machines

MDPI

Article

Multi-Agent Reinforcement Learning for Extended Flexible Job Shop Scheduling

Shaoming Peng [1,2], Gang Xiong [2,3,4,†], Jing Yang [1,2], Zhen Shen [2,5], Tariku Sinshaw Tamir [6,7], Zhikun Tao [1,2], Yunjun Han [2,3,*] and Fei-Yue Wang [8]

1 School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China; pengshaoming2021@ia.ac.cn (S.P.); yangjing2020@ia.ac.cn (J.Y.); taozhikun2023@ia.ac.cn (Z.T.)
2 State Key Laboratory of Multimodal Artificial Intelligence Systems, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China; gang.xiong@ia.ac.cn (G.X.); zhen.shen@ia.ac.cn (Z.S.)
3 Beijing Engineering Research Center of Intelligent Systems and Technology, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China
4 Guangdong Engineering Research Center of 3D Printing and Intelligent Manufacturing, Cloud Computing Center, Chinese Academy of Sciences, Dongguan 523808, China
5 Intelligent Manufacturing Center, Qingdao Academy of Intelligent Industries, Qingdao 266109, China
6 State Key Laboratory of Precision Electronic Manufacturing Technology and Equipment, Guangdong University of Technology, Guangzhou 510006, China; tamir@ia.ac.cn
7 School of Electrical and Computer Engineering, Institute of Technology, Debremarkos University, Debremarkos 269, Ethiopia
8 State Key Laboratory for Management and Control of Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China; feiyue.wang@ia.ac.cn
* Correspondence: yunjun.han@ia.ac.cn
† Gang Xiong is the co-first author.

Abstract: An extended flexible job scheduling problem is presented with characteristics of technology and path flexibility (dual flexibility), varied transportation time, and an uncertain environment. The scheduling can greatly increase efficiency and security in complex scenarios, e.g., distributed vehicle manufacturing, and multiple aircraft maintenance. However, optimizing the scheduling puts forward higher requirements on accuracy, real time, and generalization, while subject to the curse of dimension and usually incomplete information. Various coupling relations among operations, stations, and resources aggravate the problem. To deal with the above challenges, we propose a multi-agent reinforcement learning algorithm where the scheduling environment is modeled as a decentralized partially observable Markov decision process. Each job is regarded as an agent that decides the next triplet, i.e., operation, station, and employed resource. This paper is novel in addressing the flexible job shop scheduling problem with dual flexibility and varied transportation time in consideration and proposing a double Q-value mixing (DQMIX) optimization algorithm under a multi-agent reinforcement learning framework. The experiments of our case study show that the DQMIX algorithm outperforms existing multi-agent reinforcement learning algorithms in terms of solution accuracy, stability, and generalization. In addition, it achieves better solution quality for larger-scale cases than traditional intelligent optimization algorithms.

Keywords: production planning and scheduling; multi-agent reinforcement learning; flexible job shop; path flexibility; technological flexibility

Citation: Peng, S.; Xiong, G.; Yang, J.; Shen, Z.; Tamir, T.S.; Tao, Z.; Han, Y.; Wang, F.-Y. Multi-Agent Reinforcement Learning for Extended Flexible Job Shop Scheduling. *Machines* **2024**, *12*, 8. https://doi.org/10.3390/machines12010008

Academic Editor: Francisco J. G. Silva

Received: 25 November 2023
Revised: 8 December 2023
Accepted: 11 December 2023
Published: 22 December 2023

1. Introduction

Flexible job shop scheduling (FJSP) is regarded as an effective measure to deal with the challenge of mass personalized and customized manufacturing in the era of Industry 4.0, and is widely extended to many real applications [1]. In this paper, we focus on a kind of extended FJSP, which considers both technology and path flexibility (dual flexibility), varying transportation times among stations, and an uncertain environment. This extended

FJSP is termed FJSP-DT (FJSP with dual flexibility and transportation time variation). FJSP-DT occurs in real complex scenarios, such as distributed automobile assembly scheduling, multiple aircraft maintenance scheduling, and robot scheduling in a warehouse sorting center, etc. FJSP-DT takes the following characters into consideration and is abstractly depicted in Figure 1.

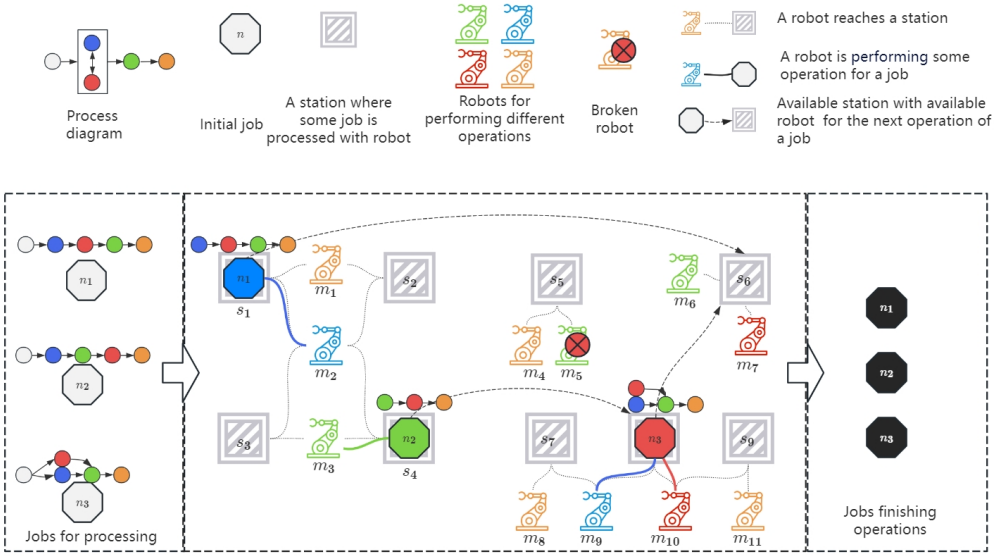


Figure 1. Illustration of FJSP characterized by dual flexibility and varied transportation time in an uncertain environment.

1. Dual flexibity. FJSP-DT considers both technology and path flexibility. As shown in Figure 1, the blue operation may be processed before the red operation, or vice versa, which results in technology flexibility. The same operations can be processed by different robots, e.g., the orange operation can be processed by Robot m_1 or Robot m_4, hence leading to path flexibility.
2. Varied transportation time. One job needs to be processed through multiple stations, and the transportation time between operations varies depending on the distance between stations.
3. Cooperative gaming and uncertain environment. Multiple jobs coordinate their operations to achieve shorter makespan employing preempt resources, such as robots, stations, etc. The environment also suffers from uncertain factors, e.g., robot failures.

Based on the thorough survey of [2] and papers related to JSP published after 2022 (e.g., [3–10]), we classified the works related to FJSP into 23 categories by the factors considered (shown in Table 1), such as machine availability, routing, varied transportation time, etc. It can be concluded that there is a lack of FJSP studies that take all the characteristics above into account. It is necessary to conduct in-depth research due to the importance of FJSP-DT applications. However, solving FJSP-DT faces significant challenges.

1. Generalization. The scheduling algorithm should be adaptable to changing conditions, such as altered operation sequences, modified stations, robot distributions, or varying job quantities.
2. Real-time. Due to the uncertainty of the environment, the scheduler needs to respond in real time (in seconds) to robot failures and make real-time scheduling.
3. Accuracy. Significant operational risks may arise in the application scenario corresponding to FJSP-DT. In automobile assembly scheduling, poor coordination between operations may lead to collisions. Therefore, the algorithm needs to ensure operational safety.
4. Curse of dimensionality. The solution space expands as the number of agents and resources (such as robots, and stations) increases, resulting in a computational dilemma. Here, M, S, and T are used to represent the sets of robots, stations, and time respec-

tively. N and O represent the sets of agents and operations respectively. The solution space of FJSP-DT is $A^{|N|\times|O|}_{|M|\times|S|\times|T|}$ which causes an explosion in combinations.

The methods of solving FJSP-related problems fall under three categories: mathematical programming, meta-heuristic algorithms, and reinforcement learning. Mathematical programming mainly applies to small-scale problems due to its long computation time [11]. Therefore, more than 60% of the publications applied evolutionary algorithms (meta-heuristic) to overcome the challenge [12]. However, these methods require high computational costs and result in unsuitability for large-scale, real-time scheduling problems [13].

Table 1. Literature related to job shop scheduling problems (JSP).

Type	Number [1]
Classical JSP	68
Dynamic JSP	48
JSP considering the machine availability	52
Flexible JSP with alternative machines (FJSP)	225
JSP with alternative routings	32
JSP considering batches	39
JSP considering setup times	69
JSP considering transportation time	1
JSP with nondeterministic or non-constant processing time	32
Distributed JSP (DSJSP)	25
JSP with dual-resource constraints (DRJSP)	30
JSP considering energy and pro-environment	132
JSP with a prior job	5
JSP with dependent jobs	2
JSP with no-wait constraint for operations on the same job	18
JSP with blocking constraint for capacities of buffer	8
JSP with reentrancy	9
JSP with preemption	2
JSP considering overtime work	2
JSP with limited buffer capacity	2
JSP considering outsourcing (subcontracting)	6
JSP considering robot or automated guided vehicle (AGV)	28
FJSP with worker cooperation flexibility	1
FJSP with technology and path flexibility (FJSP-DT)	-

[1] The statistics in the table refer to papers published between 2016 and 2023. Specifically, data before 2022 is referenced from [2], while the rest of the data is obtained through Google Scholar search, https://scholar.google.com/ (accessed on 10 December 2023).

Reinforcement learning has been widely applied to solve numerous optimization problems in the real world [14], and was regarded as one of the top five most promising algorithms [15] to solve JSP due to its strong self-learning ability. Since the Deep Q-Network was proposed, multi-agent reinforcement learning (MARL) has been a focused piece of research in solving FJSP. To map FJSP to the MARL-solving architecture, each job or robot is regarded as an agent in the scheduling environment, where multiple agents cooperate and learn their policies. After offline training, the MARL algorithm can be deployed online to make decisions based on real-time observations. Consequently, MARL has the great potential to model complex workshop scheduling problems [16], and can achieve real-time computation and adaptability [17]. Therefore, we adapt MARL to solve the FJSP-DT problem with several adaptive improvements in architecture design, value function calibration, etc. The main contributions of our paper can be summarized as follows.

1. The FJSP-DT abstracted from real and complex scenarios is proposed. As far as our literature review is concerned, the problem is new and needs to be fully investigated. The FJSP-DT is modeled as a decentralized partially observable Markov decision process in that an agent may only observe the information around it in real applications.

Then, the MARL-based method can be applied to achieve high adaptability and real-time decisions.

2. We build an adaptive and stable multi-agent learning framework by combining a graph convolutional network (GCN) and the actor–critic structure. GCN extracts embedding graph's structural features and represents system states in non-Euclidean space, thereby degrading dimension curses and adapting to various scheduling environments. The actor–critic structure can update the network parameters in a single step without running an episode, making it faster than the policy gradient algorithm.
3. A double Q-value mixing algorithm (DQMIX) under the above framework is proposed to address the challenge of fast convergence and high adaptability. The algorithm combines an unrestricted optimal network with a monotonic mixing network to improve exploration and exploitation capabilities. It also integrates mechanistic constraints into data-based learning, mitigating the curse of dimensionality by eliminating invalid actions. In addition, the reward function is designed as a function of reduction in makespan estimation to mitigate the learning challenges caused by sparse feedback.

2. Related Works

In this section, we will focus our review on the research about reinforcement learning in applications to the optimization of JSP, including single-agent reinforcement learning and multi-agent reinforcement learning.

- Single-Agent Reinforcement Learning (SARL): The algorithm only contains one agent that makes all the decisions for a control system.
- Multi-Agent Reinforcement Learning (MARL): The algorithm comprises multiple agents that interact with the environment through their respective policies.

2.1. SARL for Scheduling

SARL virtualizes an agent interacting with the scheduling environment, learning a scheduling policy, and then making decisions. The early paper applying SARL to JSP may be traced back to Zhang and Dietterich (1995) to learn a heuristic evaluation function over states [18]. Subsequently, Aydin and Öztemel (2000) [19] applied reinforcement learning to choose dispatching rules depending on the current state of a production system. Since the proposal of Deep Q-Network (DQN), using SARL to solve JSP has attracted more and more attention.

2.1.1. SARL with Value Iteration

Waschneck et al. (2018) [20] applied the DQN algorithm to solve a dynamic and flexible production problem with the objective of maximizing plant throughput. The proposed model took machine availability and processing characteristics as states and mapped the states to the station selection. Luo (2020) [21] developed an optimization algorithm based on Double DQN (DDQN) for dynamic FJSP with order insertion. The algorithm can select appropriate scheduling rules according to the job state and obtain a plan better than the general scheduling rules. Lang et al. (2020) [22] combined the DQN algorithm with discrete event simulation to solve a flexible job shop problem with process planning. Two independent DQN agents are trained. One agent selects operation sequences, while the other assigns jobs to machines. Du et al. (2021) [6] considered an FJSP with time-of-use electricity price constraint and dual-objective optimization for the makespan and total price and proposed a hybrid multi-objective optimization algorithm of estimation of distribution algorithm and DQN to solve the problem. Li et al. (2022) [5] presented dynamic FJSPs with insufficient traffic resources (DFJSP-ITR). They proposed a hybrid DQN (HDQN) that includes double Q-learning, prioritized replay, and a soft target network update policy to minimize the maximum duration and total energy consumption. Gu et al. (2023) [23] integrated DQN method into a scalp swarm algorithm (SWA) framework to dynamically tune the population parameters of SWA for JSP solving.

2.1.2. SARL with Policy Iteration

Wang et al. (2021) [24] considered the uncertainties, such as the mechanical failure of the job shop, and proposed a dynamic scheduling method based on the proximal policy optimization (PPO) to find the optimal scheduling policy. The states are defined by the job processing state matrix, the designated machine matrix, and the processing time matrix of the operations. An action set is defined as the operation selected from the candidate operation set, and the reward is related to machine utilization. The results showed that the proposed approach based on reinforcement learning obtained comparative solutions and achieved adaptive and real-time scheduling.

The application of SARL to solve the scheduling problem has some limitations. Firstly, FJSP-DT occurs in an uncertain environment, and the information obtained by the agent is likely to be incomplete, which is difficult for SARL to handle since SARL depends on global information. Secondly, SARL does not tackle communication and collaboration between jobs or machines, resulting in the loss of important scheduling information [16]. Thirdly, the action space of SARL expands with the number of jobs or machines [25], and a high action dimension can pose a challenge to policy learning. Research shows that the performance of policy gradient methods gradually lags as the action dimension increases [26].

2.2. MARL for Scheduling

MARL aims to model complex worlds where each agent can make adaptive decisions, realizing competition and cooperation with humans and other agents, and is attracting more and more attention in academia and industry [27].

From the perspective of the multi-agent system's training paradigm, agents' training can be broadly divided into distributed and centralized schemes.

- Distributed Training Paradigm (DTP): In the distributed Paradigm, agents learn independently of other agents and do not rely on explicit information exchange.
- Centralized Training Paradigm (CTP): The centralized paradigm allows agents to exchange additional information during training, which is then abandoned during tests. Agents receive only the locally observable information and independently determine actions according to their policies during execution.

2.2.1. MARL with DTP

Regarding works on MARL with DTP, Aissani et al. (2009) [28] applied MARL for adaptive scheduling in multi-site companies. The supervisor agent sends requests to inventory agents and resource agents at different sites for a solution in the company's multi-agent system. The inventory agent asks the resource agent to propose a solution. The resource agent then starts its decision-making algorithm based on the SARSA (state–action–reward–state-action) algorithm using the data system (the resource state, task duration, etc.) and sends back a solution. Martínez et al. (2020) [29] proposed a MARL tool for JSPs where machines are regarded as agents. This tool allows the user to either keep the best schedule obtained by a Q-learning algorithm or modify it by fixing some operations to satisfy certain constraints. The tool then optimizes the modified solution, taking into account the user's preferences and using the possible alternatives. Hameed et al. (2020) [30] presented a distributed reinforcement learning approach for JSPs. The innovation of this paper is that the authors modeled various relationships within the manufacturing environments (robot manufacturing cells) as graph neural networks (GNN). Zhou et al. (2021) [31] proposed a new distributed architecture with multiple artificial intelligence (AI) schedulers for the online scheduling of orders in smart factories. Each machine is taken as a scheduler agent, which collects the scheduling states of all machines as input for training separately and executes the scheduling policy, respectively. Popper et al. (2021) [32] proposed a distributed MARL scheduling method for the multi-objective optimization problem of minimizing energy consumption and delivery delay in the production process. The basic algorithm is solved by PPO, which regulates the joint behavior of each agent through a common reward function. The algorithm can schedule any

number of operations. Burggräf et al. (2022) [7] presented a deep MARL approach with distributed actor-critic architecture to solve dynamic FJSPs. The novelty of this work lies in parameterizing the state and action spaces. Zhang et al. (2022) [8] constructed a multi-agent manufacturing system (MAMS) with the capability of online scheduling and policy optimization for the dynamic FJSP. Various machine tools are modeled as agents capable of environmental perception, information sharing, and autonomous decision making.

2.2.2. MARL with CTP

Wang et al. (2021) [33] proposed a flexible and hybrid production scheduling problem and introduced a multi-agent deep reinforcement learning (MADRL) scheduling method called Multi-Agent Deep Deterministic Policy Gradient (MADDPG). Similarly, Wang et al. (2022) [9] introduced a modeling method of the decentralized partially Markov decision process (Dec-POMDP) for the resource preemption working environment (PRE), and applied QMIX to solve the PRE scheduling problem, where each job is an agent who selects its action according to current observation. Jing (2022) [10] designed a MARL scheduling framework based on GCN, namely a graph-based multi-agent system (GMAS), to solve the FJSP. First, a probabilistic model of the directed acyclic graph of the FJSP is constructed from the product processing network and workshop environment. Then, the author modeled the FJSP as a topological graph prediction process and adjusted the scheduling policy by predicting the connection probabilities between the edges.

In contrast to DTP, CTP shares information among agents through a centralized evaluation function, making learning more stable and leading to fast convergence. Therefore, regarding the solving method for FJSP-DT, MARL integrated with the CTP training paradigm bears the great potential of agility, adaptability, and accuracy, and is the focus of study therewith.

3. Problem Description and Model Formulation

This section describes the FJSP-DT problem and its modeling as a decentralized partially observable Markov decision process.

3.1. Problem Description

There are $|M|$ robots in the FJSP-DT workshop and $|S|$ stations. In the workshop, a total of $|N|$ jobs need to be processed and job n_i has $|O_i|$ operations. The relationships between two operations may be sequential, parallel, or conflicted. The processing time t_{jk} of the operation o_{ij} depends on the robot m_k. There exists a subset of robots M_j to satisfy the processing requirements of the operation o_{ij}. Each robot covers a subset of stations. Multiple robots can share the same station. The transportation time τ_{pq} from the station s_p to the station s_q for the next operation needs to be considered. The notations formulating the problem are given in Table 2.

Table 2. Notations.

Symbol	Definition
N	A set of $\|N\|$ jobs, $N = \{n_1, n_2, \ldots, n_{\|N\|}\}$.
n_i	The ith job in the set N.
O	A set of all jobs' operations, $O = \{o_1, o_2, \ldots, o_{\|O\|}\}$.
S	A set of all stations for processing, $S = \{s_1, s_2, \ldots, s_{\|S\|}\}$.
s_k	The kth station in the set S.
M	A set of $\|M\|$ robots, $M = \{m_1, m_2, \ldots, m_{\|M\|}\}$.
m_k	The kth robot in the set M.
O_i	A set of operations for job n_i, $O_i = \{o_{i1}, o_{i2}, \ldots, o_{\|O_i\|}\}$.
o_{ij}	The jth operation of job n_i.
M_j	A set of robots that can support operation o_j, $M_j = \{m_{j1}, m_{j2}, \ldots, m_{\|M_j\|}\}$.
m_{jk}	The kth robot that can support operation o_j.
M_l	A set of robots that the station s_l can cover.
m_{lk}	The kth robot that the station s_l can cover.
S_k	A set of stations that the robot m_k can reach.
s_{kl}	The lth station that the robot m_k can reach.

There are some assumptions as below.

- All jobs are available after release dates.
- A robot can only process one job at a time.
- Each operation can only be processed on one robot at a time.
- Each operation cannot be interrupted during processing.
- There is no buffer in the station, and each station can only accommodate one job at a time.

3.2. Dec-POMDP Framework

FJSP-DT can be regarded as a cooperative multi-agent Markov game due to the jobs' cooperative scheduling objective. A fully cooperative multi-agent task can be formulated as a decentralized partially observable Markov decision process (Dec-POMDP), described by a tuple $\langle \mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{P}, R, \mathcal{O}, \Omega, \gamma \rangle$, where:

- $\mathcal{N} = \{1, \ldots, n\}$ is the set of $n > 1$ agents;
- $\mathcal{S}$ is a set of states denoting all joint states possible by the multi-agents;
- $\mathcal{A} = \times_{i \in \mathcal{N}} \mathcal{A}^i$ is the set of joint actions, where $\mathcal{A}^i$ denotes the set of actions for agent i;
- $\mathcal{P}$ is the transition probability function;
- $R = R(s, a)$ is the reward function, mapping states and joint actions to real numbers;
- $\mathcal{O} = \times_{i \in \mathcal{N}} \mathcal{O}^i$ is the set of joint observations, where $\mathcal{O}^i$ is the set of observations available to agent i;
- Ω is the observation probability function;
- $\gamma \in [0, 1)$ is a discount factor.

The agents and environment interact at each time step. $t = 0, 1, 2, \cdots$. $s \in \mathcal{S}$ describes the true state of the environment, and the state at time step t is denoted by s_t. The action for agent i is denoted by $a^i \in \mathcal{A}^i$. At every time step t, each agent i takes an action a^i_t, leading to one joint action $a = \left[a^1, a^2, \ldots, a^n\right]^T, a \in \mathcal{A}$ at every time step. In the fully cooperative setting, all agents share the same reward function $R^1 = R^2 = \cdots = R^n = R(s, a)$. The state transition function $\mathcal{P} = P(s'|s, a) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ describes the dynamics of the environment. The observation function $\Omega(s, a) : \mathcal{S} \times \mathcal{A} \to \mathcal{O}$ specifies the probabilities $p(o|a, s')$ of joint observations. Due to the partial observability, each agent only draws individual observations $o^i \in \mathcal{O}^i$, defined by the observation function $\Omega(s, a)$. The observations for agent i at time step t are denoted as o^i_t. At every time step, the environment emits one joint observation $o = \left[o^1, \ldots, o^n\right]^T, o \in \mathcal{O}$. The number of joint observations is $|\mathcal{O}| = |\mathcal{O}^1 \times \mathcal{O}^2 \times \cdots \times \mathcal{O}^n| = \Pi_i |\mathcal{O}^i|$, and the number of joint actions is $|\mathcal{A}| = |\mathcal{A}^1 \times \mathcal{A}^2 \times \cdots \times \mathcal{A}^n| = \Pi_i |\mathcal{A}^i|$. And then, both state and action explosions occur.

Each agent has an action–observation history $\tau^i \in \mathcal{H} \equiv (\mathcal{O} \times \mathcal{A})^*$. At time step t, the action–observation history for agent i is denoted by $\tau^i_t = [a^i_0, o^i_1, \ldots, a^i_{t-1}, o^i_t]^T$. A joint action–observation history $\tau = \left[\tau^1, \ldots, \tau^n\right]^T$ denotes the action–observation histories of all agents. We refer to the state, action, observation, reward, and action–observation variables at time step t as S_t, A_t, O_t, R_t, and H_t, respectively.

The stochastic policy for agent i is denoted by $\pi^i(a^i|\tau^i) : \mathcal{H} \times \mathcal{A} \to [0, 1]$, then a multi-agent policy will be denoted by $\pi(a|\tau) = (\pi^1, \ldots, \pi^n)$.

A centralized training and decentralized execution paradigm [25] is adopted in our learning algorithm. Each job is taken as an agent. During training, each agent can access the environment state s and all agent's observations o and their histories τ. However, during execution, each agent can only access its own observations o^i and action–observation history τ^i to determine its action according to its policy. The modeling for FJSP-DT under the Dec-POMDP framework will be detailed in Sections 3.3–3.5.

3.3. State and Observations

The states and observations, including four 4 factors: job, operation, station, and robot, are high-dimension data. Hence, graph-embedding technology is applied to represent states and observations. Then, GCN is applied to compress states and observation

dimensions, since it is effective in reducing the computational cost of large and sparse embedding graphs.

The graph can be defined as $G = \{V, E, C, X\}$ with $|V| = |N| + |O| + |S| + |M|$ nodes and $|E|$ edges. $v_i \in V$ denotes the ith node of the graph; $e(v_i, v_j) \in E$ denotes the edge between the node $v_i \in V$ and the node $v_j \in V$. $C \in \mathbb{R}^{|V|\times|V|}$ denotes the adjacency matrix, and the element $c_{ij} \in C$ represents the weight of the edge $e(v_i, v_j)$. The feature matrix of the nodes is represented by $X \in \mathbb{R}^{|V|\times p}$, where p denotes the dimension of the feature vector. The schematic diagram of G is shown in the lower left part of Figure 2.

The types of nodes for our problem include four categories: job, operation, station, and robot. We utilize the following form to represent the feature matrix.

$$X = \begin{bmatrix} j_{11} \cdots j_{|N|1} & m_{11} \cdots m_{|M|1} & s_{11} \cdots s_{|S|1} & o_{11} \cdots o_{|O|1} \\ \vdots \ddots \vdots & \vdots \ddots \vdots & \vdots \ddots \vdots & \vdots \ddots \vdots \\ j_{1p} \cdots j_{|N|p} & m_{1p} \cdots m_{|M|p} & s_{1p} \cdots s_{|S|p} & o_{1p} \cdots o_{|O|p} \end{bmatrix}, \tag{1}$$

where j, m, s, and o represent job, robot, station, and operation, respectively, and p is the number of features. The features of the job include the job type, the number of operations remaining, and the total remaining processing time of all unfinished operations. The features of the operation include the operation type and the processing time. The features of the station and the robot include the node type and the remaining occupancy time.

The edges between operations, robots, or stations do not change during processing. However, the edges between jobs and operations, jobs and robots, and jobs and stations change as machining progresses. Hence, the weights of those dynamic edges are updated during machining, which reflects the current state of the availability of robots and stations, each job's location, and the machining progress.

The adjacency matrix C, combining the feature matrix X of the embedding graph, is of high dimension, and introduces difficulty for network parameter optimization therewith. Hence, GCN [34] is adopted to extract the structural features of the graph which is aggregated with node features to generate new node representations. Then, those new representations can be input into the network for learning. This GCN-based approach can not only improve the representation ability of node features but also compress the network's parameter size and speed up optimization.

A GCN layer can be described as

$$H^{(h+1)} = \rho\left(\tilde{D}^{-\frac{1}{2}}\tilde{C}\tilde{D}^{-\frac{1}{2}}H^{(h)}W^{(h)}\right), \tag{2}$$

where $\tilde{C} = C + I_{|V|}$ is the self-ring adjacency matrix of the graph G, and I is an identity matrix with vertices. The element $\tilde{d}_{i,i}$ in the matrix $\tilde{D}$ is equal to $\sum_j \tilde{c}_{i,j}$. $H^{(h)}$ is the input matrix of the h layer of the GCN, and the input of the first layer ($h = 0$) is the feature matrix X of the nodes. $W^{(h)}$ is a learnable parameter matrix in the hth layer.

Equation (2) represents the multi-layer GCN. In this equation, we take the normalized self-ring adjacency matrix $\tilde{D}^{-\frac{1}{2}}\tilde{C}\tilde{D}^{-\frac{1}{2}}$ as the operator of the feature matrix. Therefore, the graph embedding matrix is denoted as

$$\tilde{X} = \tilde{D}^{-\frac{1}{2}}\tilde{C}\tilde{D}^{-\frac{1}{2}}X, \tag{3}$$

where X is the feature matrix of the nodes, and $\tilde{X}$ is the embedding graph matrix as input of the learning model.

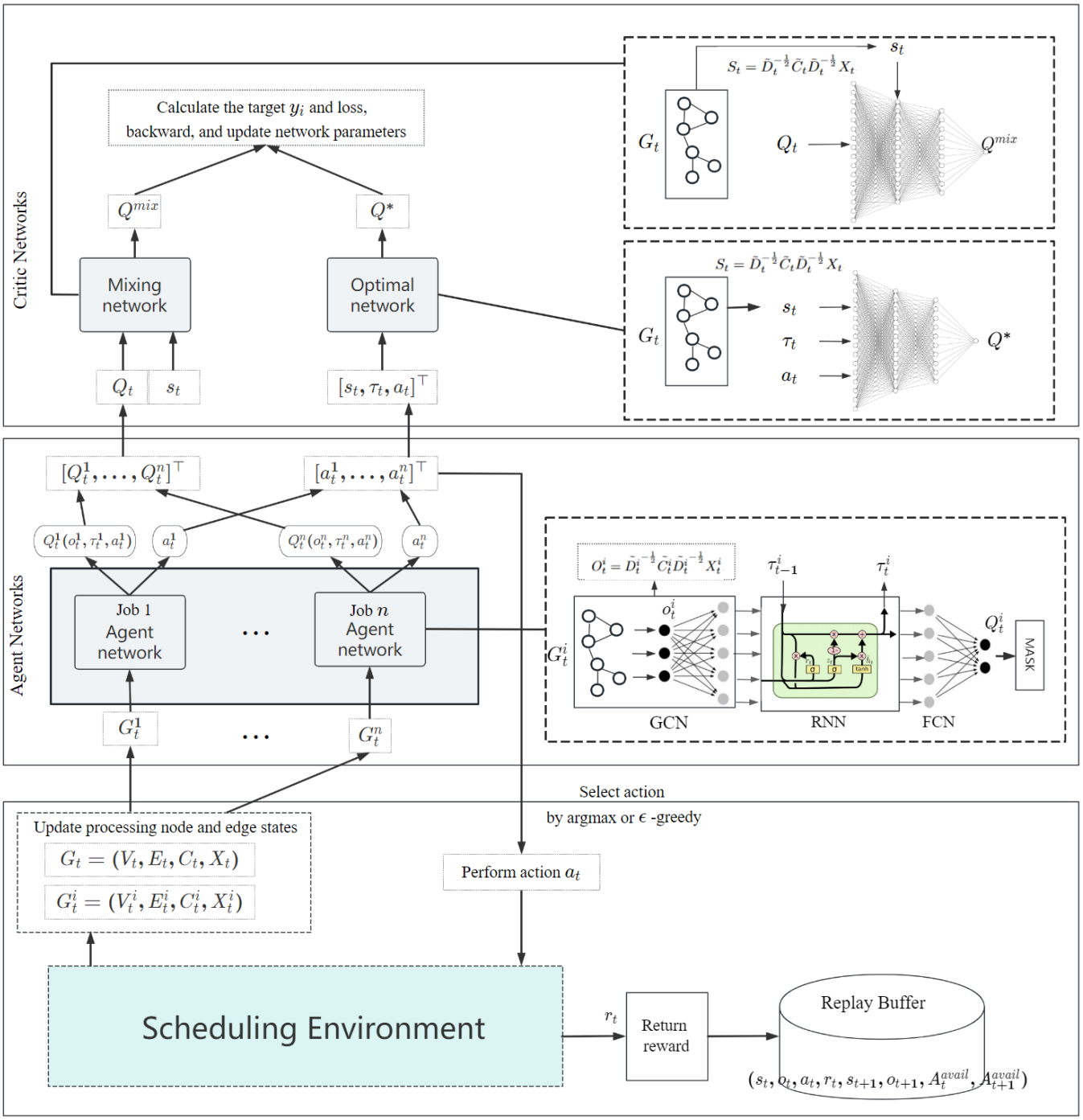


Figure 2. The scheduling algorithm architecture of solving the FJSP-DT using the DQMIX.

3.3.1. State

The state includes information about all jobs. According to Equations (1) and (3), at time step t, state S_t can be represented by the following form,

$$S_t = \tilde{D}_t^{-\frac{1}{2}} \tilde{C}_t \tilde{D}_t^{-\frac{1}{2}} X_t, \tag{4}$$

which is a matrix that can be flattened into a vector before inputting into the network. The state is utilized during algorithm training.

3.3.2. Observations

In Dec-POMDP, each agent cannot fully observe other agents' information. Thus, each agent's original observations are defined as a subgraph $G^i = \{V^i, E^i, C^i, X^i\}$, where we exclude other jobs' nodes and edges except job n_i. Its feature matrix can be rewritten to

$$X^i = \begin{bmatrix} j_{i1} & m_{11} & \dots & m_{|M|1} & s_{11} & \dots & s_{|S|1} & o_{11} & \dots & o_{|O|1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ j_{ip} & m_{1p} & \dots & m_{|M|p} & s_{1p} & \dots & s_{|S|p} & o_{1p} & \dots & o_{|O|p} \end{bmatrix}. \tag{5}$$

Then, the transformed observation matrix of agent i can be represented as

$$O_t^i = \tilde{D}_t^{i\,-\frac{1}{2}} \tilde{C}_t^i \tilde{D}_t^{i\,-\frac{1}{2}} X_t^i. \tag{6}$$

The observation defined by (6) is required during algorithm execution besides the training period.

3.4. Actions

At each decision point, the agent must choose a station and a robot as an action, called station–robot pair. Each agent needs a matrix to represent its action space, and the joint action space of multiple agents should be represented with a three-dimensional tensor. This high-dimensional action space will cause difficulties in learning policies.

Some mechanistic constraints are introduced to reduce the action space. In general, a robot can only reach some of the stations, so the number of accessible station–robot pairs is cut down. For example, if the workshop has 20 robots and 20 stations, and a robot can only reach 4 stations, then the number of station–robot pairs can be reduced to 80 (20×4), far less than 400 (20×20). Furthermore, fewer feasible station–robot pairs satisfy the constraints of operation precedence and the availability of robots and stations in the machining process. Those constraints will further reduce the number of alternative actions during model training.

Therefore, we adopt mechanic constraints to pick out feasible station–robot pairs as each agent's action space. Suppose that the number of possible combinations is K; then, the action set is represented as

$$\mathcal{A}^i = \{a_1, a_2, \cdots, a_K, a_{K+1}, a_{K+2}, a_{K+3}, a_{K+4}\} \tag{7}$$

See Table 3 for the details. At time step t, the possible actions for agent i are denoted $\mathcal{A}_t^i \subseteq \mathcal{A}^i$.

Table 3. The description of actions.

Action	Description
$a_i (i \leq K)$	Specifying a station and a robot are available for the agent to perform the next operation.
a_{K+1}	Performing the transportation action. If the current station cannot support the next operation, the agent must move.
a_{K+2}	Staying on processing. The agent must not be interrupted to carry out other things.
a_{K+3}	Taking a wait action. The agent performs nothing.
a_{K+4}	Representing stop. At this point, the agent has completed all operations.

3.5. Reward

The reward function guides the agent to minimize C_{max} or maximize $-C_{max}$, or $c - C_{max}$, where C_{max} denotes the maximum completion time of all jobs, which determines the makespan, c is a constant. However, the scheduling in FJSP-DT is a sparse reward task, and we cannot accurately compute the actual makespan C_{max} until all jobs' tasks are completed. Thus, the agents cannot receive real rewards at each decision epoch. This brings

forward learning challenges. To address this challenge, we estimate the makespan (EC_{max}) at each step that is utilized to evaluate the quality of each agent's action [35].

$$EC_{max} = \max\left(EC_j^i\right)_{1\le i\le |N|, 1\le j\le |O|}, \tag{8}$$

where EC_j^i denotes the estimated completion time of the operation $o_{i,j}$.

We employ $EC_{max}(t)$ to symbolize the EC_{max} at time step t. The immediate reward function is designed as a function of reduction in EC_{max}.

$$r_t(s,a) = EC_{max}(t) - EC_{max}(t+1). \tag{9}$$

At $t = 0$, a constant c is set as an initial value, namely $EC_{max}^0 = c$. Maximizing the immediate reward function (9) at each time step is equivalent to maximizing the accumulated reward $c - C_{max}$.

The estimated completion time of a certain operation is equal to the processing time of that operation plus the estimated completion time of its preceding operation, which is represented as follows:

$$EC_j^i = EC_{j-1}^i + t_j, \tag{10}$$

where EC_{j-1}^i denotes the estimated completion time of the operation $o_{i,j-1}$, which is the previous operation of the current operation o_{ij}. The variable t_j denotes the processing time of the current operation o_{ij}. We estimate t_j by averaging a set of processing times $\{t_{j1}, t_{j2}, \ldots, t_{jk}\}$, where $1, \ldots, k$ are indices of the robots that can support the operation o_j.

If C_{max} exceeds C_{up} which is the upper bound set by the human planner, the solution is poor, and a negative reward should be given. On the contrary, C_{max} being less than C_{up} indicates the solution is acceptable, and a positive reward should be given. Therefore, the modified reward function is represented as

$$r_t(s,a) = \begin{cases} EC_{max}(t) - EC_{max}(t+1) & \text{if } t < |T| \text{ and } EC_{max}(t) \le C_{up}, \\ \left(C_{up} - C_{max}\right) \times \alpha & \text{if } t = |T| \text{ and } EC_{max}(t) \le C_{up}, \\ -Z & \text{if } EC_{max}(t) > C_{up}. \end{cases} \tag{11}$$

where $T = \{1, 2, \cdots, C_{up}\}$, $Z > 0$, $\alpha > 0$, $\left(C_{up} - C_{max}\right) \times \alpha$ denotes a non-negative reward between the upper bound of the makespan and C_{max}. It encourages the agents to find good policies to reduce C_{max} as much as possible.

4. Algorithm

This section describes the DQMIX algorithm optimizing the above Dec-POMDP model. We expound it from four aspects: algorithm overview, agent network, critic network, and loss function.

4.1. Algorithm Overview

DQMIX addresses the challenges of FJSP-DT scheduling through three designs.

1. We utilize a learning algorithm to satisfy the real-time and adaptive requirements of FJSP-DT scheduling. The scheduling procedure of FJSP-DT is modeled as a Dec-POMDP model, and a real-time scheduling policy that can adapt to an uncertain environment is obtained through reinforcement learning.
2. Regarding accuracy, we propose double critic networks to assist agent training, enhancing the stability and quality of learning. In addition, a MASK layer is added in the DQMIX output, which outputs a set of feasible actions based on the current state and conflicted relationships, ensuring that the output actions are accurate.
3. The MARL architecture is adopted to reduce the dimensionality of action space. Each job agent only needs to choose actions based on their own observations, and

each agent model only needs to output a $1 \times |\mathcal{A}^i|$ dimensional vector instead of $n \times |\mathcal{A}^i|$ dimensional vectors. The GCN network layer is designed to compress the number of input parameters from $|V| \times |V| \times p$ to $|V| \times p$, which further alleviates the dimensionality curse.

As shown in Figure 2, the DQMIX scheduling framework includes the FJSP-DT workshop environment and MARL networks.

The FJSP-DT workshop environment is depicted in Figure 1, consisting of stations, robots, and jobs. We view the workshop state as a graph, with stations, robots, jobs, and operations as the nodes of the graph, and the state of the nodes and edges representing the processing state. We treat the job as an agent, and it determines the station and robot required for each job.

The MARL network comprises n agent networks and two critic networks, namely the mixing network and the optimal network.

The agent networks of DQMIX refer to the agents' policy models. The critic networks provide global guidance for the collaboration of the agents. It assists in optimizing the parameters of the agent networks during training.

The learning process of DQMIX is described as follows.

- Firstly, an environment instance is generated according to FJSP-DT.
- Secondly, starting at time $t = 0$, each agent chooses an action a_t^i based on its own observation o_t^i and its own policy. The actions of all agents form the joint action vector a_t.
- Thirdly, according to the processing state s_t and scheduling action a_t at time t, the environment updates the processing state s_{t+1} and observations o_{t+1} at time $t+1$. The environment outputs a reward r according to the reward function, and stores the tuple $(s_t, o_t, a_t, s_{t+1}, o_{t+1})$ as a POMDP instance in the experience memory.
- Next, the agents continue to interact with the environment until all jobs' tasks are completed, or the makespan exceeds C_{up}.
- Finally, when the scheduling ends, a complete POMDP chain is formed, and the environment gives the final reward r.

The agents repeat the above steps to obtain samples and update network parameters.

4.2. Agent Network

For each agent, there is one agent network that represents its action-value function $Q^i(\tau^i, a^i)$. The agent network includes a graph convolutional network (GCN) layer, a recurrent neural network (RNN) layer, a fully connected network (FCN) layer, and a MASK layer.

The GCN layer aggregates the state features of the job graph and adjacency matrix information into a transformed state matrix.

The RNN network receives the GCN output and the action–observation history τ_{t-1}^i at time step t and outputs a hidden vector.

The FCN layer takes the hidden vector as input and outputs an action-value vector $Q_t^i(o_t^i, \tau_t^i, \cdot)$ of dimension $|\mathcal{A}^i|$.

The MASK layer is applied to constrain the agent network output to a valid action vector. The valid action vectors $\mathcal{A}^{i,avail}$ for agent i at each step are given by the scheduling environment, and they are represented as a vector consisting of 0 and 1, $\mathcal{A}^{i,avail} = [\cdots, 0, \cdots, 1, \cdots]^T$, where 1 denotes the available action and 0 denotes the unavailable action. Then, the action value vector output by the MASK layer can be represented as

$$Q_t^{i,avail}(o_t^i, \tau_t^i, \cdot) = Q_t^i(o_t^i, \tau_t^i, \cdot) + (1 - \mathcal{A}^{i,avail}) \times M, \tag{12}$$

where M is a large negative number (e.g., $M = -1 \times 10^8$).

All agent networks share parameters for speeding up networks' training.

Each agent utilizes the agent network to choose its action at each decision point by estimating action values for a given observation.

The action values chosen from the agent network outputs also are taken as input for the mixing network. The joint action is calculated by the argmax operator and the joint hidden feature vector τ is calculated by the sum of agent networks' hidden features. The joint action and the joint hidden feature vector are taken as the input of the optimal network.

4.3. Critic Network

DQMIX integrates two critic networks: the mixing network and the optimal network, improving the fitting ability of QMIX [36].

4.3.1. Motivation of Proposing Double Critic Networks

QMIX is a MARL algorithm proposed in 2018 [36]. Although widely used, it may not achieve optimal solutions in solving non-monotonic problems due to its structural constraint of monotonicity [37]. To avoid the limitations of QMIX, QTRAN [37] introduces a joint action-value function to replace the mixing function in QMIX, which improves the fitting performance. However, it may not perform well in some complex multi-agent tasks [38]. WQMIX [39] adds an optimal value function to narrow the bias caused by QMIX, but the optimal value function in WQMIX contains a hyperparameter, which increases the learning uncertainty during training.

DQMIX integrates the monotonic function Q^{mix} while introducing a new joint action-value function Q^* without any restrictions. Q^* is an optimal joint action-value function of the state, the action–observation history, and the joint action. Q^{mix} is a mixing value function, the same as the monotonic value function in QMIX. They are represented as follows:

$$Q^{mix} = f(s, [Q^i]), \tag{13}$$

$$Q^* = f(s, \tau, a). \tag{14}$$

where τ represents the joint action–observation history and $[Q^i]$ represents the agent utilities.

The above design mainly lies in two considerations.

1. The non-negative monotonicity of the mixing value function should be kept when the optimal policy has been recovered, consistent with the goal pursued in cooperative games. Each agent's marginal return with the optimal policy is non-negative, i.e.,

$$\frac{\partial Q^{mix}(s, [Q^i])}{\partial Q^i(\tau^i, a^i)} \geq 0. \tag{15}$$

2. To overcome the limitations of Q^{mix} when the optimal policy has not yet been recovered, a joint action-value function $Q^*(s, \tau, a)$ is introduced. $Q^*(s, \tau, a)$ is a function of state s, action–observation history τ and joint action a. It reduces the dependencies on agents' utilities.

According to the above analysis, we introduce Theorem 1 [37,40] to improve QMIX and reformulate the decomposition condition of the joint action-value function as the value decomposition theorem (VDT).

The VDT says if and only if there is a joint action-value function $Q^{*\prime}(s, \tau, a)$ satisfying the conditions (16)–(19), the real action-value function $Q^*(s, \tau, a)$ can be decomposed into a set of individual action-value functions $\{Q^i\}_{i \in N}$.

$$Q^{mix}(s, \tau, \hat{a}) = Q^*(s, \tau, \hat{a}), \tag{16}$$

$$Q^{mix}(s, \tau, a) \geq Q^*(s, \tau, a), \tag{17}$$

$$\frac{\partial Q^{mix}(s, \tau, a)}{\partial Q^i(\tau_i, a_i)} \geq 0, \forall i \in N, \tag{18}$$

$$\hat{a} = \left[\underset{a_i}{\operatorname{argmax}} Q^i(\tau_i, a_i) \right]_{i \in N}. \tag{19}$$

The literature [40] provides detailed proof. The following conditions can be deduced by substitution based on the above conditions:

$$Q^*(s,\tau,\hat{a}) = Q^{mix}(s,\tau,\hat{a}) \geq Q^{mix}(s,\tau,a) \geq Q^*(s,\tau,a). \tag{20}$$

Equation (20) satisfies the individual global maximization (IGM) principle [37].

4.3.2. Mixing Network

The mixing network is a feedforward neural network that takes the agent networks' outputs as input and mixes them monotonically. To enforce the monotonicity constraint of (15), the weights (but not the biases) of the mixing network are restricted to be non-negative. The non-negative weights are produced by a set of hypernetworks [41] with the state as input.

4.3.3. Optimal Network

The true joint action-value function is approximated by the optimal network Q^*. The optimal network is not restricted to be monotonic by using non-negative weights. Therefore, we can simplify the architecture by inputting agents' observations (represented as a hidden feature vector τ), the joint action a_t, and the state (if necessary) s_t into the network.

We construct the network above to improve scalability and sample efficiency as follows. Since the joint action space is $\times_{i\in\mathcal{N}}\mathcal{A}^i$, it requires high complexity to find an optimal action as the number of agents n grows. The optimal joint action is sampled from all the agent networks by decentralized policies with linear-time individual argmax operations. The optimal network shares parameters from the agent networks, which enhances good sample efficiency.

4.4. Loss Function

We design the loss functions according to VDT as below.

$$\mathcal{L}(s,\tau,a,r,s',\tau';\theta) = \mathcal{L}_{star} + \lambda_{opt}\mathcal{L}_{opt} + \lambda_{nopt}\mathcal{L}_{nopt}. \tag{21}$$

$$\mathcal{L}_{star}(;\theta) = (Q^*(s,\tau,a) - y)^2. \tag{22}$$

$$\mathcal{L}_{opt}(;\theta) = \Big(Q^*(s,\tau,\hat{a}) - Q^{mix}(s,\tau,\hat{a})\Big)^2. \tag{23}$$

$$\mathcal{L}_{nopt}(;\theta) = \begin{cases} \Big(Q^{mix}(s,\tau,a) - y\Big)^2 & \text{if } c_1, \quad (24a) \\ \Big(Q^{mix}(s,\tau,a) - Q^{clip}(s,\tau,a)\Big)^2 & \text{if } c_2, \quad (24b) \\ \Big(Q^{mix}(s,\tau,a) - Q^*(s,\tau,a)\Big)^2 & \text{if } c_3, \quad (24c) \end{cases}$$

where Q^* denotes the output of the optimal network; Q^{mix} denotes the output of the mixing network; Q^{clip} denotes a clip function; a denotes the joint action vector performed by the agents; $\hat{a}$ denotes the optimal joint action vector calculated by the argmax operator with the maximum value of the agent networks; c_1 denotes $y > Q^*(s,\tau,a)$; c_2 denotes $y \leq Q^*(s,\tau,a)$ and $Q^*(s,\tau,a) < Q^*(s,\tau,\hat{a})$; c_3 denotes $y \leq Q^*(s,\tau,a)$ and $Q^*(s,\tau,a) \geq Q^*(s,\tau,\hat{a})$.

$$y = r + \gamma Q^*(s',\tau',\hat{a}';\theta^-), \tag{25}$$

$$\hat{a} = \left[\underset{a_i}{\operatorname{argmax}}\, Q^i(\tau^i,a^i;\theta)\right]_{i=1}^{n}, \tag{26}$$

$$\hat{a}' = \left[\underset{a^{i\prime}}{\operatorname{argmax}}\, Q^i(\tau^{i\prime},a^{i\prime};\theta^-)\right]_{i=1}^{n}, \tag{27}$$

where y denotes the output of the target network of the optimal network, θ^- denotes the parameter of the target network, and $\hat{a}'$ is the optimal actions maximizing the agent network output $Q^i(\tau^{i\prime}, a^{i\prime})$ for $i \in N$.

$$Q^{clip}(s,\tau,a) = \text{clip}\Big(Q^{mix}(s,\tau,a), Q^*(s,\tau,a), Q^*(s,\tau,\hat{a})\Big). \tag{28}$$

Equation (28) is used to enforce the output of the mixing network between the evaluation action value $Q^*(s,\tau,a)$ and the optimal action value $Q^*(s,\tau,\hat{a})$.

Equation (22) makes the optimal network maximize the reward as far as possible, and Equation (23) ensures that the optimal network and the mixing network are optimal at the same time. The equations in (24) make two networks track each other when performing non-optimal action. If the joint action value is underestimated, Equation (24a) makes the mixing network track the true value. If the joint action value is overestimated, Equations (24b) and (24c) keep the gap between the mixing network output and the optimal network output to the extent that satisfies the necessary conditions of VDT.

The training scheme consists of two loops: the outer loop is used to complete sufficient training steps, while the inner loop is used to complete an MDP instance, i.e., an episode. The details are presented in Algorithm 1.

Algorithm 1 DQMIX Algorithm.

Initialize network parameters θ, learning rate α, discount rate γ, exploration rate ϵ, replay buffer D, batch size b, the max number of episodes M, the current episode $e = 0$, the max number of steps in one episode C_{up}, the number of jobs n. Create environment model E.
while $e < M$ **do**
  Reset environment E, initialize state s_t, observations $o_0 = \{o_0^i\}_{i=1}^n$, action–observation history $\tau_{-1} = \{\tau_{-1}^i\}_{i=1}^n$, available action set $\mathcal{A}_0^{avail} = \{\mathcal{A}_0^{i,avail}\}_{i=1}^n$, episode data d, the current step $t = 0$; done = False.
  while $t < C_{up}$ **do**
    Receive the state s_t and observations o_t available actions $\mathcal{A}_t^{avail}$ from E, set the agent $i = 0$.
    while $i < n$ **do**
      Choose action by

$$a_t^i = \begin{cases} \text{randomly choose an action from } \mathcal{A}_t^{i,avail} & \text{if random()} < \epsilon; \\ \text{argmax}\, Q^{i,avail}(o_t^i, \tau_{t-1}^i, \cdot) & \text{otherwise;} \end{cases}$$

      put a_t^i to action vector a_t, $i \leftarrow i+1$
    end while
    Execute actions a_t in E to get $r, s_{t+1}, o_{t+1}, done$;
    Update available action set $\mathcal{A}_{t+1}^{avail}$;
    Store $(s_t, o_t, a_t, \mathcal{A}_t^{avail}, r, s_{t+1}, o_{t+1}, \mathcal{A}_{t+1}^{avail}, done)$ to d;
    if done **then**
      break
    end if
    $s_t \leftarrow s_{t+1}, o_t \leftarrow o_{t+1}, \mathcal{A}_t^{avail} \leftarrow \mathcal{A}_{t+1}^{avail}, t \leftarrow t+1$
  end while
  Store episode data d to replay buffer D;
  if the size of D more than the least size **then**
    Sample a batch of b episodes from D;
    Compute $[Q^i], y, Q^*, Q^{mix}$;
    Update network parameters θ by minimizing loss with loss function (21).
  end if
  $e \leftarrow e+1$
end while

5. Case Study

The experiments evaluate the performance of the proposed algorithm in terms of solution quality, convergence, scalability, and generalization. The time efficiency of the experiments is not measured because the well-trained DQMIX can solve problems in real time. The ablation experiments are performed to investigate the effectiveness of GCN and the reward function. The following experiments run on the computer with the GPU model of Tesla V100. Tesla V100 is manufactured by NVIDIA Corporation, an American technology company based in Santa Clara, CA, USA.

5.1. Case Description and Algorithm Settings

5.1.1. Case Description

A typical application for the FJSP-DT is aircraft maintenance scheduling. In the FJSP-DT, we consider aircraft as jobs. Therefore, we set up four typical cases, namely N3, N6, N8, and N12, representing three, six, eight, and twelve jobs, respectively. N3 is a small case, and N6, N8, and N12 are medium-to-large-sized cases. The base settings are given in Table 4. The detailed settings of N3 are given in Figure 3 and Tables 5–7.

Table 4. Case parameters.

Case	Number of Jobs	Number of Operations	Number of Stations	Number of Robots	C_{up}	LB [1]
N3	3	12	5	5	18	13
N6	6	41	22	35	90	54
N8	8	55	22	35	100	58
N12	12	83	22	35	120	66

[1] The lower bound of the makespan.

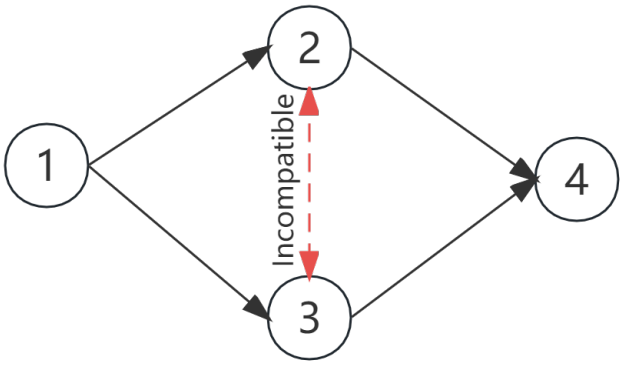


Figure 3. Process diagram.

Table 5. Alternative machines.

Operation	Alternative Robots (Processing Time)
1	M1(1), M4(1)
2	M3(5), M4(5), M5(5)
3	M1(4), M2(4)
4	M2(1), M3(1), M5(1)

Table 6. Robots to stations.

Robot	Available Stations
M1	S1, S4, S5
M2	S2
M3	S2, S3, S5
M4	S4
M5	S2, S3, S5

Table 7. Transportation time.

Station	S1	S2	S3	S4	S5
S1	0	1	1	2	2
S2	1	0	1	1	2
S3	1	1	0	1	1
S4	2	1	1	0	1
S5	2	2	1	1	0

5.1.2. Experimental Settings

To validate the effectiveness of the proposed algorithm, we selected four representative algorithms that can solve the FJSP and its extended problems, namely hGATS [42], hGAVNS [43], MGA [44], and MILP. We make adaptive modifications to the original implementation of hGATS, hGAVNS, and MGA. The initialization parameters for each algorithm are set as follows: population size is 200, crossover probability is 0.85, mutation probability is 0.15, maximum iteration is 1e4, and maximum stagnation generation is 50. Each algorithm utilizes a different random seed to run each case three times, and the average of the results from each iteration is taken as the evaluation result. The MILP model is solved using the open-source solver CBC programming. Solver CBC stands for the "constrained binary optimization" solver. It is an open-source software library used for solving optimization problems with linear and nonlinear constraints. The CBC solver is widely used in various fields such as operations research, engineering, finance, and logistics to find the optimal solution to complex problems.

To evaluate the convergence, scalability, and generalization, we utilize two well-known MARL (MARL) algorithms as baselines: centrally-weighted QMIX (CWQMIX) [39] and QTRAN [37]. CWQMIX has an unrestricted joint action-value network but takes the action values of all agents as its input, which is different from DQMIX. QTRAN also has an unrestricted joint action-value network but does not import a mixing network like DQMIX. The primary hyperparameters of the MARL algorithms are determined through several trials. For the N3 case, the algorithms converge better with a learning rate of 5×10^{-4} and a batch size of 32, but for the medium and large cases, the learning rate decreases to 5×10^{-5}, and batch size increases to 128, then the algorithms perform better. The number of training steps is 1×10^{-5} for the N3 case and 3×10^{-5} for others. The hidden layer parameters for each network are shown in Table 8. We considered the matching between the hidden and input layer dimensions and determined the settings of these parameters through multiple tests.

Table 8. Model parameters.

Network	Parameters
GCN	One layer, with the hidden dimension of 512
RNN	Two layers, with the hidden dimension of 512 in each layer.
Mixing Network	Two layers, with the hidden dimension of 512 in each layer.
Optimal Network	Three layers, with the hidden dimension of 512 in each layer.

The reward function of all MARL models is set to

$$r_t(s,a) = \begin{cases} EC_{max}(t) - EC_{max}(t+1) & \text{if } t < |T| \text{ and } EC_{max}(t) \leq C_{up}, \\ (C_{up} - C_{max}) \times 10 & \text{if } t = |T| \text{ and } EC_{max}(t) \leq C_{up}, \\ -C_{up} & \text{if } EC_{max}(t) > C_{up}. \end{cases} \quad (29)$$

where T is set to the total number of operations in the cases.

5.2. Solution Quality

The solutions of the comparative algorithms for the case of N3, N6, N8, and N12 are shown in Table 9.

Table 9. Comparison of the makespan between the comparative algorithms.

Case	DQMIX		hGATS		hGAVNS		MGA		CWQMIX		QTRAN		MILP
	BEST	MEAN	BEST	MEAN	BEST	MEAN	BEST	MEAN	BEST	MEAN	BEST	MEAN	
N3	14	14	13	13	13	13	13	14.0	14.0	17.1	14.0	16.2	13
N6	63	64.5	61	63.6	61	62	62	63.3	68.0	83.6	72.0	84.6	-
N8	68	70.3	71	71.6	67	68	70	71	73	75.7	76	78.7	-
N12	81	84.7	85	85.6	85	85.6	83	84	81	85.7	82	89.7	-

As shown in Table 9, the optimal solution for the case of N3 given by MILP is 13. hGATS, hGAVNS, and MGA obtain the optimal solution. While it does not reach the optimal solution, the DQMIX algorithm converges to a good local optimum that is very close to the optimal solution. For case N8, the performance of DQMIX in terms of makespan is better than that of hGATS and MGA (shorter makespan is better). For case N12, the solution quality of DQMIX is better than that of hGATS and hGAVNS. In other cases, the solution quality of DQMIX is comparable to that of other traditional algorithms. Compared with the MARL algorithms, the average of the solutions generated by DQMIX is superior to those of CWQMIX and QTRAN.

From the *RPD* (30) indicators (shown in Table 10), the deviation of the average solution obtained by DQMIX from the best-known solution (*BKS*) does not exceed 10%. Moreover, as the problem size increases, the deviation becomes smaller, which to some extent indicates that DQMIX is increasingly superior in solving large-scale problems.

RPD is a common indicator in workshop schedules, represented as

$$RPD = \frac{\sum_{u}^{U} (makespan_u - BKS)/BKS}{U}, \tag{30}$$

where $makespan_u$ is the makespan of the uth experiment, and U is the number of experimental runs.

Table 10. Comparison of the *RPD* for the comparative algorithms.

Case	DQMIX		hGATS		hGAVNS		MGA		CWQMIX		QTRAN		MILP
	BEST	MEAN	BEST	MEAN	BEST	MEAN	BEST	MEAN	BEST	MEAN	BEST	MEAN	
N3	7.7%	7.7%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	7.7%	31.5%	7.7%	24.5%	13
N6	3.3%	5.7%	0.0%	4.3%	0.0%	1.6%	1.6%	3.8%	11.5%	37.0%	18.0%	38.7%	61
N8	1.5%	4.9%	6.0%	6.9%	0.0%	1.5%	4.5%	6.0%	9.0%	13.0%	13.4%	17.5%	67
N12	0.0%	4.6%	4.9%	5.7%	4.9%	5.7%	2.5%	3.7%	0.0%	8.8%	1.2%	10.7%	81

Previous studies have demonstrated that when faced with large-scale problems that are beyond the capabilities of mathematical programming methods (such as MILP), intelligent search algorithms (e.g., GA) can still yield near-optimal solutions. While machine learning algorithms (e.g., reinforcement learning), exhibit strong real-time performance, their solution quality often requires enhancement. The findings from this section's experiments demonstrate that the solution quality of the DQMIX model proposed is comparable to that of advanced intelligent search algorithms (hGAVNS, hGATS, and MGA). Notably, as the problem scale expands, DQMIX's solution quality surpasses that of search-based algorithms (such as those in N12), a trend that warrants further exploration. Secondly, while DQMIX, CWQMIX, and QTRAN are all reinforcement learning algorithms, the solution quality of DQMIX surpasses that of CWQMIX and QTRAN. This superiority can be attributed to the fact that DQMIX employs a double Q-value-function optimization structure, which distinguishes it from CWQMIX and QTRAN.

5.3. Computation Time

Compared to traditional algorithms, such as genetic algorithms, machine learning models have shorter execution times and can satisfy the requirements of scenarios with high real-time requirements, such as the scenarios mentioned in this paper.

Table 11 presents the experimental data on training duration and execution duration of DQMIX model and three intelligent search algorithms (MGA, hGAVNS, hGATS) in the N8 case. According to the experimental results, the execution duration of DQMIX is in seconds, while intelligent search algorithms require more than half an hour, and hGAVNS even nearly an hour. Despite the longer training time required by DQMIX, this is somewhat compensated by the adaptability of DQMIX (see Section 5.6). A well-trained DQMIX model can adapt to changes in dynamic environments to some extent, thus meeting the needs of real-time scheduling in dynamic environments. In contrast, search algorithms must

recalculate for any instance, with the same training time and execution time, being unable to satisfy the real-time requirements under the dynamic environment.

Table 11. Training and execution time of each model in N8 case.

Model	Training Duration	Execution Duration
DQMIX	5.0 h 47.08 min	4.59 s
hGATS	47.77 min	47.77 min
hGAVNS	51.58 min	51.58 min
MGA	32.86 min	32.86 min

5.4. Convergence

Convergence analysis is mainly applicable to the learning algorithms DQMIX, CWQMIX, and QTRAN.

As shown in Figure 4, the three algorithms show good convergence in the case of N3. However, while QTRAN shows quick convergence, DQMIX outperforms CWQMIX and QTRAN in terms of convergence along the tail. In the case of N6, DQMIX demonstrates faster convergence compared to the other two algorithms, resulting in a shorter test makespan than QTRAN and CWQMIX. These results indicate that DQMIX is superior to the other two algorithms in terms of convergence.

5.5. Scalability

Scalability refers to the ability of models to maintain performance as the number of agents increases. The experimental results are presented in Figure 4c,d.

When the number of agents increases from 3 and 6 to 8 and 12, DQMIX exhibits stability, while QTRAN and CWQMIX show reduced stability. Additionally, DQMIX consistently produces superior solutions compared to the other two algorithms, as shown in Table 9. These findings suggest that DQMIX is more scalable and reliable, while also giving better solutions than the other MARL algorithms across varying numbers of agents.

Our ablation experiments reveal that DQMIX's particular network benefits the learning policies. The optimal network and the mixing network of CWQMIX takes the individual action values as input, which may restrict the networks' fitting ability and thus converge to local optima. On the other hand, QTRAIN does not have a mixing network, which may cause the network to be unstable. Therefore, compared to the first two algorithms, DQMIX maintains the mixing network and takes state and action as inputs for the optimal network, resulting in better solution quality and stability.

5.6. Generalization

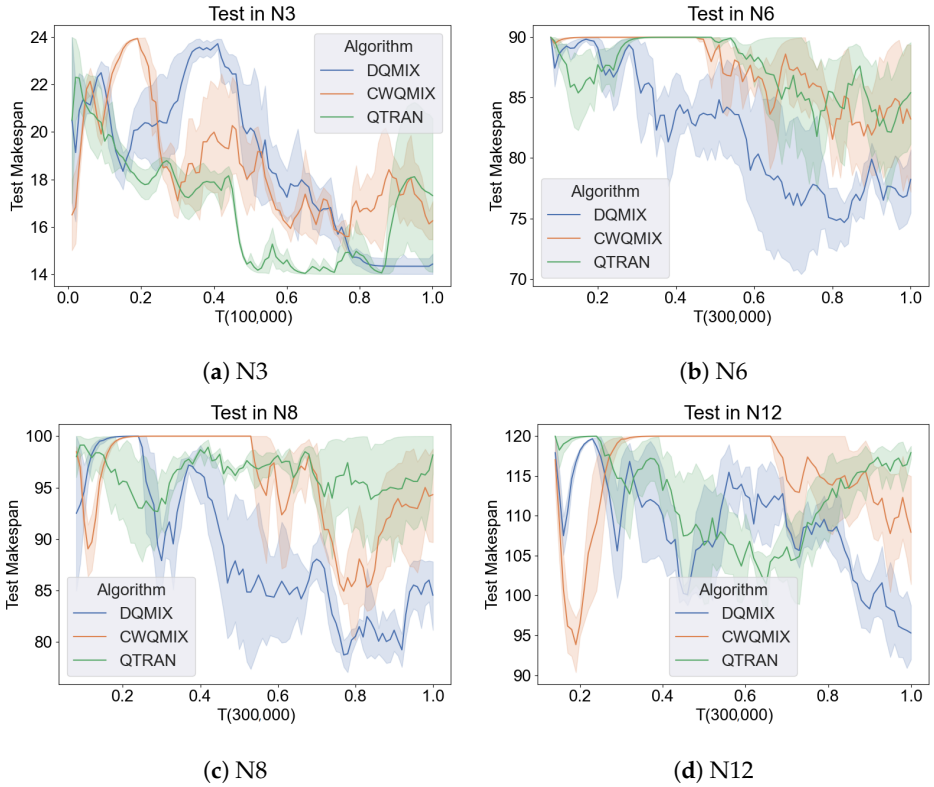
During the generalization test, the algorithms that are well trained in a static environment are evaluated in a dynamic environment. The test scenarios incorporate factors such as robot breakdowns, variations in the number of jobs, and changes in operations. The well-trained models from the N8 case are employed as the test models.

5.6.1. Generalization to Robot Breakdowns

To simulate robot breakdowns, a simulator is deployed where the probability of robot failure follows the exponential distribution with parameter λ. In this study, we simulate 150 instances of robot breakdowns. The test solution is computed as an average of 150 solutions.

Table 12 presents the results of the robot breakdown generalization test for the three algorithms. Overall, the results show that all three algorithms have good generalization under robot breakdowns. The average makespan of DQMIX is significantly lower than that of the other two algorithms, which suggests that DQMIX's solution quality is superior. However, the solving success rate of DQMIX is slightly lower than that of CWQMIX. This is because some solutions generated by DQMIX exceed the upper bound of the makespan

set by the worker. In terms of stability, the MIR of QTRAN is the smallest, and it seems that QTRAN is the most stable. However, when we take into account the deviation of the makespan from the optimal solution, as captured by the $NMIR$ metric, we observe that DQMIX performs better. In fact, DQMIX has both higher solution quality and greater stability than QTRAN when we consider $NMIR$.



(**a**) N3 (**b**) N6

(**c**) N8 (**d**) N12

Figure 4. Test performance of the MARL algorithms. T in the x-axis represents the number of training time steps, the same in the following.

Table 12. Test with robot breakdowns.

Items	**DQMIX**	**CWQMIX**	**QTRAN**
StaticMakespan [1]	69	73	76
Number of instances [2]	150	150	150
Makespan mean [3]	73.0	78.5	79.3
Makespan variance	6.0	3.7	3.6
MIR	5.7%	7.5%	4.4%
$NMIR$	5.7%	10.2%	7.2%
Success rate	99.3%	100%	98.7%

[1] The makespan solved in the static environment. [2] Cases produced by random seeds on the same exponential distribution. [3] The makespans solved under robot breakdowns.

MIR (makespan increment rate) is to measure the dynamic scheduling performance [10], as shown in Equation (31).

$$MIR = \frac{DynamicMakespan - StaticMakespan}{StaticMakespan}. \tag{31}$$

$NMIR$ (normalized MIR), represented as Equation (32), reflects the deviation of the makespan solved under the dynamic environment from the best-known makespan.

$$NMIR_i = MIR \times \frac{StaticMakespan_i - LB}{BestStaticMakespan - LB}. \tag{32}$$

where *BestStaticMakespan* is the best-known makespan solved by algorithms in a static scheduling environment. *LB* is the lower bound of the solution, which may be lower than the optimal solution.

5.6.2. Generalization to Varied Quantity of Jobs

This test investigates the algorithms' generalization to varying job quantities. The experiments are performed on the job set of $\{4, 5, 6, 7, 8\}$, and the results are shown in Table 13.

Table 13. Test with varied job quantity.

Cases	Number of Jobs	DQMIX	CWQMIX	QTRAN
N4	4	68	65	81
N5	5	68	66	79
N6	6	70	68	81
N7	7	71	79	76
N8	8	81	79	83
Mean	-	71.6	71.4	80
Variance	-	29.3	49.3	7

According to Table 13, the three algorithms still have generalization for the varied number of jobs. Specifically, the average of the solutions of CWQMIX and DQMIX are significantly lower than QTRAN, indicating that CWQMIX and DQMIX have better generalization.

5.6.3. Generalization to Changes in Operations

This test verifies the generalization of the algorithms to different operation settings. We constructed 10 cases, each containing 8 jobs, with randomly assigned operations for each job. The test results are shown in Table 14. According to Table 14, the average solution of DQMIX is lower than QTRAN and CWQMIX.

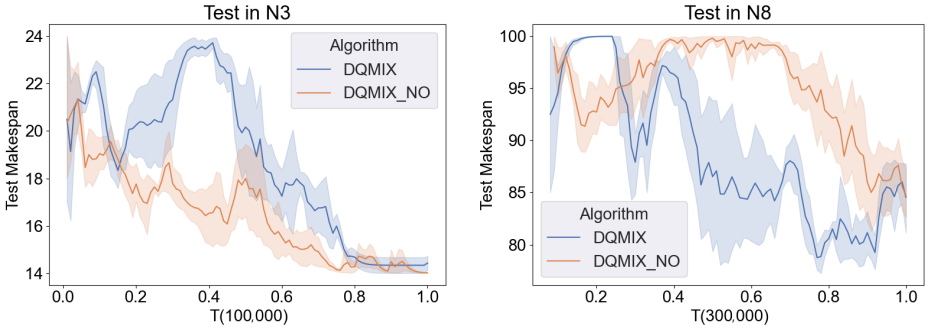
Table 14. Test with the variation in operations.

Case	DQMIX	CWQMIX	QTRAN
1	77	73	74
2	89	73	72
3	71	72	88
4	79	72	74
5	68	69	69
6	75	100	72
7	66	100	72
8	69	66	78
9	68	67	71
10	67	100	72
Mean	72.9	79.2	74.2
Success rate	100%	70%	100%

The generalization tests conducted on the three algorithms reveal that DQMIX has the best generalization.

5.7. GCN Effectiveness Verification

Compared with GCN, we artificially design a state vector and an observation vector, called state vector representations. DQMIX and DQMIX-NO represent the model based on the GCN embedding and the state vector representations, respectively. We test the two models in the cases of N3 and N8 three times on three random seeds, and the results are shown in Figure 5.



(**a**) N3 (**b**) N8

Figure 5. Test performance for GCN.

According to Figure 5a, we observe that DQMIX-NO outperforms DQMIX in terms of convergence speed and final solution quality. However, in the N8 case (Figure 5b), DQMIX converges faster and achieves better results than DQMIX-NO. Additionally, DQMIX demonstrates greater stability in both cases.

Our experiments show that DQMIX can be as effective as DQMIX-NO in small-scale cases. However, for larger-scale cases, DQMIX shows better convergence and stability, which improves the overall training effect. Another benefit of DQMIX is that it requires no domain expertise for state modeling.

5.8. Ablation on Reward Function

The reward function based on makespan increment estimation is proposed to aid model training. To verify the effectiveness of the reward function, we make a comparison with a sparse reward function.

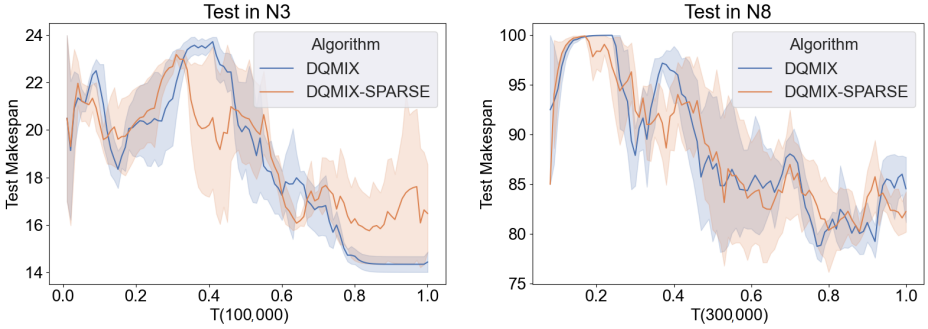
The sparse reward function is defined as follows.

$$r_t(s,a) = \begin{cases} 0 & \text{if } t < |T| \text{ and } EC_{max}(t) \leq C_{up}, \\ -Z & \text{if } EC_{max}(t) > C_{up}, \\ (C_{up} - C_{max}) \times \alpha & \text{if } t = |T| \text{ and } EC_{max}(t) \leq C_{up}. \end{cases} \tag{33}$$

The DQMIX model with the sparse reward function is labeled as DQMIX-SPARSE.

We perform ablation with the reward functions in the cases of N3 and N8, and the results are shown in Figure 6.

Figure 6a shows that in the case of N3, DQMIX achieves better convergence and solution quality than DQMIX-SPARSE, while in the case of N8 (Figure 6b), the difference between the two is not significant in terms of convergence. It is worth noting that our training policy of sampling complete trajectories in batches and using them for learning may alleviate the negative impact of sparse feedback on model training.



(**a**) N3 (**b**) N8

Figure 6. Test performance for reward functions.

6. Conclusions and Future Work

The paper introduces a complex scheduling problem called FJSP-DT, which has path and operation sequence flexibility and varied transportation time. FJSP-DT is modeled as Dec-POMDP, and GCN is applied to compress its states and observations. Mechanistic constraints are employed to exclude invalid actions and reduce the action space. We propose a MARL-based DQMIX algorithm to solve FJSP-DT. This algorithm incorporates double critic networks: the optimal network and the mixing network, enhancing its exploration and exploitation capabilities. The experimental results demonstrate that DQMIX can obtain comparable solutions to traditional algorithms. DQMIX shows high convergence and good generalization in the presence of robot breakdowns, varying quantities of jobs, and changes in operations.

Our future works will mainly focus on three aspects: (1) improving the scalability of algorithms through data and knowledge fusion-driven methods, (2) refining the reward function through hierarchical reinforcement learning, and (3) constructing a virtual–physical integrated system to support aircraft maintenance operations.

Author Contributions: Conceptualization, S.P., G.X., Z.S. and Y.H.; methodology, S.P.; software, S.P.; validation, S.P.; formal analysis, S.P.; investigation, S.P.; resources, Z.S. and Y.H.; data curation, S.P.; writing—original draft preparation, S.P.; writing—review and editing, G.X., J.Y., T.S.T., Z.T., Z.S., S.P. and Y.H.; visualization, S.P.; supervision, G.X.; project administration, F.-Y.W.; funding acquisition, G.X., Z.S. and Y.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFB3301504, in part by the National Natural Science Foundation of China under Grant U19B2029, Grant U1909204 and Grant 92267103, in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2021B1515140034, and in part by the CAS STS Dongguan Joint Project under Grant 20211600200022 and Grant 20201600200072.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data available on request from the authors.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Chaudhry, I.A.; Khan, A.A. A research survey: Review of flexible job shop scheduling techniques. Int. Trans. Oper. Res. **2016**, *23*, 551–591. [CrossRef]
2. Xiong, H.; Shi, S.; Ren, D.; Hu, J. A survey of job shop scheduling problem: The types and models. *Comput. Oper. Res.* **2022**, *142*, 105731. [CrossRef]
3. Luo, Q.; Deng, Q.; Xie, G.; Gong, G. A Pareto-based two-stage evolutionary algorithm for flexible job shop scheduling problem with worker cooperation flexibility. *Robot. Comput.-Integr. Manuf.* **2023**, *82*, 102534. [CrossRef]
4. Wei, Z.; Liao, W.; Zhang, L. Hybrid energy-efficient scheduling measures for flexible job-shop problem with variable machining speeds. *Expert Syst. Appl.* **2022**, *197*, 116785. [CrossRef]
5. Li, Y.; Gu, W.; Yuan, M.; Tang, Y. Real-time data-driven dynamic scheduling for flexible job shop with insufficient transportation resources using hybrid deep Q network. *Robot. Comput.-Integr. Manuf.* **2022**, *74*, 102283. [CrossRef]
6. Du, Y.; Li, J.Q.; Chen, X.L.; Duan, P.Y.; Pan, Q.K. Knowledge-based reinforcement learning and estimation of distribution algorithm for flexible job shop scheduling problem. *IEEE Trans. Emerg. Top. Comput. Intell.* **2023**, *7*, 1036–1050. [CrossRef]
7. Burggräf, P.; Wagner, J.; Saßmannshausen, T.; Ohrndorf, D.; Subramani, K. Multi-agent-based deep reinforcement learning for dynamic flexible job shop scheduling. *Procedia CIRP* **2022**, *112*, 57–62. [CrossRef]
8. Zhang, Y.; Zhu, H.; Tang, D.; Zhou, T.; Gui, Y. Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems. *Robot. Comput.-Integr. Manuf.* **2022**, *78*, 102412. [CrossRef]
9. Wang, X.; Zhang, L.; Lin, T.; Zhao, C.; Wang, K.; Chen, Z. Solving job scheduling problems in a resource preemption environment with multi-agent reinforcement learning. *Robot. Comput.-Integr. Manuf.* **2022**, *77*, 102324. [CrossRef]
10. Jing, X.; Yao, X.; Liu, M.; Zhou, J. Multi-agent reinforcement learning based on graph convolutional network for flexible job shop scheduling. *J. Intell. Manuf.* **2022**, 1–19. . [CrossRef]

11. Ku, W.Y.; Beck, J.C. Mixed integer programming models for job shop scheduling: A computational analysis. *Comput. Oper. Res.* **2016**, *73*, 165–173. [CrossRef]
12. Gao, K.; Cao, Z.; Zhang, L.; Chen, Z.; Han, Y.; Pan, Q. A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems. *IEEE/CAA J. Autom. Sin.* **2019**, *6*, 904–916. [CrossRef]
13. Tian, Y.; Si, L.; Zhang, X.; Cheng, R.; He, C.; Tan, K.C.; Jin, Y. Evolutionary large-scale multi-objective optimization: A survey. *ACM Comput. Surv.* **2021**, *54*, 174. [CrossRef]
14. Afshin, O.; Davood, H. A review of cooperative multi-agent deep reinforcement learning. *Appl. Intell.* **2023**, *53*, 13677–13722.
15. Lihu, A.; Holban, S. Top five most promising algorithms in scheduling. In Proceedings of the 2009 5th International Symposium on Applied Computational Intelligence and Informatics, Timisoara, Romania, 28–29 May 2009; pp. 397–404.
16. Wang, X.; Zhang, L.; Ren, L.; Xie, K.; Wang, K.; Ye, F.; Chen, Z. Brief review on applying reinforcement learning to job shop scheduling problems. *J. Syst. Simul.* **2021**, *33*, 2782.
17. Liu, Y.K.; Zhang, X.S.; Zhang, L.; Tao, F.; Wang, L.H. A multi-agent architecture for scheduling in platform-based smart manufacturing systems. *Front. Inf. Technol. Electron. Eng.* **2019**, *20*, 1465–1492. [CrossRef]
18. Zhang, W.; Dietterich, T.G. A reinforcement learning approach to job-shop scheduling. In Proceedings of the IJCAI, Citeseer, Montreal, QU, Canada, 20–25 August 1995; Volume 95, pp. 1114–1120.
19. Aydin, M.E.; Öztemel, E. Dynamic job-shop scheduling using reinforcement learning agents. *Robot. Auton. Syst.* **2000**, *33*, 169–178. [CrossRef]
20. Waschneck, B.; Reichstaller, A.; Belzner, L.; Altenmüller, T.; Bauernhansl, T.; Knapp, A.; Kyek, A. Optimization of global production scheduling with deep reinforcement learning. *Procedia CIRP* **2018**, *72*, 1264–1269. [CrossRef]
21. Luo, S. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Appl. Soft Comput.* **2020**, *91*, 106208. [CrossRef]
22. Lang, S.; Behrendt, F.; Lanzerath, N.; Reggelin, T.; Müller, M. Integration of deep reinforcement learning and discrete-event simulation for real-time scheduling of a flexible job shop production. In Proceedings of the 2020 Winter Simulation Conference (WSC), Orlando, FL, USA, 14–18 December 2020; pp. 3057–3068.
23. Gu, Y.; Chen, M.; Wang, L. A self-learning discrete salp swarm algorithm based on deep reinforcement learning for dynamic job shop scheduling problem. *Appl. Intell.* **2023**, *53*, 18925–18958. [CrossRef]
24. Wang, L.; Hu, X.; Wang, Y.; Xu, S.; Ma, S.; Yang, K.; Liu, Z.; Wang, W. Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning. *Comput. Netw.* **2021**, *190*, 107969. [CrossRef]
25. Gronauer, S.; Diepold, K. Multi-agent deep reinforcement learning: A survey. *Artif. Intell. Rev.* **2022**, *55*, 895–943. [CrossRef]
26. Chandak, Y.; Theocharous, G.; Kostas, J.; Jordan, S.; Thomas, P. Learning action representations for reinforcement learning. In Proceedings of the International Conference on Machine Learning, PMLR, Long Beach, CA, USA, 9–15 June 2019; pp. 941–950.
27. Liu, Y.; Li, Z.; Jiang, Z.; He, Y. Prospects for multi-agent collaboration and gaming: Challenge, technology, and application. *Front. Inf. Technol. Electron. Eng.* **2022**, *23*, 1002–1009. [CrossRef]
28. Aissani, N.; Trentesaux, D.; Beldjilali, B. Multi-agent reinforcement learning for adaptive scheduling: Application to multi-site company. *IFAC Proc. Vol.* **2009**, *42*, 1102–1107. [CrossRef]
29. Martínez Jiménez, Y.; Coto Palacio, J.; Nowé, A. Multi-agent reinforcement learning tool for job shop scheduling problems. In Proceedings of the International Conference on Optimization and Learning, Cadiz, Spain, 17–19 February 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 3–12.
30. Hameed, M.S.A.; Schwung, A. Reinforcement learning on job shop scheduling problems using graph networks. *arXiv* **2020**, arXiv:2009.03836.
31. Zhou, T.; Tang, D.; Zhu, H.; Zhang, Z. Multi-agent reinforcement learning for online scheduling in smart factories. *Robot. Comput.-Integr. Manuf.* **2021**, *72*, 102202. [CrossRef]
32. Popper, J.; Motsch, W.; David, A.; Petzsche, T.; Ruskowski, M. Utilizing multi-agent deep reinforcement learning for flexible job shop scheduling under sustainable viewpoints. In Proceedings of the 2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), Mauritius, 7–8 October 2021; pp. 1–6.
33. Wang, S.; Li, J.; Luo, Y. Smart scheduling for flexible and hybrid production with multi-agent deep reinforcement learning. In Proceedings of the 2021 IEEE 2nd International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA), Chongqing, China, 17–19 December 2021; Volume 2, pp. 288–294.
34. Kipf, T.N.; Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv* **2016**, arXiv:1609.02907.
35. Zhang, C.; Song, W.; Cao, Z.; Zhang, J.; Tan, P.S.; Chi, X. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 1621–1632.
36. Rashid, T.; Samvelyan, M.; Schroeder, C.; Farquhar, G.; Foerster, J.; Whiteson, S. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In Proceedings of the International Conference on Machine Learning, PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 4295–4304.
37. Son, K.; Kim, D.; Kang, W.J.; Hostallero, D.E.; Yi, Y. QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In Proceedings of the International Conference on Machine Learning, PMLR, Long Beach, CA, USA, 9–15 June 2019; pp. 5887–5896.
38. Mahajan, A.; Rashid, T.; Samvelyan, M.; Whiteson, S. MAVEN: Multi-agent variational exploration. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 7611–7622.

39. Rashid, T.; Farquhar, G.; Peng, B.; Whiteson, S. Weighted QMIX: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 10199–10210.
40. Son, K.; Ahn, S.; Reyes, R.D.; Shin, J.; Yi, Y. QTRAN++: Improved value transformation for cooperative multi-agent reinforcement learning. *arXiv* **2020**, arXiv:2006.12010.
41. Ha, D.; Dai, A.M.; Le, Q.V. HyperNetworks. In Proceedings of the 5th International Conference on Learning Representations, ICLR, Toulon, France, 24–26 April 2017.
42. Li, X.; Gao, L. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. *Int. J. Prod. Econ.* **2016**, *174*, 93–110. [CrossRef]
43. Li, X.; Gao, L.; Pan, Q.; Wan, L.; Chao, K.M. An effective hybrid genetic algorithm and variable neighborhood search for integrated process planning and scheduling in a packaging machine workshop. *IEEE Trans. Syst. Man, Cybern. Syst.* **2018**, *49*, 1933–1945. [CrossRef]
44. Liu, Q.; Li, X.; Gao, L.; Li, Y. A modified genetic algorithm with new encoding and decoding methods for integrated process planning and scheduling problem. *IEEE Trans. Cybern.* **2020**, *51*, 4429–4438. [CrossRef]