Journal of Manufacturing Systems 65 (2022) 452–468

Contents lists available at ScienceDirect

Journal of Manufacturing Systems

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

Solving task scheduling problems in cloud manufacturing via attention mechanism and deep reinforcement learning

Xiaohan Wang [a], Lin Zhang [a,*], Yongkui Liu [b], Chun Zhao [c], Kunyu Wang [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] Computer School, Beijing Information Science and Technology University, Beijing 100101, China

ARTICLE INFO

Keywords:
Scheduling problem
Deep reinforcement learning
Transformer
Multi-head attention
Cloud manufacturing

ABSTRACT

Cloud manufacturing (CMfg) offers a cloud platform for both consumers and providers, and allocating consumer tasks to service providers requires many-to-many scheduling. Deep reinforcement learning (DRL) has been gradually employed to solve CMfg scheduling problems and has achieved satisfactory performance in dynamic and uncertain cloud environments. Nonetheless, we need better scheduling algorithms and more accessible modeling methods to enable practical implementation. In this study, an end-to-end scheduling algorithm is proposed to address task scheduling problems in CMfg. The proposal extracts intercorrelations in enterprise–enterprise and enterprise–task with the multi-head attention mechanism and is trained by DRL. Our proposal has extremely low time-response compared to heuristic algorithms and can generate a scheduling solution in seconds. In contrast to other DRL algorithms, our proposal can achieve better scheduling performances and uses a more accessible modeling method: the objective function alone is sufficient to train the model stably, without the need for a step-based reward function. Applying multi-head attention and DRL to scheduling problems is an exploratory attempt to achieve positive results. Experimental results on a case of automobile structure part processing in CMfg indicated that our proposal showed competitive scheduling performances and running time compared to eight DRL algorithms, two heuristic algorithms, and two priority dispatching rules. Besides, the results proved that our proposal's generalizability and scalability were better than the other eight DRL algorithms.

1. Introduction

Cloud manufacturing (CMfg), a manufacturing mode originating from cloud computing and grid computing, provides a cloud platform to integrate manufacturing resources among manufacturing enterprises and gather consumer requirements [1,2]. Manufacturing resources are encapsulated into services and released in the cloud platform so that products can be processed with services distributed in different regions. CMfg overcomes the limitation of physical distances and demonstrates convenience for production and manufacture. Meanwhile, massive manufacturing services in cloud platforms pose a scheduling challenge to CMfg managers [3]. Consumers' manufacturing tasks require different types of manufacturing services. These services are located in various regions; therefore, the processing needs to consider the service distribution among enterprises and logistics. Additionally, the processing time, cost, and reliability of services of the same type may be diverse in different enterprises; therefore, selecting an optimal processing sequence with high-quality services is necessary. Under these circumstances, an effective scheduling algorithm that minimizes the total makespan and cost and maximizes reliability is of exceptional significance [4,5]. In contrast to other scheduling problems, such as the job shop scheduling problem (JSSP), the cloud manufacturing scheduling problem (CMfg-SP) has the characteristics of openness, dynamics, and uncertainty, which increases the requirements for scheduling algorithms [6–8]. Hence, a scheduling algorithm with satisfactory scheduling performance, time-respond, generalizability, and scalability is urgently required.

As a critical issue for realizing optimal assignments of manufacturing services, CMfg-SP has attracted considerable research interest in academia and industry [4]. Over the past decades, the most widely applied methods for CMfg-SPs have been priority dispatching rules (PDRs) and heuristic algorithms [5]. PDRs are scheduling rules based on priorities, such as first-in-first-out (FIFO). They are typically designed by experienced experts and are widely employed in various CMfg systems [9]. PDRs require a short computing time and are suitable

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

https://doi.org/10.1016/j.jmsy.2022.08.013
Received 19 February 2022; Received in revised form 19 August 2022; Accepted 29 August 2022
Available online 18 October 2022

for dynamic scheduling [10]. However, a satisfactory PDR usually requires careful design, and new PDRs need to be redesigned when facing an unseen problem [11]. Besides, scheduling performances of PDRs are easily affected by scenario distribution, and a well-defined PDR that achieves positive performance in one scenario may generate disappointing results in another. In this paper, the scheduling performance refers to the performance of the objective that needs to be optimized, and the objective for CMfg-SPs is usually to maximize the quality of service (QoS). Heuristic algorithms decompose the scheduling targets into subproblems and solve each subproblem through optimization algorithms [12]. Commonly used optimization algorithms include ant colony optimization (ACO) [13], genetic algorithms (GAs) [14], and non-dominated sorting genetic algorithm-II (NSGA-II) [15]. Heuristic algorithms usually obtain more accurate results than PDRs and rely less heavily on handcrafted data [16]. However, they are relatively more time-consuming than PDRs, and recalculation is required when the scenario changes. Heuristic algorithms may face challenges when applied to CMfg-SPs that need scheduling algorithms to be fast-responding and generalized [17]. Additionally, solving CMfg-SPs with exact methods (such as integer programming) is impractical. One reason is that complicated constraints and conditions in CMfg-SPs are non-trivial to be represented by mathematical equations. Another reason is that even though an exact method is established, solving time complexity on CMfg-SPs would be extremely high. These two reasons are why most researchers adopt PDRs and heuristic algorithms for CMfg-SPs rather than exact methods.

With the development of artificial intelligence, deep reinforcement learning (DRL) has gradually been applied to solve multiple combinatorial optimization problems [18–20]. In model-free environments, DRL is modeled as a Markov decision process (MDP), where the agent learns to schedule tasks automatically by trial and error based on its knowledge of the CMfg environment [21]. Recently, some studies have focused on using DRL algorithms to solve CMfg-SPs and have achieved positive results [20,22–24]. The advantages of DRL algorithms include three main aspects: little handcrafting, high generalizability, and low running time [25]. For DRL-based scheduling algorithms, less additional hand engineering is required after they have been constructed and deployed to the CMfg platform. The algorithm automatically learns the scheduling policy through interaction with the CMfg environment, and the model trained in one scenario can be reused in other scenarios. Owing to the powerful fitting ability of neural networks, DRL algorithms can generate satisfactory solutions with low time consumption. They have shown potential generalizability when facing dynamic and uncertain scheduling environments [26]. However, despite these advantages, DRL algorithms have not become the leading solutions for addressing CMfg-SPs. The current issues in applying DRL algorithms to CMfg-SPs are as follows: (1) Most DRL algorithms are step-based, and their performance depends heavily on the design of rewards and states. Proper design can achieve positive scheduling performance but requires an engineer to be proficient in DRL algorithms and manufacturing. (2) There are multiple intercorrelations between enterprise–enterprise and enterprise–task. Although deep neural networks provide the opportunity to extract these complex features, they struggle to effectively capture these intercorrelations simultaneously [4]. Besides, inputting all these features into a neural network may face challenges such as the high-dimension inputs. Nonetheless, many DRL scheduling algorithms still represent these features with deep neural networks, resulting in the insufficient feature extraction issue. This issue leads to poor scheduling performances, so more effective deep learning models are needed to represent the features of CMfg-SPs.

Definition 1 (*Step-based*)**.** The step-based DRL model generates a solution to the problem in steps, and each step requires a real-time state representation and reward function.

Definition 2 (*End-to-end*)**.** The end-to-end DRL model produces a solution to the problem in steps or at once, and it only needs a total objective for training.

With the motivation to resolve the two issues mentioned above and propose an effective scheduling algorithm with satisfactory scheduling performance, running time, generalizability, and scalability, we leverage the transformer and DRL to solve CMfg-SPs aiming at maximizing the QoS. Transformer is an effective deep learning architecture that has been widely adopted by various state-of-the-art (SOTA) models, such as BERT in natural language processing (NLP) [27] and swin transformer in computer vision (CV) [28]. Transformer extracts complex features by the multi-head attention (MHA) mechanism. Kool et al. utilized a transformer to solve vehicle routing problems (VRPs) and obtained SOTA results [29]. Inspired by them, we use the transformer to extract the features of enterprises and train the model using DRL. The contributions of this study are as follows: (1) CMfg-SP is modeled and formulated as an MDP. This formulation allows CMfg-SP to be represented by a transformer and trained using DRL algorithms. (2) An end-to-end scheduling model that does not require detailed definitions of the reward and state is proposed. It is an exploratory attempt to apply both the transformer and DRL to CMfg-SPs or even to the entire scope of scheduling problems. (3) We utilize the MHA on a graph representation to extract intercorrelations among enterprises in the encoder. Task features are integrated into these enterprise embeddings with MHA in the decoder. We adopt the action mask to ensure that selected actions in the decoder are reasonable. The entire model is trained using the DRL algorithm named REINFORCE with a baseline. (4) We conduct experiments based on the case of automobile structure parts in CMfg-SPs and compare the proposal with eight DRL algorithms, two heuristic algorithms, and two PDRs. Experimental results indicated that our proposal showed competitive scheduling performances and the fastest running time. The results also proved that our proposal's generalizability and scalability were better than the other eight DRL algorithms.

In contrast to other classic combinatorial optimization problems such as VRP, CMfg-SP is a practical scheduling problem with complicated procedures and diverse constraint conditions. We take the following improvements based on the previous attention model: (1) To capture intercorrelations among enterprises, we concatenate features of locations, service time, service cost, and processing efficiency in an enterprise as an input vector for the encoder. Before delivering this vector to the encoder, we normalize it by the layer normalization operation and then convert it to an initial embedding through neural networks. (2) Before the decoding, we construct a context node based on task features and other dynamic information. Then, enterprise embeddings from the encoder are integrated into the context node by MHA to form the context embedding. This method ensures that the decoder considers comprehensive information when generating scheduling solutions. (3) We formulate a masking scheme for CMfg-SPs in the output of the decoder, which prevents the decoder from selecting infeasible enterprises. (4) To represent the situation where no feasible action can be selected in the decoder, we set a dummy location with a fixed coordinate. The decoder will select this dummy location if there is no available enterprise.

The remainder of the paper is organized as follows: Section 2 presents a brief literature review of DRL scheduling algorithms. Section 3 shows the modeling and formulation of CMfg-SPs. The proposed model is described in Section 4. Comparative Experiments of DRL scheduling algorithms based on a case of automobile structure parts are conducted in Section 5 before concluding the paper in Section 6.

2. Literature review

This section briefly reviews related works of solving CMfg-SPs with DRL algorithms. As CMfg-SPs belong to scheduling problems, we also

record the applications of DRL algorithms to other similar scheduling problems.

As an attractive research direction, solving CMfg-SPs with DRL is becoming increasingly popular [5]. DRL can learn to schedule manufacturing services among service providers according to the interaction history data in the cloud platform [20], and it shows obvious advantages in solving dynamic scheduling and reducing hand-engineering [4]. Dong et al. adopted deep Q-network (DQN) to solve task scheduling problems in CMfg [22]. The optimization objective was to minimize the total makespan, and experiments indicated that their DQN-based scheduling algorithm outperformed the other four heuristic algorithms. Liang et al. solved the QoS-aware service composition in CMfg by the DRL algorithm named dueling double deep Q-network (Dueling DDQN) [20]. Dueling DDQN decouples the action selection from the action evaluation to reduce overestimations, and the last layer of the Q-network is split into two parts that estimate the value function and the advantage function, respectively. The author compared their adopted Dueling DDQN with Q-learning and DQN, and experimental results showed that their application of Dueling DDQN to service composition problems could obtain higher QoS. Chen et al. proposed a reinforcement learning-based assigning policy approach to address the multi-projects scheduling problems in CMfg [23]. The optimization objectives were minimizing the total makespan and the logistics distance. Results indicated that their proposal beat Q-learning and a heuristic algorithm. Zhu et al. addressed the resource scheduling problems in CMfg with DRL [24]. The author employed a policy gradient DRL algorithm to fit the scheduling policy, and the experiments showed that their proposal was comparable to heuristics [30]. Hu et al. leveraged DQN to solve the Petri-net-based dynamic scheduling of flexible manufacturing systems [31]. The author designed a novel graph convolution layer named the Petri-net convolution layer to extract the scheduling features of the manufacturing system state. Dong et al. solved the workflow scheduling in CMfg with the Actor-Critic algorithm [32]. The author used Pointer Network as the policy model, and experiments showed that their proposal outperformed heuristic scheduling algorithms. For works in [20,22–24,31,32], authors applied various DRL algorithms to CMfg-SPs and compared the performance to heuristic algorithms. However, these DRL algorithms require careful state, action, and reward design, as most are step-based. So it relies highly on hyperparameter adjustments to make the training converge. Besides, only heuristic algorithms and Q-learning are compared in their experiments without the comparison to other same-level DRL algorithms.

Applying DRL algorithms to other scheduling problems has also been gradually becoming a major trend [4]. Luo employed the deep Q-network with fixed Q target (DQN with Fixed Q Targets) to address the dynamic flexible job shop scheduling problems under new job insertions, and the author conducted numerous experiments to verify the performance of the proposal [12]. Samsonov et al. solved the manufacturing control in job shop environments with DRL algorithms [33]. The author divided the manufacturing controls into discrete and continuous issues and applied DQN and soft actor-critic (SAC). Chen et al. utilized the DRL algorithm named actor-critic (AC) to choose heuristic operators, and this method showed positive performances [34]. For works in [12,33,34], scheduling features were represented by scaled numbers, leading to low efficiency in feature extraction. Works in [35–37] addressed JSSPs with the DRL algorithm named proximal policy optimization (PPO). They all extracted the job features by using a graph neural network (GNN) as their policy model and trained the model with PPO. Wang et al. utilized the pointer network as the policy model to solve the multi-objective workflow scheduling in the cloud [38]. Asynchronous advantage actor-critic (A3C) was adopted to train the policy model. Mao solved the scheduling problems of data processing clusters [26]. The author regarded the tasks with graph structures as a directed acyclic graph and represented it with a graph convolution network (GCN). In [9], the author utilized GNN to represent the disjunctive graph in JSSPs. The operators were recorded as nodes, and the processing sequences were regarded as directed lines. For works in [9,26,35–38], the scheduling features were extracted effectively by GNN or the pointer network, but they consumed more running time at the same time.

Table 1
DRL algorithms for solving scheduling problems.

DRL algorithms	Types of algorithms	Applied scheduling problems
Soft Actor Critic (SAC)	Policy-based	Manufacturing controls [33], robot task allocation [39]
Proximal policy optimization (PPO)	Policy-based	Job shop scheduling problems (JSSPs) [9,35–37]
Dueling double deep Q-network (Dueling DDQN)	Value-based	QoS-aware service composition in CMfg [20]
Deep Q-network with fixed Q target (DQN with Fixed Q Targets)	Value-based	Dynamic flexible job shop scheduling problems [12]
Deep Q-network (DQN)	Value-based	Task scheduling in CMfg [22], dynamic scheduling of flexible manufacturing system [31], JSSPs [33]
Double deep Q-network (DDQN)	Value-based	Task scheduling in the cloud computing [40]
Asynchronous advantage actor critic (A3C)	Policy-based	Multi-objective workflow scheduling in the cloud [38]
Advantage actor critic (A2C)	Policy-based	JSSPs [34]

DRL algorithms provide intelligent decision-making ability for scheduling problems [17]. For CMfg-SPs, the decision is the discrete action selection of which task to process or which service provider to assign. Under this circumstance, we can apply DRL algorithms with discrete actions to CMfg-SP. There are two types of DRL algorithms, including policy-based and value-based [21]. Policy-based algorithms explicitly construct the representation of the agent policy that maps states to actions, and the policy is stored in memory during learning. Value-based algorithms build a value function rather than the policy, and the policy can be derived from the value function. DRL algorithms that have been used to CMfg-SP or other scheduling problems are listed in Table 1.

3. Problem statement

In this section, the modeling and formulation of CMfg-SP with three optimization objectives are introduced first. Then, we reformulate this problem as a Markov decision process (MDP).

3.1. Modeling and formulation of CMfg-SP

3.1.1. Modeling

In CMfg-SP, there are n^{mt} manufacturing tasks $MT = \{mt_1, mt_2, \dots, mt_{n^{mt}}\}$ to be processed in n^{me} manufacturing enterprises $ME = \{me_1, me_2, \dots, me_{n^{me}}\}$. Each task $mt_i \in MT$ contains a set of subtasks $MST_i = \{mst_{i,j} | j = 1, 2, 3, \dots\}$ and each enterprise $me_i \in ME$ provides a set of manufacturing services $MS_i = \{ms_{i,j} | j = 1, 2, 3, \dots\}$. The transition of tasks among enterprises relies on the logistics, and the logistics between me_i and me_j is noted as lg_{me_i,me_j}. To clarify these variables and their compositions in CMfg-SPs, we listed them in Table 2. To finish a manufacturing task mt_i, subtasks $\{mst_{i,j} | j = 1, 2, 3, \dots\}$ need to be processed following the sequence Seq_i. When facing complex task structures, such as the graph structure in which the processing sequences of subtasks follow the directed acyclic graph, we need to transfer them into the sequential structure Seq_i [4]. The processing of $mst_{i,j}$ needs the corresponding manufacturing service with the functional type $fc_{i,j}$. Implementing the manufacturing service $ms_{i,j}$

Table 2
Variables in CMfg-SPs.

	Compositions	Ranges	Meanings
$mt_i \in MT$	n_i^{mst}	Z^+	Number of subtasks in mt_i
	MST_i	$R^{n_i^{mst}}$	Manufacturing subtasks contained in mt_i
	Seq_i	$R^{n_i^{mst}}$	Processing sequence of subtasks in mt_i
$mst_{i,j} \in MST_i$	$fc_{i,j}$	Z^+	Functional type required for processing $mst_{i,j}$
	$pre_{i,j}$	Z^+	Successor subtask required before processing $mst_{i,j}$
$me_i \in ME$	n_i^{ms}	Z^+	Number of services in me_i
	MS_i	$R^{n_i^{ms}}$	Manufacturing services contained in me_i
	loc_i	R^2	Coordinate of me_i
$ms_{i,j} \in MS_i$	$fc_{i,j}$	Z^+	Functional type of $ms_{i,j}$
	$tm_{i,j}$	R^+	Time consumption of providing $ms_{i,j}$
	$ct_{i,j}$	R^+	Cost of providing $ms_{i,j}$
	$cp_{i,j}$	Z^+	Maximum number of $ms_{i,j}$ that can be provided concurrently
	rl_i	R^+	Reliability of $ms_{i,j}$ that only determined by me_i
	α_i	R^+	Processing efficiency of $ms_{i,j}$ that only determined by me_i
lg_{me_i,me_j}	lt_{me_i,me_j}	R^+	Logistics time of transferring any task between me_i and me_j
	lc_{me_i,me_j}	R^+	Logistics cost of transferring any task between me_i and me_j

in me_i consumes time $tm_{i,j}$ and cost $ct_{i,j}$. Additionally, more than one enterprise can provide services with the same functional type, but the time and cost will be diverse. Besides, the reliability rl_i and processing efficiency α_i are usually different in enterprises, and both of them influence the selections of enterprises during scheduling. According to modeling methods introduced in [20], the processing time of $ms_{i,j}$ is calculated by $tm_{i,j}/\alpha_i$, meaning that a higher efficiency brings lower processing time. During the scheduling, all tasks start from a depot and return this place after finishing all subtasks. The depot is named as the initial location with a fixed coordinate, and we noted it as me_0. If we consider the initial location as a special manufacturing enterprise, then $ME = \{me_0, me_1, \dots, me_{n^{me}}\}$ represents all locations for selecting. In this paper, we only focus on the scheduling procedure, methods of decomposing consumers' tasks into subtasks, determining the processing sequence of subtasks, and modeling service reliability are not discussed.

3.1.2. Objective

In CMfg-SP, n^{mt} tasks are processed with services provided by n^{me} enterprises. The target is to minimize the total makespan and cost and simultaneously maximize the reliability. Let $me_{t,i}, ms_{t,i}$ be the selected enterprise and the needed service for the task i in the t step. Our decision variable in the t step is the selected location $me_{t,i}$. Three objective functions are represented as follows:

$$\min Obj_{makespan} = \max_i\left(\sum_t (lt_{me_{t-1,i},me_{t,i}} + tm_{me_{t,i},ms_{t,i}}/\alpha_i)\right) \tag{1}$$

$$\min Obj_{cost} = \sum_i \sum_t (lc_{me_{t-1,i},me_{t,i}} + ct_{me_{t,i},ms_{t,i}}) \tag{2}$$

$$\max Obj_{rl} = \sum_i \sum_t (rl_{ms_{t,i}}) \tag{3}$$

where definitions of $lt, lc, tm, ct, \alpha, rl$ are reported in Table 2. Solving CMfg-SPs falls into the scope of multi-objective optimization problems. To solve it with DRL algorithms, we leverage the Weighted Sum Method (WSM) to balance the weights of makespan, cost, and reliability. The total optimization objective can be described as:

$$\min Obj = w1 * \|Obj_{makespan}\| + w2 * \|Obj_{cost}\| - w3 * \|Obj_{rl}\| \tag{4}$$

where $w1, w2, w3 \in (0, 1)$ are three weights for makespan, cost, and reliability. $\|Obj_x\| = \frac{Obj_x - min(Obj_x)}{max(Obj_x) - min(Obj_x)}$ means the scaling operation that scales values within the scope of $(0, 1)$, and $x \in \{makespan, cost, rl\}$. The optimization target is to minimize the total objective Obj, and $w1, w2, w3$ determines the importance of these three metrics. Previous works in [20,41] have discussed the influences of different weight values, and we directly adopt their results. In CMfg-SP, we only calculate one group of weights denoting the importance of different sub-objectives rather than the Pareto Front, so the multi-objective problem is converted into a single-objective one [20]. Quality of service (QoS) is widely utilized to measure the scheduling performance in CMfg-SP [4,42]. This paper defines the QoS as $QoS = 1/Obj$, so a higher QoS represents a better scheduling performance.

3.1.3. Assumptions

For simplicity but without losing generality, we make following assumptions [6,10]:

(1) The logistics time and cost between two enterprises are proportional to the Euclidean distance. The logistics defined by other functions may affect the decision results but not the effectiveness of our proposal.
(2) The situation that an enterprise contains no service is not taken into consideration, which is consistent with reality because each enterprise almost provides at least one service.
(3) Once a task has been processed with a service. It cannot be interrupted before finishing the processing.
(4) When the capacity of the service reaches its limitation, an enterprise cannot provide this service until the currently occupied tasks are completed.

3.2. Reformulation as an MDP

Rather than optimizing the target straightly like heuristic optimization algorithms, DRL solves sequence decision problems based on the formulation of an MDP [43]. The MDP of CMfg-SP can be described as a tuple $\langle S, A, r, \pi, \gamma \rangle$. $s \in S$ represents the state that contains features of enterprises and tasks. Our proposal uses embeddings generated by MHA to represent states to avoid explicitly defining specific states for CMfg-SPs. The states' embeddings in the t step are generated based on the following raw features:

$$\langle x_1, x_2, \dots, x_{n^{me}}, Task_t \rangle \tag{5}$$

where $x_i, i \in [0, n^{me}]$ represent enterprise features that mainly includes locations, processing efficiency, service time, and service cost. We regard enterprise features as the input to the encoder, and detailed descriptions of x_i can be found in Section 4.2. $Task_t$ represent task features in the t step that mainly includes the task's location, required service's type, and capacities of the required service in enterprises. These task features are the input of the decoder and are integrated with enterprise embeddings from the encoder. Detailed descriptions of $Task_t$ can be found in Section 4.3. $A = \{a_0, a_1, a_2, \dots a_{n^{me}}, a_{n^{me}+1}\}$ denotes the discrete action space, where a_0 represents the initial location me_0, and $a_1, a_2, \dots, a_{n^{me}}$ represent enterprises $me_1, me_2, \dots, me_{n^{me}}$. $a_{n^{me}+1}$ is the dummy location. It indicates that all enterprises are infeasible to

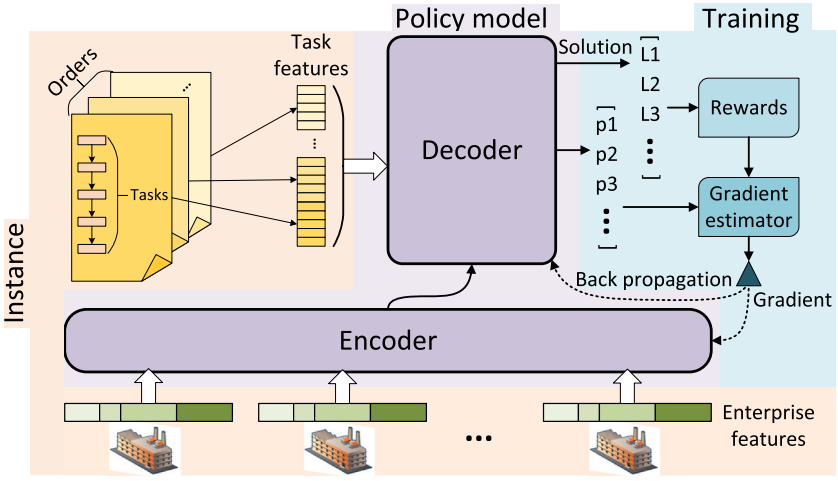

Fig. 1. Overview of our solution for CMfg-SP. The framework has three parts: The pink area, including task and enterprise features, is named as an instance. The purple area shows our policy model with an encoder–decoder architecture. The blue area denotes the training method. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

process the current task, and the agent will select the dummy location when all feasible locations are masked. $r(s,a) : S \times A \rightarrow R$ is the reward function. π is the policy model, and the relationship between s, a, π in the t step is $a_{t+1} = \pi(s_t)$. The transition between s_t and s_{t+1} is triggered by the action a_t and means that a task has just finished a subtask. $\gamma \in [0, 1]$ is the discount factor.

There are n^{mt} tasks needed to be processed concurrently in the cloud platform, but the MDP only supports one agent to make decisions. Inspired by [18,19,29,44], we form a task queue $[mt_1, mt_2, \dots, mt_{n^{mt}}]$ by connecting all tasks so that it can be regarded as an agent. The agent's policy π_θ generates the next location according to the current state of both tasks and enterprises. We aimed to learn a policy π_θ and allocate a high probability to a scheduling solution $L = \{l_1, l_2, \dots\}$ with higher rewards [18]. Let $P_{\pi_\theta}(L|S)$ be the probability of generating the solution L based on the policy π_θ, where S means the state space for a CMfg-SP. Then $P_{\pi_\theta}(L|S)$ is represented as:

$$P_{\pi_\theta}(L|S) = \prod_t P_{\pi_\theta}(l_t|s_t, l_{t-1}) \tag{6}$$

where $s_t \in S, l_t \in L$ are the state and the selection in the t step, respectively. Based on this reformulation, we can represent the policy π_θ with the transformer model to maximize $P_{\pi_\theta}(L|S)$ in which L can obtain high rewards and train the policy with DRL algorithms.

4. Methodology

In this section, the overview of the proposal is introduced first, which follows an encoder–decoder architecture like transformer. Then, the encoder and the decoder are presented in detail. Finally, the training algorithm is provided.

4.1. Overview

Our proposal represents the policy model π_θ based on an encoder–decoder architecture. As shown in Fig. 1, the inputs are information and features of a specific CMfg-SP, which is named as an instance. The output L is a scheduling solution to this instance, where $L = \{l_1, l_2, \dots\}$ and $l_t \in L$ represents the selected location in the t step. The policy model has two parts: the encoder and the decoder. The encoder converts enterprise features into embeddings based on the attention mechanism. The decoder integrates task features into the enterprise embeddings by attention and generates the solution L step by step. In each decoding step, we utilize a context embedding to represent dynamic features during the scheduling and predict the following selection based on this embedding and the previous selection. The attention in our proposal is the multi-head attention (MHA) utilized in the transformer model, which extracts intercorrelations of features by matrix operations of query, key, and value [45].

4.2. Problem encoder

The encoder converts raw enterprise features (such as coordinates and owned services) and their relationships into efficient representations of embeddings through MHA [46]. In CMfg-SPs, there are intercorrelations among enterprises: (1) The logistics-involved relationship exists in enterprises because they are located in different regions. (2) Enterprises with diverse types of services have a complementary relationship when they process a task together. (3) Enterprises with the same type of services exhibit a potential competition relationship because of the diversity of time and cost.

In transformer [45], the encoder adopts the positional encoding to record the input sequence, but we want to represent the global graph relationship rather than the sequential relationship of enterprises. So we abandon the positional encoding to put each enterprise in equal position, meaning that the embeddings of the decoder can be invariant to the input sequence of enterprises. In contrast to VRP or TSP, CMfg-SP is more complex in representation. The encoder is shown in Fig. 2. The inputs $X = \{x_0, x_1, x_2, \dots, x_{n^{me}}, x_{n^{me}+1}\}$ represent raw features of $n^{me} + 2$ locations, including the initial location me_0, enterprises $me_1, me_2, \dots, me_{n^{me}}$, and the dummy location $x_{n^{me}+1}$. Each $x_i \in X$ is represented by:

$$x_i = \begin{cases} \langle loc_i, ef_i, tm_i, ct_i \rangle, \forall i \in [1, n^{me}] \\ loc_i, \forall i \in \{0, n^{me} + 1\} \end{cases} \tag{7}$$

where loc_i denotes the two-dimension coordinate, and ef_i is the processing efficiency. To represent the service distribution in me_i, we use $tm_i = \{tm_{i,j} | j \in [1, n^{ms}]\}$ to indicate the service owned by me_i, where n^{ms} denotes the number of service types. $tm_{i,j}$ equals to the processing time of the service j if me_i has the service j otherwise $tm_{i,j} = 0$. Similar to tm_i, ct_i represents the service cost distribution in me_i. $\langle \cdot \rangle$ is the horizontal concatenation operator to concatenate vectors together. For the initial location and the dummy location, we only utilize the loc_i to represent their features. To align these features, we adopt a layer of neural network to project $\{x_0, x_1, x_2, \dots, x_{n^{me}}, x_{n^{me}+1}\}$ to the same dimension by:

$$h_i^{(0)} = \begin{cases} LayerNorm(W^{(0)} x_i + b^{(0)}), \forall i \in [1, n^{me}] \\ W^{(0),\prime} x_i + b^{(0),\prime}, \forall i \in \{0, n^{me} + 1\} \end{cases} \tag{8}$$

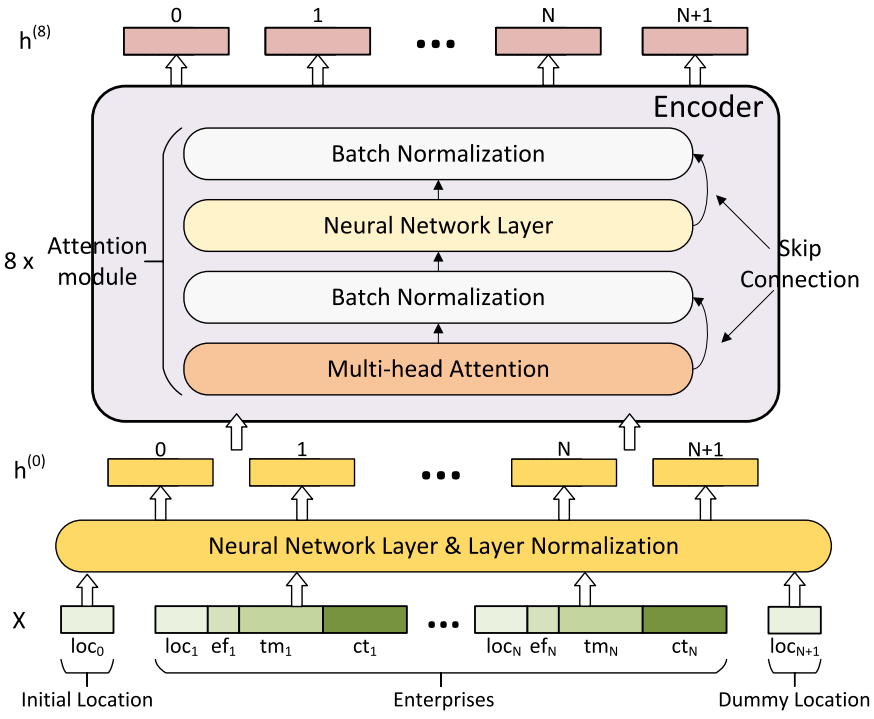

Fig. 2. The model structure of the encoder. We use N instead of n^{me} to represent the number of enterprises for readability. Attention embeddings $h_0^{(8)}, h_2^{(8)}, \ldots, h_{n^{me}+1}^{(8)}$ of $n^{me}+2$ locations are computed through eight attention modules (The setting of eight attention modules is the same as that in literature [29], and the detailed structure in each attention module follows the MHA in the transformer model [45]).

where $h_i^{(0)}$ represents the embedding of the location i, and $W^{(0)}, b^{(0)}$, $W^{(0),\prime}, b^{(0),\prime}$ are parameters of neural networks. $LaterNorm$ is the layer normalization operation. Then, embeddings $h_0^{(0)}, h_1^{(0)}, \ldots, h_{n^{me}+1}^{(0)}$ are delivered to the encoder to extract intercorrelations. As shown in Fig. 2, there are eight attention modules in the encoder, and each module contains two sublayers: a MHA layer and a neural network layer [45]. For each module $l \in \{1, 2, \ldots, 8\}$, attention embeddings are calculated by:

$$\hat{h_i^{(l)}} = BatchNorm(h_i^{(l-1)} + MHA^{(l)}(h_0^{(l-1)}, \ldots h_{n^{me}+1}^{(l-1)})), \forall i \in [0, n^{me}+1] \quad (9)$$

$$\widetilde{h_i^{(l)}} = ReLU(W_1^{(l)} \hat{h_i^{(l)}} + b_1^{(l)}), \forall i \in [0, n^{me}+1] \quad (10)$$

$$h_i^{(l)} = BatchNorm(\hat{h_i^{(l)}} + \widetilde{h_i^{(l)}}), \forall i \in [0, n^{me}+1] \quad (11)$$

where $BatchNorm$ represents the batch normalization operation [47], and $ReLU(\cdot)$ means the ReLU activation function. $\hat{h_i^{(l)}}, \widetilde{h_i^{(l)}}$ are the hidden embeddings of the enterprise i in the l attention layer calculated by the MHA layer and the neural network layer, respectively. $W_1^{(l)}, b_1^{(l)}$ are parameters of the neural network in the l attention layer. $h_i^{(l-1)}, h_i^{(l)}$ are final embeddings of the location i in attention layers of $l-1$ and l, respectively. The addition operation in Eq. (11) means the skip-connection to accelerate the training [48]. The essence of MHA is to extract intercorrelations of features with matrix operations of query, key, and value, and detailed calculations of $MHA(\cdot)$ can be found in Appendix A. Attention embeddings $h_0^{(8)}, h_2^{(8)}, \ldots, h_{n^{me}+1}^{(8)}$ are obtained after the encoding.

4.3. Solution decoder

Based on attention embeddings $h_0^{(8)}, h_2^{(8)}, \ldots, h_{n^{me}+1}^{(8)}$ of $n^{me}+2$ locations from the encoder, we introduce the decoder that generates the solution L step by step. We execute the decoding based on embeddings from the encoder and utilize the MHA again to integrate task features into enterprise embeddings. The task features do not appear in the encoder (see Fig. 3). Specifically, we construct a context node $h^{(c)}$ with current task features, the enterprise embeddings, and other related information. $h^{(c)}$ changes dynamically during the decoding, which is represented as:

$$h_t^{(c)} = [\overline{h^{(8)}}, h_{\pi_{t-1}}^{(8)}, mst_t, v_{mst_t}] \quad (12)$$

where $\overline{h^{(8)}} = \frac{\sum_{i=0}^{n^{me}+1} h_i^{(8)}}{n^{me}+2}$ is the average embedding. $h_{\pi_{t-1}}^{(8)}$ represents the embedding of the location selected by the policy π_{t-1} in the last step. mst_t denotes the required type of service of the current processing task in the t step. $v_{mst_t} = \{v_{mst_{t,j}} | j \in [1, n^{me}]\}$ represents the capacities of the required service mst_t in n^{me} enterprises. To facilitate the expression of dimensions of vectors, we use $v \in R^d$ to represent that the vector v is d-dimension and each value in v belongs to Real number space R. Given the dimension of enterprise embeddings from the encoder is d_h, then we have $\overline{h^{(8)}} \in R^{d_h}$, $mst_t \in R^1$, and $v_{mst_t} \in R^{n^{me}}$. So $h_t^{(c)} \in R^{(2*d_h+1+n^{me})}$. To integrate enterprise embeddings from the encoder into the context node, we calculate the context embedding $\overline{h_t^{(c)}}$ based on $MHA(h_t^{(c)}, h_0^8, h_1^8, \ldots, h_{n^{me}+1}^8)$. In the decoder, we only calculate the context embedding $\overline{h_t^{(c)}}$ of $h_t^{(c)}$ through MHA, rather than all attention embeddings of $h_t^{(c)}, h_0^8, h_1^8, \ldots, h_{n^{me}+1}^8$. Specifically, the key and value in MHA are from embeddings $h_0^{(8)}, h_1^{(8)}, \ldots, h_{n^{me}+1}^{(8)}$, but the query is constructed based on $h_t^{(c)}$. We omit the subscript t in the following expressions. The query $q_{(c)}$, key k_i, and value v_i in MHA for calculating the context embedding $\overline{h_t^{(c)}}$ are represented as:

$$q_{(c)} = W^q h^{(c)}, k_i = W^k h_i^{(8)}, v_i = W^v h_i^{(8)} \quad (13)$$

where W^q, W^k, W^v are matrix parameters of query, key, and value. $i \in [0, n^{me}+1]$ represents the location i, and $h_i^{(8)}$ is the embedding of it from the encoder. $h^{(c)}$ is the context node. The detailed calculation of these query, key, and value in $MHA(\cdot)$ can be found in Appendix A. We only use one layer of MHA in the decoder. With this MHA, we convert the context node that contains raw task features and other

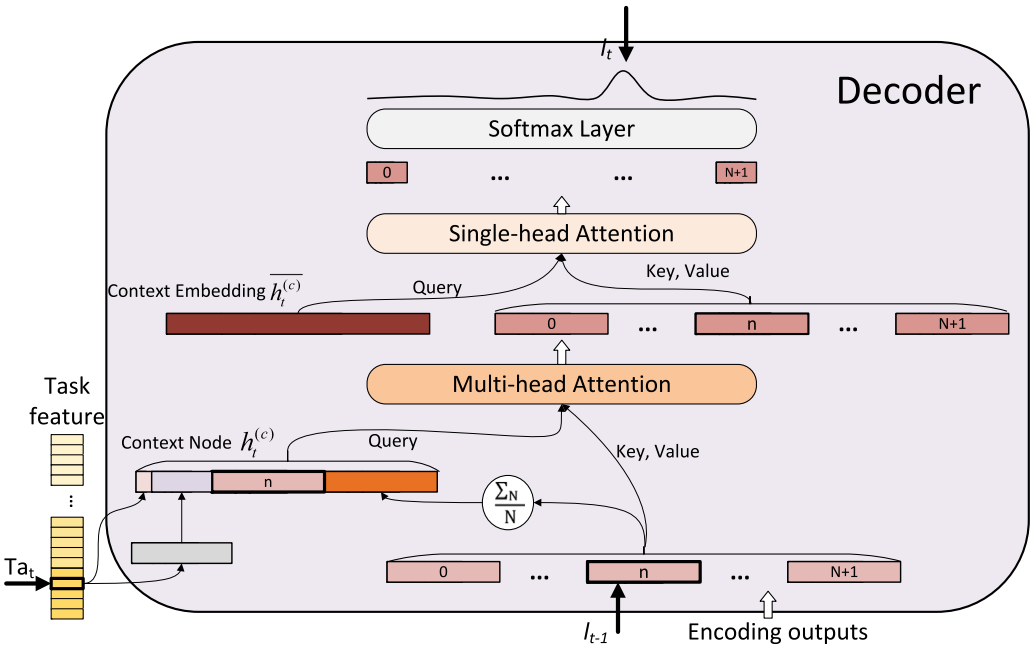

Fig. 3. The model structure of the decoder. This figure demonstrates the procedure of selecting l_t. We use N denotes n^{me} for readability. In contrast to the encoder that only computing once for an instance, the decoder generates the selection l_i step by step until Ta_{i+1} points at the terminal task.

dynamic features into the context embedding $\overline{h_t^{(c)}}$, and $\overline{h_t^{(c)}}$ represents the intercorrelation of the context node and enterprise embeddings.

To calculate probabilities of selecting $n^{me}+2$ locations, we utilize a single-head attention layer to compute the compatibility a_i:

$$a_i = tanh(\frac{\overline{q_{(c)}^T} \cdot k_i}{\sqrt{d^k}}), \forall i \in [0, n^{me}+1] \tag{14}$$

where the query $\overline{q_{(c)}} = W^q \cdot \overline{h^{(c)}}$, and W^q is the matrix parameter of the query. k_i is the key as reported in Eq. (13). d^k is the dimension of k_i. The a_i represents the compatibility of the location i to the context embedding $\overline{h_t^{(c)}}$, and it can be interpreted as an unnormalized log-probability for choosing the location i. CMfg-SPs contain hard constraints when selecting a location. Leaving these hard constraints for the policy model to learn is inefficient, so the masking scheme is necessary to ensure the rationality of the decoder's output [49]. A location is infeasible if there is no needed service or the capacity of the service is full at this time. The masking algorithm is listed in algorithm 1, and the mask result $M = \{1, -\infty, \dots\} \in R^{(n^{me}+2)}$ is finally obtained where $-\infty$ denotes this location is infeasible. Finally, the probability P_i is calculated by:

$$P_i = softmax(a_i \cdot M), \forall i \in [0, n^{me}+1] \tag{15}$$

Based on the calculation of P_i with Eq. (15), we obtain the probability vector $\{P_0, \dots, P_{n^{me}+1}\}$, and select P_i with the largest probability as the action. With the probability of the selected action in each step, the probability $P_{\pi_\theta}(L|S)$ can be calculated by Eq. (6).

To conclude, the decoder utilizes two layers of attention to output selecting probabilities of locations. The first layer of MHA converts the context node into the context embedding that contains the intercorrelation of the context node and enterprise embeddings. Then, the second single-head attention is used to calculate the probabilities of selecting $n^{me}+2$ locations. The decoding procedure terminates when all n^{mt} tasks are scheduled.

4.4. Model training

Sections 4.2 and 4.3 describe the policy model with an encoder–decoder framework that generates a solution L based on an instance

Algorithm 1: CMfg-SP Masking Algorithm

Input:
ms_t: required service in the t time;
t: current time step;
$EC(me_i, S_t)$: service capacity of the service S_t in me_i;
$Ot(i, S_t) = \{[ot^j_{min}, ot^j_{max}] | j >= 0\}$: historical occupied periods of the service S_t in me_i;
Output:
$Mask \in R^{n^{me}+2}$: the mask vector of $n^{me}+2$ locations;
if *A task is just finished* **then**
  $Mask[0] \leftarrow 1$ and $Mask[j] \leftarrow -\infty, \forall j \in [1, n^{me}+1]$;
else
  $Mask[0] \leftarrow -\infty$;
  foreach $i \in [1, n^{me}]$ **do**
    if $EC(i, S_t) \neq 0$ **then**
      $Num \leftarrow 0$;
      Calculate the logistics time lt_i between the current location and the enterprise n;
      foreach $[ot_{min}, ot_{max}] \in Ot(i, S_t)$ **do**
        if $t + lt_i > ot_{min}$ *or* $t < ot_{max}$ **then**
          $Num \leftarrow Num + 1$;
        end
        if $Num < EC(me_i, S_t)$ **then**
          $Mask[i] \leftarrow 1$;
        else
          $Mask[i] \leftarrow -\infty$;
        end
      end
    else
      $Mask[i] = -\infty$;
    end
  end
  if $Mask[j] == -\infty, \forall j \in [0, n^{me}]$ **then**
    $Mask[-1] \leftarrow 1$;
  else
    $Mask[-1] \leftarrow -\infty$;
  end
end
Return $Mask$;

with the state space S. To train this policy model through the interaction data in the cloud platform, we adopt the reinforcement learning algorithm REINFORCE with a baseline [50]. REINFORCE maximize the cumulative reward $G = \sum_t r_t$ of the agent by Policy Gradient Theorem [50]. In our proposal, the gradient estimator of $\nabla Loss$ with a rollout baseline is represented by:

$$\nabla Loss = E_{\pi_\theta}\{[G_{\pi_\theta}(S) - G_b(S)]\nabla log P_{\pi_\theta}(L|S)\} \tag{16}$$

where $P_{\pi_\theta}(L|S)$ is computed by Eq. (6). $G_{\pi_\theta}(S), G_b(S)$ represent cumulative rewards obtained by the current policy π_θ and the baseline model b, respectively. The rollout baseline b is the historical trained policy with the best performance. Given the current training step is t, then $b_t \in \{\pi_{\theta(1)}, \pi_{\theta(2)}, \dots, \pi_{\theta(t-1)}\}$ and $G_{b_t}(S) \geq G_{\pi_{\theta(i)}}(S), i = 1, 2, \dots, t-1$. So the baseline model has the same structure as the policy model, but its parameters are frozen to the historical policy with the best performance. At the end of each training epoch, parameters of the baseline model b_t will be replaced with $\theta(t)$ if $\pi_{\theta(t)}$ outperforms b_t, otherwise b_t remains unchanged. To compare performances of b_t and $\pi_{\theta(t)}$, we introduce the paired *t-test* on 100 separate scheduling instances. The $t - test(\pi_{\theta(t)}, \pi_{b_t}) < 0.05$ means that the training policy outperforms the baseline model with high probability [51]. The training algorithm is reported in algorithm 2, where *t-test* denotes the paired *t-test*, S_i represents an instance in a batch, and L_i, L_i^b mean solutions of S_i by π_θ, b, respectively. Based on the gradient estimator described in Eq. (16), the policy is trained with Adam optimizer [52].

In our proposal, we directly adopt Obj in the terminal step of an episode as the cumulative reward, meaning that r_t in our MDP modeling is defined as:

$$r_t = \begin{cases} 0 & t \leq t_{done} \\ Obj & t = t_{done} \end{cases} \tag{17}$$

where t_{done} means the terminal step, Obj is the objective function in Eq. (4). We use the objective value in the terminal step as the reward. The policy is expected to learn to maximize the cumulative reward by optimizing the decision-making based on the reasoning ability. Under this circumstance, the gradient estimator of ∇L can also be represented as:

$$\nabla Loss = E_{\pi_\theta}\{[Obj_{\pi_\theta}(S) - Obj_b(S)]\nabla log P_{\pi_\theta}(L|S)\} \tag{18}$$

where $Obj_{\pi_\theta}(S), Obj_b(S)$ denote total objectives obtained by π_θ and b, and the total objective is calculated by Eq. (4). According to Eq. (17), we do not need to define step-based rewards.

Algorithm 2: Training Algorithm

Input:
π_θ: the training policy with parameters θ
b: the baseline model
N: number of epochs
M: number of steps
B: number of batches
foreach $epoch = 1 : N$ **do**
  foreach $step = 1 : M$ **do**
    Sample an instance S_i randomly from the training dataset, $\forall i \in [1, B]$;
    Generate the solution L_i based on π_θ and calculate $G_{\pi_\theta}(S_i), P_{\pi_\theta}(L_i|S_i), \forall i \in [1, B]$;
    Generate the solution L_i^b based on b and calculate $G_b(S_i), \forall i \in [1, B]$;
    $\nabla Loss \leftarrow \sum_{i=1}^{B}(G_{\pi_\theta}(S_i) - G_b(S_i))\nabla log P_{\pi_\theta}(L_i|S_i)$;
    $\theta \leftarrow Adam(\theta, \Delta Loss)$;
    Implement $t - test$ for π_θ, b on 100 evaluation instances;
    if $t - test(\pi_\theta, b) < 0.05$ **then**
      $b \leftarrow \pi_\theta$;
    end
  end
end

5. Case study

In this section, we use eight DRL scheduling algorithms, two heuristic algorithms, and two PDRs as compared algorithms. First, we introduce the experimental configuration and detailed hyper-parameters of compared algorithms. Second, we conduct comparative experiments on scheduling performances to show each algorithm's QoS, makespan, cost, reliability, and running time. Then, to verify the effectiveness of our proposal compared to other DRL-based algorithms, the generalizability to unseen instance-sets and the scalability of the total subtasks' quantity are compared. Finally, we discuss these experiments' results. The development platform is based on Python 3.8.1 programming language and PyTorch 1.8.2 with CUDA 11.1. A server computer with XEON Gold 6240 @3.8 Hz CPU and NVIDIA RTX 3080 GPU is adopted as the physical equipment.

5.1. Experimental configuration and compared algorithms

Section 3 has described the modeling and formulation of CMfg-SPs, and this section introduces the detailed configuration of an automobile structure part scheduling case in CMfg. There are 20 automobile manufacturing enterprises providing processing services for automobile structure parts. 16 types of manufacturing services for automobile structure part processing are listed in Table 3, where CNC means computer numerical control. Tasks from consumers are compositions of subtasks that need to be processed by k types of services, where $k \in [1, 16]$. Additionally, we randomly generate other parameters to simulate the uncertainty and evaluate the general performance on different distributions rather than that of a specific parameter setting. The two-dimension coordinates (x, y) of enterprises are randomly sampled within $x \in (0, 1), y \in (0, 1)$, and the initial location is fixed at $(0, 0)$. Capacities of services in each enterprise are randomly sampled within $[0, 10]$. After analyzing the data distribution of processing time and cost of the same service in different manufacturing enterprises, the processing time of the service i in enterprise j is sampled within $\frac{Random(5,10)}{\alpha_j} + Normal(2, 4)$, the cost is within $Random(10, 30) * \alpha_j + Normal(1, 3)$, where the efficiency α_j is randomly generated from $[1, 3]$. The reliability rl_i of me_i is randomly sampled within $[2, 4]$. We compute the total number of subtasks with $\sum_{i=1}^{n^{mt}} n_i^{mst}$, where n^{mt} is the number of tasks and n_i^{mst} is the number of subtasks in mt_i. For convenience, we named instance-sets with the total number of subtasks $100, 200, 300, 400$ as $T100, T200, T300, T400$. In the following experiments, an instance-set contains 2000 instances generated by the method described above. For this case, the makespan and cost are more significant than reliability so that the weights w_1, w_2, w_3 in Eq. (4) are assigned with $2, 2, 1$.

Table 3
Manufacturing services of automobile structure parts

No.	1	2	3	4
Abbreviation	FO	TR	FL	PI
Meanings	Forming	Trimming	Flanging	Piercing
No.	5	6	7	8
Abbreviation	WE	CL	WC	AL
Meanings	Welding	Cleaning	Wire-cut	Aligning
No.	9	10	11	12
Abbreviation	DE	PH	DR	MI
Meanings	Degreasing	Phosphating	Drying	Milling
No.	13	14	15	16
Abbreviation	CNCT	CNCM	TU	CA
Meanings	Turning with CNC	Milling with CNC	Turning	Casting

Table 4
Hyper-parameter settings of compared algorithms.

Hyper-parameters	SAC	PPO	Dueling DDQN	DQN with Fixed Q Target	DQN	DDQN	A3C	A2C
Running mode	GPU	CPU	GPU	GPU	GPU	GPU	CPU	CPU
Training episodes	40 000	62 000	60 000	60 000	60 000	60 000	62 000	62 000
Learning rate	/	0.02	0.01	0.01	0.01	0.01	/	/
Actor learning rate	0.0001	/	/	/	/	/	0.0001	0.0001
Critic learning rate	0.0001	/	/	/	/	/	0.0001	0.0001
Batch size	256	/	512	256	256	512	256	256
Buffer size	/	/	40 000	40 000	40 000	40 000	/	/
Length of experience replay	/	/	1	1	1	1	/	/
Discount factor	0.99	0.95	0.99	0.99	0.99	0.99	0.99	0.99
Linear hidden units	/	[20, 20]	[30, 10]	[30, 10]	[30, 10]	[30, 10]	/	/
Actor linear hidden units	[64, 64]	/	/	/	/	/	[64, 64]	[64, 64]
Critic linear hidden units	[64, 64]	/	/	/	/	/	[64, 64]	[64, 64]
Soft updated rate	0.005	/	/	0.01	/	/	/	/
Gradient clipping	5	7	5	5	5	5	5	5

As for the parameters of our proposal, the learning rate is 0.0002 with a decay factor of 0.995, and the batch size is 512. For convenience, we called the proposed model SAM in the following, meaning the Scheduling Attention Model. It is prohibitively time-consuming to compute optimal solutions for CMfg-SPs through exact methods, so we choose to utilize DRL scheduling algorithms, heuristic algorithms, and PDRs as the compared algorithms [4].

For DRL scheduling algorithms, typical DRL algorithms applied to CMfg-SPs are reported in Table 1, and we implement all of them to measure our proposal's performance. However, as these DRL algorithms are step-based, extensional definitions of reward functions and state representations are needed. Hyper-parameter adjustments are also necessary to make all these DRL algorithms positively perform on CMfg-SPs. We have conducted numerous experiments and deeply reviewed related literature to search for feasible reward functions and state representations. In these DRL algorithms, some algorithms are improved versions of others. For example, PPO is an algorithm for optimizing the AC policy, so it should outperform A2C in theory. However, these improvements were verified on the control of video games or robots rather than scheduling problems. Problem domains, data distributions, and reward signals can all influence results. So we compare all of them to show their practical performances in CMfg-SPs. These DRL algorithms also use the weighted sum method to integrate three objectives as a total objective, which is considered in the reward function. The definitions of reward and state for these DRL algorithms can be found in Appendix B. The hyper-parameters of these DRL algorithms are reported in Table 4.

For heuristic algorithms, genetic algorithm (GA) and ant colony optimization (ACO) are implemented, and each is executed for 200 iterations. In GA, the population size is 100, the crossover rate is 0.4, and the mutation rate is 0.1. In ACO, the population size is 100, the pheromone drop amount per step is 0.001, the evaporate rate is 0.1, the pheromone factor is 1, and the heuristic factor is 3. For PDRs, we apply the weighted objective greedy rule (WOGR) and the average resource first rule (ARFR) [10]. In each step, WOGR selects the next enterprise based on the greedy strategy to maximize the weighted objective in Eq. (4). ARFR chooses the next enterprise having the maximum number of idle services and prioritizes the assignment of services to the tasks with the maximum number of remaining subtasks.

5.2. Comparative experiments on scheduling performances and running time

We first compare SAM with heuristic algorithms and PDRs, and the results are shown in Fig. 4. Subfigures in Fig. 4 numbered (1) to (4) represent results on instance-sets of $T100, T200, T300, T400$. For each instance-set, we use six instances to evaluate the QoS. In Fig. 4(1)-(4), we found that GA outperforms others in most instances. SAM beats GA in individual instances and shows the second-highest QoS. ACO indicates the third-best performance and beats the two PDRs in most instances. Additionally, WOGR beats ARFR in (1) and (2), but the results are reversed in (3) and (4). It means that PDRs perform diverse results when facing different scheduling scenes.

Then, we compare SAM with eight DRL scheduling algorithms, and Fig. 5 shows the results. All algorithms are trained with the same instance-sets, and their training curves converge to stable values.[1] In Fig. 5(1), SAM outperforms others in two instances. PPO shows the best performance in three instances. In (2), SAM possesses the highest QoS in three instances, and SAC shows the best performance in two instances. In (3), SAM outperforms others in four instances, but PPO and SAC show the best performance in the fourth and fifth instances, respectively. In (4), SAM has the best performance in five instances, and DQN with fixed Q target has the highest QoS in the fourth instance. Based on results in Fig. 5, we found that SAM outperforms other DRL algorithms in most instances. Besides, with the increasing number of tasks, the performance of SAM has become superior to others.

To take a glimpse at the scheduling performances of all algorithms, we listed the average values of metrics in Table 5. QoS contains three objectives, including makespan, cost, and reliability. One crucial requirement on the scheduling algorithm in CMfg is the time-respond [4], so we also compare the running time of algorithms. In terms of the scheduling performance, GA shows the highest QoS in most instances, and SAM indicates the second-best scheduling performance. However, the running time of GA is nearly thousands of seconds, which cannot satisfy the requirement of low time-respond in CMfg. In contrast, DRL algorithms solve the problem in seconds. To conclude, SAM shows the competitive scheduling performance (only next to GA) and has the lowest running time when solving CMfg-SPs.

5.3. Comparative experiments on generalizability and scalability

Compared to other approaches, DRL-based scheduling algorithms have advantages of the generalizability to unseen instance-sets and the scalability of the total subtasks' quantity. Focusing on these two abilities, we compare our proposal with other DRL-based algorithms.

We first evaluate the generalizability of algorithms. For the CMfg-SP of automobile structure parts, six elements are unpredictable, including (1) locations of enterprises, (2) service distributions, (3) reliabilities, (4) service processing time, (5) service cost, and (6) subtask sequences. To evaluate the generalizability, we conduct six instances-sets with six corresponding elements changing and implement all DRL algorithms on them. These six instance-sets are numbered (1) – (6), and all instances are unseen to the training. The QoS results are shown in Fig. 6. The horizontal coordinate refers to the algorithms, and the vertical refers to the QoS. The box line of each algorithm represents QoS results of ten instances in each instance-set. Subfigures from Fig. 6(1) to (6) indicate that SAM shows higher QoS than others on average, and these results are more evident in (3) and (5). SAM has the highest medians and

[1] The training curves can be found in Appendix C.

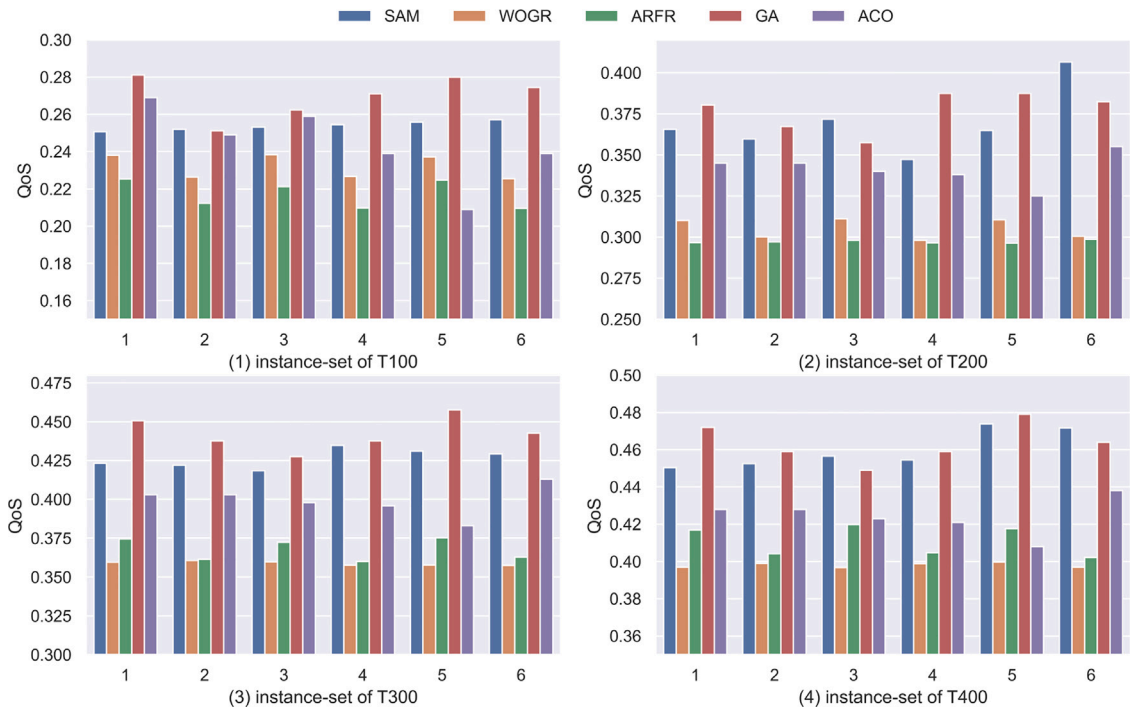

Fig. 4. QoS of our proposal, heuristic algorithms, and PDRs.

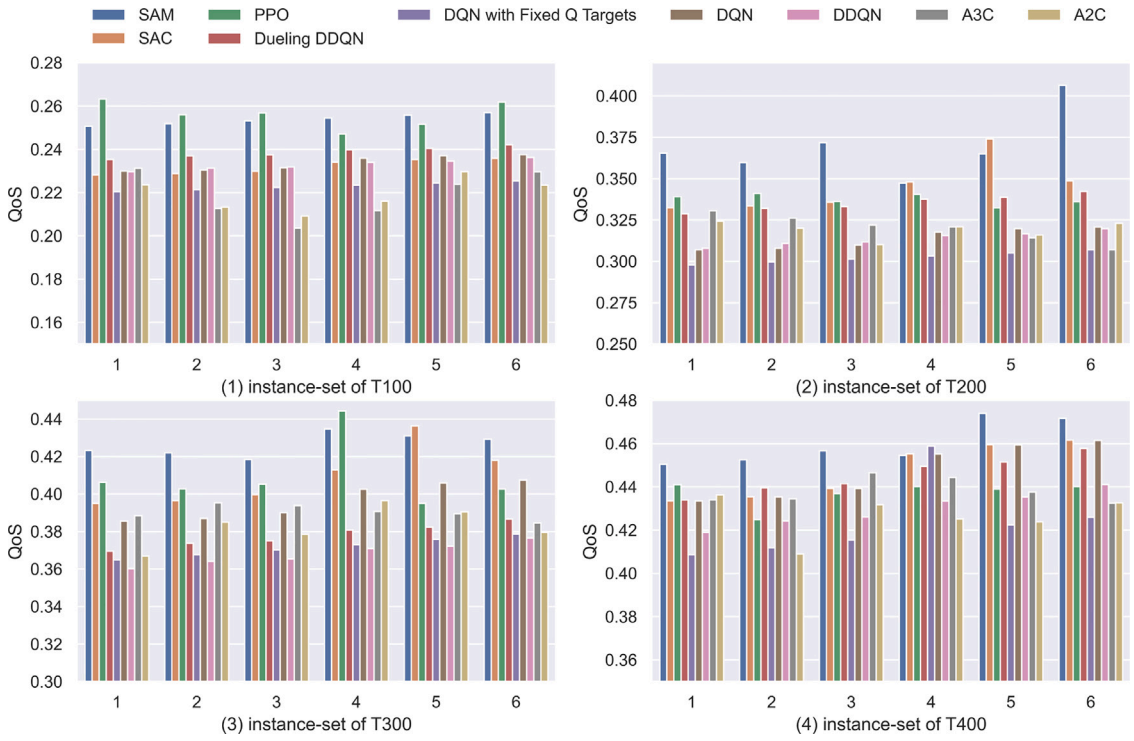

Fig. 5. QoS of our proposal and other DRL scheduling algorithms.

maximums in all these six sets. Besides, although PPO and SAC obtain positive results as described in Section 5.2, their generalizability is not as good as those on training sets, which means that they may overfit the training instance-sets.

Then, we evaluate the scalability of algorithms. The quantity of total subtasks is unpredictable due to the complexity and uncertainty of the CMfg environment, so scalability needs to be considered. Based on trained models on the instance-set of $T400$, we conduct experiments on smaller instance-sets of $T300, T350$ and larger instance-sets of $T450, T500, T550, T600$. As shown in Fig. 7, each instance-set has ten instances, and the vertical coordinate refers to QoS. SAM has the highest medians and maximums in (1)–(6), and the gap between SAM and others is pronounced in Fig. 7-(3), (4), and (6). So in terms of scalability, SAM outperforms the other eight DRL algorithms. Furthermore, we also compare the running time. As shown in Fig. 8, the running

Table 5
Comparison of QoS, makespan, cost, reliability, and running time.

Scenes	Algorithms	QoS	Makespan (scaled)	Cost (scaled)	Reliability (scaled)	Running time (s)
	SAM	0.2538	1.3397	1.0531	0.8461	0.2104
	SAC	0.2320	1.4760	1.0925	0.8266	0.2862
	PPO	0.2560	1.2854	1.0601	0.7849	0.7224
	Dueling DDQN	0.2387	1.4516	1.0796	0.8730	0.2643
	DQN with Fixed Q Targets	0.2229	1.5391	1.0757	0.7437	0.3391
	DQN	0.2337	1.4310	1.0724	0.7277	0.3341
T100	DDQN	0.2329	1.4996	1.0911	0.8886	0.3511
	A3C	0.2183	1.5728	1.0862	0.7380	0.4207
	A2C	0.2190	1.5369	1.0840	0.6756	0.4081
	WOGR	0.2321	1.2962	1.0432	0.3783	0.2531
	ARFR	0.2173	1.5485	1.0937	0.6831	0.2584
	GA	0.2612	1.1094	1.1104	0.6111	1384
	ACO	0.2197	1.5799	1.0367	0.6827	980.8
	SAM	0.3655	0.6470	1.0488	0.6556	0.4722
	SAC	0.3454	0.7766	1.0706	0.7994	0.7620
	PPO	0.3375	0.7711	1.0712	0.7220	1.2690
	Dueling DDQN	0.3354	0.8110	1.0874	0.8154	0.6034
	DQN with FixedQ Targets	0.3023	0.8935	1.0915	0.6625	0.8711
	DQN	0.3137	0.8975	1.0976	0.8029	0.7041
T200	DDQN	0.3137	0.8847	1.0973	0.7763	0.7211
	A3C	0.3199	0.8181	1.0885	0.6872	1.0348
	A2C	0.3190	0.8233	1.0866	0.6848	0.9983
	WOGR	0.3065	0.7774	1.0555	0.4034	0.5629
	ARFR	0.303	0.8988	1.0933	0.6845	0.6104
	GA	0.3774	0.54919	1.0886	0.6263	5245
	ACO	0.3355	0.7682	1.0405	0.6371	2291
	SAM	0.4077	0.5301	1.0646	0.7365	0.8158
	SAC	0.4098	0.5187	1.0800	0.7570	1.1778
	PPO	0.4094	0.5289	1.0679	0.7509	1.8269
	Dueling DDQN	0.3780	0.6146	1.0893	0.7621	1.1110
	DQN with FixedQ Targets	0.3717	0.6021	1.0965	0.7067	1.4224
	DQN	0.3962	0.5317	1.0839	0.7073	1.2343
T300	DDQN	0.3682	0.6010	1.0943	0.6744	1.1659
	A3C	0.3904	0.5482	1.0883	0.7113	1.5524
	A2C	0.3826	0.5618	1.0873	0.6848	1.5771
	WOGR	0.3652	0.5192	1.0619	0.4247	0.8997
	ARFR	0.3675	0.6003	1.0919	0.6639	0.9511
	GA	0.4476	0.3962	1.0273	0.6133	5245
	ACO	0.3933	0.5522	1.0378	0.6377	3915
	SAM	0.4597	0.3971	1.0700	0.7598	1.1409
	SAC	0.4471	0.3937	1.0883	0.7276	1.6787
	PPO	0.4369	0.4263	1.0837	0.7311	2.4604
	Dueling DDQN	0.4455	0.3743	1.0674	0.6386	1.7031
	DQN with FixedQ Targets	0.4238	0.4313	1.0780	0.6593	2.0586
	DQN	0.4471	0.4186	1.0811	0.7626	1.6346
T400	DDQN	0.4297	0.4182	1.0801	0.6694	1.9259
	A3C	0.4381	0.4017	1.0882	0.6975	2.2624
	A2C	0.4263	0.4239	1.0853	0.6724	2.2614
	WOGR	0.4042	0.3952	1.0659	0.4485	1.3089
	ARFR	0.4113	0.4504	1.0877	0.6455	1.3889
	GA	0.4691	0.3459	1.0259	0.6131	7704
	ACO	0.4182	0.4677	1.0461	0.6371	5943

time increases as the number of total subtasks rises. SAM shows the lowest running time compared to others, and it does not exceed two seconds even facing the largest task number of $T600$. Dueling DDQN shows the second lowest time consumption when implemented on an instance-set of $T600$. Considering the gap between the one with the longest running time consumed and SAM, DQN with Fixed Q Target consumes four times longer than SAM when facing $T600$.

5.4. Discussion

We have conducted comparison experiments from three perspectives to comprehensively evaluate our proposal's performance, including scheduling performance, generalizability, and scalability. Our proposal indicates competitive scheduling performances and is only next to GA compared to eight DRL scheduling algorithms, two heuristic algorithms, and two PDRs. Compared to GA, our proposal shows an obvious running-time advantage and can solve CMfg-SPs in seconds, while GA requires thousands of seconds to search for a satisfying result. Among DRL-based scheduling algorithms, our proposal possesses the lowest running time and highest QoS in CMfg-SPs. Additionally, compared to other DRL scheduling algorithms, our proposal indicates better performances on the generalizability to unseen instance-sets and the scalability of the total subtasks' quantity. These advantages benefit from the policy model's attention mechanism, which captures complex relationships between enterprises and converts raw features of enterprises and tasks into effective attention embeddings.

6. Conclusion and future work

In this paper, we proposed an end-to-end scheduling algorithm to address the CMfg-SP through the attention mechanism and DRL to maximize the QoS. To this end, we modeled the CMfg-SP as an MDP and formulated it to suit the transformer model. We utilized the MHA on a graph representation to extract intercorrelations among enterprises in the encoder. Then, we integrated task features into enterprise embeddings with MHA in the decoder. MHA has been rarely applied to

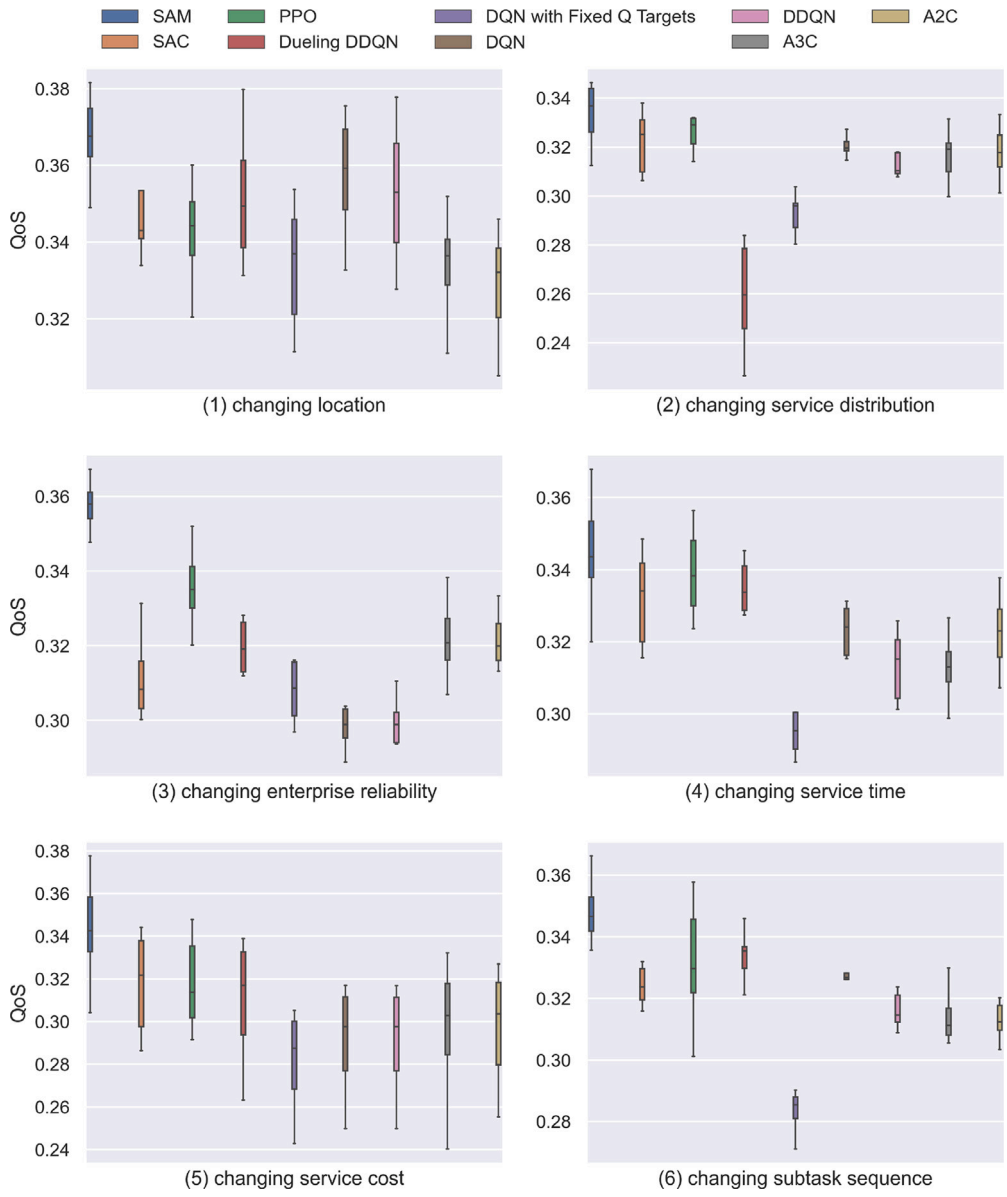

Fig. 6. QoS of DRL algorithms on six unseen instance-sets.

CMfg-SPs or even general scheduling problems. Experiments on the automobile structure part scheduling in CMfg were conducted. Eight other mainstream DRL algorithms, two heuristic algorithms, and two PDRs were implemented as compared algorithms. The experimental results indicated that our proposal showed competitive scheduling performances compared to eight DRL algorithms, two heuristic algorithms, and two priority dispatching rules. The results also proved that our proposal's generalizability and scalability were better than other DRL algorithms.

We aim to advance the application of DRL algorithms to CMfg-SPs by providing an improved model and easier modeling. Our approach belongs to a DRL-based scheduling algorithm because REINFORCE with a baseline trains the model, and the decoding requires immediate online feedback of the environment during the scheduling. We replace the cumulative rewards in REINFORCE with a terminal objective to avoid the definition of the step-based reward function. Benefit from the powerful reasoning ability of the attention mechanism, our approach can still converge and achieve competitive performances. To further promote the application of DRL algorithms to CMfg-SPs, future works will focus on two aspects. (1) As a multi-objective optimization problem, CMfg-SP needs to consider balancing all objectives rather than integrating them into one objective. An alternative solution is to define several reward functions for these objectives and train the model using ensemble learning. (2) In our proposal, the features of enterprises and tasks are represented by the transformer model, and the DRL algorithm trains the entire model. In the future, we will integrate both of them into one unified model using a recently proposed architecture named decision transformer [53].

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.

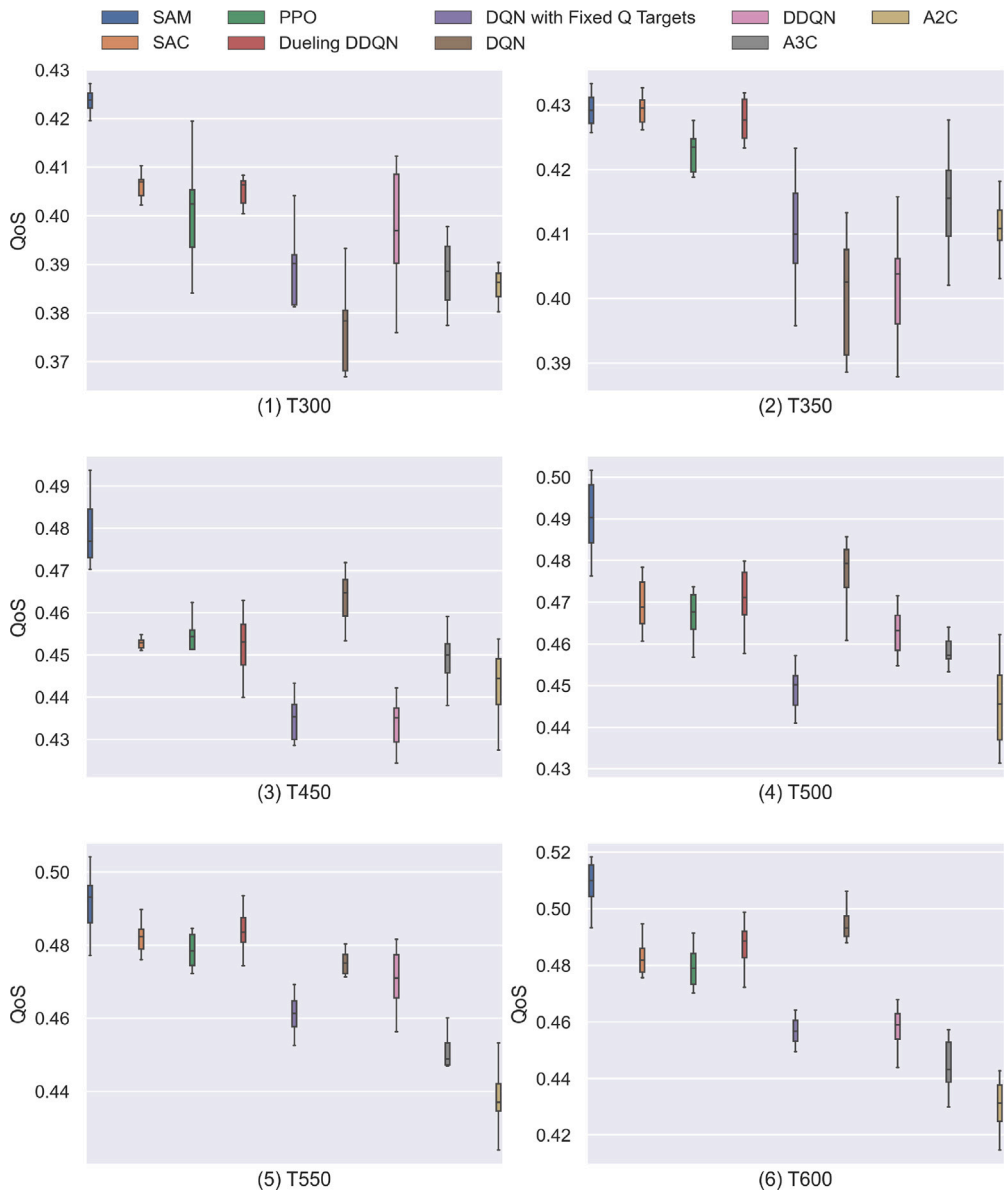

Fig. 7. QoS of DRL algorithms on six instance-sets of different total task quantities.

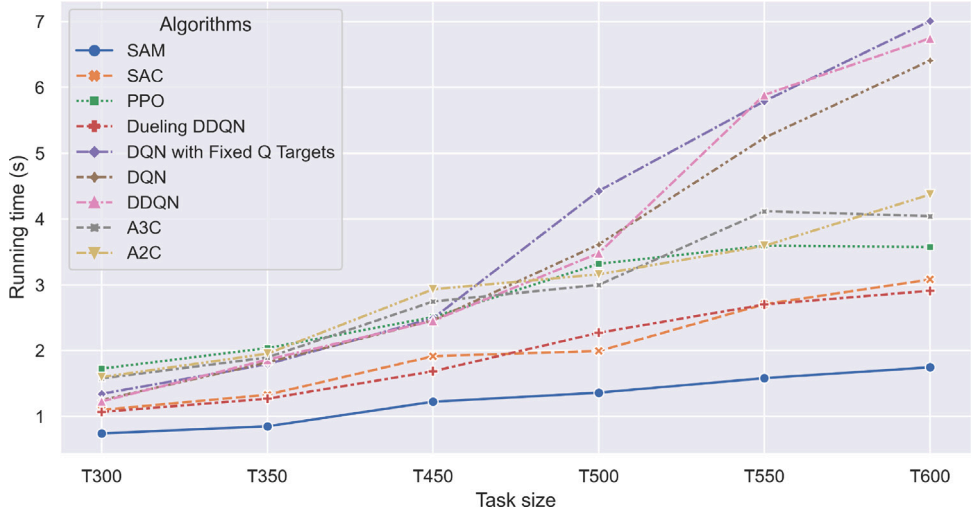

Fig. 8. Running time of DRL algorithms implemented on instance-sets of $T300, T350, T450, T500, T550, T600$. All algorithms are trained on the instance-set of $T400$.

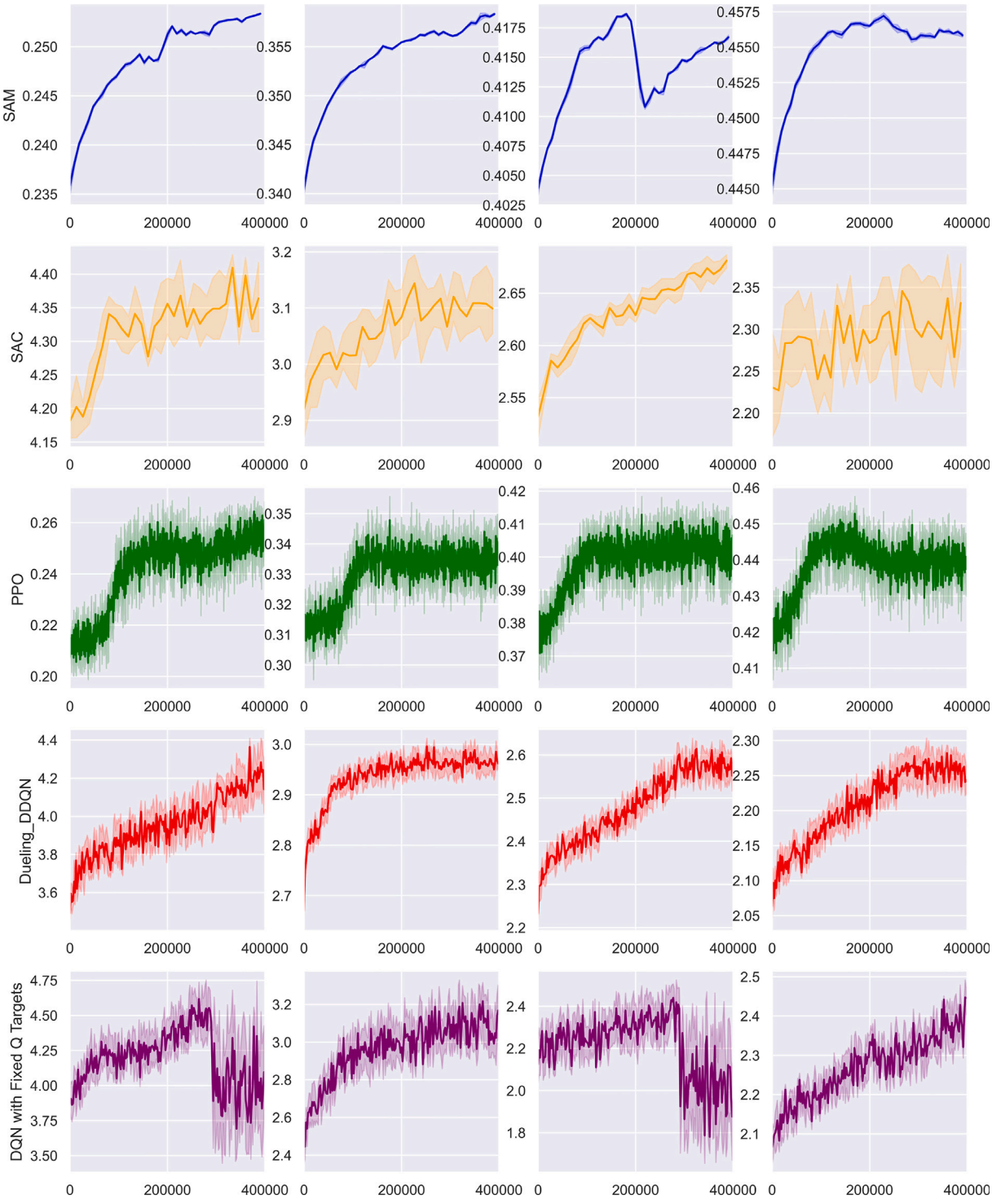

Fig. C.9. Training curves of SAM, SAC, PPO, Dueling DQN, DQN with Fixed Q Target.

Appendix A. Multi-Head Attention (MHA)

To get an intuition for query, key, and value in MHA, we start with an example. The query-key–value concept is analogous to retrieval systems. For example, when you search for websites on Google, the search engine will map your query (text in the search bar) against a set of keys (web title, description) associated with candidate websites in their database, then present you the best matched websites (values). Specifically, $MHA^{(l)}(h_0^{(l-1)} \dots h_{n^{me}+1}^{(l-1)})$ represents the MHA operation that extracts different types of information among $h_0^{(l-1)}, \dots h_{n^{me}+1}^{(l-1)}$ with multi-heads. $MHA^{(l)}$ means the MHA operation in the l module, and the inputs $h_0^{(l-1)}, \dots h_{n^{me}+1}^{(l-1)}$ are from the $l-1$ module. MHA has been proven to be efficient for information extraction [46]. $MHA^{(l)}(h_0^{(l-1)}, \dots h_{n^{me}+1}^{(l-1)})$ is computed by:

$$MHA^{(l)}(h_0^{(l-1)}, \dots, h_{n^{me}+1}^{(l-1)}) = Concat(Head_1^{(l)}(h_0^{(l-1)}, \dots, h_{n^{me}+1}^{(l-1)}), \dots, Head_H^{(l)}(h_0^{(l-1)}, \dots, h_{n^{me}+1}^{(l-1)})) \quad \text{(A.1)}$$

where $Concat(Head_i^{(l)}) = W^c Head^{(l)}$, and W^c is a weighting parameter matrix. $Head_i^{(l)} \in \{Head_1^{(l)}, \dots, Head_H^{(l)}\}$. H represents the number of

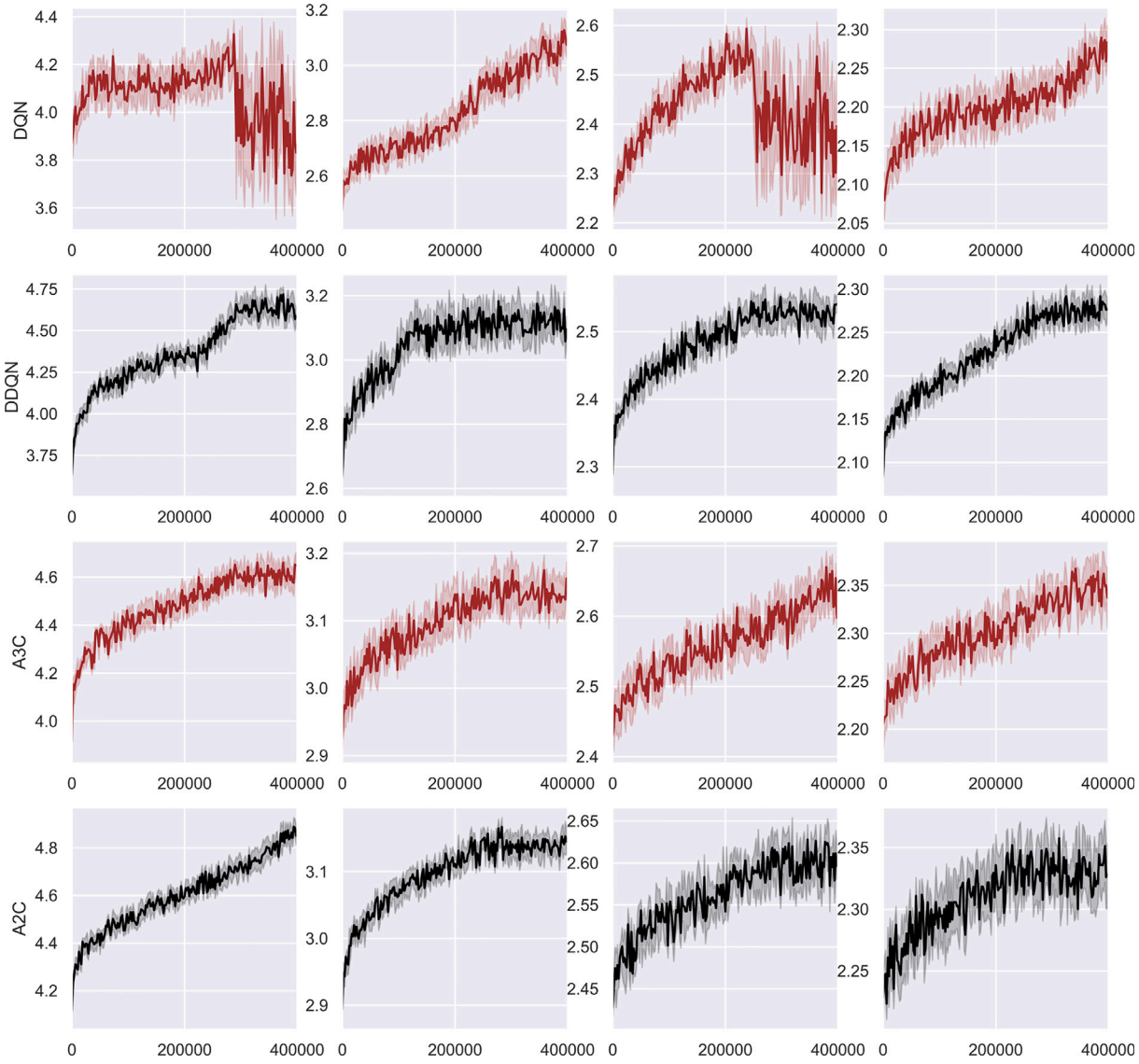

Fig. C.10. Training curves of DQN, DDQN, A3C, A2C.

heads. $Head_i^{(l)}$ represents the one-head attention operation, and H heads of $Head_1^{(l)}(\cdot), \ldots, Head_H^{(l)}(\cdot)$ means that they extract information from H perspectives with H heads of attention operations. Attention $Head^{(l),i}$ of the location i in each head is computed by query, key, and value, and they are obtained from the $h_i^{(l-1)}$. Query q_i, key k_i, and value v_i of a one-head attention are represented by:

$$q_i = W^q h_i^{(l-1)}, k_i = W^k h_i^{(l-1)}, v_i = W^v h_i^{(l-1)} \tag{A.2}$$

where W^q, W^k, W^v are parameters for query, key, and value, respectively. For $i, j \in [0, n^{me}+1]$, the compatibility $a_{i,j}$ is represented by [45]:

$$a_{i,j} = softmax(\frac{q_i^T k_j}{\sqrt{d_k}}) \tag{A.3}$$

where $softmax(x_{i,j}) = \frac{e^{x_{i,j}}}{\sum_j e^{x_{i,j}}}$, and d_k is the dimension of the key. Finally, $Head^{(l)}(h_0^{(l-1)}, \ldots, h_{n^{me}+1}^{(l-1)})$ is calculated as:

$$Head^{(l),i}(h_0^{(l-1)}, \ldots, h_{n^{me}+1}^{(l-1)}) = \sum_j a_{i,j} v_j, \forall i \in [0, n^{me}+1] \tag{A.4}$$

where v_j is denoted as Eq. (A.2), and $Head^{(l),i}, \forall i \in [0, n^{me}+1]$ means the one head attention for i. Then, we can obtain the MHA by Eq. (A.1) based on results of one-head attentions.

Appendix B. State representation and reward function for step-based DRL algorithms

For these step-based DRL algorithms, defining the state and reward is of great significance. According to the definitions described in [20,22,23] and numerous implementations of experiments, we represent the state St_t in the t step as:

$$St_t = [sr_t, st_t, pg_t, cl_t, locs_t, m_t] \tag{B.1}$$

where sr_t is the manufacturing service needed in the t step, st_t is the vector of service processing time of sr_t in enterprises. $pg_t = \frac{current\ \ finished\ \ tasks}{all\ \ tasks} \in [0,1]$ is a ratio of the number of current finished tasks and all tasks indicating the progress of scheduling. cl_t denotes the two-dimension coordinate of the current location, and $locs_t$ is a vector representing the distances between the current location and other locations of enterprises. $m_t = [0, 0, 1, 1, \ldots]$ is the mask that

masking the infeasible selections as 0 otherwise 1. The state St_t is changing during the scheduling and represents the current situation of scheduling. Besides, the state st_t is normalized to accelerate the training.

Based on the reward definition method described in [54,55], the reward function for these step-based DRL algorithms is described as algorithm 3. The Obj in algorithm 3 is the total objective in Eq. (4). We set some leading signals in the reward function to promote the convergence of DRL algorithms.

Algorithm 3: Reward function r_t for step-based DRL algorithms

Input: $done_t$: Boolean value indicating whether this episode is finished or not in the t step
$action_{t+1}$: the selected action for the $t+1$ step.
Obj_t: the total objective in the t step.
Output: r_t: The reward in the t step.
Initialize: $last\_Obj \leftarrow 0$
if $done_t \leq true$ **then**
  if $action_{t+1}$ *points at the initial location or the dummy location* **then**
    if $action_{t+1}$ *points at the initial location* **then**
      $r_t \leftarrow -2$
    else
      $r_t \leftarrow -5$
    end
  else
    $\delta Obj \leftarrow Obj_t - last\_Obj$
    $r_t \leftarrow -\frac{\delta Obj}{10}$
    $last\_Obj \leftarrow Obj_t$
  end
else
  $r_t \leftarrow -15 \cdot Obj$
end

In Section 5, we adopt the same state representation and reward function for SAC, PPO, Dueling DDQN, DQN with Fixed Q Targets, DQN, DDQN, A3C, and A2C. For value-based algorithms, including Dueling DDQN, DQN with Fixed Q Targets, DQN, and DDQN, we set the length of experience replay as 1. Mnih set the length as 4 and represented the state of the Atari video games with 4 frames of images [56]. We tried to increase the length of experience replay when training algorithms for CMfg-SPs, but the performances are not as satisfactory as that of 1 length.

Appendix C. Training results for experiments

This section presents the training curves of our proposal and the compared DRL algorithms. As these DRL algorithms are different in calculating the episode reward, we record the QoS instead of the episode reward to indicate the changes in training. The training curves of SAM, SAC, PPO, Dueling DDQN, DQN with Fixed Q Targets DQN, DDQN, A3C, and A2C are shown in Fig. C.10. We train each algorithm with four instance-sets of $T100, T200, T300, T400$ repeatedly for three times. The shaded area represents the fluctuation of the values during the three-times training results, and the solid line denotes their average values. Most of them converge to stable values after training for 400000 episodes (see Fig. C.9).

References

[1] Zhang L, Luo Y, Tao F, Li BH, Ren L, Zhang X, Guo H, Cheng Y, Hu A, Liu Y. Cloud manufacturing: a new manufacturing paradigm. Enterprise Inform Syst 2014;8(2):167–87.

[2] Li BH, Zhang L, Wang SL, Tao F, Cao J, Jiang X, Song X, Chai X. Cloud manufacturing: a new service-oriented networked manufacturing model. Comput Integr Manufact Syst 2010;16(1):1–7.

[3] Xu X. From cloud computing to cloud manufacturing. Robot Comput-Integr Manuf 2012;28(1):75–86.

[4] 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.

[5] Halty A, Sánchez R, Vázquez V, Viana V, Piñeyro P, Rossit DA. Scheduling in cloud manufacturing systems: Recent systematic literature review. 2020.

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

[7] Wang X, Zhang L, Liu Y, Li F, Chen Z, Zhao C, Bai T. Dynamic scheduling of tasks in cloud manufacturing with multi-agent reinforcement learning. J Manuf Syst 2022;65:130–45.

[8] Hu Y, Shi C, Lv W, Liu Y, Wang XV. Optimization of manufacturers based on agent in cloud manufacturing. Internat J Model Simul Sci Comput 2022.

[9] Park J, Chun J, Kim SH, Kim Y, Park J. Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning. Int J Prod Res 2021;59(11):3360–77.

[10] 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.

[11] 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.

[12] Luo S. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. Appl Soft Comput 2020;91:106208.

[13] Dorigo M, Birattari M, Stutzle T. Ant colony optimization. IEEE Comput Intell Mag 2006;1(4):28–39.

[14] Mirjalili S. Genetic algorithm. In: Evolutionary algorithms and neural networks. Springer; 2019, p. 43–55.

[15] Deb K, Agrawal S, Pratap A, Meyarivan T. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: International conference on parallel problem solving from nature. Springer; 2000, p. 849–58.

[16] Nasiri MM, Salesi S, Rahbari A, Meydani NS, Abdollai M. A data mining approach for population-based methods to solve the JSSP. Soft Comput 2019;23(21):11107–22.

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

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

[19] Nazari M, Oroojlooy A, Snyder LV, Takáč M. Reinforcement learning for solving the vehicle routing problem. 2018, arXiv preprint arXiv:1802.04240.

[20] 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.

[21] Henderson P, Islam R, Bachman P, Pineau J, Precup D, Meger D. Deep reinforcement learning that matters. In: Proceedings of the AAAI conference on artificial intelligence, vol. 32. 2018.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] Devlin J, Chang M-W, Lee K, Toutanova K. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018, arXiv preprint arXiv:1810.04805.

[28] Liu Z, Lin Y, Cao Y, Hu H, Wei Y, Zhang Z, Lin S, Guo B. Swin transformer: Hierarchical vision transformer using shifted windows. In: Proceedings of the IEEE/CVF international conference on computer vision. 2021, p. 10012–22.

[29] Kool W, Van Hoof H, Welling M. Attention, learn to solve routing problems!. 2018, arXiv preprint arXiv:1803.08475.

[30] Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M. Deterministic policy gradient algorithms. In: International conference on machine learning. PMLR; 2014, p. 387–95.

[31] Hu L, Liu Z, Hu W, Wang Y, Tan J, Wu F. Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinforcement learning with graph convolutional network. J Manuf Syst 2020;55:1–14.

[32] 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.

[33] Samsonov V, Kemmerling M, Paegert M, Lütticke D, Sauermann F, Gützlaff A, Schuh G, Meisen T. Manufacturing control in job shop environments with reinforcement learning.. In: ICAART (2). 2021, p. 589–97.

[34] Chen X, Tian Y. Learning to perform local rewriting for combinatorial optimization. 2019.
[35] Zhang C, Song W, Cao Z, Zhang J, Tan PS, Xu C. Learning to dispatch for job shop scheduling via deep reinforcement learning. 2020, arXiv preprint arXiv:2010.12367.
[36] Hameed MSA, Schwung A. Reinforcement learning on job shop scheduling problems using graph networks. 2020, arXiv preprint arXiv:2009.03836.
[37] Seito T, Munakata S. Production scheduling based on deep reinforcement learning using graph convolutional neural network.. In: ICAART (2). 2020, p. 766–72.
[38] Wang B, Li H, Lin Z, Xia Y. Temporal fusion pointer network-based reinforcement learning algorithm for multi-objective workflow scheduling in the cloud. In: 2020 international joint conference on neural networks (IJCNN). IEEE; 2020, p. 1–8.
[39] Tang H, Wang A, Xue F, Yang J, Cao Y. A novel hierarchical soft actor-critic algorithm for multi-logistics robots task allocation. IEEE Access 2021;9:42568–82.
[40] Tong Z, Ye F, Liu B, Cai J, Mei J. DDQN-TS: A novel bi-objective intelligent scheduling algorithm in the cloud environment. Neurocomputing 2021.
[41] Tong H, Zhu J. A two-layer social network model for manufacturing service composition based on synergy: A case study on an aircraft structural part. Robot Comput-Integr Manuf 2020;65:101933.
[42] Velmurugan P, Ashok B. Improving the quality of service by continuous traffic monitoring using reinforcement learning model in vanet environment. Internat J Model Simul Sci Comput 2022.
[43] Hausknecht M, Stone P. Deep recurrent q-learning for partially observable mdps. In: 2015 aaai fall symposium series. 2015.
[44] Vinyals O, Fortunato M, Jaitly N. Pointer networks. 2015, arXiv preprint arXiv:1506.03134.
[45] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. In: Advances in neural information processing systems. 2017, p. 5998–6008.
[46] Li J, Tu Z, Yang B, Lyu MR, Zhang T. Multi-head attention with disagreement regularization. 2018, arXiv preprint arXiv:1810.10183.
[47] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning. PMLR; 2015, p. 448–56.
[48] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016, p. 770–8.
[49] Wu Y-C, Tseng B-H, Rasmussen CE. Improving sample-efficiency in reinforcement learning for dialogue systems by using trainable-action-mask. In: ICASSP 2020-2020 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE; 2020, p. 8024–8.
[50] Peters J. Policy gradient methods. Scholarpedia 2010;5(11):3698.
[51] Rosner B. A generalization of the paired t-test. J R Stat Soc Ser C Appl Stat 1982;31(1):9–13.
[52] Zhang Z. Improved adam optimizer for deep neural networks. In: 2018 IEEE/ACM 26th international symposium on quality of service (IWQoS). IEEE; 2018, p. 1–2.
[53] Chen L, Lu K, Rajeswaran A, Lee K, Grover A, Laskin M, Abbeel P, Srinivas A, Mordatch I. Decision transformer: Reinforcement learning via sequence modeling. 2021, arXiv preprint arXiv:2106.01345.
[54] Sutton RS, Barto AG. Reinforcement learning: An introduction. MIT Press; 2018.
[55] Ng AY, Harada D, Russell S. Policy invariance under reward transformations: Theory and application to reward shaping. In: Icml, vol. 99. 1999, p. 278–87.
[56] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, et al. Human-level control through deep reinforcement learning. Nature 2015;518(7540):529–33.