time from machine *u* to machine *v*
Tt_{0m}	transfer time from raw material storage to machine *m*
$Tt_{m\ r+1}$	transfer time from machine *m* to final product storage
Ast_m	available starting time of machine *m*
Aet_m	available stop time of machine *m*
$MATY_m$	set of processable type of machine *m*
Sp_m	processing speed of machine *m*
Sot_{ij} Eot_{ij}	start and end time of operation *i* in job *j*

IV. Method

In this section, we present the rationale of our method. We first formulate Markov Decision Process model in our learning algorithm. Then, we design a knowledge guidance method to learn the scheduling policy, followed by the introduction of training algorithm. Our algorithm proposes an innovative knowledge-guided deep reinforcement learning framework, in which reconfiguration time is explicitly integrated into two key components: the reward function and the knowledge guidance. By embedding reconfiguration-related knowledge into these critical stages, the algorithm can rapidly learn dynamic policies and derive convergent and robust scheduling solutions for CPP scheduling. The workflow of the proposed knowledge guided DRL algorithm is shown in Fig. 3.

The scheduling process is actually a series of consecutive decisions assigning jobs to the corresponding machines. The value of β_{jr}^{i} is determined one by one during this process. The scheduling process considered here works as follows. At the beginning, the CPP scheduling instance is initialized into the state vector S_0. Then, the job vector and machine vector are aggregated together to form the action vector. The knowledge network and policy network will calculate the action probabilities, respectively, after applying the masking method. The system will then sample the action α_0 for this time step based on the combined probability. Once the decision is made, the system will continue scheduling if the process is not yet complete. At each decision step t (time 0 or when an operation is completed), the agent observes the current system state S_t and makes a decision α_t. Then, the environment transits to the next decision step t + 1. The process iterates until all the operations are scheduled. The corresponding MDP is defined as follows.

State: The state is used to represent the status of the system and guide decision-making. The state vector at time *t* is denoted as a node vector $S_t = \{N_t(i)\}, i \in \{J, M\}$, which consists of both machine nodes and job nodes. *J* and *M* represent all CPP jobs and machines respectively. To comprehensively characterize the workshop scenario, each node vector is composed of six features $N_t(i) = (wb_t, at_t, nn_t, ut_t, co_t, cs_t)_i, \forall i \in \{J, M\}$. In our framework, the nodes consist of both machines and jobs, and the feature vector is constructed to incorporate critical attributes from both types. A well-designed feature should capture historical, current, and future information. In traditional FJSP solutions, node utilization is used to represent historical information, while idle time characterizes future information [30]. For current information, typical descriptors include processing speed, available resources, operational status, and task operations [31]. To ensure that the features comprehensively reflect the production state, all of these aspects are incorporated into the feature vector. A detailed explanation of the selected features is provided in Table II.

TABLE II. FEATURE EXPLANATION

Feature name	explanation
wb	working binary of node
at	available time(completion time) of node
nn	number of available accessable nodes
ut	utility of node
co	the current operation number of node
cs	current speed of node

Action: This article combines the operation selection and the machine assignment as a composite decision. An action a_t is defined as a feasible operation-machine pair and None type action $a_t = \{(o_{ij}, M_k)\} \cup \{None\}$ at step t, where O_{ij} is eligible and M_k is idle and can process O_{ij} **.**The action vector is concatenated by operation and machine vector, $S_t^{'} = concat(N_t(J), N_t(M))$. In RCSPs, sometimes when the machine chooses a "None" action, enabling it to subsequently select the action with shorter reconfiguration time, thereby reducing the makespan. Therefore, when using the actor to generate a policy, we divide the action vector into available and unavailable parts by masking method and output the corresponding probabilities through policy network respectively.

Transition: The state vector depends on the previous state and action, determining the transition function. Notably, when all jobs are still being processed, the state does not change. A new state is generated at the moment a new operation is completed after time t, marking this time as t+1. At this point, S_{t+1} is updated based on the new node vector. The next action, a_{t+1}, is then selected based on S_{t+1}, and this iterative process continues until all jobs in the manufacturing workshop are completed.

Reward: Reward function is designed to estimate the action and optimize the policy. In the state-of-art research on jobshop scheduling, the reward function is commonly designed as the difference between the estimated completion times of S_t and S_{t+1}. In our study, we have adopted this approach. Additionally, we have considered the impact of machine reconfiguration time to guide the agent in learning strategies that minimize reconfiguration time:

$$r(S_t, S_{t+1}, a_t) = T(S_t) - T(S_{t+1}) - trantime(a_t) \tag{14}$$

where $T(S)$ is current time of state S and *trantime* is machine reconfiguration time of available.

Policy: A policy $\pi(a_t \mid S_t)$ defines a probability distribution over the action set for each state. Our policy network output avail and nonavail action probability separately based on state vector at each decision time. To learn effective strategies faster, we deployed a knowledge-guided structure in outputting action probability. The knowledge network outputs a probability distribution of actions, and actions are sampled based on the knowledge guidance probability. Detailed structure of knowledge network is discussed in next section.

A. Knowledge Guidance Structure

The Flexible Job-Shop Scheduling Problem is strongly NP-hard, and the inclusion of machine reconfiguration time further increases its complexity. To address this large-scale problem, a knowledge-guided structure is integrated into algorithm. The efficacy of knowledge-guided structure in bolstering the efficacy of training results has been empirically validated [32]. The if-then rule is a prevalent form of general advice, this approach has been empirically validated to improve training results and accelerate algorithm training process [33].

To minimize makespan, a general heuristic is proposed to speed up training and improve scheduling results. This heuristic is influenced by the principles of the IF A THEN B structure and e-greedy strategy. it can be summarized as follows: IF the available action has a shorter machine reconfiguration time and a heavier sequential workload, THEN the guidance probability for selecting this action is higher. This method balance exploration and exploitation during learning, preventing the system from settling on locally optimal solutions. Considering the possibility of zero reconfiguration time, the general advice probability distribution of available action is:

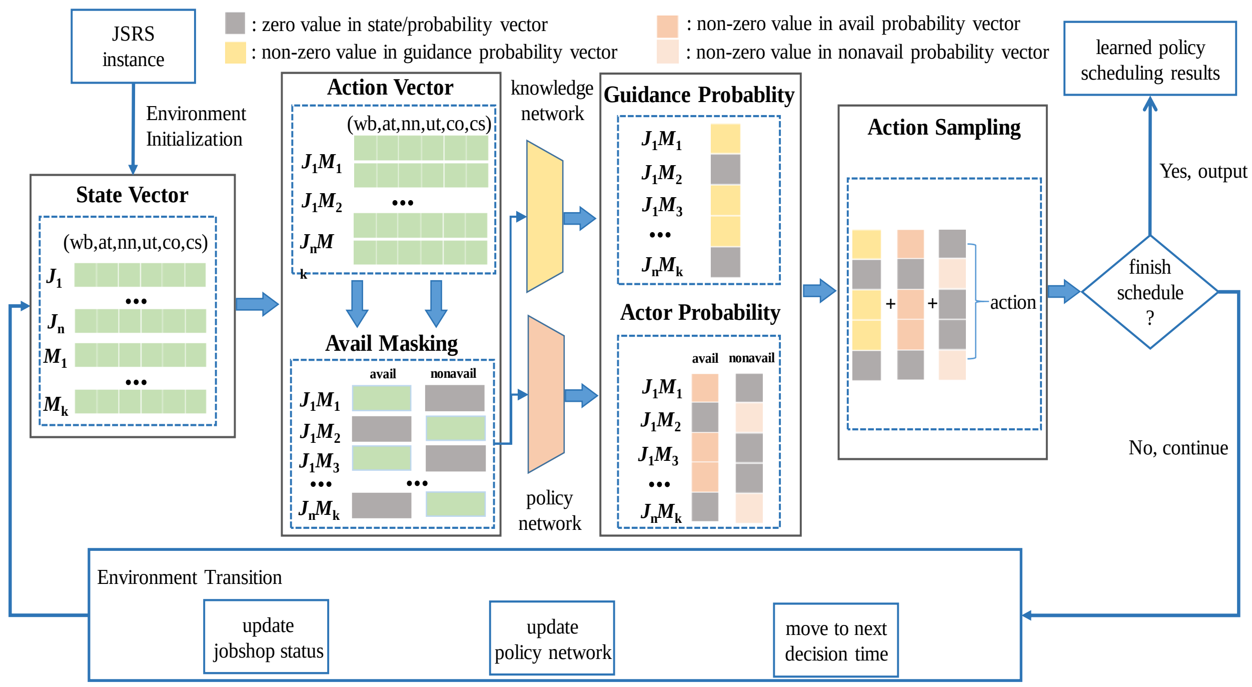

Fig. 3. Workflow of proposed knowledge guided DRL method

$$S_1 = trantime(a_t)/\left\|trantime(a_t)\right\|_2 \tag{15}$$

$$S_2 = workload(a_t)/\left\|workload(a_t)\right\|_2 \tag{16}$$

$$p_{a_t} = \alpha_1(e^{-S_1}/\left\|e^{-S_1}\right\|_1) + \alpha_2(e^{S_2}/\left\|e^{S_2}\right\|_1) \tag{17}$$

where *trantime* is machine reconfiguration time of available action and *workload* is remaining processing time of the available action. The guidance probability for a non-available action is represented as a zero vector, so the knowledge network outputs a single probability vector, as shown in Fig. 3. In summary, the knowledge network serves as the output function of p_{a_t}. Due to the high variance of reconfiguration and processing time between different jobs, using standard activation functions like softmax can lead to probability distributions that are either too concentrated (0-1distribution) or too scattered (uniform distribution) to be effective. Therefore, in this case, we adopt the 1-norm activation form to obtain a more stable probability distribution of p_{a_t} .

B. Training

KGDRL uses Proximal Policy Optimization (PPO) structure for training, which deployed an actor-critic structure. Actor is the policy network π_ω , and critic v_ϕ predicts the value of state S_t . Both actor and critic are designed as MLPs with two hidden layers, π_ω is deployed with a L1-norm activation as mentioned before, the overall training structure is shown in Fig. 3. As shown in Algorithm 1, the training is performed in I iterations and β independent instances during each iteration, we compute the guidance probabilities for each state and incorporate them into the probabilities outputted by the policy network, to optimize the agents' sampling strategy. The actor probability of avail action is calculated using avail state vector, and similarly for nonavail action.

```
Algorithm 1 : KGDRL
Initialize Policy network $\pi_\omega$ and Critic network $V_\phi$ with trainable parameters $\omega$ and $\phi$
Initialize $\beta$ independent instances
  For $iter = 1,2,...I$ do :
    For $b = 1,2,...\beta$ do :
      Initialize $S_t$ based on instance b
      While $S_t$ is not terminal do :
        Initialize $a_t$, mask vector $F_{avail}$ and $F_{nonavail}$ based on $S_t$
          $F_{avail} = 1, F_{nonavail} = 0$ if $O_{ij} \in a_t$
        Compute action distribution based on policy network
          $p_{avail} = \pi_\omega(F_{avail}(O_{ij})S_t), p_{nonavail} = \pi_\omega(F_{nonavail}(O_{ij})S_t)$
        Compute general advice distribution $p_{advice} = p_{a_t}$
        Sample action $a$ based on $p = p_{advice} + p_{avail} + p_{nonavail}$
        Reverie reward $r_t$ and next state $S_{t+1}$
          $S_t \leftarrow S_{t+1}$
      Compute GAE $A_t$ based on $r_t$, $S_{t+1}$ for each step
      Compute PPO loss $\Delta$ based on $A_t$
    Update network parameters $\omega$ and $\phi$ based on PPO loss $\Delta$
Return
```

V. EXPERIMENTS

In the experiments, we evaluate our proposed KGDRL model on RCSPs. Unlike dynamic scheduling problem with setup time, the machine reconfiguration time is related to operation stages, but not job sequence. Therefore, we are incapable of verifying our algorithm utilizing public datasets, compelling us to independently construct RCSP instances for testing. To ensure the effectiveness of the KGDRL method across all CPP problems, a robustness test has been conducted. Finally, we analysis the features of machine reconfiguration, particularly the effectiveness across different instances.

A. Experiment Setting

1) Instance Construction

In real-world CPP scheduling settings, each job consists of multiple processing operations, and the workload required for each operation varies. Additionally, different stages require different machine components, leading to variations in machine processing speeds. Furthermore, due to different reconfiguration methods between components, reconfiguration times are not uniform across operations on the same machine. As a result, three key variables randomly vary in real-world reconfigurable production instances: operation processing time, machine reconfiguration time, and processing speed. Based on the scheduling model, the operation processing time is $p_{\text{ij}} = q_{ij} \times Pt_r\left(OPTY_{ij}\right)$. According to the Central Limit Theorem, the distribution of the mean of a random variable will follow a normal distribution when the data size is large. Based on this, we use a normal distribution function to generate three key random parameters. It is important to note that reconfiguration time is always shorter than operation processing time, as machines only need to switch their components during reconfiguration, so the average reconfiguration time is set to 0.1 times the average processing time and the average machine processing speed is defined as 1, as shown in Table III.

TABLE III. INSTANCE DETAILS

Size(n*m)	p_{ij} (s)	Rt_r (s)	Sp_r (s)
5*5, 10*5, 20*5, 5*10, 10*10, 20*10	N(100,0.1)	N(10,0.1)	N(1,0.1)

2) Configuration

We deployed one hidden layer in both the actor and the critic network. For training, the number of iterations and instance batch size are set to $I = 1000$ and $\beta = 10$ to achieve the best scheduling results, min and max replay buffer is 128 and 2048 respectively, allowing the agent to learn an effective policy in less time. For PPO loss, the clip ratio and coefficients of entropy is 0.1 and 0.01 respectively to ensure the convergent scheduling policies. The PPO epochs are set to 15. These hyperparameters are empirically tuned on the 5 × 5 instance and fixed on the remaining sizes.

3) Baselines

The learned policies are compared to three widely used heuristic rules and three well-known DRL algorithms.

Random: Select action randomly.

MWKR: Select action with most work remaining.

FIFO: Select action firstly arrive in jobshop.

DDQN: Double Deep Q Learning [34].

A2C: Advantage Actor Critic [35].

DDPG: Deep Deterministic Policy Gradient [36].

Given that each stage requires identical processing resources, the reconfiguration time remains constant between identical stages. Furthermore, the reconfiguration time is zero when sequentially processing operations within the same stage.

B. Performance

1) Convergence of Algorithms

Based on the generation rules provided in Tab.III, we generated 10 instances for each of the six dimensions to train our proposed KGDRL algorithm. We then calculated the average makespan for the 10 instances in each size. The training process of the KGDRL method is stable and converged rapidly. As shown in Fig. 4, the training curve for six different sizes exhibit some fluctuations occur within the first 200 iterations, followed by swift convergence. With the help of the knowledge-guided network, the DRL agent can find an effective search policy during scheduling. The convergence performance is consistent across all instance sets, highlighting the necessity of the proposed knowledge-guided mechanism.

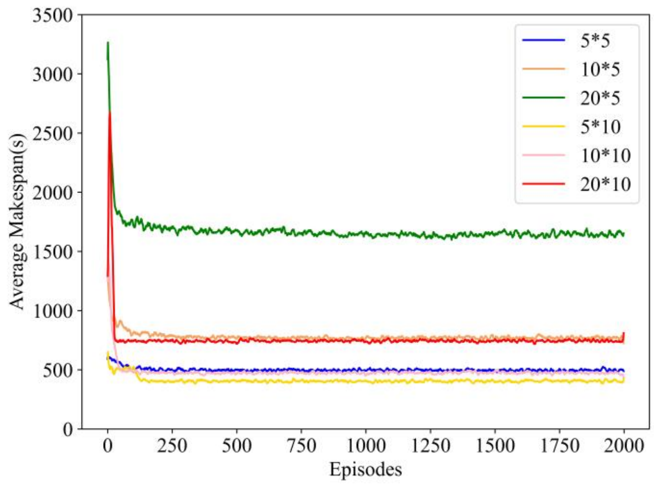

Fig. 4. Training curve of KGDRL

To demonstrate the potential of KGDRL, we compared their training results with other baseline methods, as discussed earlier. Specifically, we evaluated the convergence of all baseline algorithms by calculating their standard makespan after 1000 iterations. The standard makespan is computed by dividing the makespan of each instance by the ratio parameter (defined as [n/m], where n is the number of jobs and m is the number of machines). For example, the ratio parameter is 1 for a 5*10 instance. As shown in Fig.5, for converged training policies, we observe a stable standard makespan across all instances after 1000 iterations. Specifically, KGDRL and DDQN exhibit stable policies, with their makespans consistently close to 400 across all six instance sizes. In contrast, the standard makespan for most instances trained by A2C and DDPG exceeds 800 and increases significantly with dimensionality, indicating poor scalability and convergence.

The high training fluctuations observed in these two algorithms indicate that their training results are unreliable, as they continue to search for actions randomly and have not learned an effective search policy. Consequently, we select the DDQN algorithm and other three rules for comparison in scheduling results due to the unconvincing results provided by A2C and DDPG.

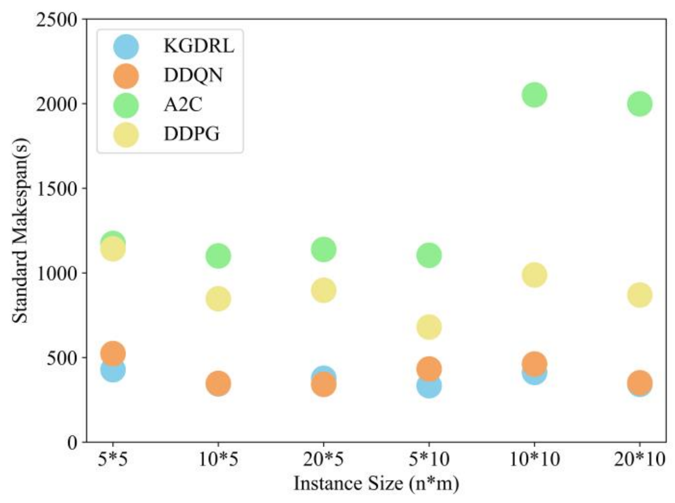

Fig. 5. Standard makespan of baselines after 100 iterations

2) Scheduling Results

For CPP production, a shorter makespan indicates superior performance. The total makespan is the maximum completion time among all jobs, we compared the makespan results of all baselines in six size, as shown in Fig. 6. We can find that as the number of jobs increases, adding more machines results in greater improvements in makespan. This finding highlights the value of reconfiguration. When the number of available job-machine action pairs increases, dynamic reconfiguration effectively mitigates workload imbalances, accelerating the optimization of makespan. In detail, our proposed KGDRL has achieved better scheduling results compared to the baselines. By noticing that MKWR rules achieved the best scheduling results of all heuristic rules in all size instances, KGDRL significantly outperform MWKR in five instances, indicating the value of DRL when solving high dimension scheduling problems like CPP. KGDRL also provides a more stable training policy compared to other DRL algorithms. KGDRL performed better in most cases and is more stable compared to DDQN. Although the DDQN algorithm performs well in the 20*5 case, it significantly underperforms the MWKR rule on high-dimensional cases, exhibiting a lack of sufficient robustness.

C. Robustness

1) Training Time

To verify the insight that KGDRL maintains stable scheduling results compared to DDQN, we conducted a detailed robustness check. The key robustness indicator for these algorithms is their training time, which we compared across all baseline methods, as shown in Fig. 7. KGDRL demonstrates stable training times across all instance sizes, while DDQN exhibits substantial fluctuations, with a discrepancy of up to 6000 seconds across the six instance sizes. The training time pattern of DDQN is similar to that of A2C and DDPG, which, as previously analyzed, fail to select effective actions during training. This suggests that the training process of DDQN relies heavily on random selection, resulting in unstable training outcomes.

The advanced mechanism of KGDRL supports stable training. KGDRL used a non-available action mask mechanism, which effectively filters out unavailable actions during scheduling across all instances, thus facilitating efficient learning.

2) Scalability on Multi-Scale Instances

To further evaluate the robustness of KGDRL, we considered the variance in reconfiguration time, which is a critical factor in real-world applications. Given the significant volatility of machine reconfiguration times in CPP scheduling, it is essential to verify whether our approach delivers favorable scheduling performance across various reconfiguration time scenarios. For robustness check, we initially trained KGDRL using the training set with 5 different categories with 5*5 instances, and then deployed the learned strategy on a validation set, comparing it with our previously constructed baselines. Detailed information of robustness instance is shown in Tab.IV. Overall, the proposed KGDRL method not only achieves the best scheduling outcomes but also demonstrates excellent robustness, with both the mean and standard deviation of the makespan on the validation set significantly lower than all baseline methods.

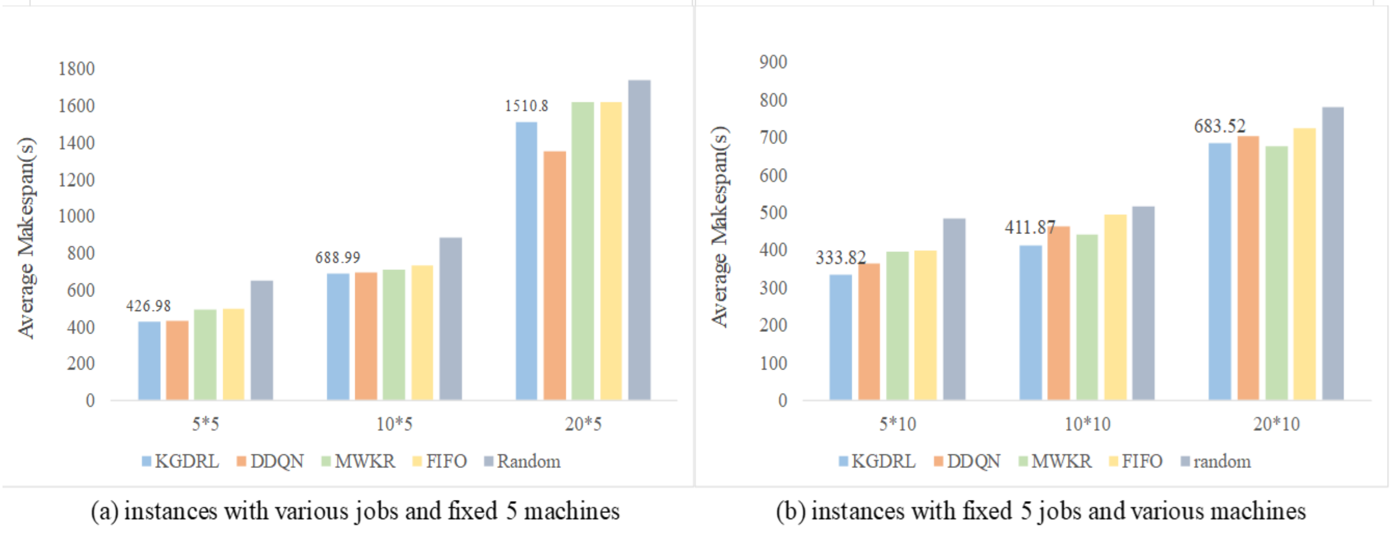

Fig. 6. Average Makespan of six sizes

Fig. 8 presents the average makespan on the validation set. Among all methods, KGDRL exhibits the most concentrated distribution, indicating its ability to consistently achieve high-quality scheduling results under the guidance of the knowledge network. In contrast, DDQN exhibits a larger standard deviation and a higher mean makespan, aligning closely with random rules. This further validates the reliance on random selection during the DDQN training process. As the action space expands with increasing instance size, the action vector gradually becomes sparse, with most actions having the same, but highest, probabilities. As a result, the DRL agent cannot select the most suitable action and is forced to randomly choose one. To address this, KGDRL implemented an aggregation mechanism for three types of action probabilities vectors to eliminate the sparsity, enabling the agent to select the appropriate action at each decision point.

D. *Further Analysis*

1) *Reconfiguration Details*

Machine reconfiguration operations play a crucial role in shaping future scheduling decisions. Through effective reconfiguration, a dynamically reconfigurable workshop can efficiently reorganize diverse and customized jobs, and reduce the overall completion time of the workshop. We presented a 5×5 instance as an example, the detailed job and machine parameters for this instance are provided in Tables V and VI. This example illustrates the dynamic nature of CPP instances, as evidenced by significant variations in processing times across operations (up to 100 seconds), processing speed differences (up to 80% on Machine M1), and reconfiguration time fluctuations (up to 33% on Machine M5). Due to the limited reconfigurability of machines, conventional scheduling methods result in low work efficiency for machines M3 and M4, as most jobs are assigned to highly reconfigurable machines. This imbalance significantly increases the overall scheduling time of the workshop. However, after implementing reconfiguration operations, machines M3 and M4 can still complete jobs within their processing range, as shown in the Gantt chart in Fig. 9, ensuring overall production efficiency.

To further analyze the impact of reconfiguration on CPP, we calculated the average makespan and reconfiguration time for 25 validation instances, as shown in Fig. 10. It can be observed that KGDRL performs well, even though the average reconfiguration time is greater than the processing time. The trained policy strikes a balance between reconfiguration and processing, thereby maximizing the operational efficiency of the job shop. When dealing with instances that have high reconfiguration times, the production workshop is more likely to experience an unbalanced workload, as machines need much more reconfiguration time to process essential jobs. To address this, our knowledge-guided rules incorporate workload design. By considering machine workload, the agent can better understand the processing sequence, thereby reducing the impact of reconfiguration time. The agent may opt for actions with longer processing times but shorter reconfiguration times when the reconfiguration time of target action is extremely long, allowing KGDRL to achieve the best balance between processing and reconfiguration, ultimately resulting in better CPP scheduling policies.

2) *Flexibility Analysis*

Given that machine flexibility greatly varied in real-world manufacturing contexts, we analyzed its impact on scheduling strategies. For reconfigurable machines, the flexibility level of jobshop is defined as the sum of the processable stages of each machine divided by the total processing stages.

To ensure the conclusions are representative, we analyze the scheduling results of the validation dataset with different machine reconfiguration time. All instances are divided into three groups: low, medium and high. As shown in Fig. 11, KGDRL achieves the best results across all flexibility levels, while DDQN performs worse than both MWKR and FIFO rules at all levels. The performance gap is particularly larger at medium and high flexibility levels. These results highlight the limitations of Deep Q networks when handling large-scale data. As flexibility increases, the state vector expands rapidly, and the Q network struggles to effectively capture key features of high-dimensional tensors, leading to low action selection efficiency throughout the training process, which degrades policy performance. In our proposed KGDRL algorithm, the masking operation and knowledge-guided mechanism are employed to address this issue. Regardless of the state vector and action set size, the KGDRL agent can effectively learn a convergent policy and achieve the best scheduling results.

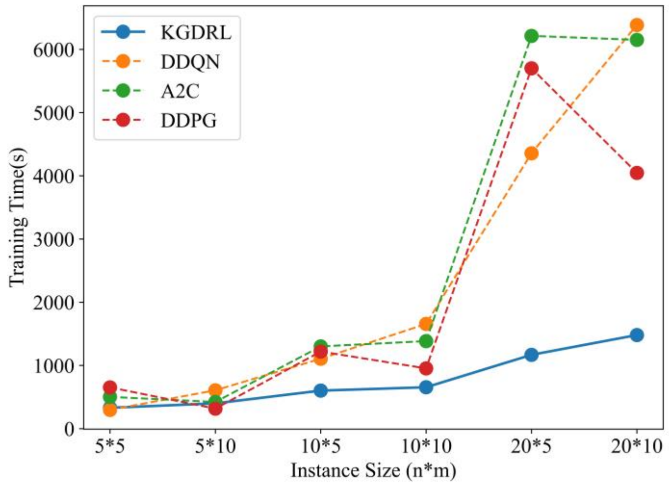

Fig. 7. Training time of baselines after 100 iterations

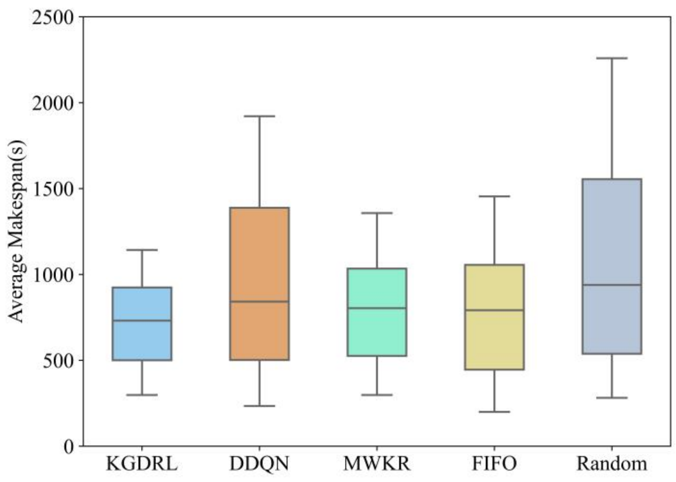

Fig. 8. Boxplot of average makespan on 5*5 validation instance

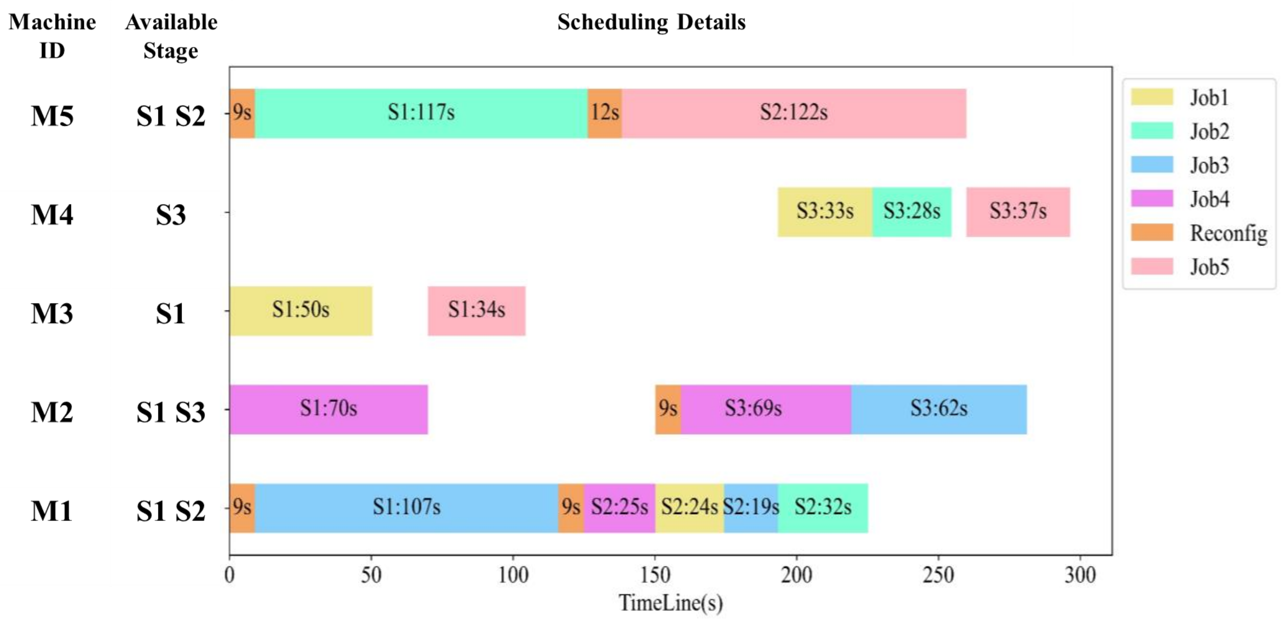

Fig. 9. Reconfiguration example of gantt chart on 5*5 instance

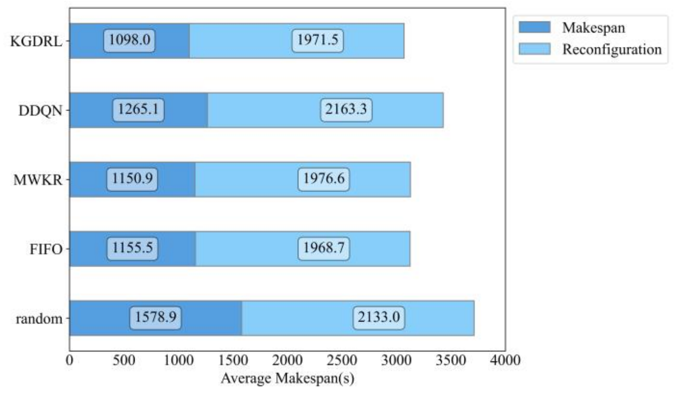

Fig. 10. Makespan reconfiguration comparison on 5*5 instances

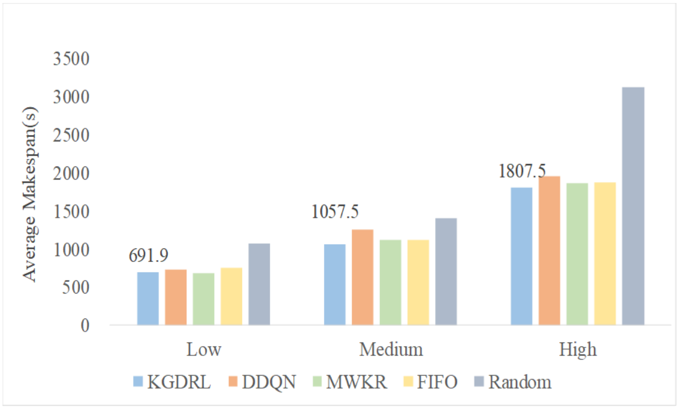

Fig. 11. Policy variance on different flexibility instances

VI. CONCLUSIONS AND FUTURE RESEARCH

This paper addresses the resource reconfiguration scheduling problem for processing services in cloud manufacturing through cloud-edge collaboration and knowledge-guided deep reinforcement learning. The main conclusions are summarized as follows:

1) The ultra-flexible CMfg framework is proposed to support dynamic machine reconfiguration for varying processing requirements and to enable smart decision-making based on intelligent scheduling models and algorithms.

2) We developed a CPP model that incorporates reconfiguration variables to gain a comprehensive understanding of reconfiguration time variance and its impact on scheduling, enabling more effective problem-solving.

3) Prior knowledge about reconfiguration time and machine workload is embedded in the training network, making the learning policy of KGDRL more efficient than the baselines. It remains robust across various instance sizes, indicating significant potential for solving complex CPP scheduling problems in cloud manufacturing.

In the future, digital twins can be introduced for real-time monitoring of CPP scheduling. Uncertainties related to reconfiguration can be tracked throughout the entire process, as digital twins can be deployed on both machines and jobs to accurately diagnose these uncertainties. Once identified, the digital twin can transmit the cause and type of the issue to the manufacturing cloud, enabling the system to generate appropriate solutions and thus ensure safe and efficient production.

TABLE IV. ROBUSTNESS INSTANCE DETAILS

Category	C1	C2	C3	C4	C5
Rt_r (s)	N(1,0.1)	N(10,0.1)	N(20,0.1)	N(100,0.1)	N(1000,0.1)
Train Number	/	10	/	10	/
Val Number	5	5	5	5	5

TABLE V. JOB PARAMETER OF PROPOSED INSTANCE

Processing time(s)		**Jobs**				
		J1	**J2**	**J3**	**J4**	**J5**
Operation	**O1**	O11:126	O21:69	O31:107	O41:100	O51:86
	O2	O12:121	O22:159	O32:95	O42:126	O52:81
	O3	O13:111	O23:93	O33:62	O43:60	O53:122

TABLE VI. MACHINE PARAMETER OF PROPOSED INSTANCE

Machines	**M1**	**M2**	**M3**	**M4**	**M5**
Available stages	S1 S2	S1 S3	S1	S3	S1 S2
Processing speed	1.0 0.2	0.7 1.0	0.4	0.3	1.7 1.5
Reconfiguration time(s)	S1-S2:9 S2-S1:9	S1-S3:9 S3-S1:8	/	/	S1-S2:12 S2-S1:9

REFERENCES

[1] Yang, Chen, et al. "Towards product customization and personalization in IoT-enabled cloud manufacturing." Cluster Computing 20 (2017): 1717-1730.

[2] Vlachos I, Pascazzi R M, Ntotis M, et al. Smart and flexible manufacturing systems using Autonomous Guided Vehicles (AGVs) and the Internet of Things (IoT)[J]. International Journal of Production Research, 2024, 62(15): 5574-5595.

[3] Zhou Y, Du S, Liu M, et al. Machine-fixture-pallet resources constrained flexible job shop scheduling considering loading and unloading times under pallet automation system[J]. Journal of Manufacturing Systems, 2024, 73: 143-158.

[4] Javaid M, Haleem A, Singh R P, et al. Enabling flexible manufacturing system (FMS) through the applications of industry 4.0 technologies[J]. Internet of Things and Cyber-Physical Systems, 2022, 2: 49-62.

[5] Milisavljevic-Syed J, Li J, Xia H. Realisation of responsive and sustainable reconfigurable manufacturing systems[J]. International Journal of Production Research, 2024, 62(8): 2725-2746.

[6] Rohaninejad M, Vahedi-Nouri B, Hanzálek Z, et al. An integrated lot-sizing and scheduling problem in a reconfigurable manufacturing system under workforce constraints[J]. International Journal of Production Research, 2024, 62(11): 3994-4013.

[7] Li Y, Li X, Gao L, et al. Multi-agent deep reinforcement learning for dynamic reconfigurable shop scheduling considering batch processing and worker cooperation[J]. Robotics and Computer-Integrated Manufacturing, 2025, 91: 102834.

[8] Pang, Shibao, et al. "Mass personalization-oriented integrated optimization of production task splitting and scheduling in a multi-stage flexible assembly shop." Computers & Industrial Engineering 162 (2021): 107736.

[9] Lei, Kun, et al. "Large-scale dynamic scheduling for flexible job-shop with random arrivals of new jobs by hierarchical reinforcement learning." IEEE Transactions on Industrial Informatics 20.1 (2023): 1007-1018.

[10] Delorme X, Fleury G, Lacomme P, et al. Modelling and solving approaches for scheduling problems in reconfigurable manufacturing systems[J]. International Journal of Production Research, 2024, 62(7): 2683-2704.

[11] Yang, Chen, et al. "Metaverse: Architecture, Technologies, and Industrial Applications." 2023 IEEE 19th International Conference on Automation Science and Engineering (CASE). IEEE, 2023.

[12] Zhao Z, Zhang M, Chen J, et al. Digital twin-enabled dynamic spatial-temporal knowledge graph for production logistics resource allocation[J]. Computers & Industrial Engineering, 2022, 171: 108454.

[13] Wang K J, Lee T L. Designing a digital-twin based dashboard system for a flexible assembly line[J]. Computers & Industrial Engineering, 2024, 196: 110491.

[14] Li Q, Tang W, Li Z. Leveraging Industry 4.0 for Sustainable Manufacturing: A Quantitative Analysis Using FI-RST[J]. Applied Sciences, 2024, 14(20): 9545.

[15] Koren, Yoram, et al. "Reconfigurable manufacturing systems." CIRP annals 48.2 (1999): 527-540.

[16] Zhang, Lixiang, et al. "Distributed real-time scheduling in cloud manufacturing by deep reinforcement learning." IEEE Transactions on Industrial Informatics 18.12 (2022): 8999-9007.

[17] Hu, Y., Zhang, L., Zhang, Z., Li, Z., & Tang, Q. (2024). Flexible assembly job shop scheduling problem considering reconfigurable machine: A cooperative co-evolutionary matheuristic algorithm. Applied Soft Computing, 166, 112148.

[18] Fu B, Bi M, Umeda S, et al. Digital Twin-based Smart Manufacturing: Dynamic Line Reconfiguration for Disturbance Handling[J]. IEEE Transactions on Automation Science and Engineering, 2025.

[19] Yelles-Chaouche A R, Gurevsky E, Brahimi N, et al. Minimizing task reassignments under balancing multi-product reconfigurable manufacturing lines[J]. Computers & Industrial Engineering, 2022, 173: 108660.

[20] Chen X, Li Y, Wang L, et al. Multi-objective grey wolf optimizer based on reinforcement learning for distributed hybrid flowshop scheduling towards mass personalized manufacturing[J]. Expert Systems with Applications, 2025, 264: 125866.

[21] Mahmoodi E, Fathi M, Tavana M, et al. Data-driven simulation-based decision support system for resource allocation in industry 4.0 and smart manufacturing[J]. Journal of Manufacturing Systems, 2024, 72: 287-307.

[22] Khou S A, Olamaei J, Hosseini M H. Strategic scheduling of the electric vehicle-based microgrids under the enhanced particle swarm optimization algorithm[J]. Scientific Reports, 2024, 14(1): 30795.

[23] Serrano-Ruiz J C, Mula J, Poler R. Job shop smart manufacturing scheduling by deep reinforcement learning[J]. Journal of Industrial Information Integration, 2024, 38: 100582.

[24] Du Y, Li J. A deep reinforcement learning based algorithm for a distributed precast concrete production scheduling[J]. International Journal of Production Economics, 2024, 268: 109102.

[25] Yang, S., Wang, J., & Xu, Z. (2024). Learning to schedule dynamic distributed reconfigurable workshops using expected deep Q-network. Advanced Engineering Informatics, 59, 102307.

[26] Lu J, Yang J, Li S, et al. A2C-DRL: Dynamic scheduling for stochastic edge–cloud environments using A2C and deep reinforcement learning[J]. IEEE Internet of Things Journal, 2024, 11(9): 16915-169.

[27] Zhang, F., Li, R., & Gong, W. (2024). Deep reinforcement learning-based memetic algorithm for energy-aware flexible job shop scheduling with multi-AGV. Computers & Industrial Engineering, 189, 109917.

[28] Liu, Xiaoyu, et al. "Multi-agent deep reinforcement learning for end—edge orchestrated resource allocation in industrial wireless networks." Frontiers of Information Technology & Electronic Engineering 23.1 (2022): 47-60.

[29] Zhou, Junlong, et al. "Dependable scheduling for real-time workflows on cyber–physical cloud systems." IEEE Transactions on Industrial Informatics 17.11 (2020): 7820-7829.

[30] Zhou T, Zhu H, Tang D, et al. Reinforcement learning for online optimization of job-shop scheduling in a smart manufacturing factory[J]. Advances in Mechanical Engineering, 2022, 14(3): 16878132221086120.

[31] Song W, Chen X, Li Q, et al. Flexible job-shop scheduling via graph neural network and deep reinforcement learning[J]. IEEE Transactions on Industrial Informatics, 2022, 19(2): 1600-1610.

[32] Liu L, Zhou W, Guan K, et al. Knowledge-guided machine learning can improve carbon cycle quantification in agroecosystems[J]. Nature communications, 2024, 15(1): 357.

[33] Kuhlmann, Gregory, et al. "Guiding a reinforcement learner with natural language advice: Initial results in RoboCup soccer." The AAAI-2004 workshop on supervisory control of learning and adaptive systems. 2004.

[34] Liu R, Piplani R, Toro C. Deep reinforcement learning for dynamic scheduling of a flexible job shop[J]. International Journal of Production Research, 2022, 60(13): 4049-4069.

[35] Monaci M, Agasucci V, Grani G. An actor-critic algorithm with policy gradients to solve the job shop scheduling problem using deep double recurrent agents[J]. European Journal of Operational Research, 2024, 312(3): 910-926.

[36] Wang J, Zhou H, Guo J, et al. A Q-Learning-based Deep Deterministic Policy Gradient Algorithm for the Re-entrant Hybrid Flow Shop Joint Scheduling Problem with Dual-gripper[J]. Engineering Letters, 2025, 33(5).