Multi-D3QN: A Multi-strategy Deep Reinforcement Learning for Service Composition in Cloud Manufacturing

Jun Zeng(✉), Juan Yao, Yang Yu, and Yingbo Wu

School of Big Data and Software Engineering, Chongqing University, Chongqing, China
{zengjun, yaojuan, yuyang96, wyb}@cqu.edu.cn

Abstract. Service composition is an indispensable technology in the cloud manufacturing process to ensure the smooth execution of tasks. To implement effective and accurate service composition strategies, many researchers choose to use Meta-heuristics algorithms with strong optimization capabilities. However, as users' demand of personalized products increasing, dynamic service composition is essential. Meta-heuristics algorithms lack dynamic adaptability, so they are not suitable for solving complex and dynamic service composition problems. Deep Reinforcement Learning (DRL) algorithm is difficult to reach a stable state, when the hyper-parameters and rewards in the algorithm are not properly designed. To solve these problems, we propose a Multi-strategy Deep Reinforcement Learning (DRL) algorithm, named Multi-D3QN, which combines the basic DQN algorithm, the dueling architecture, the double estimator and the prioritized replay mechanism. Meanwhile, we add some strategies such as instant reward, the ε-greedy policy and a heuristic strategy to ensure better performance of the algorithm in dynamic environment. Experiments show that our proposed method not only adapt to the dynamic environment, but also obtain a better solution.

Keywords: Cloud manufacturing · Dynamic service composition · Quality of service · Deep reinforcement learning

1 Introduction

Today, the quality of life is constantly improving, and the demand of users for personalized products is also increasing [1]. However, due to resource constraints, the manufacturing resources and capabilities of a single enterprise have been unable to meet user's needs. In order to solve this problem, enterprises need to collaborate effectively by sharing manufacturing resources and capabilities [2]. A new service-oriented intelligent manufacturing model known as Cloud manufacturing (CMfg) has been proposed [3]. In the CMfg platform, the shared manufacturing resources and capabilities by enterprises are encapsulated into services and provided to users through the internet for selection. Service composition and optimization selection (SCOS) [4, 5] is considered to be the critical technology to realize the sharing function of resources and capabilities in the CMfg platform [6, 7]. Based on different composition structures,

H. Gao and X. Wang (Eds.): CollaborateCom 2021, LNICST 407, pp. 225–240, 2021.
https://doi.org/10.1007/978-3-030-92638-0_14

SCOS integrates various fine-grained services with different functions into coarse-grained services with comprehensive functions to deal with complex manufacturing tasks, and then meets the needs of users. Widespread attention has been received dealing with the quickly and effectively optimal combination strategy problem.

So far, many researchers have proposed a lot of heuristic methods for the SCOS problem [8, 9]. Although these methods have promoted the research work of SCOS, they lack adaptability to dynamic environments. For instance, when the environments change, the algorithms may need to be redesigned. Therefore, SCOS problems require new and innovative methods. Some researchers considered that Reinforcement Learning (RL) is adaptable to dynamic environments, and tried to use RL to solve the SCOS environment change problems [10–12]. Wang et al. [10] used Deep Reinforcement Learning (DRL) to solve the SCOS problems, in which the state set is divided into two state subsets, and behavior strategies are selected in different types of states. These methods can re-adjust the system to reach a stable state through adaptability when the environment changes suddenly. And there is still a key issue that needs to be addressed: If the hyper-parameters and rewards in the algorithm are not properly designed, it is difficult to return to a stable state. Therefore, some strategies need to be adopted to avoid this problem.

To solve the problems above, a Multi-strategy DRL algorithm, named Multi-D3QN, that combines the basic DQN algorithm, the dueling architecture, the double estimator and the prioritized replay mechanism is proposed. Specifically, the dueling architecture can improve the convergence speed of the algorithm. The double estimator can overcome the estimation problem, and the Prioritized replay mechanism can accelerate the learning speed of the algorithm. Moreover, three different strategies are added to the model, which leads the algorithm to return to a stable state and get better solutions in a dynamic environment. Based on Multi-D3QN, the dynamic SCOS problem in cloud manufacturing is studied. Experimental results reveal that the Multi-D3QN method can achieve better performance in cases with hyper-parameters and rewards that are inappropriate.

In summary, our contributions in this paper are as follows:

- Aiming at the problem that the meta-heuristic methods have complex design flow in the dynamic environment, we proposed Multi-D3QN, which combines DQN, the dueling architecture, the double estimator and the prioritized replay mechanism, and generates an algorithm with a better solution in the dynamic environment by integrating their advantages.
- Aiming at the shortcoming that DRL is difficult to return to a stable state when the service is unavailable, due to the values of the parameters are not appropriate. We developed a strategy, which according to heuristic rules, to shield unavailable services.
- To get the best performance of our algorithm, we compared different strategies in experiments to determine the final algorithm.

The remainder of this paper is organized as follows: Sect. 2 introduced the related work. The problem description and MDP-based CMfg service composition are presented in Sect. 3. The detailed description of framework is given in Sect. 4. In Sect. 5, the comparative experiment and analysis are performed to verify the proposed approach. Finally, the conclusion is provided in Sect. 6.

2 Related Work

As a key issue for sharing and collaborating of manufacturing resources and capabilities among enterprises on the CMfg platform, SCOS is gaining ever-increasing significance. In previous work, there have been many studies on SCOS.

2.1 Meta-heuristics-based Service Composition

Cloud manufacturing SCOS problem is an NP-hard problem, and it is difficult to find the optimal solution in a limited time. Many researchers attention to the strong optimization ability of the Meta-heuristics algorithms, which can find the optimal or nearly optimal solution [7]. Yang et al. [5] proposed an improved gray wolf optimizer algorithm (IGWO) by improving the control factor and location update method in the gray wolf optimizer algorithm, which improved the search ability of the algorithm and ensured the accuracy of the scheme. On this basis, they also proposed an enhanced gray wolf optimization algorithm (EMOGWO) [4] for multi-objective SCOS problems, which made three improvements to the basic multi-objective GWO and overcame the shortcoming of local optimum and less diversity in multi-objective CMfg SCOS problems. In Zhou et al. [13], according to the high flexibility and complexity characteristics of CMfg, a hybrid artificial colony method (HABC) is designed to solve the large-scale service composition solution space problem by introducing archimedean copula estimation of distribution algorithm (ACEDA) probability model and the chaos operators of global best-guided artificial bee colony, which has the advantages of high performance and high stability. Fazeli et al. [14] combined genetic algorithm (GA), particle swarm optimization (POS) and social spider algorithm (SSO), an ensemble optimization approach (EOA) is proposed. In this approach, the algorithm combination process is regarded as a black box and used a new operator to summarize the results. The algorithm has the characteristics of flexible expansion and easy composition, which improved the QoS value of the service composition.

2.2 RL/DRL-Based Service Composition

Lei et al. [15] proposed a reinforcement learning of Time-based Learning method, named TL, which can effectively control exploration and exploitation, so the service composition success rate is improved. Li et al. [12] based on Q-Learning algorithm, a multi-task oriented algorithm named multi-Q learning is proposed to realize subtask-assistance strategy for large-scale and adaptive service composition. It has faster learning speed and convergence compared with other methods. In Wang et al. [16] the model combines on-policy reinforcement learning and game theory, a new model for large-scale and adaptive

service composition based on multi-agent reinforcement learning is proposed, which can make the algorithm highly adapt to the dynamic environment, and multi-agent cooperation can quickly find the optimal solution. In Wang et al. [17] the service composition process is modeled as a Markov Decision Process (MDP) and trained with the LSTM-based deep Q-network method to find the optimal service composition. Liang et al. [11] established a cloud manufacturing service composition model based on deep reinforcement learning, the service composition process is modeled as a Markov Decision Process (MDP), and the reward function with the consideration of logistics is designed.

So far, the existing studies have not achieved satisfactory results whether using meta-heuristics algorithms or using reinforcement learning methods to solve SCOS problems. Especially when the hyper-parameters and rewards design in reinforcement learning are inappropriate, it is difficult to reach a stable state again when the environment changes. We are trying to find some strategies to change this situation. Therefore, a Multi-strategy DRL algorithm is developed in this paper.

3 Service Composition Problem Description and MDP-Based CMfg Service Composition

3.1 Problem Description

The process of SCOS can be divided into the following steps [5]: (1) Task decomposition, (2) Service discovery and matching, (3) Service composition and optimization selection. The detailed description of the SCOS process can be seen in the existing study [18]. This paper focuses mainly on the third step. In the process of performing tasks, due to the influence of many factors such as the change of tasks or users' requirements and unavailability services, the service composition process is highly dynamic. It is necessary to study these factors. The impacts of service unavailability are studied in this paper. In cloud manufacturing, there are many reasons for service unavailability, such as machine failures, enterprise departure, and service shutdown, etc. Hence, when there are unavailable services in the service chains, the SCOS algorithms need to quickly and effectively find the optimal service again from the service sets to replace the unavailable service, which can ensure the smooth completion of the task. As shown in Fig. 1.

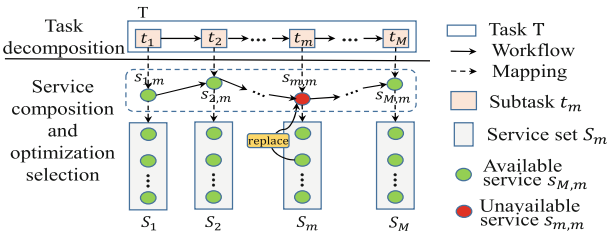

Fig. 1. Dynamic service composition flowchart

3.2 MDP-Based CMfg Service Composition

In the cloud manufacturing environment, a task T is usually decomposed into several subtasks t, T = $\{t_1, t_2, \ldots, t_i, \ldots, t_n\}$, where N is the number of subtasks of T, t_i is the i-th subtask of T. The service composition process for T can be modeled as a MDP.

State. In MDP an optimal(or near-optimal) service is selected for each subtask t_i, i = 1, 2, …, N. There are two states for t_i and as the service composition progresses, the states of the subtasks and the services will change. The detailed description of the states is comprehensively explained in the existing study [11].

Action. There is an action set for each t_i. An action corresponds the selection of a service from a service set for t_i, and the result of an action is that a service is selected. In the whole system, the number of action sets is equal to the number of service sets, and the size of the action set for each subtask is equal to the size of the service set corresponding to the subtask.

Reward Function. The reward function plays an important role in SCOS by guiding the DRL agent to find the optimal service composition solution. When a service is selected from the service set in the current state, we get a reward r from the environment after performing the action. Because the measurement standards and units of each QoS attribute are different. Before calculating the reward, it is necessary to normalize the QoS value of each indicator. The reward is calculated based on the normalized QoS values:

$$Q_i^+ = \begin{cases} \frac{q_i - q_{min}^i}{q_{max}^i - q_{min}^i}, q_{min}^i \neq q_{max}^i \\ 1, \ q_{min}^i = q_{max}^i \end{cases} \tag{1}$$

$$Q_i^- = \begin{cases} \frac{q_{max}^i - q_i}{q_{max}^i - q_{min}^i}, q_{min}^i \neq q_{max}^i \\ 1, \ q_{min}^i = q_{max}^i \end{cases} \tag{2}$$

In Eqs. (1) and (2) q_{min}^i and q_{max}^i indicate the minimum and maximum aggregated values of the i-th QoS attribute for all possible composition paths, q_i is the aggregated values of the i-th QoS attribute. The aggregation functions of QoS attributes are shown in [19]. For positive indices (Q_i^+), the bigger value of QoS is, the better quality is, like reliability, security and availability, and the others are just opposite (Q_i^-), such as time

and cost. In this paper, a bigger value means the performance is better for all indices through normalization. The QoS value of the normalized attribute can be integrated into an overall QoS value through a simple weighting method [20].

4 Proposed Algorithm Framework

In this paper, we propose a Multi-strategy DRL algorithm (Multi-D3QN), which combines the basic DQN algorithm, the dueling architecture, the double estimator and the prioritized replay mechanism. Specifically, the dueling architecture can shield some actions that have little influence on the optimal value function, so that the algorithm can quickly obtain the optimal value function and improve the convergence speed of the algorithm. However, the actions found by the dueling architecture may have overestimation, which can be avoided by the double estimator. The prioritized replay mechanism is used to find the sample data effectively that needs to be learned, which can promote faster learning of the algorithm. In addition, a heuristic strategy is added in the algorithm to shield unavailable services to adapt to the dynamic environment. The overall framework of our method is shown in Fig. 2, and the specific operations are indicated in Algorithm 1. Then, we will introduce more details of the framework.

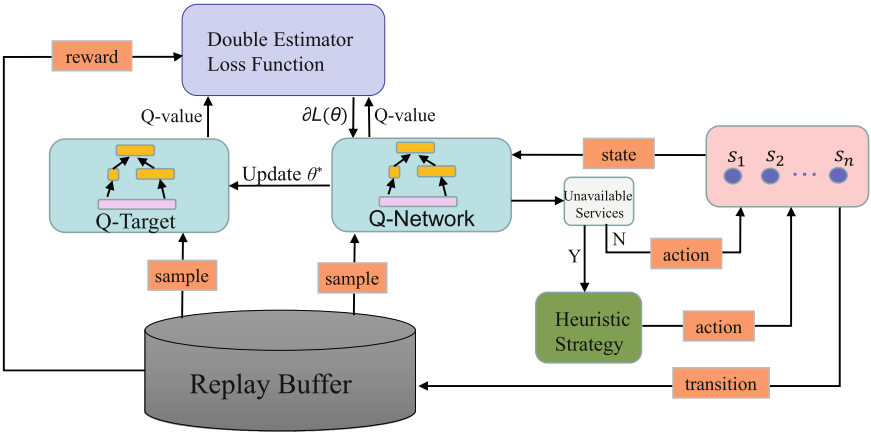

Fig. 2. Framework of the Multi-D3QN

Algorithm 1. Multi-D3QN Algorithm

Input: initial exploration and final exploration ε; learning rate α; mini-batch size b; sampling weights β; discounted rate γ.
Output: reward r.

// Initialization
Initialize replay memory D with capacity N and initial priority p_1=1;
Initialize action-value function Q with Q-network parameters θ;
Initialize action-value function Q* with Q-target parameters θ*;
for each episode **do**;
  for each t **do**;
    // The Dueling Architecture and The Strategy for Adaptability
    if services unavailability **then**
      Into The shielding algorithm:
      With probability ε select a random action a_t;
        if $a_t(o)$<0 then // $a_t(o)$ is the Qos value of the corresponding service
          Shield unavailability services;
          Select a random action a_t;
        end if
      Otherwise select $a_t = \arg max_a Q(s_t, a, \theta)$;
        if $a_t(o)$<0 then
          Shield unavailability services;
          Select a random action a_t;
        end if
    else
      With probability ε select a random action a_t;
      Otherwise select $a_t = \arg max_a Q(s_t, a, \theta)$;
    end if
    // The Prioritized Replay Mechanism
    Execute action a_t, observe reward r_t and next state s_{t+1}, $s_t = s_{t+1}$;
    Store transition (s_t, a_t, r_t, s_{t+1}) in D with maximal priority $p_1 = max_{i<t} p_i$;
    Sample mini-batch of transition (s_j, a_j, r_j, s_{j+1}), for each sample $j \sim p(j) = p_j / \sum_j p_i$;
    Compute importance-sampling weight $w_j = \left(N * p(j)\right)^{-\beta} / \max_i(w_i)$
    // The Double Estimator
    if episode terminates at step $j+1$ **then**
      set $y_j = r_j$;
    else
      $y_j = r_j + \gamma Q(s_{t+1}, \arg max_a Q(s_{t+1}, a, \theta), \theta^*)$
    end if
    Update Q-network parameters θ with a loss function of $\frac{1}{b}\sum_{j=1}^{b} w_j (y_j - Q(s_t, a_t, \theta))^2$;
    // The Prioritized Replay Mechanism
    Compute TD-error $\delta_j = y_j - Q(s_t, a_t, \theta)$;
    Update transition priority $p_j \leftarrow |\delta_j|$;
    Every c steps reset $\theta^* \leftarrow \theta$;
  end for
  Continue to iterate until the convergence condition is satisfied
end for

4.1 DQN Algorithm

DQN, in addition to Q-Learning, is the basic algorithm in Reinforcement Learning (RL). It uses deep neural network to approximate a value function instead of Q-table in Q-Learning, which overcomes the shortcoming that Q-Learning can't deal with a large

amount of data. In DQN, two deep neural networks are constructed: Q-network and Q-target. It uses Q-network to estimate the current value function, and Q-target to generate the target Q-value. During learning, DQN uses random sampling to extract samples from the Replay Buffer, then update the parameters in Q-network according to the loss function, and finally get the maximum future rewards [11]. The methods used in Sects. 4.2, 4.3, 4.4 and 4.5 are all improvements to improve the DQN framework.

4.2 The Dueling Architecture

The dueling architecture is an improvement of the deep learning network structure in DQN, so that the algorithm can learn something faster. The core idea is that the last layer of the network is generated with two quantities: the value of the state V(s) and the advantage of actions in this state A(s, a), i.e.

$$\mathrm{Q}(s,a) \;=\; V(s)+A(s,a) \tag{3}$$

This means that $A(s,a)$ is a gain that implies how much Q-value exceeds the expected value when action a is selected [21]. Advantage $A(s,a)$ could be positive or negative. The division of the value and the advantage in the network architecture improves the stability and convergence speed of the algorithm.

4.3 The Double Estimator

The traditional RL such as DQN and Q-leaning use the single estimator to update parameters, which will lead to overestimation. This means that the algorithm will suffer a lot of negative effects. In order to solve this problem, Hasselt et al. [22] proposed a double estimation method. Its core idea can be briefly summarized as: In the learning phase, two different network parameters are used to estimate the value function, in form:

$$y_j \;=\; r_j+\gamma Q(s_{t+1}, argmax_a Q(s_{t+1},a,\theta),\theta^*) \tag{4}$$

Where r_j is the j-th reward take from replay buffer D; s_{t+1} is the next state at iteration $t+1$. θ^* and θ represent the parameters of the Q-network and the Q-target, respectively. γ is discount rate.

The calculation formula of the loss function is as follows:

$$L(\theta) \;=\; \frac{1}{b}\sum\nolimits_{j=1}^{b} w_j\big(y_j - Q(s_t,a_t,\theta)\big)^2 \tag{5}$$

where $Q(s_t,a_t,\theta)$ is the output of the Q-network, s_t, a_t represent the state and action at iteration t, respectively. b is sample mini-batch from replay buffer D, w_j is the sample weight.

4.4 The Prioritized Replay Mechanism

DQN uses a random sampling strategy to learn, that is, samples are randomly selected from the experience replay buffer, which ignores the importance of conversion. Some scholars have proposed priority methods to make experience playback more efficient [23]. The core idea is that RL agent can learn more efficiently from experiences. Specifically, more samples of experience with high expectations are measured by TD error.

$$\delta_j = y_j - Q(s_t, a_t, \theta) \tag{6}$$

Where y_j is the Q-value in the double estimator at iteration $t+1$.

4.5 The Strategy for Adaptability

In this subsection, a heuristic strategy is developed for the dynamic SCOS problems. It is well-known that when an agent interacts with the environment, given a state, an agent possibly gets an action set, which has different values. In this paper, the value of an action is the QoS value of service from the service set.

According to the existing research [10, 11], when services are unavailability, the QoS value of the whole service chain will become smaller, so we give the following definition:

Unavailability service: In a state, when the QoS values of some actions become anomaly (according to many experiments we performed, in this paper, we have observed that the abnormal value of an action has a negative QoS value), it can be judged that the service corresponding to the action is an unavailability service.

If a service chain contains unavailability services, then the service chain will be greatly affected. Not only the overall QoS value is reduced, but also the execution of the task will be affected. However, for the most part, the DRL method to return to a stable state within a limited time is very hard, due to the fact that the appropriate values of hyper-parameters and rewards have not been found. Therefore, in order to ensure the smooth progress of the task, these unavailability services need to be shielded by heuristic strategies.

The heuristic strategy will be elaborated in this section. As the program starts, the average QoS value m_1 will be calculated. If the action values are less than zero, the services will be judged unavailable. At the same time, the average QoS value m_2 will also be calculated. When it comes to the condition that m_2 is less than m_1, which means the model is not adaptive to a stable state, the heuristic strategy will start. The pseudo code is as Algorithm 2.

Algorithm 2: The Shielding Algorithm

Input: action sets
Output: optimal action
1: **for** each episode **do**
2: **for** each t **do**;
3: Calculate the average QoS value m_1
4: **if** the action values <0 **then**
5: Calculate the average QoS value m_2
6: **end if**
7: **if** m_2<m_1 after a period of time **then**
8: Find out actions with QoS value less than zero
9: Shield all actions with values less than zero
10: Randomly select an action as the optimal action
11: **end if**
12: Continue to iterate until the convergence condition is satisfied
13: **end for**
14: **end for**

5 Experiments

We conduct the experiments to assess the proposed Multi-strategy Deep Reinforcement Learning algorithm (Multi-D3QN) on three aspects: effectiveness, adaptability and robustness. Experiment results obtained with Multi-D3QN are compared with those results with other three competitive methods DQN, dueling-DQN and double-DQN.

5.1 Experiment Setting

- **Dataset.** In the experiment, we mainly consider five QoS attributes, including reliability, security, availability, time and cost. Since there is no publicly available dataset for cloud manufacturing services, without loss of generality, all QoS attributes values are randomly generated between [0.7, 0.95] [13, 14]. Due to the QoS value of the data from the dataset calculated by Eqs. (1) and (2) are too small, the comparison of results between the algorithms is not obvious. Therefore, the weights of QoS attributes are equal to 1 instead of decimals which usually used. Besides, each task is decomposed into 30 subtasks, each subtask corresponds to 30 candidate services.
- **Network Structure.** The neural network for Multi-D3QN model has an input layer, a LSTM layer and two fully connected hidden layer with LSTM layer having 64 neurons and the other layers having 30 neurons. The hyper-parameters used for Multi-D3QN training are shown in Table 1.

Table 1. Hyper-parameters for the training of Multi-D3QN

Parameters	Value	Description
N	10000	The capacity of experience replay buffer
c	500	Target network update step
b	32	Mini-batch size
α	0.01	Initial learning rate used by Adam
lp	1000	Decay step used by exponential decay
lr	0.96	Decay rate used by exponential decay
γ	0.9	Discount factor
ε_{min}	0.01	Probability of initial exploration
ε_{end}	0.9	Probability of final exploration
β	0.4	Sampling weights

The ε-greedy policy is used with ε annealed linearly from 0.9 to 0.01

5.2 Result Analysis

(1) Learning Rate

Learning rate plays a very important role in algorithm performance. If the learning rate is too high, the model will not converge. However, if the learning rate is too low, the model will converge too slowly or fail to learn. So it is very important to choose a suitable learning rate. The experimental results are shown in Fig. 3. It can be observed that D3QN with $\alpha = 0.01$ performs best than that for $\alpha = 0.001$ and $\alpha = 0.1$, because the result of $\alpha = 0.01$ obtains a higher QoS value and a quicker convergence speed. Specifically, the D3QN algorithm with $\alpha = 0.01$ converged at about 46000^{th}, but at about 50000^{th} and 65000^{th}, respectively, for $\alpha = 0.001$ and $\alpha = 0.1$; with $\alpha = 0.01$, the D3QN algorithm can achieve a higher QoS values than that for $\alpha = 0.1$ and $\alpha = 0.001$. It can also be seen that D3QN with $\alpha = 0.01$ and $\alpha = 0.001$ have almost the same the range of fluctuations, while $\alpha = 0.01$ has a higher Qos value. This is because the

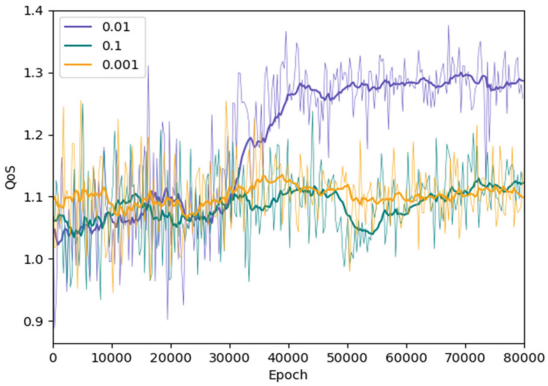

Fig. 3. Different learning rate for D3QN

learning rate is too small, which leads to the slower learning of the algorithm. However, the D3QN with $\alpha = 0.1$ the range of fluctuations is larger than the other two learning rates, which is because the large learning rate leads to the difficulty of model convergence. Therefore, $\alpha = 0.1$ is selected for the following experiments.

(2) Effectiveness and Efficiency

In order to verify the effectiveness of our proposed method. The D3QN is compared with DQN, dueling DQN and Double DQN. Experimental results are obtained with four different algorithms, as shown in Fig. 4. According to the results, there is little difference between the results of the algorithms in the initial stage, but with the number of iterations increasing, D3QN obtains a larger QoS value than DQN, dueling DQN and double DQN. For the convergence of the algorithms, DQN converges at the 55000^{th}, and double DQN converges at around the 48000^{th}. Dueling DQN and D3QN converge at around the 45000^{th} and 46000^{th}, respectively, which indicates that D3QN has the advantage of faster convergence speed of dueling DQN. Compared with other algorithms, D3QN not only has the advantage of faster convergence, but also can get better solutions. The reason is that the dueling architecture enables to boost the convergence of the algorithm faster, the double estimator can overcome the shortcomings of overestimation and the prioritized replay mechanism can improve the learning speed of the sample to promote faster learning of the algorithm.

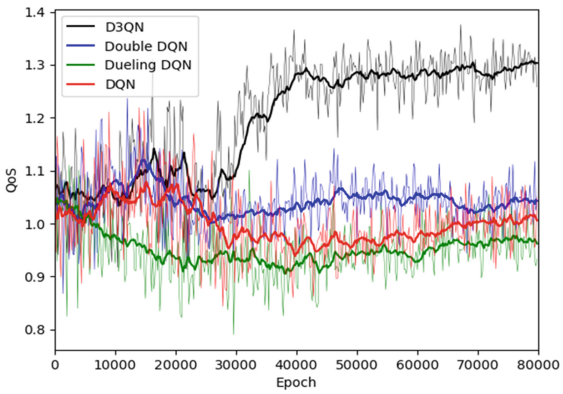

Fig. 4. Training curves for D3QN, DQN, dueling DQN and double DQN

(3) Influence of Strategies

To verify that our proposed algorithm is suitable for dynamic environments, we simulate a dynamic environment by randomly disabling the percentage of services. And to get the best performance of the algorithm, we study the influence of different strategies on the algorithm, i.e. D3QN (without reward, the ε-greedy policy and heuristic strategy), NS-D3QN (D3QN without the ε-greedy policy), Multi-D3QN (D3QN with reward, the ε-greedy policy and heuristic strategy). In particular, D3QN with reward means that the reward is equal to the QoS value of the selected service, after an action is executed. While D3QN without reward, it means that the reward is equal to zero.

These strategies start to change when the service is disabled. The detailed process is as follows: When the algorithm reaches a stable state, randomly disable 10% of the services at a certain episode, the episode is applied to the algorithm at 60000th episode, in this paper.

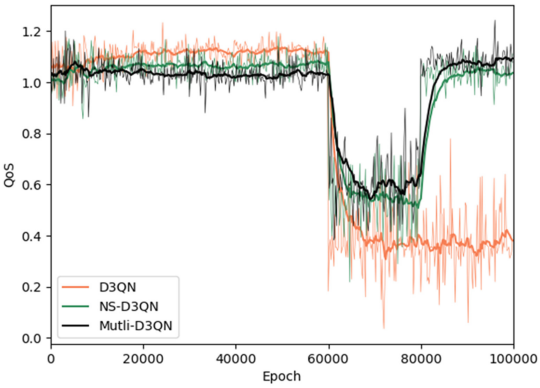

Fig. 5. The influence of different strategies

Figure 5 shows the experimental result. According to the results, the Multi-D3QN has the best results in this experiment. Similarly, NS-D3QN and D3QN provide the second and the worst results, respectively. Careful observation reveals that when the services are unavailable, the QoS values of the optimal solution are reduced. However, at 80000th, the Multi-D3QN and NS-D3QN return to a stable state, while this does not happen to the D3QN. This proves that unavailable services are shielded by the heuristic strategies in the Multi-D3QN and NS-D3QN, so that there are no unavailable services in the service compositions, thereby improving the QoS values. It can be observed that Multi-D3QN obtains a higher QoS value than NS-D3QN after 60000th, because the algorithm has a chance to jump out of the local optimal through the policy. In terms of reward, the algorithms with reward can find a higher solution than the algorithm without reward, the reason is that DRL can effectively adjust the parameters according to the real reward value. Through experiments, it can be concluded that Multi-D3QN with multiple strategies is better in terms of performance.

(4) Adaptability and Robustness

To verify the adaptability and robustness of the proposed algorithm, the Multi-D3QN is compared with other algorithms. In this experiment, the experiment setting is the same as 3. Figure 6 shows the experimental result of the Multi-D3QN, double DQN, DQN and dueling DQN, respectively. It can be observed that the DQN has the best QoS values when the environment is static (before 60000th episode). Multi-D3QN, double DQN, and dueling DQN get the second, third, and the worst QoS values, respectively. However, when the services are unavailable (between 60000th episode and 80000th episode). The Multi-D3QN has higher QoS values than other algorithms, this is because the reward strategy and the ε-greedy policy in the Multi-D3QN enable the

algorithm to find a better solution. The dueling DQN gets the worst QoS values during the whole process, which indicates that the dueling DQN has an overestimation problem. While, the double DQN obtains the second QoS values, due to the double estimator can overcome the overestimation problem. The QoS values of DQN reduces the most, which shows that the DQN is very unstable when the environment change. After a period of time, the Multi-D3QN returns to a stable state, while the other three algorithms do not have this change. The reason is that Multi-D3QN triggers the heuristic strategy, which shielding the unavailable services in the service chains. Furthermore, it can be observed from Fig. 6 that the range of fluctuations for Multi-D3QN is also smaller than that for double DQN, DQN and dueling DQN, which proves that our method is robust.

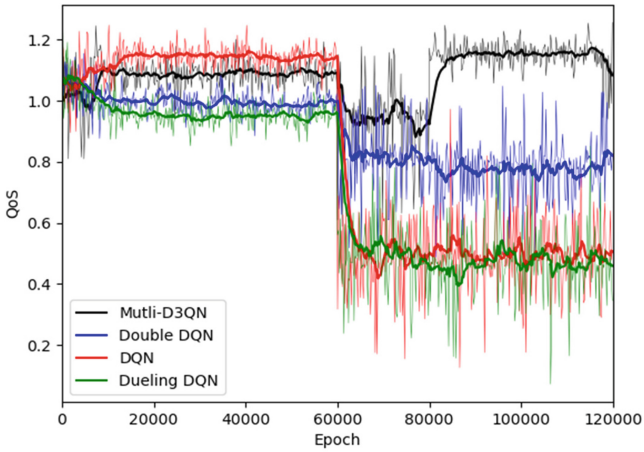

Fig. 6. Adaptability for Multi-D3QN, double DQN, DQN and dueling DQN

6 Conclusions

In this paper, we proposed the Multi-D3QN method by combining the basic DQN algorithm, the dueling architecture, the double estimator and the prioritized replay mechanism. Most importantly, some strategies, i.e. a heuristic strategy, instant reward and the ε-greedy policy, are added to our method to improve the performance of the algorithm. Specifically, the heuristic strategy can shield some unavailable services, and overcome the shortcomings of the algorithm that cannot be restored to a stable state due to the improper design of hyper-parameters and rewards. Instant reward allows DRL to effectively adjust parameters according to the real reward value, and the ε-greedy policy allows the algorithm to have a chance to jump out of the local optimum. The experiment shows that our proposed method not only has the advantage of strong adaptability in the dynamic environment, but also can find a better solution than other deep reinforcement learning such as DQN, dueling DQN and double DQN.

Acknowledgments. This work was supported in part by the National Key Research and Development Project under grant 2019YFB1706101, the Natural Science Foundation of Chongqing, China (No. cstc2020jcyj-msxmX0900).

References

1. Mourad, M.H., Nassehi, A., Schaefer, D., Newman, S.T.: Assessment of interoperability in cloud manufacturing. Robotics and Computer-Integrated Manufacturing 61, (2020)
2. Bouzary, H., Chen, F.F.: A classification-based approach for integrated service matching and composition in cloud manufacturing. Robotics and Computer-Integrated Manufacturing 66, (2020)
3. Zhang, L., et al.: Cloud manufacturing: a new manufacturing paradigm. Enterprise Information Systems **8**, 167–187 (2014)
4. Yang, Y., Yang, B., Wang, S., Jin, T., Li, S.: An enhanced multi-objective grey wolf optimizer for service composition in cloud manufacturing. Applied Soft Computing 87, (2020)
5. Yang, Y., Yang, B., Wang, S., Liu, W., Jin, T.: An Improved Grey Wolf Optimizer Algorithm for Energy-Aware Service Composition in Cloud Manufacturing. The International Journal of Advanced Manufacturing Technology **105**(7–8), 3079–3091 (2019). https://doi.org/10.1007/s00170-019-04449-9
6. Akbaripour, H., Houshmand, M., van Woensel, T., Mutlu, N.: Cloud manufacturing service selection optimization and scheduling with transportation considerations: mixed-integer programming models. The International Journal of Advanced Manufacturing Technology **95** (1–4), 43–70 (2017). https://doi.org/10.1007/s00170-017-1167-3
7. Liu, Y., Wang, L., Wang, X.V., Xu, X., Zhang, L.: Scheduling in cloud manufacturing: state-of-the-art and research challenges. Int. J. Prod. Res. **57**, 4854–4879 (2019)
8. Lartigau, J., Xu, X., Nie, L., Zhan, D.: Cloud manufacturing service composition based on QoS with geo-perspective transportation using an improved Artificial Bee Colony optimisation algorithm. Int. J. Prod. Res. **53**, 4380–4404 (2015)
9. Que, Y., Zhong, W., Chen, H., Chen, X., Ji, X.: Improved adaptive immune genetic algorithm for optimal QoS-aware service composition selection in cloud manufacturing. Int. J. Adv. Manuf. Technol. **96**(9–12), 4455–4465 (2018). https://doi.org/10.1007/s00170-018-1925-x
10. Wang, H., et al.: Adaptive and large-scale service composition based on deep reinforcement learning. Knowl.-Based Syst. **180**, 75–90 (2019)
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. **67**, 101991 (2021)
12. Quan, L., Wang, Z.-L., Liu, X.: A real-time subtask-assistance strategy for adaptive services composition. IEICE Trans. Inf. Syst. **E101D**, 1361–1369 (2018)
13. Zhou, J., Yao, X.: A hybrid artificial bee colony algorithm for optimal selection of QoS-based cloud manufacturing service composition. Int. J. Adv. Manuf. Technol. **88**(9–12), 3371–3387 (2016). https://doi.org/10.1007/s00170-016-9034-1
14. Fazeli, M.M., Farjami, Y., Nickray, M.: An ensemble optimisation approach to service composition in cloud manufacturing. Int. J. Comput. Integr. Manuf. **32**, 83–91 (2018)
15. Yu, L., Zhou, J., Wei, F., Gao, Y., Yang, B., Zhu, H.: Web Service Composition Based on Reinforcement Learning (2015)

16. Wang, H., Chen, X., Wu, Q., Yu, Q., Zheng, Z., Bouguettaya, A.: Integrating on-policy reinforcement learning with multi-agent techniques for adaptive service composition. In: Franch, X., Ghose, A.K., Lewis, G.A., Bhiri, S. (eds.) ICSOC 2014. LNCS, vol. 8831, pp. 154–168. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45391-9_11
17. Wang, H., Gu, M., Yu, Q., Fei, H., Li, J., Tao, Y.: Large-scale and adaptive service composition using deep reinforcement learning. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds.) ICSOC 2017. LNCS, vol. 10601, pp. 383–391. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69035-3_27
18. Yuan, M., Zhou, Z., Cai, X., Sun, C., Gu, W.: Service composition model and method in cloud manufacturing. Robot. Comput. Integr. Manuf. **61**, 101840 (2020)
19. Liu, Z.Z., Song, C., Chu, D.H., Hou, Z.W., Peng, W.P.: An approach for multipath cloud manufacturing services dynamic composition. Int. J. Intell. Syst. **32**, 371–393 (2017)
20. Zhou, J., Yao, X.: Multi-population parallel self-adaptive differential artificial bee colony algorithm with application in large-scale service composition for cloud manufacturing. Appl. Soft Comput. **56**, 379–397 (2017)
21. Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., Freitas, N.: Dueling network architectures for deep reinforcement learning. In: Maria Florina, B., Kilian, Q.W. (eds.) Proceedings of The 33rd International Conference on Machine Learning, vol. 48, pp. 1995–2003. PMLR, Proceedings of Machine Learning Research (2016)
22. van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double Q-learning. In: AAAI (2016)
23. Schaul, T., Quan, J., Antonoglou, I.: Prioritized experience replay, arXiv preprint arXiv: 1511.05952 (2015)