Journal of Manufacturing Systems 65 (2022) 130–145

Contents lists available at ScienceDirect

Journal of Manufacturing Systems

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

Technical paper

Dynamic scheduling of tasks in cloud manufacturing with multi-agent reinforcement learning

Xiaohan Wang [a], Lin Zhang [a,*], Yongkui Liu [b], Feng Li [c], Zhen Chen [a], Chun Zhao [d], Tian Bai [a]

[a] School of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, China
[b] School of Mechano-Electronic Engineering, Xidian University, Xi'an 710071, China
[c] School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798, Singapore
[d] Computer School, Beijing Information Science and Technology University, Beijing 100101, China

ARTICLE INFO

MSC:
00-01
99-00

Keywords:
Cloud manufacturing
Dynamic scheduling
Multi-agent reinforcement learning
Graph convolution network
Smart manufacturing

ABSTRACT

Cloud manufacturing provides a cloud platform to offer on-demand services to complete consumers' tasks, but assigning tasks to enterprises with different services requires many-to-many scheduling. The dynamic cloud environment puts forward higher requirements on scheduling algorithms' real-time response and generalizability. Additionally, complex manufacturing tasks with flexible processing sequences also increase the decision-making difficulty. The existing approaches either have difficulty meeting the requirements of dynamics and fast-respond or struggle to effectively capture features of tasks with flexible processing sequences. To address these limitations, we develop a novel scheduling algorithm to solve a dynamic scheduling problem in the group service cloud manufacturing environment. Our proposal is formulated and trained by multi-agent reinforcement learning. The graph convolution network encodes tasks' graph-structure features, and the recurrent neural network records each task's processing trajectories. We independently design the action space and the reward function and train the algorithm with a mixing network under the centralized training decentralized execution architecture. Multi-agent reinforcement learning and graph convolution networks are rarely used to cloud manufacturing scheduling problems. Contrast experiments on a case study indicate that our proposal outperforms the other six multi-agent reinforcement learning-based scheduling algorithms in terms of scheduling performance and generalizability.

1. Introduction

As an open and sharing platform, cloud manufacturing (CMfg) integrates enterprises' manufacturing resources and gathers consumers' requirements [1,2]. In CMfg, manufacturing resources are encapsulated into services and assigned to different consumers over the Internet. Hence, CMfg breaks through the physical distance limitations of traditional manufacturing and provides convenience for high-end products such as aircraft structural parts. There are two phases in the CMfg scheduling, including the first and second phases. The first phase matches task batches with groups of similar enterprises, and the second phase assigns a batch of tasks to specific enterprises [3,4]. The first phase of matching has been studied in multitudinous works over the years [5], and corresponding research is relatively mature. However, the second phase of scheduling still faces many challenges due to environmental dynamics and task complexity. As the scheduling target of the second phase is assigning a batch of tasks to a group of enterprises, we name this problem the group service CMfg (GSCMfg) scheduling problem. This problem has three primary challenges: (1) The service types in an enterprise may change dynamically during the scheduling, and the dynamic is unpredictable and irregular. (2) A manufacturing task contains several subtasks requiring different services, and the processing sequence of subtasks in a task follows the structure of the directed acyclic graph (DAG). (3) Manufacturing tasks need to be scheduled simultaneously within a short period, which requires the fast-responding ability for multi-task scheduling.

Previous research proposed various scheduling algorithms for CMfg scheduling problems, which can be divided into three categories: dispatching rules, meta-heuristic algorithms, and reinforcement learning-based (RL-based) algorithms. Some researchers designed dispatching rules for CMfg scheduling to resolve dynamics and obtained favorable results [6,7] because rules possess a fast-responding ability due to their low computation consumption. However, effective dispatching rules usually require an experienced person's careful design and cannot always guarantee a satisfactory solution. To alleviate these issues of dispatching rules, most researchers choose to solve CMfg scheduling problems by meta-heuristic algorithms. Among them, ant colony

* Corresponding author.
E-mail address: zhanglin@buaa.edu.cn (L. Zhang).

https://doi.org/10.1016/j.jmsy.2022.08.004
Received 23 November 2021; Received in revised form 23 July 2022; Accepted 13 August 2022
Available online 13 September 2022

optimization (ACO) and genetic algorithm (GA) are two representative algorithms to optimize the single-objective in CMfg scheduling problems [3,8]. Meta-heuristic algorithms automatically find the local-optimal solution according to the established goal, and the scheduling results are more accurate than dispatching rules in general. However, the high computation consumption of meta-heuristic algorithms brings a slow-responding issue, and the generalizability of meta-heuristic algorithms is relatively low. When the environment changes, they need to recalculate the scheduling scheme. So meta-heuristic algorithms are rarely applied to dynamic CMfg scheduling problems. With the development of artificial intelligence (AI), reinforcement learning (RL)[1] has shown apparent strength for solving multiple combinatorial optimization (CO) problems benefitting from the training-based framework and powerful non-linear fitness capability [9,10]. In RL-based algorithms, the scheduling problem is modeled as a Markov decision process (MDP), where necessary components, including the action, the state, and the reward function, are defined to describe the interaction between agents and the environment [11]. RL-based algorithms have positive generalizability, so they are widely applied to dynamic scheduling problems [12]. Besides, because RL-based algorithms are training-based and can save trained models offline, they obtain the fast-responding ability through reusing pre-trained models. Nevertheless, current RL-based scheduling methods can hardly address the challenge of capturing tasks' graph-structure features when facing the GSCMfg scheduling problem. Additionally, these RL-based scheduling algorithms are all based on single-agent architectures. The primary issue is that a single-agent architecture would consist of all combination actions of all tasks, leading to a vast action space requiring a deep neural network with high complexity [13]. A multi-agent architecture can address this issue by shifting the complexity of action space to the difficulty of collaboration between agents.

With the motivation for solving the dynamic GSCMfg scheduling problem, we propose a multi-agent graph convolution integrated scheduler (MAGCIS). MAGCIS addresses the three challenges mentioned above based on the combination of multi-agent reinforcement learning (MARL) and graph convolution network (GCN): (1) MAGCIS can resolve the dynamic challenge in the GSCMfg environment because MARL can suit the dynamic environment, and the trained models possess the fast-responding ability. (2) Graph features of tasks are captured by GCN to represent the implicit processing information of subtasks, and the information mainly includes the types and sequences of subtasks. (3) MAGCIS shifts the complexity of action space to the difficulty of collaboration between agents by applying a multi-agent architecture. MARL and GCN are new algorithms and are rarely used to CMfg scheduling problems. The proposed method represents an early exploration in this aspect and obtained positive performance on the GSCMfg scheduling problem. Contributions of this paper can be summarized as follows: (1) We model the GSCMfg environment as a decentralized partially observable Markov decision process (Dec-POMDP) and divide the entire system into the environment and the multi-agent system (MAS). (2) For each agent in the MAS, we extract DAG structures of tasks with GCN and record the trajectories of task processing with RNN. (3) Our unique designs of the action space and the reward function for GSCMfg scheduling problems are proven effective. (4) The training method uses a mixing network to calculate the total Q-value and the temporal difference (TD) error to generate the loss under the centralized training decentralized execution (CTDE) architecture. (5) Numerical experiments on a case study are conducted to verify the effectiveness of our proposal from four perspectives, and we compare MAGCIS with other MARL-based scheduling algorithms. Results indicate that MAGCIS outperforms others in scheduling performance and generalizability.

[1] RL described in this paper mainly refers to deep reinforcement learning (DRL), which combines the traditional RL with neural networks.

The remainder of this paper is organized as follows: Section 2 presents the literature review of the solutions for CMfg scheduling problems, the advances in RL/MARL, and RL/MARL-based scheduling. Section 3 defines the background and statement of the GSCMfg scheduling problem. Section 4 elaborates on the model structure and the training method. The comparison experiments on a case study are implemented and analyzed in Section 5 before concluding the paper in Section 6.

2. Literature review

CMfg scheduling problems have been studied for many years [14]. This section briefly reviews related works on scheduling in CMfg, advances in RL/MARL, and RL/MARL-based scheduling algorithms.

2.1. Scheduling in CMfg

As a critical problem for realizing the optimal assignment of tasks and services in CMfg, scheduling has attracted considerable research interest [15]. Some works focused on the multi-objective optimization problem in scheduling scenarios. In Ref. [3], the two-level scheduling optimization algorithm was proposed based on the two-level multi-task scheduling model, which contained task-level scheduling and subtask-level scheduling. Ref. [16] proposed an extended Gale–Shapley-algorithm-based approach for service composition, which allowed the generalization of multiple service composition solutions. Ref. [5] studied the multi-phase integrated scheduling of hybrid tasks and compared it with the traditional step-by-step decision. Besides, six representative multi-objective evolutionary algorithms were adopted in the proposal, including Pareto-dominance-based, decomposition-based, indicator-based, and many-objective methods. Ref. [17] and Ref. [18] provided ACO-based multi-objective algorithm, NSGA-II-based multi-objective algorithm, and cooperative particle swarm optimization algorithm to optimize scheduling makespan, cost, and QoS. Other scheduling problems such as logistics service and supply chain scheduling were widely studied. Ref. [19] created a win-win situation for all clients involved through client coordination in the process of service booking and scheduling. Aiming at the high real-time and low latency requirements, a cloud edge-based two-level hybrid scheduling learning model was proposed in Ref. [20]. For logistics service scheduling, Ref. [21] established a logistics scheduling service model after analyzing scheduling issues from tasks, production services, logistics services, and optimization objectives.

These existing CMfg scheduling methods are based on dispatching rules or meta-heuristic optimization. However, designing favorable rules requires expert experience and careful consideration, which requires lots of hand-engineering work. Besides, meta-heuristic optimization algorithms rely on long-term iterative optimization, lacking the fast-responding ability to dynamic environments.

2.2. Advances in RL and MARL

Reinforcement learning (RL) describes and solves the decision-making problems in the process of interacting with the environment [22]. RL algorithms can be divided into value-based and policy-based. Value-based algorithms learn the value function $V(s)$ or the Q-value function $Q(s,u)$ and generate their policies based on exploration strategies. Typical value-based algorithms include Q-Learning, Deep-Q-Network (DQN), Double DQN, Dueling DQN [23]. Policy-based algorithms tend to directly learn the policy π, which mainly include proximal policy optimization (PPO), REINFORCE, soft actor–critic (SAC), and actor–critic (AC) [24].

Unlike RL focusing on the single-agent architecture, MARL solves the sequential decision-making problem of multiple autonomous agents operating in a common environment. Each agent aims to optimize the long-term return by interacting with the environment and other

Table 1
Mainstream MARL algorithms under the CTDE architecture.

MARL algorithms	Brief introduction
VDN [28]	Summing $Q\text{-}value$ of each agent to obtain Q^{total} straightly
QMIX [29]	Utilizing the mixing network to convert each $Q\text{-}value$ into Q^{total}
QTRAN [30]	Leveraging $V\text{-}value$ to assist in obtaining Q^{total} to solve the problem of non-monotonicity.
REINFORCE [31]	A policy-based training architecture that uses returns of all agents to estimate the loss function by the policy gradient theorem.
Central_V [32]	An AC training architecture that uses a central state and temporal-difference (TD) errors to estimate the advantages for training.

agents [25]. Expanding the single-agent RL algorithms to multiple agents, such as independent Q-learning (IQL) [26], does not perform as well as other hybrid architectures that are specifically proposed for multi-agents. In MARL, the centralized training and decentralized execution (CTDE) architecture has become the mainstream architecture [24]. In CTDE, the MARL algorithm's training architecture and execution architecture are inconsistent, and CTDE has been proven effective in solving multi-agent decision-making problems [27]. We report mainstream state-of-the-art (SOTA) MARL algorithms under the CTDE architecture in Table 1

2.3. RL/MARL-based scheduling algorithms

This section reviews the related literature in two categories: RL-based CMfg scheduling and MARL-based general scheduling. Applying RL to CMfg scheduling problems is becoming more and more popular [33], but currently, there are few MARL-based CMfg scheduling algorithms. The absence of MARL-based scheduling algorithms can be attributed to two primary reasons. First, MARL requires users to have more experience and skills in hyper-parameter tuning because training a MARL algorithm to be convergence is more complicated than that of an RL algorithm. Second, converting a multi-agent problem to a single-agent one is also applicable in some CMfg scheduling problems, although scheduling performance may decrease because of this conversion.

Some researchers have recently adopted RL algorithms to solve CMfg scheduling problems. A workflow scheduling algorithm was proposed in Ref. [34], and the algorithm was developed based on the AC algorithm. A DQN-based task scheduling mechanism was proposed in Ref. [35] to minimize the makespan in CMfg scheduling problems. However, Ref. [34] and Ref. [35] studied the static scheduling problem rather than the dynamic one. Some researchers proposed RL-based scheduling algorithms to solve the dynamic scheduling in the CMfg environment. To assist SaaS providers in making near-optimal service selection decisions in a dynamic and stochastic cloud environment, Ref. [36] proposed a workflow application scheduling approach based on the Q-learning algorithm. Ref. [12] studied the dynamic multi-objective scheduling problem in the CMfg environment and proposed an RL-based method. Their method converted scheduling problems with multiple resources into one learning target and utilized a blacklist mechanism to improve the learning performance. To solve issues of the uneven distribution of the services and the uncertain arrival of projects, Ref. [37] proposed an RL-based assigning policy (RLAP) approach to obtain a non-dominated solution set. The authors designed a dynamic state representing algorithm for agents to determine their decision context. However, these methods face the challenge of the high dimension of the action space when applied to more complex problems [15]. Besides, they struggle to represent graph-structure features of tasks when solving GSCMfg scheduling problems.

A few works have focused on applying MARL algorithms to scheduling problems. Ref. [13] thought that a multi-agent architecture and MARL algorithms could handle training and high-dimension action issues. Still, the paper only provided a conceptual-level description. Ref. [38] used the MARL algorithm, named MADDPG, in a distributed-agent architecture to provide demand response for discrete manufacturing systems energy management, and experiments showed that the proposed method was better than traditional mathematical solutions. Ref. [39] applied the RL algorithm PPO with the generalized advantage estimator (GAE) in a distributed-agent architecture to address the dynamic energy scheduling problem, which outperformed meta-heuristic algorithms. Ref. [40] regarded each bus as an agent and formed a distributed bus agent system to provide dynamic holding control. Ref. [41] modeled the scheduling problem in the cloud environment as a Markov decision process and regarded each workflow as an agent. They defined that each agent chose the following action according to the states of surrounding agents, and experiments showed that their solution outperformed meta-heuristic algorithms. However, these methods are not proposed for CMfg scheduling problems but for addressing particular issues in their focused environments. Besides, these MARL-based methods do not consider the extraction of graph-structure features.

3. Problem statement

In this section, the CMfg environment is introduced first, followed by the formulation of the GSCMfg scheduling problem.

3.1. CMfg environment

As shown on the right side of Fig. 1, there are mainly two phases for the entire scheduling process in CMfg, including matching and scheduling phases [15]. In the first phase, numerous tasks and services are clustered into different task batches and service groups, respectively. Then, each task batch is matched with a group of services based on the similarity [3,42]. In the second phase, a batch of tasks is assigned to enterprises with a group of services. Then enterprises process the assigned tasks and deliver the processing information to the scheduler. It is worth noticing that the interaction frequency between the scheduler and enterprises is by step rather than by episode, meaning that each scheduling action will receive an immediate response after the assignment.

Definition 1. Step: One step is triggered by an event that a subtask is completed. It also represents one interaction between the MAS and the GSCMfg environment.

Definition 2. Episode: An episode represents an entire scheduling procedure, and the terminal of an episode is reached when all subtasks of all tasks are completed.

In this paper, the task scheduling in the second phase is mainly considered. The scheduler provides scheduling solutions to assign tasks to available enterprises in each step. In the GSCMfg scheduling problem, each enterprise has multiple services for processing different subtasks. However, the number of services in an enterprise may increase or decrease during the scheduling, leading to a dynamic service distribution. For example, existing services may break out due to unforeseen issues, and new services may be added.

A task can also be regarded as a workflow consisting of several subtasks [34]. This paper only considers the scheduling of processing-level tasks, which means that the product processed by a task cannot be split for parallel processing. For example, Fig. 2 demonstrates the processing task of the rotating assembly. As an aircraft engine's core component, the rotating assembly's processing requires multiple high-precise manufacturing services. The entire processing contains five stages: forging, turnery, heat-treat, computerized numerical control

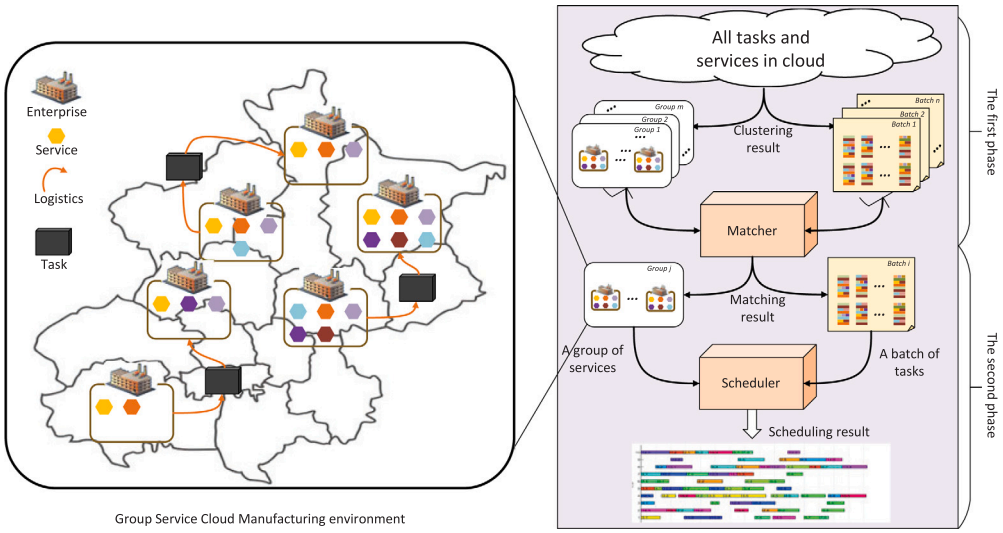

Fig. 1. Two phases in the entire CMfg scheduling process. The first phase utilizes a matcher to match each batch of tasks with a group of services, and then the second phase schedules tasks to services. The left side of the figure shows a scheduling demonstration, where tasks are processed and transported among enterprises with different services in the Beijing service group. These enterprises are located in different districts of Beijing, and the transfer of tasks depends on logistics.

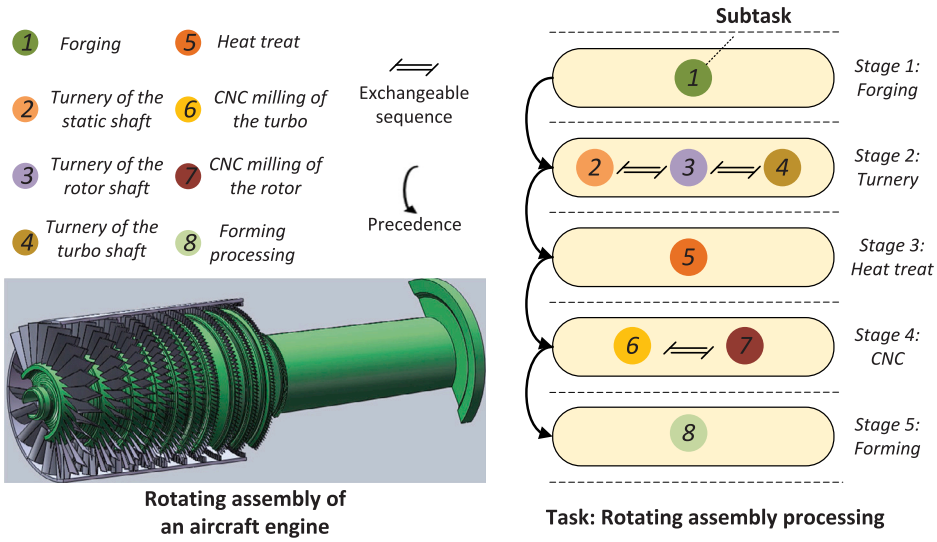

Fig. 2. Processing task of the rotating assembly. The left side shows the subtasks numbered 1–8 and their required processing services. The right side shows the processing sequence of the task.

(CNC), and forming. Each stage may have more than one subtask. For instance, there are three subtasks in the turnery stage: turnery of the static shaft, turnery of the rotor shaft, and turnery of the turbo shaft. Processing sequences of subtasks in the same stage are flexible without constraints, meaning that these sequences are exchangeable. However, there are precedence relationships between stages, which means that subtasks in a stage can only be processed when all subtasks in the last stage are finished. Different subtasks require various services, and there is a deterministic matching relationship between subtasks and services. Subtasks can be processed in one or many enterprises. A task is completed when all its subtasks are finished.

3.2. Problem formulation

The GSCMfg environment differs from classical scheduling environments in the task representation and the dynamic service distribution. There is a batch of n tasks $T = \{T^1, T^2, \dots, T^n\}$ to be scheduled among k enterprises $E = \{E^1, E^2, \dots, E^k\}$. For each task $T^i \in T$, it has m_i stages $\{St^{i,1}, St^{i,2}, \dots, St^{i,m_i}\}$, and each stage $St^{i,j} \in St^i$ contains $a^{i,j}$ subtasks that need to be processed by corresponding services. Enterprises provide services, and each enterprise $E^i \in E$ provides w_i types of services. The service distribution in E^i is represented as ϕ^{E^i}, and $\phi^{E^i} = \{S^{i,1}, S^{i,2}, \dots, S^{i,w_i}\}, i \in [1, k]$. We note the service distribution of E^i in the $t+1$ timestep as $\phi^{E^i}_{t+1}$. $\phi^{E^i}_{t+1}$ may dynamically change during the scheduling, and this dynamic is represented by:

$$\phi^{E^i}_{t+1} = \begin{cases} \phi^{E^i}_t + \Delta\phi^+_{t+1} & \delta^i_{t+1} > \epsilon 1 \\ \phi^{E^i}_t - \Delta\phi^-_{t+1} & \epsilon 2 < \delta^i_{t+1} \le \epsilon 1 \\ \phi^{E^i}_t & \delta^i_{t+1} \le \epsilon 2 \end{cases} \quad \forall i \in [1, k] \tag{1}$$

where $\Delta\phi^+_{t+1}, \Delta\phi^-_{t+1}$ represent an increased service and a decreased service, respectively. $\Delta\phi^+_{t+1} \notin \phi^{E^i}_t$, and $\Delta\phi^-_{t+1} \in \phi^{E^i}_t$. $\delta^i_{t+1} \in (0, 1)$ represent a random variable, and $\epsilon 1, \epsilon 2 \in (0, 1)$ are two threshold constants. Eq. (1) means that service distributions of enterprises may change dynamically in each timestep, resulting in an increase or decrease in services. After finishing the current processing in the enterprise E^i, a task may be delivered to the next enterprise E^j for the next processing. The logistics time between E^i and E^j is denoted as $l^{E^i,E^j} \in l$, where l indicates the logistics time matrix.

The objective of the CMfg scheduling problem is to minimize the total makespan M^{total}:

$$M^{total} = \max_{i=1}^{n} \sum_{j=1}^{m_i} \sum_{k=1}^{a^{i,j}} (T^k_{process} + l^{E^{k-1},E^k} + T^k_{waiting}) \quad (2)$$

where i, j, k represent the task i, the stage j, and the subtask k, respectively. $T^k_{process}$ is the processing time of the subtask k, $T^k_{waiting}$ is the waiting time before processing, and l^{E^{k-1},E^k} denotes logistics time from the enterprise of processing the subtask $k-1$ to that of k. Eq. (2) means that a batch of n tasks is processed simultaneously in the GSCMfg environment, and the latest completed task determines the total makespan. We model each task as an agent, and each agent selects the next enterprise to process its following subtask. Nevertheless, this one-level selection requires two-level information for making a proper decision. The upper-level information is the service distributions of all enterprises $\{E^1, E^2, \dots, E^k\}$, and the lower-level information is the subtasks to be processed in the current stage. The feature extraction of this two-level information influences the decision-making. From Eq. (2), we can infer that choosing a satisfying enterprise for a task puts forward the following requirements on the scheduler:

(1) The decision-making procedure of an agent is influenced by both the environment state and other agents, so the scheduler need to consider the global situations. It requires the scheduler to have a 'wide' sight.
(2) The required services for the current and next stages must be considered. For example, selecting an enterprise with adequate services to process subtasks of different stages can save logistics time. It requires the scheduler to have a 'deep' sight.

3.3. Problem features and assumptions

In contrast to other scheduling systems, CMfg is a service-oriented manufacturing mode [7]. Characteristics of the GSCMfg scheduling problem are summarized as follows:

(1) Each enterprise provides more than one service, and the provided services may increase or decrease dynamically during the processing, which cannot be predicted in advance.
(2) The DAG structures of tasks are diverse, and the diversity reflects in different service requirements and distinguishable processing sequences.
(3) The mapping relationship between subtasks and services is many-to-many, meaning that a subtask can be completed in many services, and one service can process many subtasks.
(4) Because the GSCMfg is a cross-enterprise system with various distributed enterprises, logistics are needed to facilitate the task transfer. So the logistics time should be taken into the calculation of the total makespan.

In the GSCMfg scheduling problem, the total makespan is the most concern objective. For simplicity but without losing generality, some assumptions are listed as follows:

Assumption 1. One type of service can only process one type of subtasks, and the same type of services in different enterprises have the same processing time.

Assumption 2. The scheduling of one batch of tasks will not be influenced by the next coming batches.

Assumption 3. Once a subtask is being processed, it cannot be interrupted by inside or outside factors.

Assumption 4. The logistics time is proportional to the distance, and the logistics time between the two enterprises is determined.

Assumption 5. The minimum time interval between two dynamic changes is higher than the time consumption of state perception and decision-making, and the scheduler can always obtain the real-time state of the cloud environment.

4. Overview framework and detailed design

This section will introduce the proposed MAGCIS model in detail, including the overview framework, the multi-agent system model, and the training architecture.

4.1. Overview framework

The overview framework of the proposal is presented at Fig. 3. A batch of tasks is assigned to enterprises with different services, and the scheduling terminates when all tasks are completed. Each task is regarded as an agent, and the agent chooses actions based on the immediate response from the GSCMfg environment in each step. Hence, a batch of tasks is modeled as a multi-agent system (MAS).

A decentralized partially observable Markov decision process (Dec-POMDP) is constructed, which consists of $\langle G, U, P_{env}, r, O, Z, n, \pi, \gamma \rangle$. $g \in G$ is the global state. At each step, each task agent $T^i \in T$ chooses an action $u^i \in U$, forming a joint action $u^1, u^2, \dots, u^n$. The joint action causes a state transition according to the state transition function P_{env}. All task agents share the same reward function $r(g,u) : G \times U \rightarrow R$. The task agent T^i has an observation $o^i \in O$ according to observation function $Z(g,u) : G \times U \rightarrow O$. $\pi^i \in \pi$ represents the policy model of the task agent i, and $\gamma \in [0,1]$ is the discount factor. In Dec-POMDP, the optimization objective described in Eq. (2) is considered in the reward function r.

4.2. Multi-agent system model

In contrast to other single RL-based scheduling methods, each task in MAGCIS is regarded as an agent. Hence, a batch of tasks is represented as a MAS. As shown in Fig. 4, each agent i takes the task T^i with the DAG structure of current unprocessed subtasks as its input and outputs the chosen action u^i indicating the chosen enterprise for its next processing. MAGCIS can be adaptive to the number of tasks in a batch automatically. Given that the model is trained with n agents, then the trained model can also generate scheduling solutions when facing m agents even though $n \neq m$. However, if $n < m$, the scheduling performance of the model may be inferior to that of a model trained with m agents. This section presents the observation encoding in the encoding part and the action selection in the output part.

4.2.1. Observation encoding

In MAGCIS, an agent takes the current processing progress of the task as its observation. To represent the DAG structure of the task T^i, we use $node^{i,1}, node^{i,2}, \dots, node^{i,e}$ to represent e nodes in T^i. Each node in a task DAG represents a subtask. We connect nodes in different stages with edges, and the $(e \times e)$-dimensional adjacency matrix $A_{(e,e)}$ denotes the directed connection relationship. The graph encoding based on GCN is applied to represent the graph-structure features. To simplify the equation representation below, we omit the footmark t denoting the t step because all operations in the observation encoding occur in the t step.

As a typical graph neural network (GNN), GCN generates a node representation by aggregating neighbors' features. The aggregating method of GCN uses graph convolution layers to extract high-level node representations [43]. We aggregate graph information according to the reverse-direction sequence of the original DAG to propagate information from children to parent nodes. Under this circumstance, the embedding of the current processing node h^k contains information of nodes $Ch(h^k)$, where $Ch(h^k)$ represents children nodes of h^k, as shown in Fig. 5.

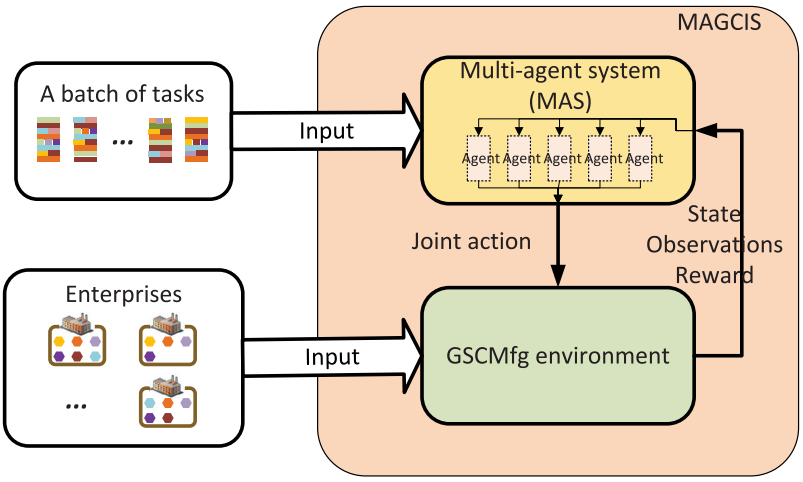

Fig. 3. Overview framework of the proposed MAGCIS model. Each task is regarded as an agent in the MAS.

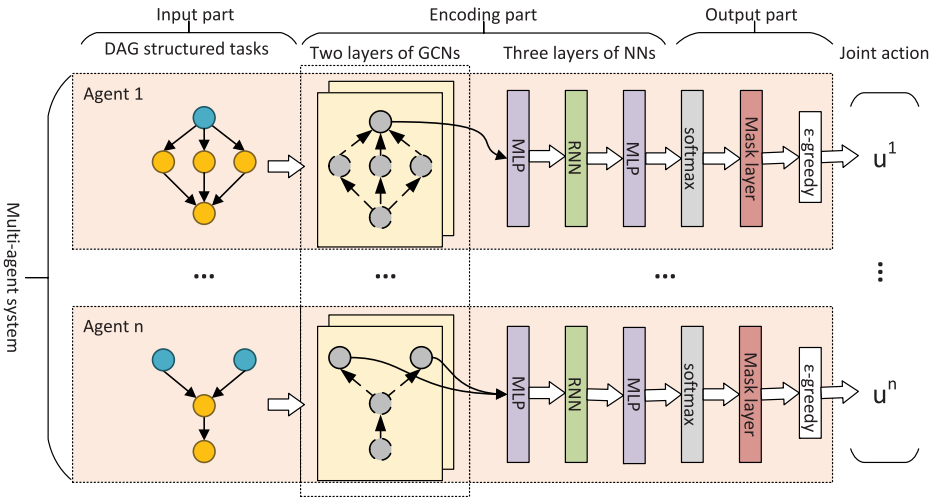

Fig. 4. Model structure of the MAS. Each agent contains three parts: the input part, the encoding part, and the output part. MLP is the abbreviation of the multi-layer perceptron.

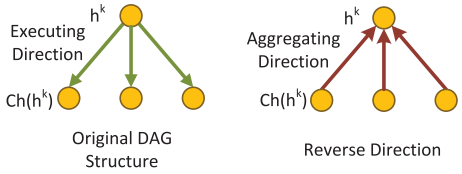

Fig. 5. The aggregating direction in GCN.

For the DAG graph $A^i_{(e,e)}$ with e nodes (simplified as A^i) of the task i, we adopt two layers of GCNs to extract the graph feature [44]:

$$H^{i,1} = ReLU(\widetilde{D^i}^{-\frac{1}{2}}\widetilde{A^i}\widetilde{D^i}^{-\frac{1}{2}}X^iW^1) \tag{3}$$

$$H^{i,2} = ReLU(\widetilde{D^i}^{-\frac{1}{2}}\widetilde{A^i}\widetilde{D^i}^{-\frac{1}{2}}H^{i,1}W^2) \tag{4}$$

where $\widetilde{A^i} = A^i + I$, I is the identity matrix with the same shape of A^i. $\widetilde{D^i}$ is the degree matrix of A^i. X^i is the initial node input, which is initialized by the binary encoding features of the processing time of all subtasks in the task T^i. For example, a subtask node feature x^k requiring a 6 hours service processing is initialized as $x^{i,k} = [0,0,0,1,1,0]$. Hence, $X^i = \{x^{i,1}, x^{i,2}, \ldots, x^{i,e}\}$. W^1, W^2 are the weight parameters needed to be learned, and $H^{i,1}, H^{i,2}$ are outputs of these two GCN layers, respectively. $ReLU(x) = max(0,x)$ is the activation function. $H^{i,2}$ contains embeddings of e nodes:

$$H^{i,2} = \{h_1^{i,2}, h_2^{i,2}, \ldots, h_e^{i,2}\} \tag{5}$$

Then, we calculate the average node embedding in the stage $St^{i,j}$ by:

$$H^{i,2,\prime} = \frac{h_1^{i,2} + h_2^{i,2} + \cdots + h_{a^{i,j}}^{i,2}}{a^{i,j}} \tag{6}$$

where $a^{i,j}$ is the number of subtasks in the stage $St^{i,j}$. Then, we scale the value of $H^{i,2,\prime}$ by:

$$\widetilde{H^{i,2}} = \frac{H^{i,2,\prime} - -min(H^{i,2,\prime})}{max(H^{i,2,\prime}) - min(H^{i,2,\prime})} \tag{7}$$

The agent processes subtasks following the DAG sequence, and there exists a sequential relationship among steps. RNN can represent this relationship by facilitating the sequence memory storing. So we use RNN to capture the sequential feature after obtaining the average node embedding:

$$H^{i,3} = ReLU(W^3\widetilde{H^{i,2}}) \tag{8}$$

$$H_l^{i,4} = ReLU(H^{i,3} + W^4H_{l-1}^{i,4}) \tag{9}$$

$$Q^i = ReLU(W^5H_l^{i,4}) \tag{10}$$

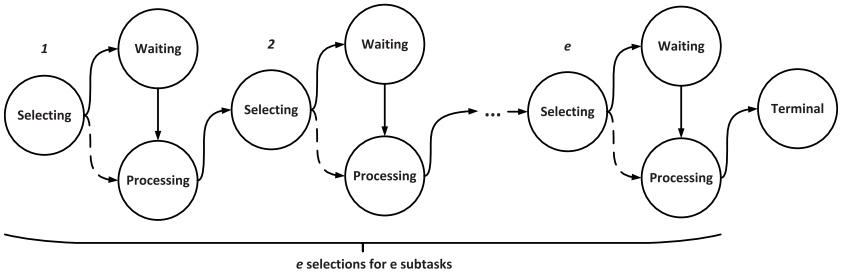

Fig. 6. The action trajectory of an agent. Dotted lines mean that the agent does not have to wait when there are available enterprises.

where W^3, W^4, W^5 are weight parameters to be learned. $H_l^{i,4}$ denotes the RNN output after l units. Q^i represents the final embedding of the observation.

To conclude, the agent i converts the unstructured DAG task information into an embedding vector Q^i through the observation encoding. Q^i contains the information on the current processing subtasks, future processing subtasks, and the observation history.

4.2.2. Action selection

We first introduce the definition of the action space for GSCMfg scheduling problems. Actions are divided into four categories: the waiting action, the selecting action, the processing action, and the terminal action. The waiting action represents no available enterprise, so the agent must wait. The selecting action means that the agent has selected an enterprise for its subsequent processing. The processing action represents that the agent is processing the subtask now, and the terminal action denotes that the agent has completed all subtasks. As shown in Fig. 6, an agent with e subtasks implements selecting actions for e times in an episode.

To represent different actions, we construct the action set $U = \{1, 2, \dots, k, k+1, k+2, k+3\}$, where k is the number of enterprises. For each agent i, $u^i \in U$ can be represented by:

$$u^i = \begin{cases} x, x \in \{1, 2, \dots, k\} & if \quad u^i \in selecting \quad actions \\ k+1 & if \quad u^i \in waiting \quad action \\ k+2 & if \quad u^i \in processing \quad action \\ k+3 & if \quad u^i \in terminal \quad action \end{cases} \tag{11}$$

where k is the number of enterprises. The agent i chooses the next action u_{t+1}^i based on the Q_t^i obtained from the observation encoding in the t step. The selection of the action $u^i = x, x \in \{1, 2, \dots, k\}$ represents that the agent chooses the u^i enterprise. However, the enterprise u^i may not contain the needed services in the task agent i. So we add a mask layer to remove irregular actions and calculate $Q_t^{i,avail}$ by:

$$Q_t^{i,avail} = softmax(Q_t^i) \cdot Mask_t^i \tag{12}$$

where the $softmax(\cdot)$ activation function limits the Q_t^i within the range of $(0, 1)$, and the mask layer $Mask_t^i = [x_1^i, x_2^i, \dots, x_j^i, \dots, x_k^i], x_j^i \in \{0, 1\}, 0 \le j \le k$. In the $Mask_t^i$, $x_j^i = 0$ means that the enterprise j is not available for the agent i. Finally, the action is selected based on the $\epsilon - greedy$ strategy [45]:

$$u_{t+1}^i = \begin{cases} argmax_{u_t^i} Q_t^{i,avail} & e_t > \epsilon \\ random & e_t \le \epsilon \end{cases} \tag{13}$$

where e_t represents a random variable within the range of $(0, 1)$. $\epsilon \in (0, 1)$ is a given parameter named the exploration rate. With the $\epsilon - greedy$ strategy, the agent always has the chance to explore new actions to avoid overfitting the observations.

4.3. Training architecture

In this section, we introduce the training architecture of MAGCIS, including the reward function design, the training procedure, and training tricks.

4.3.1. Reward function design

Since the objective is to minimize the total makespan, the reward function is defined to lead the agent to learn a makespan-reducing policy. All agents in MARL share the same reward function. In MAGCIS, we define the reward function as:

$$r_t = \begin{cases} N_t^1 - \frac{k_1}{\sum_t T^t * N_t^2}, & N_t^2 \neq 0 \quad and \quad t \neq t_{done} \\ N_t^1 - k_2 * \sum_t T^t, & N_t^2 = 0 \quad and \quad t \neq t_{done} \\ \frac{k_3}{M^{total}}, & t = t_{done} \end{cases} \tag{14}$$

where k_1, k_2, k_3 are given parameters within the range of $(0, +\infty)$. T^t means the time consumption in the t step, and $\sum_t T^t$ represents the cumulative makespan until the t step. N_t^1, N_t^2 are the number of selecting actions and waiting actions selected by agents in the MAS. t_{done} denotes the terminal step.

In RL-based scheduling algorithms, some researchers prefer directly using the optimization objective as the reward function [9]. However, we found that this method is not applicable to MARL-based solutions. To gradually lead MAGCIS to learn a makespan-reducing policy, we add some leading signals to the reward function. In training, more selecting actions means a more reasonable policy, so N_t^1 is positively related to the reward. In contrast, the waiting actions are expected to be less because choosing a waiting action means that there may be a conflict in allocating services. Besides, the current makespan $\sum_t T^t$ is expected to be minimized in each step, and the total makespan will greatly influence the final reward in the terminal step.

4.3.2. Training with the mixing network

The training procedure is shown in Fig. 7, where agents $1, 2, \dots, n$ generate $Q^1, Q^2, \dots, Q^n$ after the observation encoding. We first concatenate $Q^1, Q^2, \dots, Q^n$ to form a Q^{total} and then calculate the loss with the TD method. The mixing network mix_θ with parameters θ is used to concatenate $Q^1, Q^2, \dots, Q^n$, which can be represented by [29]:

$$Q^{total} = mix_\theta(g, Q^1, Q^2, \dots, Q^n) \tag{15}$$

where $g = [v_1, v_2, \dots, v_k]$ is the global state. $v_i \in g$ is the one-hot vector $[0, 1, 1, 0, 1 \dots]$ denoting the currently available services in the enterprise E_i, so the global state changes dynamically during the scheduling. Calculating Q^{total} with the mixing network realizes the integration of $Q^1, Q^2, \dots, Q^n$ in a non-linear way, which improves the model fitting capability [46].

After obtaining the Q^{total}, the loss function can be calculated based on the TD learning method [47]:

$$L = \sum_{i=1}^{b} [(r + \gamma max_{u'} Q_{\phi^-}^{total} - -Q_\phi^{total})^2] \tag{16}$$

where b is the batch size of training samples, and r is the reward. ϕ, ϕ^- are two groups of weight parameters of the evaluation model and the target model, and $Q_{\phi^-}^{total}, Q_\phi^{total}$ are total Q-values calculated by these two models.

Based on Eq. (16), MAGCIS can be trained and the weight parameter W^l in the l layer can be updated by: $W^l \leftarrow W^l - \eta \Delta W^l$, where η is the learning rate and $\Delta W^l = \frac{\partial L}{\partial W^l}$. At the end of the backpropagation, we

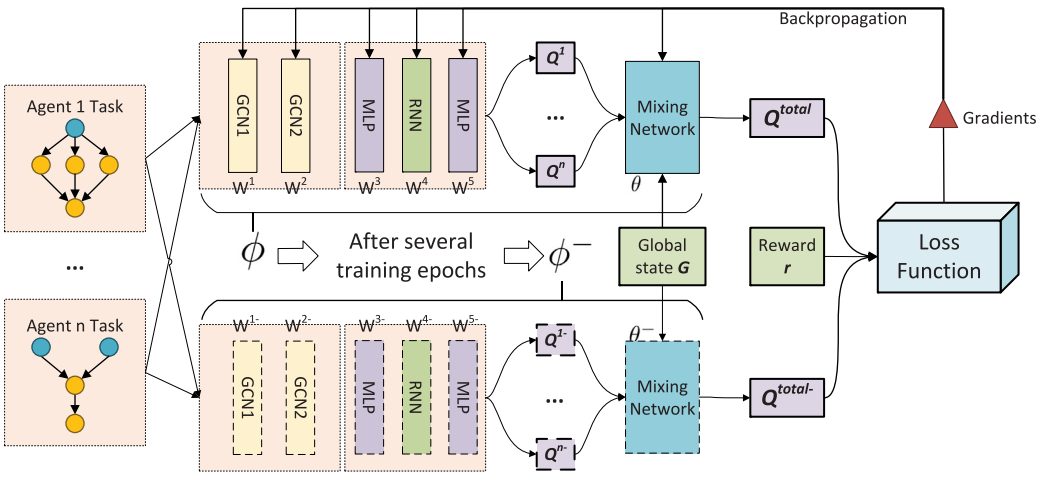

Fig. 7. The training framework. All agents in the MAS share the same group of weight parameters $\phi = \langle W^1, W^2, W^3, W^4, W^5, \theta \rangle$, and two groups of weight parameters ϕ, ϕ^- represent the evaluation model and the target model, respectively. The training process utilizes the reward, the Q^{total}, and the Q^{total-} to generate the loss for backpropagation. The training target is to learn parameters in ϕ, and ϕ will be copied to ϕ^- after several training epochs.

derive gradients of W^1, W^2 for the two layers of GCN. The gradient of $\frac{\partial L}{\partial W^2}$ is given as:

$$\frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial Q^{total}} \sum_{i=1}^{n} \frac{\partial Q^{total}}{\partial Q^i} \frac{\partial Q^i}{\partial H^{i,4}} \frac{\partial H^{i,4}}{\partial H^{i,3}} \frac{\partial H^{i,3}}{\partial H^{i,2}} \frac{\partial H^{i,2}}{\partial W^2} \tag{17}$$

$$= \sum_{i=1}^{n} \frac{\partial L}{\partial Q^{total}} \frac{\partial Q^{total}}{\partial Q^i} \frac{\partial Q^i}{\partial H^{i,4}} \frac{\partial H^{i,4}}{\partial H^{i,3}} \frac{\partial H^{i,3}}{\partial H^{i,2}} \frac{\partial H^{i,2}}{\partial W^2} \tag{18}$$

The gradient of $\frac{\partial L}{\partial W^1}$ is given as:

$$\frac{\partial L}{\partial W^1} = \sum_{i=1}^{n} \frac{\partial L}{\partial Q^{total}} \frac{\partial Q^{total}}{\partial Q^i} \frac{\partial Q^i}{\partial H^{i,4}} \frac{\partial H^{i,4}}{\partial H^{i,3}} \frac{\partial H^{i,3}}{\partial H^{i,2}} \frac{\partial H^{i,2}}{\partial H^{i,1}} \frac{\partial H^{i,1}}{\partial W^1} \tag{19}$$

Our proposal is trained based on the loss function in Eq. (16), and the procedure is shown in Fig. 7. We utilize the auto-derivation in PyTorch to calculate gradients in each layer.

4.3.3. Training tricks

A primary challenge for training MAGCIS is training efficiency. Although GCN obtains SOTA performance in multiple graph or node classification tasks, it has low training efficiency, especially when trained with RL [48]. For example, authors in Ref. [49] spent nearly five weeks training their GCN scheduling model with RL. We utilize four training tricks to improve the training efficiency.

Parameter sharing. n agents in the GSCMfg environment possess n policy models, and we utilize one group of weight parameters $\{W^l | 1 \leq l \leq 5\}$ to represent these policy models. It means that agents share the same policy model for training and executing. To distinguish agents, we utilize the one-hot vector $O^i = \{0, 0, 0, \ldots, 1, 0\}$ to identify the agent i and concatenate it with the graph embedding in the observation encoding.

Fine-tuning training. To accelerate the training, we first train a baseline model in the GSCMfg environment. Then, new models are trained based on this base model to adjust the weight parameters. This trick can reduce the early exploration process and promote convergence.

Graph integrating. The essence of GCN is to learn the aggregation importance relationship of adjacent nodes, but not the connection relationship [44]. However, different connection relationships represented by the adjacency matrix A may influence the training efficiency. We integrate all task graphs in the same batch into a large DAG graph and train the integrated one through MAGCIS.

Node reducing. In MAGCIS, the node is aggregated along the reverse direction of the executing sequence, and only unprocessed nodes are regarded as the GCN output. GCN does not calculate the parent nodes of the unprocessed nodes, so we remove them from the graph during training.

Table 2
Time consumption of services.

Service	FO	CM1	TU1	TU2	FG	TU3	CM2	CM3	HT
Time consume (hour)	9	10	14	5	7	3	4	9	13

5. Case study

In this case study, the processing of aircraft structural parts is considered in the CMfg platform, and the scheduling is consistent with that described in Section 3. In this case, subtasks need nine types of services totally, including FOrming (FO), CNC Milling-1 (CM1), TUrnery-1 (TU1), TUrnery-2 (TU2), ForGing (FG), TUrnery-3 (TU3), CNC Milling-2 (CM2), CNC Milling-3 (CM3), and Heat Treat (HT). The time consumption of these services can be found in Table 2. The processing time of the same type of service is consistent in different enterprises: The processing of the aircraft parts needs to follow stricter national standards, so there is little difference in the processing time in diverse enterprises. Besides, in GSCMfg scheduling problems, the processing abilities of enterprises in the same group are similar. There are nine types of tasks in total, and each task corresponds to a processing workflow of an aircraft structural part. These task types and their required services can be found in Table 3, where different types of tasks are numbered from Task1 to Task9. We utilize Python 3.7.4 programming language as the development platform for this case study. The simulation environment is event-triggered, where the state transition happens when some task has finished a subtask. The dynamic changing of service distributions occurs randomly before the state transition. A server computer with the configuration of Intel XEON Gold 6240 @3.8 Hz CPU, 128 GB RAM, and NVIDIA RTX 3080 GPU is applied to train models. The deep learning platform is PyTorch with CUDA 11, and the GCN is constructed based on the PyTorch Geometric library [50].

Other six MARL-based scheduling algorithms are executed for comparison, including QMIX [29], VDN [28], QTRAN_alt [30], QTRAN_base [30], REINFORCE [31], and Central_V [32]. We keep the definitions of observations, the reward, the global state, and actions of these six methods consistent with MAGCIS. For VDN, REINFORCE, and Central_V that have no global state, we add the global state information into the observation. Besides, other parameters that influence performance,

Table 3
Tasks of processing of aircraft structural parts.

No.	Name of aircraft structural parts	Processing stages				
		Stage 1	Stage 2	Stage 3	Stage 4	Stage 5
Task1	Rear-wheel axle	FG	TU1, TU2	CM3	FO	/
Task2	Front-wheel axle	FG	TU3, TU2	CM2	FO	/
Task3	Spoiler shaft	FG	TU3, TU1	HT	CM2	/
Task4	Flaps shaft	FG	TU3, TU1	CM1	HT	FO
Task5	Aileron shaft	FG	TU3, TU1, TU2	HT	CM2	FO
Task6	Elevator shaft	FG	TU3, TU2	HT	CM2	FO
Task7	Rudder shaft	FG	TU3, TU1, TU2	HT	CM2, CM1	FO
Task8	Rotating assembly of engine I	FG	TU1, TU2	HT	CM2, CM3, CM1	FO
Task9	Rotating assembly of engine II	FG	TU3, TU1, TU2	HT	CM2, CM3, CM1	FO

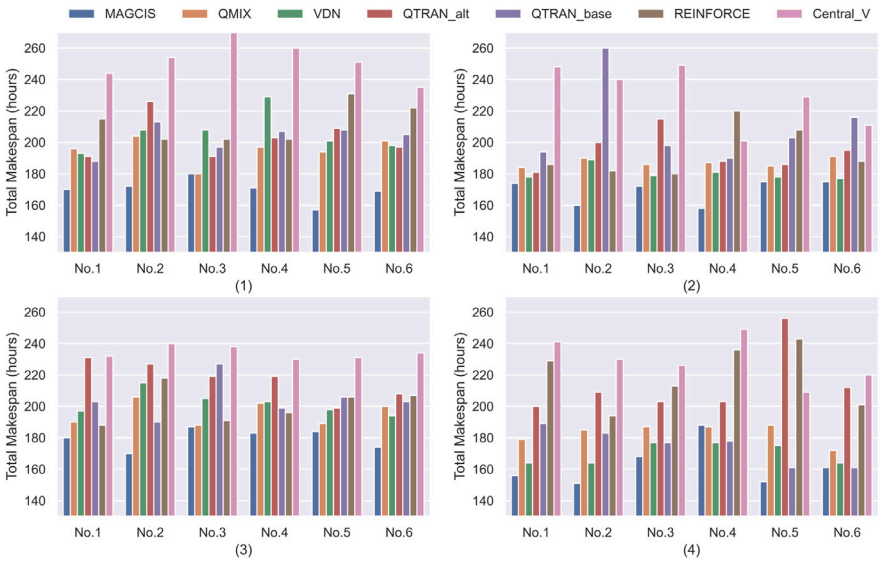

Fig. 8. Dynamic scheduling performances of seven MARL-based algorithms in the GSCMfg scheduling problem. (1)–(4) represent four different scenes, and six repeat experiments are executed in each environment to reduce randomness.

Table 4
Differences in four scenes.

Scenes	Number of enterprises	Initial service distribution	Task types	Task quantity
(1)	18	$\boldsymbol{\Phi}^1$	Task1–Task4	8
(2)	18	$\boldsymbol{\Phi}^2$	Task1–Task4	8
(3)	18	$\boldsymbol{\Phi}^1$	Task5–Task9	8
(4)	18	$\boldsymbol{\Phi}^2$	Task5–Task9	8

such as the training episodes and the learning rate, are also consistent in different algorithms.[2] In the next sections, we compare and analyze the experimental results from coarse to fine. In this section, the time units of all results are hours.

5.1. Comparison of dynamic scheduling performance

The dynamic scheduling performance is the primary metric to evaluate the overall performance of the scheduler. Cloud environments contain two changeable factors, including the initial service distribution and the DAG structures of tasks. To evaluate the scheduling performance in general, we set up four main scenes, as listed in Table 4. ϕ represents the enterprise's initial service distribution, and the explicit content of ϕ^1, ϕ^2 is described in Appendix. We use superscripts to distinguish different initial service distributions. Four types of tasks (Task1–Task4) are considered in scenes (1) and (2), and five types of tasks (Task5–Task9) are considered in scenes (3) and (4). Due to limitations of computing resources and computing capabilities, we set the task quantity as 8 in four scenes. This paper focuses on the second phase of the CMfg scheduling, where tasks are split into batches for scheduling. The task quantity of GSCMfg problems would be much less than that of CMfg problems, so 8 tasks in a batch can still be representative.

In all four scenes, each enterprise has a chance to change its current service distribution in each step, resulting in a dynamic environment. We utilize the total makespan M^{total} to measure the scheduling performance of the seven scheduling algorithms. A lower makespan means a better scheduling performance. To reduce the randomness, we train each algorithm in each scene for six models numbered $No.1$–$No.6$, as shown in Fig. 8 (1)–(4). Fig. 8 (1)–(4) indicate that the proposed MAGCIS (blue bars) outperforms the other six MARL-based scheduling algorithms in all four scenes. In scenes (1) and (3), QMIX shows the second satisfying results with an average makespan of 195.3 and 195.8, respectively. MAGCIS outperforms QMIX in these two scenes by 13.5% and 8.2%. In scenes (2) and (4), VDN shows the second best performance, and MAGCIS outperforms it by 6.3% and 4.4%. We also found that Central_V and REINFORCE always show the two worst performances in this problem. However, as the improvements of both QMIX and VDN, QTRAN_alt and QTRAN_base do not show competitive results. To conclude, MARL-based scheduling algorithms perform

[2] The detailed hyper-parameter settings can be found in the Appendix.

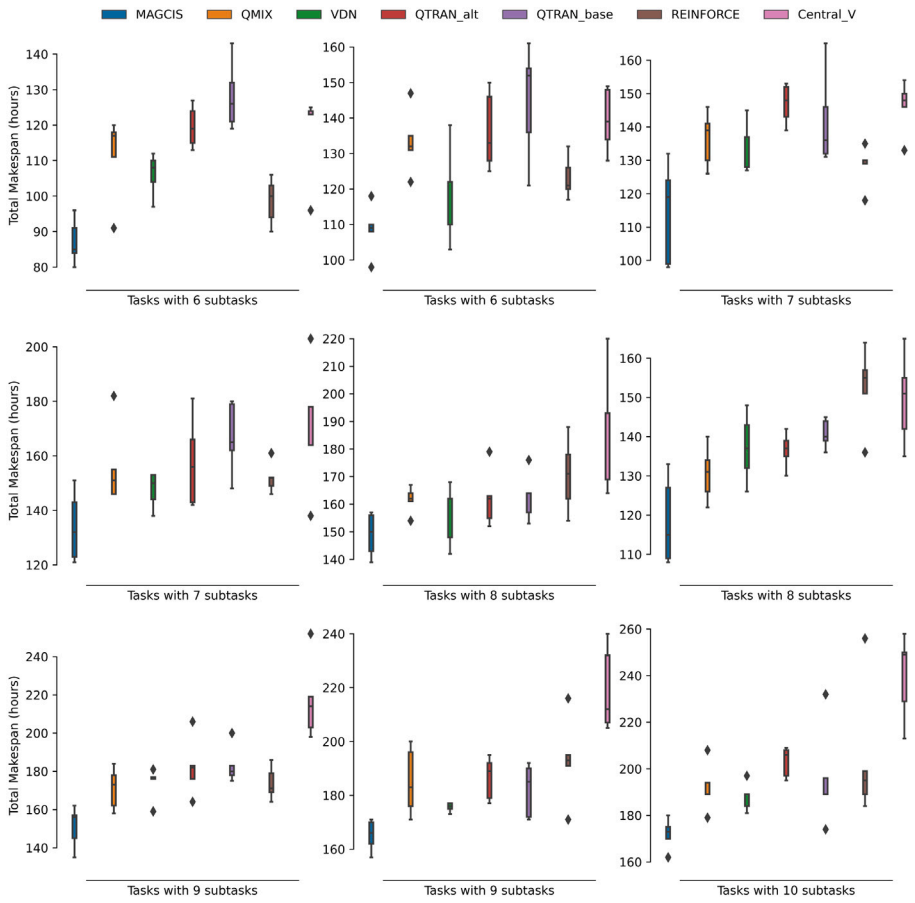

Fig. 9. Generalization performances of seven algorithms in the GSCMfg environment. The vertical axis is the total makespan. These nine subfigures represent nine types of tasks with different subtask quantities and structures, and all these tasks are unseen in training.

distinguishable performances, and the proposed MAGCIS outperforms others in all scenes in terms of the total makespan.

5.2. Generalization for different DAG task structures

Different DAG structures of tasks must be considered when applying the scheduling algorithms, especially structures unseen in training. So generalizability of trained models needs to be evaluated.[3] To measure the generalizability of different algorithms, we utilize trained models of these seven algorithms in the scene (1) as described in Table 4 to evaluate their executing results when facing nine groups of input tasks with different subtask quantities and structures. These nine groups of tasks form nine experiments as shown in Fig. 9. It is worth noticing that all the tasks in these nine experiments are unseen in training, meaning that they do not belong to the training data. We take five repeat implementations for each scheduling algorithm to reduce the random influence. The comparison results are shown in Fig. 9, and the box lines indicate that MAGCIS shows the lowest total makespan in all nine experiments. Besides, the disparity between MAGCIS and other scheduling algorithms is significant when the unseen task becomes complex. For example, the gap is nearly 15 h when the subtask quantity is 6, while the gap becomes almost 25 h when the subtask quantity reaches 10. So MAGCIS shows the competitive generalizability when facing different DAG task structures.

[3] Model generalizability means that a model trained in dataset1 can also has a positive performance in dataset2.

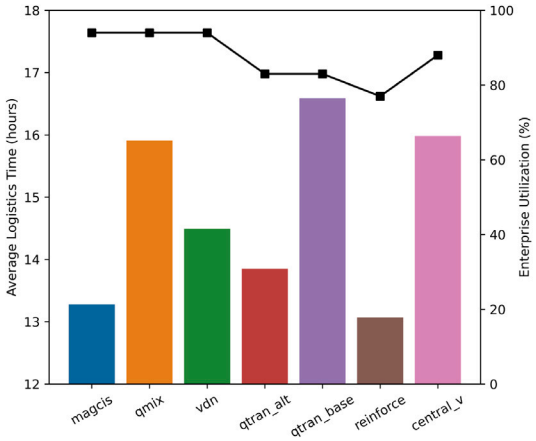

Fig. 10. ALT and EU of seven scheduling algorithms. Bar charts indicate the ALT, and the curve shows the EU.

5.3. Comparison and analysis in other dimensions

To dig deep into the reasons for the different performances of these seven MARL-based algorithms, we compare the detailed scheduling information in an episode. In this problem, the logistics time between two enterprises influences the scheduling efficiency, so average logistics time (ALT) is utilized to measure the scheduling efficiency:

$$ALT = \frac{Total\ \ Logistics\ \ Time\ \ (TLT)}{Total\ \ Logistics\ \ Number\ \ (TLN)} \tag{20}$$

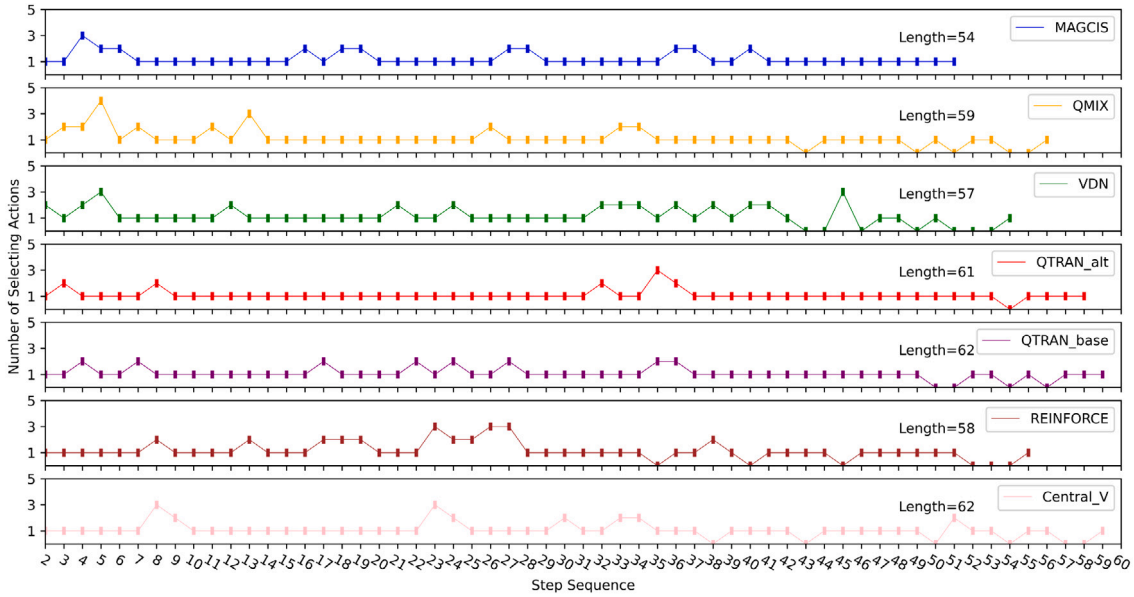

Fig. 11. NSA of the MAS controlled by different scheduling algorithms in an episode. The abscissa represents steps, and MASs controlled by different scheduling algorithms have diverse lengths of an episode.

A lower ALT usually means that less time is spent on transportation. However, a low ALT can also be caused by low enterprise utilization. For example, ALT will be extremely low when agents only choose nearby enterprises. Under this circumstance, we also measure the enterprise utilization (EU) of each algorithm, where EU is represented by:

$$EU = \frac{Number \quad of \quad Utilized \quad Enterprises}{Number \quad of \quad all \quad Enterprises} \tag{21}$$

As shown in Fig. 10, the ALT of MAGCIS is 13.3 h, lower than QMIX, VDN, QTRAN_alt, QTRAN_base, and Central_V. REINFORCE obtains the lowest ALT at 12.9 h, but its EU only reaches 77%. It means that REINFORCE only chooses nearby enterprises and leaves farther enterprises idle. Fig. 10 indicates that MAGCIS has both a low ALT and a high EU, verifying that MAGCIS has learned a more efficient scheduling policy than others.

Furthermore, we record the number of selecting actions (NSA) in an episode. Selecting actions represent effective actions, as defined in Eq. (11). A positive scheduling policy should ensure that each step contains at least one selecting action. On the contrary, $NSA = 0$ means that agents have to wait for available enterprises in this step, resulting in low scheduling efficiency. As shown in Fig. 11, we implement seven comparison experiments of seven algorithms in the same environment. MGCIS only takes 54 steps to finish the episode and gets $NSA > 0$ in each step, while others all possess steps where $NSA = 0$. The results indicate that our proposal has higher scheduling efficiency than others.

Furthermore, the objective M^{total} only represents the maximal makespan, which is the time consumption of the latest finished task. To compare the time consumption of all tasks during scheduling, we compare makespans of all agents, and the results are shown in Fig. 12. MAGCIS obtains the lowest makespans for all agents, as shown by the blue line in Fig. 12. The disparity between two nearby completed agents is also worth noticing for each algorithm. As shown in Fig. 13, we find that the interval time consumption between two nearby completed agents is relatively smooth in MAGCIS, which ranges in $[1, 10]$. The second smooth range $[3, 16]$ is in QMIX, and the worst one $[0, 30]$ is in VDN. Results in Fig. 13 indicate that MAGCIS optimizes the M^{total} by globally reducing the makespan of each agent simultaneously, so the scheduler obtains the 'wide' sight as mentioned in Section 3.2.

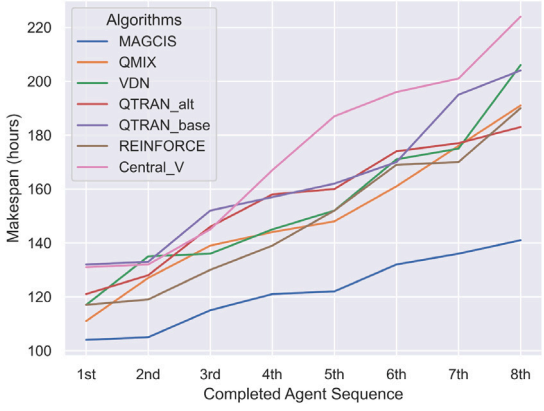

Fig. 12. Total makespan of all eight agents for completing their tasks. The horizontal axis represents the sequence of completed agents in the MAS. For example, 4th means the fourth completed agent consumes 120 h to finish its task.

5.4. One-step evaluation of execution procedure

We set up a targeted scene to show our proposed algorithm's detailed data transfer process. In this scene, only one task with five subtasks of $1, 2, 3, 4, 5$ needs to be assigned to two enterprises E^1, E^2. This task has four stages of $[1], [2, 3], [4], [5]$, and the services in E^1, E^2 are $[1, 2, 3]$ and $[1, 2, 3, 4, 5]$, respectively. The logistics time for delivering the task from the depot to E^1, E^2 are 2 and 10, respectively. The agent will complete its task faster if it chooses E^2 because E^1 cannot provide enough services. However, the low logistics time for choosing E^1 may lead the agent to choose E^1 if the agent only focuses on its next processing subtask 1 in the first stage. We set this one-step scene to (1) verify the forward-looking ability of MAGCIS and (2) show the action decision-making procedure with real matrixes and vectors.

As shown in Fig. 14, initial embeddings of subtasks $1, 2, 3, 4, 5$ are represented as a matrix X with the shape of $(4, 5)$. After two layers of GCNs that aggregate the initial embeddings from the child 5 to

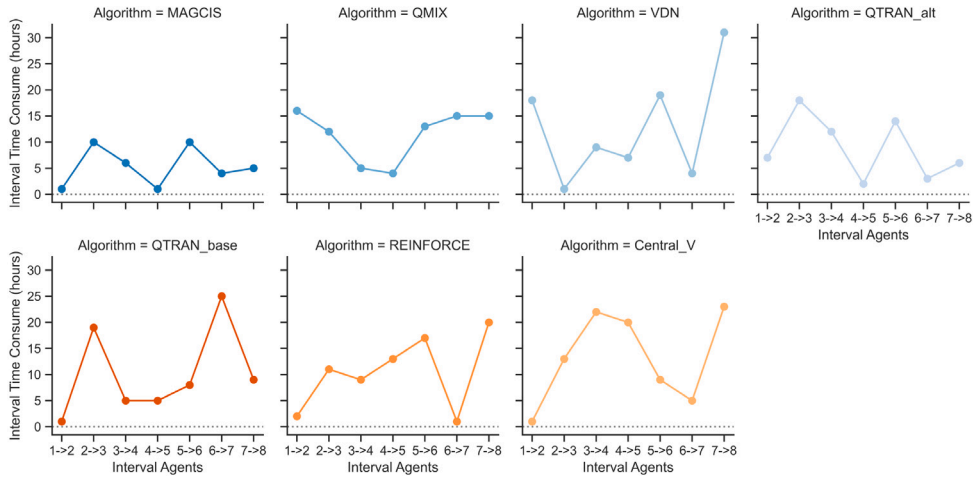

Fig. 13. Interval makespan between two agents completing the tasks. There are eight agents and seven agent intervals in each MAS.

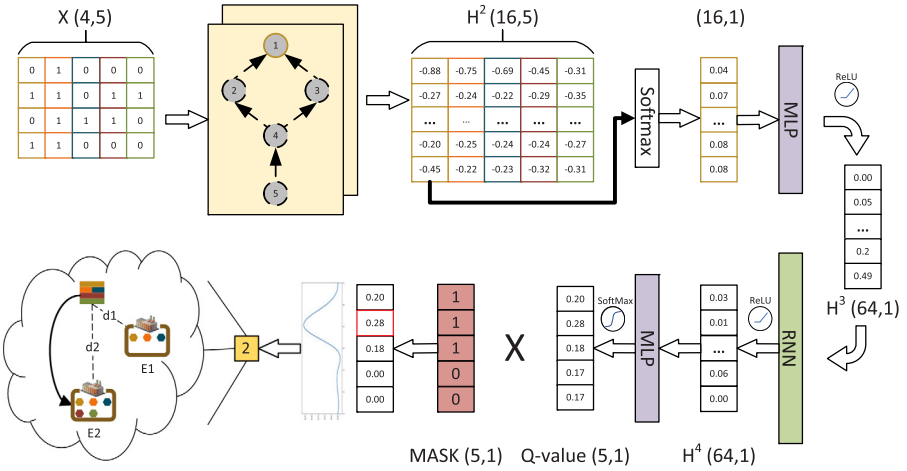

Fig. 14. The framework of the decision procedure of choosing the next enterprise based on the current observation. ReLU, SoftMax are the activation functions.

the root 1, we obtain the embeddings H^2 with the size of $(16, 5)$. Then, the embedding of the next processing node 1 in the task is delivered to MLP, RNN, and MLP in sequence, and *Q-value* is obtained as the output. The five values in *Q-value* correspond to the agent's selecting probability of five actions: selecting E^1, selecting E^2, waiting action, processing action, and terminal action. Finally, after multiplying $MASK$ and *Q-value*, we can obtain the final distribution and use the *argmax* operation to obtain the final action numbered 2. This one-step evaluation shows that MAGCIS possesses the forward-looking ability to consider all subtasks in its DAG sequence. It also indicates that MAGCIS has a 'deep' sight during the scheduling.

6. Conclusion

In this paper, we formulated the GSCMfg scheduling problem and proposed MAGCIS to solve this problem. In MAGCIS, each task is regarded as an agent, and a batch of tasks is modeled as the MAS. The multi-agent architecture shifts the complexity of action space to the difficulty of collaboration between agents. GCN and RNN are leveraged to encode the agent observation, which can extract DAG structures of tasks and record the trajectories of task processing procedures, respectively. The reward function is defined to lead MAGCIS towards convergence, and the proposed model is trained with the mixing network under a CTDE architecture. The aircraft structural part processing case study indicated that MAGCIS outperformed the other six MARL algorithms in scheduling performance and generalizability. MAGCIS can also be applied to other scheduling environments similar to the GSCMfg environment, and this paper has provided sufficient information about how to train and execute it.

Our future works will focus on two aspects: (1) Although MAGCIS provides positive results and some training tricks are adopted, the training efficiency still needs to be improved. GCN requires a high computation cost, and the training of MARL needs both CPU and GPU. So we will conduct experiments based on distributed computing devices and evaluate the scheduling performance when scheduling decades or hundreds of tasks. (2) Because the essence of MAGCIS is a knowledge-based scheduler, more data about task DAG structures and human evaluations of scheduling results are needed to improve the algorithm performance.

Table A.5
Service distribution.

Enterprise No.	Φ^1					Φ^2				
E1	TU3	CM2	CM3	HT		TU1	TU2	FG	TU3	CM2
E2	FO	CM1	TU1	TU2	FG	FO	CM1	CM3	HT	
E3	TU3	CM2	CM3	HT		TU1	TU2	FG	TU3	CM2
E4	FO	CM1	TU1	TU2	FG	FO	CM1	CM3	HT	
E5	FO	TU1	FG	CM2	HT	TU1	TU2	FG	TU3	CM2
E6	CM1	TU2	TU3	CM3		FO	CM1	CM3	HT	
E7	FO	CM1	TU1	CM3	HT	TU1	TU2	FG	TU3	CM2
E8	TU2	FG	TU3	CM2		FO	CM1	CM3	HT	
E9	FO	CM1	TU1	CM3	HT	TU2	FG	TU3	CM2	
E10	TU2	FG	TU3	CM2		FO	CM1	TU1	CM3	HT
E11	FO	CM1	TU1	CM3	HT	TU2	FG	TU3	CM2	
E12	TU2	FG	TU3	CM2		FO	CM1	TU1	CM3	HT
E13	FO	CM1	TU1	CM3	HT	TU2	FG	TU3	CM2	
E14	TU2	FG	TU3	CM2		FO	CM1	TU1	CM3	HT
E15	FO	CM1	TU1	CM3	HT	CM1	TU1	TU2	FG	
E16	TU2	FG	TU3	CM2		FO	TU3	CM2	CM3	HT
E17	FO	CM1	TU1	CM3	HT	CM1	TU1	TU2	FG	
E18	TU2	FG	TU3	CM2		FO	TU3	CM2	CM3	HT

Table A.6
Parameter settings of different algorithms.

	MAGCIS	QMIX	VDN	QTRAN_alt	QTRAN_base	REINFORCE	Central_V
episode	4e5	4e5	4e5	4e5	4e5	4e5	4e5
learning rate	5e−4	5e−4	5e−4	5e−4	5e−4	/	/
learning rate actor	/	/	/	/	/	1e−4	1e−5
learning rate critic	/	/	/	/	/	1e−3	1e−4
td_lambda	/	/	/	/	/	0.8	/
epsilon	1	1	1	1	1	/	/
min_epsilon	0.05	0.05	0.05	0.05	0.05	0.5	0.02
gcn_hidden_dim	10	/	/	/	/	/	/
rnn_hidden_dim	64	64	64	64	64	64	64
critic_dim	/	/	/	/	/	128	128
mix_hidden_dim	32	32	32	32	32	/	/
hyper_hidden_dim	64	64	64	/	/	/	/
qtran_hidden_dim	/	/	/	64	64	/	/
evaluate_cycle	1e3	1e3	1e3	1e3	1e3	1e3	1e3
batch_size	32	32	32	32	32	/	/
buffer_size	5e3	5e3	5e3	5e3	5e3	/	/
target_update_cycle	200	200	200	200	200	200	200
lambda_opt	/	/	/	1	1	/	/
lambda_nopt	/	/	/	1	1	/	/
grad_norm_clip	10	10	10	10	10	10	10

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by the National Key R&D Program Funded Projects of China (2020AAA0109202) and by the National Natural Science Foundation of China (NSFC) under Grant Nos. 61873014 and 61973243.

Appendix A. Experimental settings

In the case study, two initial service distributions Φ^1, Φ^2 are reported in Table A.5. The initial service distributions represent types of services owned by each enterprise before scheduling. During the scheduling, the service distributions might be changed due to the dynamics, but the fluctuation range will keep any enterprise at least two services.

The hyper-parameter settings of MAGCIS, QMIX, VDN, QTRAN_alt, QTRAN_base, REINFORCE, and Central_V are listed in Table A.6. We keep the common hyper-parameters consistent to provide a fair comparison.

Appendix B. Training curves

We train each algorithm adequately to avoid model underfitting. The original training curves of the seven MARL algorithms are shown in Fig. B.15. The ordinate in Fig. B.15 represents the accumulated episode reward, which reflects the accumulated reward obtained by the MAS in an episode. Curves fluctuate significantly due to the characteristics of MARL algorithms. A higher curve means better performance as all algorithms utilize the same reward function defined in Eq. (14).

The accumulated episode reward of each algorithm converges to a relatively stable value after 400000 episode training.

Appendix C. Numerical results

Original results shown in Figs. 8 and 9 are listed in Table C.7 and Table C.8 respectively.

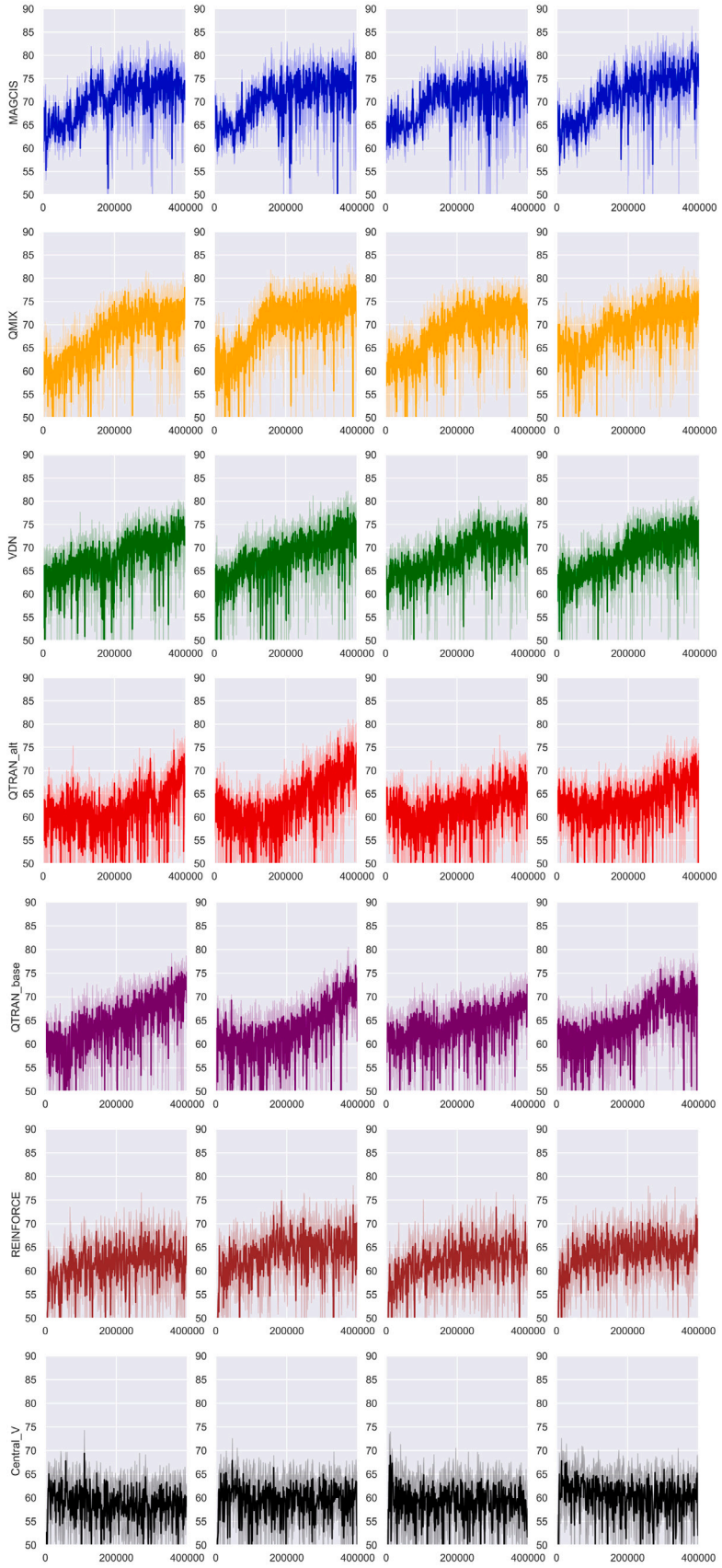

Fig. B.15. Training curves of MAGCIS, QMIX, VDN, QTRAN_alt, QTRAN_base, REINFORCE, and Central_V. Each algorithm is trained in four scenes as described in Table 4.

Table C.7
Numerical results in Section 5.1.

Algorithms	Scenes	No. 1	No. 2	No. 3	No. 4	No. 5	No. 6	Average
MAGCIS	Env(1)	170	172	180	171	157	169	**169.8**
MAGCIS	Env(2)	174	160	172	158	175	175	**169.0**
MAGCIS	Env(3)	180	170	187	183	184	174	**179.7**
MAGCIS	Env(4)	156	151	168	188	152	161	**162.7**
QMIX	Env(1)	196	204	180	197	194	201	195.3
QMIX	Env(2)	184	190	186	187	185	191	187.2
QMIX	Env(3)	190	206	188	202	189	200	195.8
QMIX	Env(4)	179	185	187	187	188	172	183.0
VDN	Env(1)	193	208	208	229	201	198	206.2
VDN	Env(2)	178	189	179	181	178	177	180.3
VDN	Env(3)	197	215	205	203	198	194	202.0
VDN	Env(4)	164	164	177	177	175	164	170.2
QTRAN_alt	Env(1)	191	226	191	203	209	197	202.8
QTRAN_alt	Env(2)	181	200	215	188	186	195	194.2
QTRAN_alt	Env(3)	231	227	219	219	199	208	217.2
QTRAN_alt	Env(4)	200	209	203	203	256	212	213.8
QTRAN_base	Env(1)	188	213	197	207	208	205	203.0
QTRAN_base	Env(2)	194	260	198	190	203	216	210.2
QTRAN_base	Env(3)	203	190	227	199	206	203	204.7
QTRAN_base	Env(4)	189	183	177	178	161	161	174.8
REINFORCE	Env(1)	215	202	202	202	231	222	212.3
REINFORCE	Env(2)	186	182	180	220	208	188	194.0
REINFORCE	Env(3)	188	218	191	196	206	207	201.0
REINFORCE	Env(4)	229	194	213	236	243	201	219.3
Central_V	Env(1)	244	254	277	260	251	235	253.5
Central_V	Env(2)	248	240	249	201	229	211	229.7
Central_V	Env(3)	232	240	238	230	231	234	234.2
Central_V	Env(4)	241	230	226	249	209	220	229.2

Table C.8
Numerical results in Section 5.2.

Scheduling Algorithm	TN=6	TN=6	TN=7	TN=7	TN=8	TN=8	TN=9	TN=9	TN=10
MAGCIS	85	109	**98**	143	157	109	145	157	180
MAGCIS	**80**	110	119	132	**139**	133	**157**	166	175
MAGCIS	84	118	99	123	156	**108**	162	170	**162**
MAGCIS	91	**98**	124	**121**	150	127	**135**	171	170
MAGCIS	96	108	132	151	143	115	156	162	173
QMIX	120	122	141	151	164	122	162	171	189
QMIX	91	132	126	155	154	126	158	176	179
QMIX	117	147	130	146	161	131	173	183	194
QMIX	111	135	146	146	162	140	184	200	189
QMIX	118	131	139	182	167	134	178	196	208
VDN	97	110	145	153	162	148	181	177	184
VDN	104	138	127	144	142	143	176	175	184
VDN	110	122	128	138	168	137	176	177	181
VDN	108	103	137	153	162	132	177	173	189
VDN	112	122	137	150	148	126	159	176	197
QTRAN_alt	119	146	143	181	179	137	206	195	209
QTRAN_alt	115	150	153	166	155	139	182	192	197
QTRAN_alt	127	133	148	143	163	130	176	189	208
QTRAN_alt	113	125	139	156	152	135	164	179	206
QTRAN_alt	124	128	152	142	162	142	183	177	195
QTRAN_base	132	152	146	162	176	144	175	190	196
QTRAN_base	119	154	165	165	157	139	180	185	174
QTRAN_base	126	136	136	179	153	136	200	171	189
QTRAN_base	121	161	132	180	164	145	183	192	189
QTRAN_base	143	121	131	148	164	140	178	172	232
REINFORCE	103	117	135	161	188	157	164	216	199
REINFORCE	94	132	129	152	171	164	169	171	256
REINFORCE	90	120	130	149	162	136	186	193	195
REINFORCE	100	126	130	146	154	155	171	195	189
REINFORCE	106	121	118	152	178	151	179	191	184
Central_V	96	134	148	178	193	165	198	205	229
Central_V	125	148	150	164	220	142	203	207	213
Central_V	123	149	146	138	169	135	214	212	258
Central_V	123	139	154	203	164	155	252	240	250
Central_V	124	128	133	178	193	151	219	232	249

References

[1] Li B, Zhang L, Wang S, Tao F, Cao J, Jiang X, et al. Cloud manufacturing: a new service-oriented networked manufacturing model. Comput Integr Manuf Syst 2010;16(1):1–7.

[2] Zhang L, Luo Y, Tao F, Li BH, Ren L, Zhang X, et al. Cloud manufacturing: a new manufacturing paradigm. Enterp Inf Syst 2014;8(2):167–87.

[3] Li F, Liao T, Zhang L. Two-level multi-task scheduling in a cloud manufacturing environment. Robot Comput-Integr Manuf 2019;56:127–39.

[4] Zhao C, Zhang L, Liu Y, Zhang Z, Yang G, Li BH. Agent-based simulation platform for cloud manufacturing. Int J Model Simul Sci Comput 2017;8(03):1742001.

[5] Laili Y, Lin S, Tang D. Multi-phase integrated scheduling of hybrid tasks in cloud manufacturing environment. Robot Comput-Integr Manuf 2020;61:101850.

[6] Zhou L, Zhang L. A dynamic task scheduling method based on simulation in cloud manufacturing. In: Theory, methodology, tools and applications for modeling and simulation of complex systems. Springer; 2016, p. 20–4.

[7] Zhou L, Zhang L, Ren L, Wang J. Real-time scheduling of cloud manufacturing services based on dynamic data-driven simulation. IEEE Trans Ind Inf 2019;15(9):5042–51.

[8] Wei Y, Tian L. Research on cloud design resources scheduling based on genetic algorithm. In: 2012 international conference on systems and informatics. IEEE; 2012, p. 2651–6.

[9] Bello I, Pham H, Le QV, Norouzi M, Bengio S. Neural combinatorial optimization with reinforcement learning. 2016, arXiv preprint arXiv:1611.09940.

[10] Mazyavkina N, Sviridov S, Ivanov S, Burnaev E. Reinforcement learning for combinatorial optimization: A survey. Comput Oper Res 2021;105400.

[11] Liang H, Wen X, Liu Y, Zhang H, Zhang L, Wang L. Logistics-involved qos-aware service composition in cloud manufacturing with deep reinforcement learning. Robot Comput-Integr Manuf 2021;67:101991.

[12] Zhu H, Li M, Tang Y, Sun Y. A deep-reinforcement-learning-based optimization approach for real-time scheduling in cloud manufacturing. IEEE Access 2020;8:9987–97.

[13] Baer S, Bakakeu J, Meyes R, Meisen T. Multi-agent reinforcement learning for job shop scheduling in flexible manufacturing systems. In: 2019 second international conference on artificial intelligence for industries. IEEE; 2019, p. 22–5.

[14] Halty A, Sánchez R, Vázquez V, Viana V, Piñeyro P, Rossit DA. Scheduling in cloud manufacturing systems: Recent systematic literature review. American Institute of Mathematical Sciences; 2020.

[15] Liu Y, Wang L, Wang XV, Xu X, Zhang L. Scheduling in cloud manufacturing: state-of-the-art and research challenges. Int J Prod Res 2019;57(15–16):4854–79.

[16] Li F, Zhang L, Liu Y, Laili Y. Qos-aware service composition in cloud manufacturing: A gale–Shapley algorithm-based approach. IEEE Trans Syst Man Cybern Syst 2018;50(7):2386–97.

[17] Li F, Zhang L, Liao TW, Liu Y. Multi-objective optimisation of multi-task scheduling in cloud manufacturing. Int J Prod Res 2019;57(12):3847–63.

[18] Jian C, Wang Y. Batch task scheduling-oriented optimization modelling and simulation in cloud manufacturing. Int J Simul Model 2014;13(1):93–101.

[19] Chen J, Huang GQ, Wang J-Q, Yang C. A cooperative approach to service booking and scheduling in cloud manufacturing. European J Oper Res 2019;273(3):861–73.

[20] Jian C, Ping J, Zhang M. A cloud edge-based two-level hybrid scheduling learning model in cloud manufacturing. Int J Prod Res 2020;1–15.

[21] Zhou L, Zhang L, Fang Y. Logistics service scheduling with manufacturing provider selection in cloud manufacturing. Robot Comput-Integr Manuf 2020;65:101914.

[22] Sutton RS, Barto AG, et al. Introduction to reinforcement learning, Vol. 135. MIT press Cambridge; 1998.

[23] Arulkumaran K, Deisenroth MP, Brundage M, Bharath AA. Deep reinforcement learning: A brief survey. IEEE Signal Process Mag 2017;34(6):26–38.

[24] Hernandez-Leal P, Kartal B, Taylor ME. A survey and critique of multiagent deep reinforcement learning. Auton Agents Multi-Agent Syst 2019;33(6):750–97.

[25] Zhang K, Yang Z, Başar T. Multi-agent reinforcement learning: A selective overview of theories and algorithms. Handb Reinf Learn Control 2021;321–84.

[26] Tan M. Multi-agent reinforcement learning: Independent vs. Cooperative agents. In: Proceedings of the tenth international conference on machine learning. 1993, p. 330–7.

[27] Du W, Ding S. A survey on multi-agent deep reinforcement learning: from the perspective of challenges and applications. Artif Intell Rev 2021;54(5):3215–38.

[28] Sunehag P, Lever G, Gruslys A, Czarnecki WM, Zambaldi V, Jaderberg M, et al. Value-decomposition networks for cooperative multi-agent learning. 2017, arXiv preprint arXiv:1706.05296.

[29] Rashid T, Samvelyan M, Schroeder C, Farquhar G, Foerster J, Whiteson S. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In: International conference on machine learning. PMLR; 2018, p. 4295–304.

[30] Son K, Kim D, Kang WJ, Hostallero DE, Yi Y. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In: International conference on machine learning. PMLR; 2019, p. 5887–96.

[31] Zhang J, Kim J, O'Donoghue B, Boyd S. Sample efficient reinforcement learning with REINFORCE. 2020, arXiv preprint arXiv:2010.11364.

[32] Foerster J, Farquhar G, Afouras T, Nardelli N, Whiteson S. Counterfactual multi-agent policy gradients. In: Proceedings of the AAAI conference on artificial intelligence, Vol. 32. (1). 2018.

[33] Mao H, Schwarzkopf M, Venkatakrishnan SB, Meng Z, Alizadeh M. Learning scheduling algorithms for data processing clusters. In: Proceedings of the acm special interest group on data communication. 2019, p. 270–88.

[34] Dong T, Xue F, Xiao C, Zhang J. Workflow scheduling based on deep reinforcement learning in the cloud environment. J Ambient Intell Humaniz Comput 2021;1–13.

[35] Dong T, Xue F, Xiao C, Li J. Task scheduling based on deep reinforcement learning in a cloud manufacturing environment. Concurr Comput: Pract Exper 2020;32(11):e5654.

[36] Wei Y, Kudenko D, Liu S, Pan L, Wu L, Meng X. A reinforcement learning based workflow application scheduling approach in dynamic cloud environment. In: International conference on collaborative computing: networking, applications and worksharing. Springer; 2017, p. 120–31.

[37] Chen S, Fang S, Tang R. A reinforcement learning based approach for multi-projects scheduling in cloud manufacturing. Int J Prod Res 2019;57(10):3080–98.

[38] Lu R, Li Y-C, Li Y, Jiang J, Ding Y. Multi-agent deep reinforcement learning based demand response for discrete manufacturing systems energy management. Appl Energy 2020;276:115473.

[39] Roesch M, Linder C, Zimmermann R, Rudolf A, Hohmann A, Reinhart G. Smart grid for industry using multi-agent reinforcement learning. Appl Sci 2020;10(19):6900.

[40] Wang J, Sun L. Dynamic holding control to avoid bus bunching: A multi-agent deep reinforcement learning framework. Transp Res C 2020;116:102661.

[41] Wang Y, Liu H, Zheng W, Xia Y, Li Y, Chen P, et al. Multi-objective workflow scheduling with deep-q-network-based multi-agent reinforcement learning. IEEE Access 2019;7:39974–82.

[42] Feng W, Yin C, Li X, Li L. A classification matching method for manufacturing resource in cloud manufacturing environment. Int J Model Simul Sci Comput 2017;8(02):1750057.

[43] Wu Z, Pan S, Chen F, Long G, Zhang C, Philip SY. A comprehensive survey on graph neural networks. IEEE Trans Neural Netw Learn Syst 2020;32(1):4–24.

[44] Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. 2016, arXiv preprint arXiv:1609.02907.

[45] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, et al. Human-level control through deep reinforcement learning. Nature 2015;518(7540):529–33.

[46] Samvelyan M, Rashid T, De Witt CS, Farquhar G, Nardelli N, Rudner TG, et al. The starcraft multi-agent challenge. 2019, arXiv preprint arXiv:1902.04043.

[47] Taylor ME, Stone P. Behavior transfer for value-function-based reinforcement learning. In: Proceedings of the fourth international joint conference on autonomous agents and multiagent systems. 2005, p. 53–9.

[48] Jiang J, Dun C, Huang T, Lu Z. Graph convolutional reinforcement learning. 2018, arXiv preprint arXiv:1810.09202.

[49] Seito T, Munakata S. Production scheduling based on deep reinforcement learning using graph convolutional neural network. In: ICAART, Vol. 2. 2020, p. 766–72.

[50] Fey M. Towards Effective Graph Representation Learning. Technical Report for Collaborative Research Center SFB 876 Providing Information By Resource-Constrained Data Analysis, 2019, p. 47.