A DQN-based Joint Spectrum and Computing Resource Allocation Algorithm for Multi-Service MEC Networks

Feifan Zhou[1], Jun Zheng[1,2,3], Luyinru Yang[1], and Feng Yan[1,2]
[1]National Mobile Communications Research Laboratory
School of Information Science and Engineering
Frontiers Science Center for Mobile Information Communication and Security
Southeast University, Nanjing, 210096, China
[2]Zhejiang Lab, Hangzhou 31121, China
[3]Purple Mountain Laboratories, Nanjing 211111, China
Email: {ffzhou, lyryang, junzheng, feng.yan}@seu.edu.cn

ICC 2022 - IEEE International Conference on Communications | 978-1-5386-8347-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICC45855.2022.9838558

***Abstract*—This paper considers the network resource allocation problem in multi-service MEC networks, and in particular considers joint spectrum and computing resource allocation in an edge network service system. The resource allocation problem under consideration is formulated as an optimization problem, which takes into account users' minimum service requirements in terms of the service delay, transmission rate, and computation rate, with an objective to minimize the average service delay of a user's service request. To solve the problem, we propose a deep-Q-network (DQN) based joint spectrum and computing resource allocation algorithm, which attempts to optimize the resource allocation for each service request by learning a more efficient allocation scheme so as to better adapt to dynamic traffic arrivals, diverse service requirements, and complex environmental conditions. Simulation results show that the proposed resource allocation algorithm can efficiently reduce the average service delay compared with two benchmark algorithms.**

Keywords—DQN; mobile edge computing, resource allocation, reinforcement learning

I. INTRODUCTION

With the development and popularity of intelligent user devices, a variety of mobile applications have been emerging continually. Many applications need the support of network services and different applications have diverse service requirements, which presents a big challenge for the transmission and computing capabilities of mobile networks. Mobile edge computing (MEC) moves the computing, caching, and control capabilities of a network from a centralized cloud platform to the edge side of the network, which can provide end users with low delay, high bandwidth and localized network services[1-2]. In an MEC-based network, end users may have diverse service requirements in terms of transmission and computing. Due to the limitation of network resources, a mobile network needs to efficiently manage its resource allocation in order to meet users' service requirements[3-4]. For this purpose, a network may perform joint allocation of different network resources to enhance the efficiency of resource allocation[5]. On the other hand, a network may introduce advanced artificial intelligence (AI) technologies to enhance the flexibility of resource allocation in order to make resource allocation more adaptable to a complex network environment and thus efficient. It is with this consideration that motivated us to conduct this work.

This work is being supported by Open Research Projects of Zhejiang Lab (No. 2021LC0AB0).

In this paper, we consider the network resource allocation problem in multi-service MEC networks, and in particular consider joint spectrum and computing resource allocation in an edge network service system. We formulate the resource allocation problem as an optimization problem, which takes into account users' minimum service requirements in terms of the service delay, transmission rate, and computation rate, with an objective to minimize the average service delay of a user's service request. To solve the problem, we propose a deep-Q-network (DQN) based algorithm for joint allocation of the spectrum resources and computing resources in the system. The proposed algorithm attempts to optimize the resource allocation for each service request by learning a more efficient allocation scheme so as to better adapt to dynamic traffic arrivals, diverse service requirements, and complex environmental conditions[6]. Simulation experiments were conducted to evaluate the performance of the proposed algorithm and simulation results show that the proposed DQN-based resource allocation algorithm outperforms two benchmark algorithms, a random algorithm and a greedy algorithm, in terms of the average service delay.

The rest of this paper is organized as follows. Section 2 and Section 3 describes the system model and formulates the resource allocation problem considered in this paper, respectively. Section 4 presents the proposed DQN-based resource allocation algorithm. Section 5 shows simulation results to evaluate the performance of the proposed algorithm. Section 6 concludes this paper.

II. SYSTEM MODEL

A. Overview

Consider an edge network service system with a single edge node and a number of mobile users, as shown in Fig. 1. The edge node is composed of a base station (BS) and an associated server[7]. Users request for computing services from the edge server, and the edge server provides computation offloading services for the users. Users' service requests arrive in the system following a Poisson distribution with mean arrival rate λ .

Assume that the spectrum resources in the system are denoted by a set of N_B channels, $\mathbf{B} = \{B_1, B_2, \cdots, B_i, \cdots B_{N_B}\}$, where $B_i (i = 1,2,...,N_B)$ denotes the i-th channel, and its bandwidth is denoted by W_{B_i} . The bandwidth of each channel is equal, i.e., $W_{B_i} = W_B, i = 1,2,...,$. The computing resources

of the server in the system are represented by a set of N_C virtual machines, $\mathbf{C}=\{C_1,C_2,\cdots,C_j,\cdots C_{N_C}\}$, where C_j ($j=1,2,\cdots,N_C$) denotes the *j*-th virtual machine, and its CPU frequency is represented by F_{C_i}. The CPU frequency of each virtual machine is also equal, i.e., $F_{C_j}=F_C, j=1,2,...,N_C$.

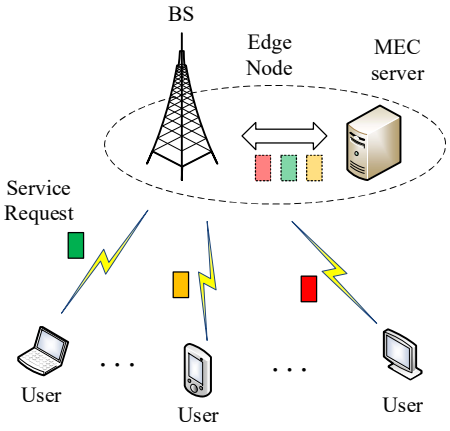

Fig. 1 System model

B. User Side

Each user randomly generates service requests. Assume that there are *M* types of service requests, which are denoted by $\mathbf{X}=\{\chi_1,\chi_2,...,\chi_M\}$. Each service type $\chi_i\in\mathbf{X}$ has different service requirements, which are denoted by a triple $(\tau^i_{\max},R^i_{\min},E^i_{\min})$. The elements in the triple, i.e., $\tau^i_{\max}$, $R^i_{\min}$, and $E^i_{\min}$, respectively, represent the minimum service requirements of the service type in terms of the service delay, transmission rate, and computation rate. Different service requirements have different demands for the spectrum resources and computing resources allocated by the system. For a type-*i* service request, it is considered that the allocated spectrum resources and computing resources satisfy its requirements only if its achieved performance (τ^i,R^i,E^i) satisfies the following conditions at the same time after resource allocation:

$$\tau^i\le\tau^i_{\max},R^i\ge R^i_{\min},E^i\ge E^i_{\min}$$

Otherwise, the requirements are considered as not satisfied. Moreover, the data size of a type-i service request is denoted by D_i and is assumed to be uniformly distributed in $[d^i_{\min},d^i_{\max}]$.

C. Edge Side

The BS performs spectrum resource allocation for each service request and the server performs computing resource allocation for each service request that arrives at the server, both in a first-come-and-first-served (FCFS) manner. Meanwhile, the server maintains a request queue for all service requests that have arrived at the server. In each allocation, the BS or server first decides the number of channels or virtual machines that need to be allocated for a service request according to the corresponding service requirements of the request, and then perform allocation following a resource allocation policy.

Let $n_B(t)$ and $n_C(t)$ denote the number of idle channels and the number of idle virtual machines available in the system at time t, respectively, $u^i_k(t)$ denote a type-i service request generated by user *k* at time *t*, $m_k(t)$ and $n_k(t)$ denote the number of channels that the BS decides to allocate for $u^i_k(t)$ and the number of virtual machines that the server decides to allocate for $u^i_k(t)$, respectively. Also, let $L(t)$ denote the queue length at time t and $L_{\max}$ denote the maximum queue length. Thus, the resource allocation policy used in the system is described as follows:

1) For a service request $u^i_k(t)$, if $n_B(t)>=m_k(t)$, the BS will allocate $m_k(t)$ channels for the request; otherwise, the request will be rejected.

2) For a service request $u^i_k(t)$, if $n_C(t)>n_k(t)$, the server will allocate $n_k(t)$ virtual machines for the request; if $n_C(t)<n_k(t)$, the request has to wait for resource allocation. In this case: if $L(t)<L_{\max}$, the request will be put into the queue and wait; if $L(t)>=L_{\max}$, the request will be rejected.

III. Problem Formulation

In this paper, we consider the joint spectrum resource and computing resource allocation problem in the edge network system described in the previous section. Given the total amount of spectrum resources and computing resources in the system, the resource allocation problem under consideration is to jointly allocate the spectrum resources and computing resources for each user request such that the average service delay of all users' service requests in the system is minimized while the service requirements of each service request are satisfied, τ_{avg}, namely:

$$\begin{aligned}&\text{Objective:} && \min\{\tau_{avg}\} && (1)\\ &\text{s.t.} && \mathrm{C}_1:\ \tau^i\le\tau^i_{\max},\forall\chi_i\in\mathbf{X}\\ &&& \mathrm{C}_2:\ R^i\ge R^i_{\min},\forall\chi_i\in\mathbf{X}\\ &&& \mathrm{C}_3:\ E^i\ge E^i_{\min},\forall\chi_i\in\mathbf{X}\end{aligned}$$

where constraint C_1, C_2, and C_3 indicate that, for any service type χ_i in $\mathbf{X}$, the service delay τ^i cannot exceed the threshold $\tau^i_{\max}$, and the transmission rate R^i and computation rate E^i cannot be smaller than the corresponding thresholds $R^i_{\min}$ and $E^i_{\min}$, respectively.

In the formulated problem, τ_{avg} represents the average service delay for a service request, which is defined as the average delay that a service request experiences in the system under the given resource allocation policy, namely:

$$\tau_{avg}=\frac{1}{N}\sum_{k=1}^{N}\tau_k \qquad (2)$$

where N denotes the total number of users' service requests that arrive in the system during a period of time. For each service request $u^i_k(t)$, its service delay τ_k consists of three

components: transmission delay τ_k^t, computation delay τ_k^c, and queuing delay τ_k^q, i.e.,

$$\tau_k = \tau_k^t + \tau_k^c + \tau_k^q . \quad (3)$$

The transmission delay can be expressed as

$$\tau_k^t = \frac{D_k^i}{R_k^i}, \quad (4)$$

where D_k^i denotes the data size of $u_k^i(t)$, and R_k^i denotes the transmission rate for $u_k^i(t)$. According to Shannon's formula, we have

$$R_k^i = B_k^i \log(1+\frac{P_k^i}{N_0 B_k^i}), \quad (5)$$

where P_k^i denotes the signal power for transmitting $u_k^i(t)$, N_0 denotes the noise power spectral density of the channel(s) occupied by $u_k^i(t)$, and B_k^i denotes the bandwidth of the channel(s) occupied by $u_k^i(t)$.

The computation delay can be expressed as

$$\tau_k^c = \frac{D_k^i}{E_k^i} \quad (6)$$

where E_k^i denotes the computing rate for $u_k^i(t)$, which is given by

$$E_k = \frac{C_k^i}{\mu_0} \quad (7)$$

where C_k^i denotes the CPU frequency of the virtual machine(s) to be allocated to $u_k^i(t)$, and μ_0 represents the number of cycles that the CPU runs for processing 1bit data.

Let $\mathbf{T}_w\{w_1, w_2, \cdots, w_{N_C}\}$ denote a set of waiting time for each virtual machine to become idle at the arrival of $u_k^i(t)$, where if an element is 0, it means that the corresponding virtual machine is idle; otherwise, the virtual machine is occupied. After removing all elements with 0 value from $\mathbf{T}_w$, a non-zero set, i.e., $\mathbf{T}_w^*$, can be obtained. Also, let l denote the number of idle virtual machines that the current service request needs to wait for, which is given by

$$l = \frac{C_k^i - C_k^*}{F_C}, \quad (8)$$

where C_k^i denotes the CPU frequency to be allocated to $u_k^i(t)$, C_k^* denotes the remaining CPU frequency in the system, and F_C denotes the CPU frequency of a single virtual machine. The queuing delay can be expressed as

$$\tau_k^q = \max(w_1', w_2', ..., w_l'), \quad (9)$$

where $w_1', w_2', ..., w_l' \in \mathbf{T}_w^*$ are the smallest l waiting time values in $\mathbf{T}_w^*$, which correspond to of l virtual machines.

IV. DQN-Based Resource Allocation Algorithm

In this section, we propose a DQN-based resource allocation algorithm to solve the resource allocation problem formulated in Section II.

A. Overview

The proposed resource allocation algorithm is intended for solving the joint spectrum resource and computing resource allocation problem formulated in the previous section. The algorithm is based on a Deep Q Network (DQN) method, which is a combination of Q learning and DNN (Deep Neural Network). This method incorporates the decision-making ability of reinforcement learning and the perception ability of deep learning[8]. Using the DQN method can help to represent a large number of action value functions in the current network state. For an edge network system with high input-data dimensions, DQN solves the massive Q-value calculation and processing problems that cannot be solved by traditional Q learning.

The establishment of DQN architecture is divided into two steps: 1) establish the basic framework of Q learning; 2) introduce a neural network structure and a memory replay mechanism.

B. Basic Framework

A Q-learning process can be divided into five parts: initializing a Q-value table, selecting actions, executing actions, calculating rewards, and updating the Q-value table, as shown in Fig. 2. Every time the Q-value table is read, an agent obtains information from an external environment; selected actions and execution actions represent the agent's exploration and decision-making of the environment; a calculated reward reflects the environment's response to the agent's decision-making. During this process, the agent gets deeper understanding of the environment, and the decisions made by the agent gradually approach a globally optimal solution, until the process converges and the globally optimal solution is output.

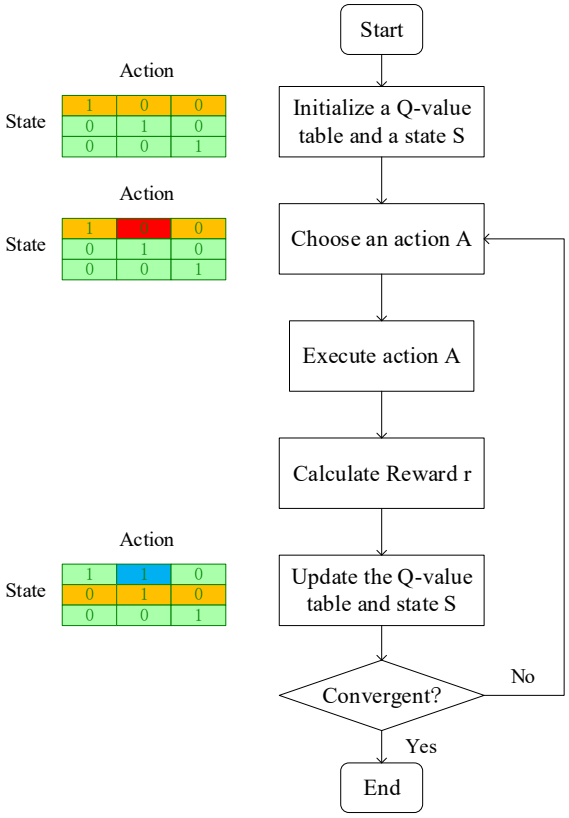

Fig. 2 The basic process of Q learning

A Q-learning process is usually modeled as a Markov decision process, represented by a five-element variable group $\{S, A, T, R, \gamma\}$, where S represents a set of all possible states of the current environment, A represents a set of actions currently selectable by an agent, T represents a set of state transition probabilities, R represents a reward function, and γ represents an attenuation factor. Next we first define the five elements of the Q learning process.

- State (S)

A state at time t is defined as a set of variables, $S(t) = \{\chi_i, D_i, n_B(t), n_C(t)\}$, where each element was defined earlier.

- Action (A)

An action that the agent can take is defined as the allocation of a resource tuple $(m_k(t), n_k(t))$ corresponding to the current state $S(t)$.

- Transition probability (T)

In the Q learning process, each action taken by the agent follows the principle of maximizing the Q value of an action. Under this principle, each state transition is determined in accordance with the Q-value table of all actions in each state, and the transition probability is 1; correspondingly, the probabilities of the transitions to other states are 0. This principle will result in insufficient exploration of the action space, which will often make the output fall into a locally optimal solution. To prevent this from happening, the agent can use an epsilon-greedy strategy, with which the agent has the probability of $1-\varepsilon$ to take the action with the largest Q value from all actions in a state and transit to the corresponding state, and the probability of ε to take other actions so as to achieve a dynamic balance between action exploration and action exploitation in the Q learning process. Assuming that there are a total of r actions in a state, the probability that the agent takes another action and transits to the corresponding state is given by $\varepsilon/(r-1)$.

- Reward function (r)

In the Q-learning process, a reward function is defined to update the Q-value table of all actions in each state. For the spectrum resource allocation, the reward function is defined as

$$r_k^s = \begin{cases} 1/\tau_k^t, & \tau_k^i \le \tau_{\max}^i, R_k^i \ge R_{\min}^i, B_k^i \le B_k^* \\ -D_k^i, & B_k^i > B_k^* \\ 0, & otherwise \end{cases}, \quad (10)$$

where B_k^* denotes the remaining available bandwidth in the system. Similarly, the reward function for the computing resource allocation is defined as

$$r_k^c = \begin{cases} 1/\tau_k^i, & \tau_k^i \le \tau_{\max}^i, E_k^i \ge E_{\min}^i, C_k^i \le C_k^* \\ -D_k^i, & C_k^i > C_k^* \\ 0, & otherwise \end{cases}. \quad (11)$$

- Attenuation factor (γ)

The attenuation factor is used to reduce the Q-value oscillation problem caused by an improper learning rate in the process of searching for an optimal solution. In most engineering practices, the attenuation factor is an empirical value of 0.9.

C. Neural network and Memory Replay

According to the five elements defined in the Q-learning process, the input data dimension of a state and all actions in the state are high, the data samples of states and actions are very sparse, and the Q values of actions in most states are never Updated during the whole learning process. For these reasons, it is not suitable to use a Q-value table for data storage. It is necessary to introduce a neural network structure on the basis of Q learning, and use the network to fit the Q values of the actions in each state. Meanwhile, it is also necessary to introduce a memory replay mechanism to remove the correlation between data samples so that the samples can be repeatedly learned, thereby improving the convergence speed of the neural network.

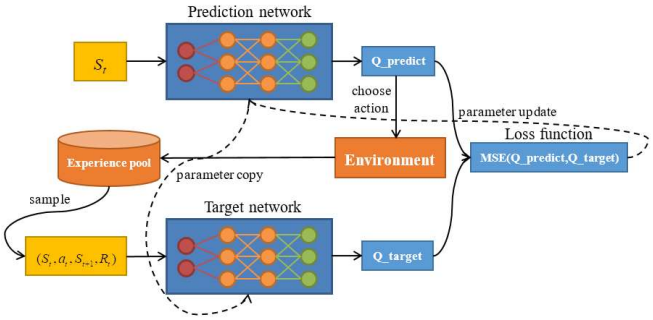

Fig. 3 Basic structure of a DQN

Fig. 3 shows the logical structure of a DQN, which is composed of upper and lower neural networks. The upper neural network is used as a prediction network to generate an output, i.e., Q_predict; the lower neural network is used as a target network to guide the update of the prediction network. The operation process of the DQN is described as follows:

1) Input a state to the prediction network, and output the predicted Q values of all actions in the state, i.e., Q_predict.

2) According to the output Q values (Q_predict), the agent performs the following steps:
   a) select an action from a set of possible actions;
   b) interact with the environment, i.e., obtain the reward of the action and the next state;
   c) combine the obtained action reward and the next state with the input state and the selected action to form a tuple (S_t, a_t, S_{t+1}, r_t);
   d) Store the tuple (S_t, a_t, S_{t+1}, r_t) in a fixed-length experience pool.

3) The target network samples a tuple from the experience pool and outputs a set of target values, i.e., Q_target.

4) Calculate the loss function of the DQN and update the Q-values in the prediction network.

The loss function of the DQN is defined as

$$Loss = [r_t + \gamma \max_{a^*} Q(S_{t+1}, a^*) - Q(S_t, a_t)]^2, \quad (12)$$

where $Q(S_t, a_t)$ denotes the Q value of state S_t and action a_t, $Q(S_{t+1}, a^*)$ denotes the Q value of the next state S_{t+1} and action a^*, and γ is an attenuation factor. Moreover, a loss threshold $Loss_{\max}$ is introduced to judge whether the DQN converges.

The Q-value update process in the prediction network can be described as

$$Q(S_t,a_t) \leftarrow Q(S_t,a_t) + \alpha[r_t + \gamma \max_{a^*} Q(S_{t+1},a^*) - Q(S_t,a_t)]^2 ,$$

where α denotes a learning rate.

After the update of the prediction network, if the maximum value of the loss function does not exceed the threshold $Loss_{\max}$, the DQN is considered convergent. In this case, for each input state S_t, under the premise of satisfying all service requirements, the action corresponding to the maximum Q value is selected as the optimal resource allocation decision for S_t.

V. Simulation Results and Performance Evaluation

In this section, we evaluate the performance of the proposed DQN-based resource allocation algorithm through simulation results. The simulation experiments were conducted using Python. In the experiments, we consider three service types and for simplicity specifically consider type-1 ($\tau^1_{\max}$, 0, 0), type-2 (∞, $R^2_{\min}$, 0), and type-3 (∞,∞, $E^3_{\min}$), where type-1 only has a delay requirement, type-2 only has a transmission rate requirement, and type-3 only a computation rate requirement. All service types and request data sizes are generated by a Python emulator. One simulation experiment uses a total of 20 groups of data, and each group of data is composed of 100 service requests in sequence. Each result is obtained by averaging the average delay for the 20 groups of data.

For evaluation, we compare the proposed DQN-based algorithm with a couple of benchmark algorithms, a random algorithm and a greedy algorithm. A random algorithm randomly selects a bandwidth level corresponding to the number of channels to be allocated from a set of available bandwidth levels and a frequency level corresponding to the number of virtual machines to be allocated from a set of available frequency levels in each resource allocation. A greedy algorithm always selects the largest bandwidth level and the largest frequency level in each resource allocation. Moreover, we use the average service delay of a service request as the main performance metric. The main parameters used in the simulation experiments are given in table I.

TABLE I. Main Simulation Parameters

Parameter	*Value*
Max service delay ($\tau^1_{\max}$)	100ms
Min transmission rate ($R^2_{\min}$)	8.0kbps
Min computation rate ($E^3_{\min}$)	80kbps
User data size $[d^i_{\min}, d^i_{\max}]$	[100KB,1MB]
BS bandwidth (B_{total})	100MHz
CPU frequency (F_{total})	1.0GHz
Noise power spectral density(N_0)	-116dBm/Hz
Number of cycles per bit(μ_0)	0.125
A set of bandwidth levels	{0MHz, 2MHz, 5MHz, 10MHz, 20MHz, 50MHz}
A set of CPU frequency levels	{0MHz, 20MHz, 50MHz, 100MHz, 200MHz,500MHz}
Request arrival rate (λ)	0.5
Learning rate (α)	0.01
Attenuation factor (γ)	0.9

Fig. 4 shows the DQN loss with the proposed DQN-based algorithm under different update iterations. It is seen that during the first 50 rounds of update iterations, the DQN loss drops dramatically and then fluctuates between 50 rounds to 400 rounds, and finally converges after about 400 rounds of iterations. This shows the good convergence of the algorithm.

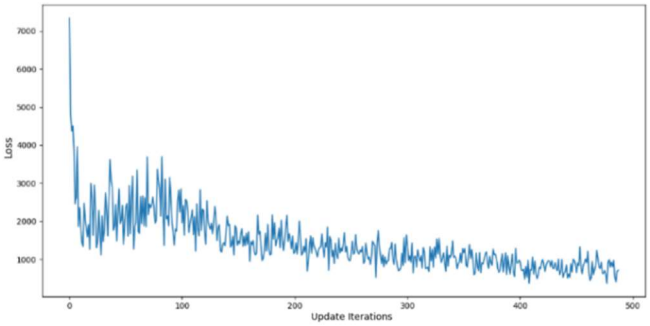

Fig. 4 Loss vs iterations

Fig.5 shows the average service delay with the three algorithms, respectively, under different request arrival rates (λ). It is seen that as the value of λ increases, the average service delay with all the three algorithms increases as well. On the other hand, the delay with the proposed DQN-based algorithm is always smaller than that with the greedy algorithm or the random algorithm. In particular, when $\lambda \le 0.125$, the average delay with the greedy algorithm and that with the DQN-based algorithm are very close. When $\lambda > 0.667$, the average delay with the DQN-based algorithm starts to increase more quickly and approach the delay with the random algorithm. This is because when $\lambda \le 0.125$, the number of idle channels is sufficient. As a result, both the DQN-based algorithm and the greedy algorithm can allocate the maximum number of idle channels for each service request. When $\lambda > 0.667$, the idle channels become insufficient to accommodate all service requests. Some service requests would not be able to obtain idle channels, thus causing the average delay with the DQN-based algorithm to approach that with the random algorithm.

Fig.6 shows the average service delay with the three algorithms, respectively, under different request data sizes. It is seen that as the request data size increases, the average service delay with all the three algorithms increases as well. This indicates that the request data size has a similar effect on the average service delay as λ .

Fig.7 shows the average service delay with the three algorithms, respectively, under different total BS bandwidth (B_{total}). It is seen that as the value of B_{total} increases, the average service delay with all the three algorithms decreases. This is because the increase in the total spectrum resource makes each service request have a larger probability of obtaining more spectrum resource. On the other hand, the delay with the DQN-based algorithm remains smaller than that with the other algorithms until B_{total} reaches 250MHz. When $B_{total} \ge 250MHz$, the average service delay with the greedy algorithm is almost the same as that with the DQN-based algorithm. This is because when the total spectrum resource increases to a certain extent, all service requests can be supported to obtain the maximum amount of spectrum resource, which makes the DQN-based algorithm become a greedy algorithm.

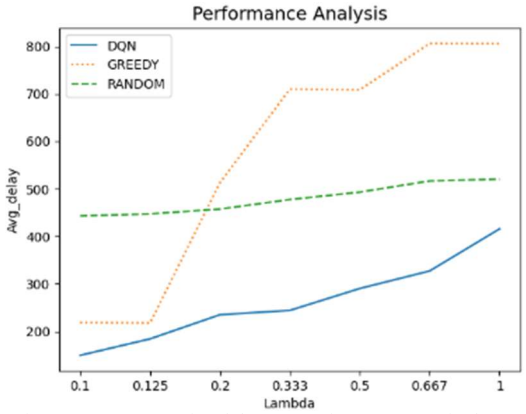

Fig. 5 Average service delay vs service request arrival rate.

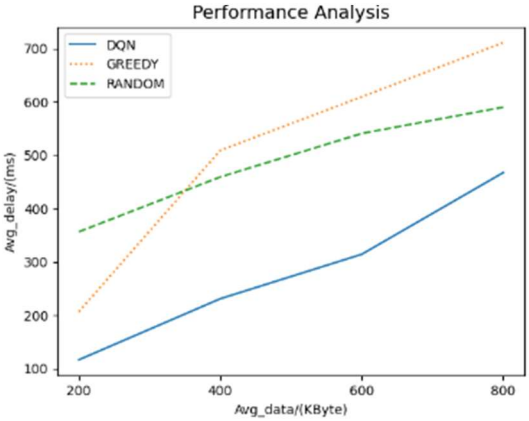

Fig. 6 Average service delay vs average request data size.

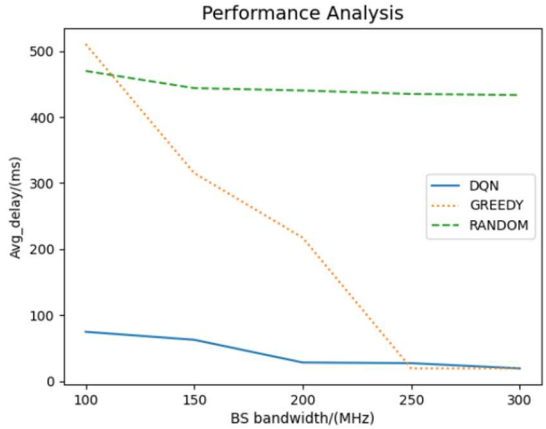

Fig. 7 Average service delay vs BS bandwidth.

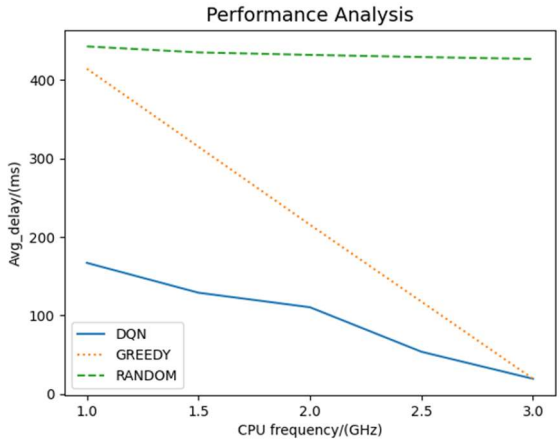

Fig. 8 Average service delay vs CPU frequency.

Fig.8 shows the average service delay with the three algorithms, respectively, under different total CPU frequency (F_{total}). It is seen that as the value of F_{total} increases, the average service delay with all the three algorithms decreases, which shows that F_{total} has an similar effect on the average service delay as B_{total} . On the other hand, the delay with the DQN algorithm remains smaller than that with the other algorithms until F_{total} reaches 3.0GHz. The reason is that when the total CPU frequency increases to a certain extent, all service requests can obtain the maximum amount of CPU frequency, which is almost the same as that of Fig.7.

VI. Conclusion

In this paper, we considered the joint spectrum and computing resource allocation problem in an edge network service system. The considered problem was formulated as an optimization problem, which takes into account users' minimum service requirements in terms of the service delay, transmission rate, and computation rate, with an objective to minimize the average service delay of a user's service request. A DQN-based algorithm was proposed to solve the formulated problem. The proposed algorithm attempts to optimize the resource allocation for each service request by learning a more efficient allocation scheme so as to better adapt to dynamic traffic arrivals, diverse service requirements, and complex environmental conditions. The simulation results show that the proposed DQN-based resource allocation algorithm outperforms a random algorithm and a greedy algorithm in terms of the average service delay.

References

[1] C. Liu, M. Bennis, M. Debbah and H. V. Poor, "Dynamic Task Offloading and Resource Allocation for Ultra-Reliable Low-Latency Edge Computing," *IEEE Transactions on Communications*, vol. 67, no. 6, pp. 4132-4150, Jun. 2019.

[2] A. Abouaomar, S. Cherkaoui, Z. Mlika and A. Kobbane, "Resource Provisioning in Edge Computing for Latency-Sensitive Applications," *IEEE Internet of Things Journal*, vol. 8, no. 14, pp. 11088-11099, 15 Jul.15, 2021.

[3] H. Chien, Y. Lin, C. Lai and C. Wang, "End-to-End Slicing as a Service with Computing and Communication Resource Allocation for Multi-Tenant 5G Systems," *IEEE Wireless Communications*, vol. 26, no. 5, pp. 104-112, Oct. 2019

[4] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji and M. Bennis, "Optimized Computation Offloading Performance in Virtual Edge Computing Systems Via Deep Reinforcement Learning," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4005-4018, Jun. 2019.

[5] H. Liao et al., "Learning-Based Context-Aware Resource Allocation for Edge-Computing-Empowered Industrial IoT," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4260-4277, May 2020.

[6] Y. -C. Wu, T. Q. Dinh, Y. Fu, C. Lin and T. Q. S. Quek, "A Hybrid DQN and Optimization Approach for Strategy and Resource Allocation in MEC Networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 7, pp. 4282-4295, Jul. 2021.

[7] J. Feng, Q. Pei, F. R. Yu, X. Chu, J. Du and L. Zhu, "Dynamic Network Slicing and Resource Allocation in Mobile Edge Computing Systems," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 7, pp. 7863-7878, Jul. 2020.

[8] S. Nath and J. Wu, "Deep Reinforcement Learning for Dynamic Computation Offloading and Resource Allocation in Cache-Assisted Mobile Edge Computing Systems," *Intelligent and Converged Networks*, vol. 1, no. 2, pp. 181-198, Sept. 2020, Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68-73.