entropy

MDPI

Article

An Edge Server Placement Method Based on Reinforcement Learning

Fei Luo [1], Shuai Zheng [1], Weichao Ding [1,*], Joel Fuentes [2] and Yong Li [1,*]

[1] School of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China; luof@ecust.edu.cn (F.L.); jesse2017315@gmail.com (S.Z.)
[2] Department of Computer Science and Information Technologies, Universidad del Bío-Bío, Chillán 3780000, Chile; jfuentes@ubiobio.cl
* Correspondence: weich@ecust.edu.cn (W.D.); yongli@ecust.edu.cn (Y.L.)

Abstract: In mobile edge computing systems, the edge server placement problem is mainly tackled as a multi-objective optimization problem and solved with mixed integer programming, heuristic or meta-heuristic algorithms, etc. These methods, however, have profound defect implications such as poor scalability, local optimal solutions, and parameter tuning difficulties. To overcome these defects, we propose a novel edge server placement algorithm based on deep q-network and reinforcement learning, dubbed DQN-ESPA, which can achieve optimal placements without relying on previous placement experience. In DQN-ESPA, the edge server placement problem is modeled as a Markov decision process, which is formalized with the state space, action space and reward function, and it is subsequently solved using a reinforcement learning algorithm. Experimental results using real datasets from Shanghai Telecom show that DQN-ESPA outperforms state-of-the-art algorithms such as simulated annealing placement algorithm (SAPA), Top-K placement algorithm (TKPA), K-Means placement algorithm (KMPA), and random placement algorithm (RPA). In particular, with a comprehensive consideration of access delay and workload balance, DQN-ESPA achieves up to 13.40% and 15.54% better placement performance for 100 and 300 edge servers respectively.

Keywords: edge computing; markov decision process; reinforcement learning; access delay; workload balance

Citation: Luo, F.; Zheng, S.; Ding, W.; Fuentes, J.; Li Y. An Edge Server Placement Method Based on Reinforcement Learning. *Entropy* **2022**, *24*, 317. https://doi.org/10.3390/e24030317

Academic Editors: Jaroslaw Krzywanski, Karolina Grabowska, Marcin Sosnowski and Dorian Skrobek

Received: 29 December 2021
Accepted: 18 February 2022
Published: 23 February 2022

1. Introduction

Mobile cloud computing is the combination of cloud computing and mobile computing to bring rich computational resources to end mobile users, network operators, and cloud computing providers. It has also become one of the new hotspots of the mobile Internet services, where problems such as network overload/underload uplink and downlink arise between the cloud and the mobile devices [1]. Mobile edge computing (MEC) addresses these problems by localizing communication, storage and computing resources near the user side, and providing resources with lower network latency and reduced network congestion.

MEC and important parts of its related research have been focused on performance improvements for task offload [2–4], service migration [5–7], and quality-of-service (QoS) prediction [8]. Most of the published work consider the premise that the location of edge server has been previously determined, without considering edge server placement. However, improper edge server placement may cause some edge servers to be overloaded or underloaded [9–11], which leads to a QoS decrease. In the contrast, a reasonable edge server placement scheme can provide edge servers more balanced workloads, and can also make the overall delay lower, achieving better QoS for users [12,13].

Currently, the edge server placement problem (ESPP) is mainly approached by transforming it into a multi-objective optimization problem, which is further solved by mixed

integer programming, heuristic algorithms or meta-heuristic algorithms [11,12,14]. However, these methods have profound limitations and performance implications. For instance, mixed integer programming cannot solve large-scale problems in a limited time [13]. Heuristic algorithms depend on their specific problem models, and it is easy to fall into local optimal solutions. Meta-heuristic algorithms are not easily reusable due to a significant amount of problem-specific parameter tuning [15].

Different from the above methods, this paper proposes a novel edge server placement algorithm based on deep reinforcement learning, dubbed DQN-ESPA. First, the ESPP is modeled as a Markov decision process (MDP), where the goal is to balance edge server workloads and minimize the access delay between the mobile user and edge server. Then, reinforcement learning is applied to find the optimal solution. Specifically, an agent continuously learns online and improves its placement decisions through a trial-and-error learning scheme. The introduction of a deep neural network (DNN) helps to deal with the exponential growth of states and actions when exploring optimal control in high-dimensional spaces, known as the curse of dimensionality [16].

The main contributions of this paper can be summarized as follow:

- A new solution model is proposed for the ESPP based on MDP and reinforcement learning. In this model, location sequences of edge servers are modeled as states, decisions of move direction of edge servers are modeled as actions, and the negative access delay and standard deviation of workloads are modeled as rewards. By maximizing the cumulative long-term reward, the ESPP can be solved by the reinforcement learning algorithm.
- An edge server placement algorithm based on deep q-network (DQN), named DQN-ESPA, is proposed to solve the ESPP by combining a deep neural network and a Q-learning algorithm.

The remainder of this paper is organized as follows. Related work is presented in Section 2. The ESPP is modeled in Section 3, which abstracts the edge computing architecture into a network structure composed of users, base stations and edge servers. The details of DQN-ESPA algorithm are described in Section 4. In Section 5, experiments are carried out and the performance of the proposed algorithm is evaluated with the other state-of-date algorithms. Finally, conclusions are drawn and the directions of future work are presented in Section 6.

2. Related Work

To meet the requirements of high resource utilization and small network delay, the ESPA selects an appropriate geographical location for the edge server in MEC. For convenience and consistency, edge devices with rich computing and storage resources, such as the roadside units (RSUs), cloudlets and the MEC edge servers, are uniformly called as edge servers [17].

Classical approaches have been used for the ESPP, such as the K-Means placement algorithm (KMPA) [18], the Top-K placement algorithm (TKPA) and the Random placement algorithm (RPA) [12,19]. Taking the network coverage and deployment cost as optimization indexes, Liang et al. [20] studied the deployment of RSUs in a two-dimensional vehicle network using linear programming. Wang et al. [21] introduced the concept of centrality in social networks into the deployment of RSUs, and proposed a method of RSUs deployment based on centrality. First, the RSUs deployment problem was abstracted into a linear programming problem, and then the RSUs problem was transformed into a 0-1 knapsack problem. Compared to the random deployment method, the efficiency of RSUs deployment was significantly improved. At present, a large number of research activities on RSUs placement are mainly from the perspective of communications [20,22–24], while less attention is paid to the demand of computing resources for vehicle applications. Premsankar et al. [25] considered the placement of RSUs from two aspects of network coverage and computing resources, and solved the placement problem of RSUs using mixed integer programming. Zhang et al. [14] put forward an asynchronous particle swarm optimization algorithm to op-

timize the RSUs deployment from the perspective of positioning accuracy and deployment cost, hoping to use the least RSUs to obtain the best vehicle positioning accuracy.

Some approaches on the optimal configuration of cloudlets were also studied. Ma et al. studied in [9] the problem of cloudlet placement in large-scale wireless metropolitan area networks. By abstracting the problem of cloudlet placement into the problem of integer linear programming, and considering the limitation that integer linear programming cannot solve the large-scale problem, the algorithm of k-medoids was used to solve the problem of cloudlets' placement. Xu et al. [10] considered the placement of cloudlets with different computing resources, and proposed a fast and effective heuristic algorithm to minimize access delay. Authors in [11] considered the impact of user's dynamic request on cloudlet placement, adopting approximate and iterative algorithms to obtain the optimal placement scheme. Observing that the deployment cost was not considered in some studies, Fan et al. [26] established a linear programming model by considering both the deployment cost and the access delay.

In view of the strategies and configuration used in the problems listed above, subsequent research focused on workload balance [27] or access delay [28–30]. Zhao et al. [12] placed edge servers comprehensively considering both the workload balance and the access delay. By constructing the placement as a multi-objective optimization problem, they solved the placement of edge servers through mixed integer planning. Subsequently, Guo et al. [13] proposed an approximation algorithm based on K-means and mixed integer quadratic programming. From the perspective of minimizing the number of edge servers deployed and QoS requirements, Zeng et al. [31] utilized the simulated annealing algorithm and the greedy placement algorithm, named as SAPA.

Considering the ESPP in particular, the described work have either used mixed integer programming algorithms, heuristic algorithms, and/or meta-heuristic algorithms. However, different problems arise for various scenarios commonly found in MEC, i.e. mixed integer programming algorithms cannot solve large-scale problems in a limited time; heuristic algorithms rely on specific problem models and easily to fall into the local optimal solution; meta-heuristic algorithms require high levels of parameter-tuning.

Reinforcement learning is commonly used to solve sequence decision-making problems and find the optimal strategy by maximizing long-term rewards [32]. In edge computing, it has been used to solve task offloading and resource allocation problems [4,33–35]. To the best of our knowledge, there are no studies on edge server placement in MEC environments using reinforcement learning. Therefore, this paper proposes a model and algorithm to solve the ESPP using reinforcement learning, providing users with lower latency and more balanced workload among edge servers. Typical comparison algorithms are used to demonstrate the superiority of the proposed algorithm in sequential decision-making circumstances, such as SAPA, KPMA, TKPA and RPA. Especially, as a variant of top-k algorithms, the bubble-sort algorithm is implemented in the TKPA experiments for comparison.

3. System Model

In this section we introduce the MEC model, which simplifies the edge computing architecture into a network structure consisting of users, base stations and edge servers. Then, a detailed description of the ESPP is provided. Additionally, we formalize the workload balance model and access delay, which will be used to evaluate the performance of the proposed algorithm.

3.1. MEC Model

In order to reduce the delay and solve the user's limited processing capacity, MEC migrates the computing and storage capability from the remote data center to the network edge close to the user. In the mobile edge computing environment, the edge computing system model can be defined as an undirected graph $G = (V, E)$, which contains users, base stations and edge servers. As it is shown in Figure 1, bs_i represents a base station, es_i

represents an edge server and md_i represents a mobile device. The mobile device forwards the request to the edge server through the base station, and the edge server returns the processed result to the mobile device through the base station. The set of points in the undirected graph G is represented by V. $V = B \cup S$, where B is the set of base stations and S is the set of edge servers placed in candidate locations. E is the set of connected edges between the base station and the edge server in G. User requests are forwarded to the appropriate edge server nearby through the base station, and the edge server returns the processed results to the user.

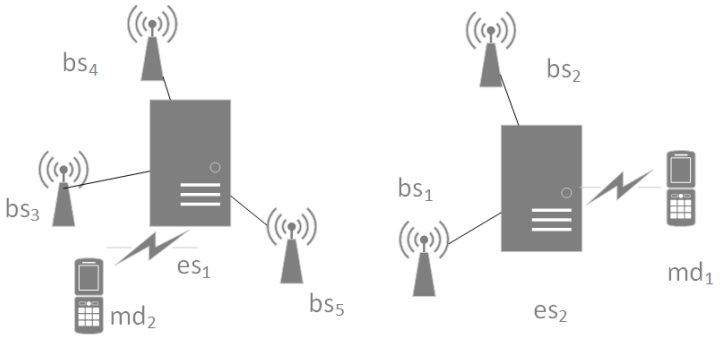

Figure 1. Edge computing system model.

3.2. Problem Description

In an edge computing system, all edge servers and base stations are represented by (S, B), where $S = \{s_1, s_2, \ldots, s_m\}$ represents a set of m edge servers. $B = \{b_1, b_2, \ldots, b_n\}$ represents a set of n base stations. The workload of each base station b_i is represented as w_i, and the distance between the base station b_i and the edge server s_j is represented as $d(b_i, s_j)$. In order to simplify the problem, five assumptions are given as follows.

1. Each edge server is homogeneous and has the same processing and storage capabilities.
2. Let the set of base stations covered by each edge server be B_i, the set of base stations covered by each edge server does not intersect, i.e., $B_i \cap B_j = \varnothing$. The union of the set of base stations covered by all edge servers is $B = \sum_{i=1}^{m} B_i$.
3. An edge server is placed next to a base station, and n base stations have n positions.
4. Only one edge server can be placed in each location.
5. Each edge server is only responsible for processing requests uploaded by base stations within the signal coverage, and each base station within the signal coverage of each edge server can only forward requests to the edge server.

m edge servers are placed in the location set L with m locations through the placement strategy, where L = $\{l_1, l_2, \ldots, l_m\}$. Therein, l_i represents the placement location of the edge server s_i, which minimizes the delay and workload balance of the entire system. The workload balance and access delay of the system are described below.

- *Workload balance*
  The edge server s_i is responsible for processing network service requests forwarded through all base stations in the base station set B_i. The workload of each base station is represented as w_j, and the workload of the edge server s_i is represented as W_i which is the sum of the workloads of all base stations in B_i, as shown in Formula (1).

$$W_i = \sum_{b \in B_i} w_j \tag{1}$$

  Then the standard deviation of all the edge servers' workloads is shown in Formula (2), where $\bar{W} = \frac{1}{m} \sum_{i=1}^{m} W_i$ is the average workload of m edge servers.

$$WSD = \sqrt{\frac{\sum_{i=1}^{m} (W_i - \bar{W})^2}{m}} \tag{2}$$

- *Access delay*
  Each base station can directly request the edge server to obtain network services through the link connection. Assuming that the positions of the base station and the edge server are denoted as l_b and l_s, respectively, the length of the connecting edge between the two can be obtained based on the longitude and latitude information of the base station, as shown in Formula (3). In this paper, we use the length of the edge to measure the data transmission delay between the base station and the edge server.

$$d(b_i, s_j) = \left\| l_{b_i} - l_{s_j} \right\|_2 \tag{3}$$

  Then the average access delay is represented as Formula (4).

$$MD = \frac{\sum_{i=1}^{m} \sum_{b_j \in B_i} d(b_j, s_i)}{n} \tag{4}$$

4. Algorithm Design

Reinforcement learning consists of an agent and an environment. The agent can change the current environmental state by executing an action, and the environment gives the agent a reward according to the action executed by the agent. The goal of reinforcement learning is to learn the optimal strategy by maximizing the cumulative long-term reward, which is usually established as an Markov decision process model. The MDP model is denoted as a state-action-reward-state transition sequence, specifically a 5-tuple (S, A, T, R, γ) described as follows:

- S is a limited state space.
- A is a limited action space.
- $T : S * A * S \rightarrow [0, 1]$, it is the state transition model. Specifically, $T(s, a, s')$ represents the probability distribution that the agent transitions to state s' after executing an action a in the state s.
- $R : S * A \rightarrow \mathbb{R}$, it represents the reward function. For example, $R(s, a)$ represents the reward value given by the environment after the agent executes the action a in the state s.
- $\gamma \in [0, 1]$ is a discount factor used to balance the importance of immediate and long-term rewards.

In the remainder of this section the MDP model is formalized based on the MEC environment, and the state space, action space and reward function are subsequently defined. Afterwards, the novel edge server placement algorithm, DQN-ESPA, is introduced.

4.1. MDP Model

4.1.1. State

Consider the scenario of placing m edge servers in n positions, and where only one edge server can be placed in each position. Each placement is a solution to the problem, and it is denoted as a different state. Suppose that the location of each edge server is (lat_i, lon_i), where lat_i is the latitude value of the location of the edge server s_i, and lon_i is the longitude value of the location of the edge server s_i. The placement sequence will be $((lat_1, lon_1), (lat_2, lon_2), \cdots, (lat_m, lon_m))$ after edge serves are placed. We define the state space S, where $S = ((lat_1, lon_1), (lat_2, lon_2), \cdots, (lat_m, lon_m))$. In particular, the placement sequence varies as the placement of the edge server changes and a new state is generated.

4.1.2. Action

Every time the placement positions of m edge servers are randomly selected from n positions to get the initial state it is assumed that only one edge server placement is changed at a time. One additional restriction is that each edge server can only move to a nearby location. As an example, consider the diagram shown in Figure 2. There are nine base stations $(b_1, b_2, \cdots, b_9)$, and the edge server s_i can be placed at any position of the nine

base stations. The initial position of the edge server s_i is at the position of the base station b_5, and the edge server s_i can move up, down, left, and right to reach the new placement position. Since the latitude and longitude of each base station are known, the neighbor base stations of each base station are also known. Hence, moving upward means moving in the direction of increasing latitude, moving downward means moving in the direction of decreasing latitude, moving left means moving in the direction of decreasing longitude, and moving right means moving in the direction of increasing longitude.

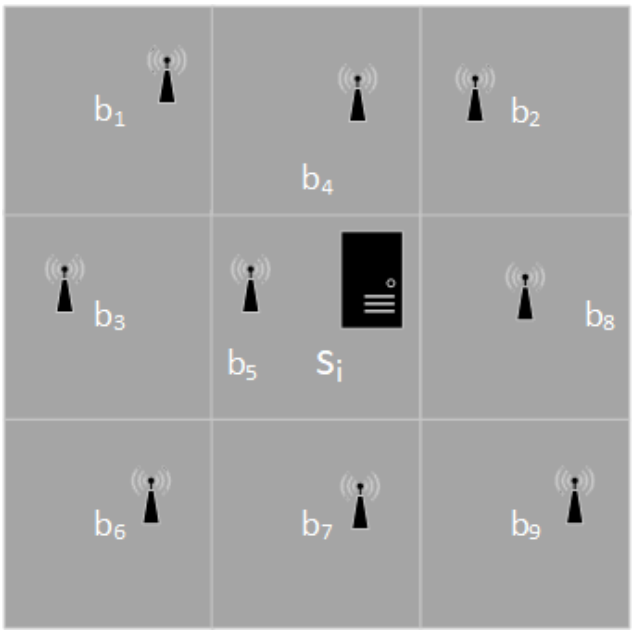

Figure 2. Diagram with location changes of an edge server.

To normalize the action, the target edge server that needs to be relocated is selected and the direction of its movement is determined. On one side, in the problem of decision-making of the target server, the action space is defined as the set of edge servers. Then the action space can be represented as $A_1 = (0, 1, 2, \cdots, n)$, where n represents the sequence number of the nth edge server. On the other side, in the problem of decision-making of the server's movements, the action space corresponds to the direction set of the server's movement, which is defined as $A_2 = (0, 1, 2, 3)$. 0 means moving upwards, 1 means moving downwards, 2 means left, and 3 means moving right. Therefore, it yields to two discrete action spaces, i.e., A_1 and A_2. In order to simplify the problem model, we combine these two action spaces into one, which is the multiplication cross of A_1 and A_2. Because there are n edge servers and each edge server can move in 4 directions, there are 4 * n actions in total. Therefore, the action space is described as $A = (0, 1, 2, \cdots, 4 * n)$. It means that for the input state sequence with n edge servers, there are $4 * n$ output actions, and the optimal action should be selected from $4 * n$ actions.

Based on action A, the target edge server that needs to be relocated is calculated as Formula (5).

$$s_i = A/4 \tag{5}$$

Then the direction of the movement for the edge server s_i is calculated as Formula (6), where % is a mod operator.

$$d_i = A\%4 \tag{6}$$

4.1.3. Reward

We evaluate the placement performance of the edge server from two aspects: average access delay and workload standard deviation. The lower the average delay, the smaller workload standard deviation, indicating the better placement performance. First, the

average access delay and the workload standard deviation are standardized. We use the Z-Score standardization method. Assuming a sequence $(x_1, x_2, \cdots, x_n)$, the standardized formula is:

$$\begin{cases} \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \\ sd = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \\ z_i = \frac{x_i - \bar{x}}{sd} \end{cases} \tag{7}$$

By combining Formulas (2) and (4), the normalization values of the workload balance and the average access delay of the system can be obtained with their respective historical data sets $(WSD_1, WSD_2, \ldots, WSD_n)$ and $(MD_1, MD_2, \ldots, MD_n)$. Then the reward function is shown in Formula (8).

$$R = -(\alpha Z_{WSD} + \beta Z_{MD}) \tag{8}$$

Therein, Z_{WSD} and Z_{MD} represent the normalization values of the workload balance and the average access delay, respectively; α and β represent the weights of two factors, and their sum is 1.

4.2. DQN-ESPA

Q-learning is a classic non-model-based reinforcement learning algorithm [36] which is used in a variety of problems, including the optimal control problem in the Markov decision process [37], human-machine dialogue [38], robot navigation [39], production scheduling [40], traffic control [41], and so on. The Q-learning algorithm takes the timely reward value r leaving the current state and the maximum Q value $\max_{a'} Q(s', a')$ of the next state s' as a complete Markov decision sequence gain. We use the Bellman optimal equation [42] to update the Q value, as shown in Formula (9).

$$Q(s,a) = Q(s,a) + \alpha(s,a)(r + \gamma \max_{a'} Q(s', a') - Q(s,a))) \tag{9}$$

In this formula, $\alpha(s,a) \in [0,1]$ is the learning rate and it is related to the state and action; $\gamma \in [0,1]$ is the weight used to balance the timely reward and the long-term reward.

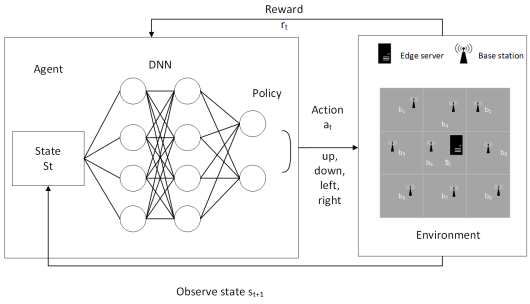

Figure 3. Deep Q-network used in DQN-ESPA.

After initializing the edge computing environment, m placement positions are randomly selected from the n candidate positions, and the initial state is obtained according to the state space defined in the MDP model. At the beginning, the agent executes an action with a random strategy. Each time an action is randomly selected from the action space defined in the MDP model, the agent enters the next state after executing the action. Then the environment gives the agent a timely reward according to the reward function defined in the MDP model. Notice that according to Formula (9) the state action value is updated. This is reflected as the agent continuously interacts with the environment, i.e., the strategies learned by the agent improve persistently, and thus the strategy will converge from a random strategy to the optimal strategy.

During the interaction between the agent and the environment, the agent stores the state action value $Q(s,a)$ of the action a executed in the state s in a two-dimensional table. When this table has all state action values, the agent can execute the optimal action

accordingly. As shown in [16], when the state space is large, a curse of dimensionality problem arises. To overcome this problem we use DNN in our DQN to estimate the value function of Q-learning algorithm [43]. Then, the optimal offload action can be directly obtained by choosing the one that has the maximum Q-value. Figure 3 depicts the general DQN flow in our algorithm. Our DQN consists of one hidden layer of 50 neurons.

Our edge server placement algorithm, DQN-ESPA, is described in Algorithm 1. DQN-ESPA updates the θ parameter to let the approximate function $\widehat{Q}$ approximately represent the Q value, as shown in Formula (10).

$$Q(s_t, a_t) \approx \widehat{Q}(s_t, a_t, \theta) \tag{10}$$

Algorithm 1 DQN-ESPA

1: Initialization of Replay memory M
2: Initialization of Q function with random parameters θ
3: Initialization of $\hat{Q}$ function with parameters θ^-
4: Randomly select m placement positions from n candidate placement positions
5: Construct the initial state s_1 based on step 4
6: $t \leftarrow 1$
7: **while** $t \leq T$ **do**
8: generate a random number rnd in [0,1]
9: **if** $rnd > \epsilon$ **then**
10: Select $a_t = argmax_a Q_t(s_t, a; \theta)$
11: **else**
12: Select action a_t randomly
13: **end if**
14: Execute action a_t, observe r_t, s_{t+1}
15: Store experience (s_t, a_t, r_t, s_{t+1}) in M
16: Sample minibatch of (s_t, a_t, r_t, s_{t+1}) randomly from M
17: **if** t + 1 is the final one **then**
18: Set $Z_t = r_t$
19: **else**
20: Set $Z_t = r_t + \gamma \max_{a_{t+1}} \hat{Q}_t(s_{t+1}, a_{t+1}; \theta^-)$
21: **end if**
22: Gradient descent step is executed on $(Z_t - Q_t(s_t, a; \theta))^2$ by θ
23: Reset $\hat{Q} = Q$ in every C steps
24: $t \leftarrow t + 1$
25: **end while**

At time step t, the state $s_t = ((lat_1, lon_1), (lat_2, lon_2), \ldots, (lat_m, lon_m))$ is set as the input of the neural network to get the current state action $Q_t(s_t, a_t; \theta_i)$ value, as shown on lines 4 and 5 in Algorithm 1. From line 7 to line 13, the algorithm selects an action according to the $\epsilon - greedy$ policy. In order to reduce the connection between the estimated value and the target value and improve training efficiency, DQN-ESPA adds a target network. The target network and the estimated network have the same parameters. Every C time steps, the agent assigns the parameters of the estimated network to the target network (line 23 in Algorithm 1). The target value Z is defined in Formula (11):

$$Z = r_t(s_t, a_t) + \gamma \max_{a_t+1} Q_t(s_{t+1}, a_{t+1}; \theta_i^-) \tag{11}$$

θ_i^- represents the parameters of the target network before a certain iteration time. The loss function L_i in DQN-ESPA is described in Formula (12):

$$L_i(\theta_i) = \mathbb{E}_{s_t, a_t, r_t, s_{t+1}}[(Z - Q_t(s_t, a_t; \theta_t))^2] \tag{12}$$

DQN-ESPA uses an experience pool M to store state action transition sequences (s_t, a_t, r_t, s_{t+1}). According to the experience replay strategy, DQN-ESPA randomly selects a

batch of samples from the experience pool for training at each iteration step (lines 15 and 16 in Algorithm 1).

To differentiate the gradient from the loss function, we define it as Formula (13):

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s_t, a_t, r_t, s_{t+1}}[(Z - Q_t(s_t, a_t; \theta_t))\nabla_{\theta_i} Q_t(s_t, a_t; \theta_i)] \tag{13}$$

Thus, the gradient $\nabla_{\theta_i} Q_t(s_t, a_t; \theta_i)$ guides loss function to be reduced at a feasible direction. Finally, we can update the parameters as:

$$\theta_{i+1} = \theta_{i+1} + \alpha \nabla_{\theta_i} L_i(\theta_i) \tag{14}$$

where α is the learning rate in (0,1).

As it can be seen on line 14 in Algorithm 1, the agent's experience (s_t, a_t, r_t, s_{t+1}) is stored in replay memory, therefore there is no need for transition probabilities.

5. Performance Evaluation

The average access delay and workload balance are adopted as evaluation indicators to evaluate the performance. The proposed DQN-ESPA is compared to classical algorithms such as SAPA, KMPA, TKPA and RPA. All the experiments were carried out using a real dataset, and the impact of different edge server numbers and placement performance carefully studied.

5.1. Configuration of the Experiments

The parameters used in the experiments are set uniformly: the storage space $M = 100,000$, the learning rate $\alpha = 0.01$, the batch size $K = 64$, the parameter update frequency $C = 100$, and the discount factor $\gamma = 0.9$.

In order to evaluate the overall results of the experiments, the experimental results will be normalized through the log function shown in Formula (15):

$$norm_i = \lg(x)/lg(max) \tag{15}$$

where x is the sample value that needs to be normalized and max is the maximum value in the sample. Then the overall evaluation indicator is defined as:

$$index(ad, wb) = \mu norm_{ad} + (1 - \mu) norm_{wb} \tag{16}$$

In this formula, $norm_{ad}$ is the normalized value of average delay, $norm_{wb}$ is the normalized value of workload balance, and μ is the weighting factor, $\mu \in [0, 1]$. In this paper μ is set as $\mu = 0.5$.

All the algorithms, including KMPA, TKPA, RPA, and DQN-ESPA, were implemented in Python 3.6, The benchmarks were executed on a computer system with Intel (R)Xeon (R) CPU E5-26200@2.00 Ghz, 128 GB memory and operated in Linux distribution Ubuntu 16.04.

5.2. Dataset Description

The dataset used in the experiments was obtained from Shanghai Telecom (http://sguangwang.com/TelecomDataset.html, accessed on 17 February 2022). It contains anonymous calls/Internet information from mobile user requests to 3233 base stations. The exact location of all base stations is included. The dataset contains 4.6 million call records, and 7.5 million Internet traffic records of approximately 10,000 anonymous mobile users gathered for 6 consecutive months. Each call/traffic record contains detailed start time and end time for each mobile user and its corresponding base station. From the Shanghai Telecom dataset we selected the data of 3000 valid base station.

5.3. Experimental Results

In order to study the performance effect on the variety of the number of edge servers to be placed, the placement performance are evaluated when the number of edge servers varies between 100 and 300.

The access delay and workload balance of DQN-ESPA are shown in Figure 4a,b when placing 100 edge servers. As the number of iterations increases, the average delay of the entire system continues to decline. It can also be seen that the workload standard deviation continuously decreases. The average access delay and the workload standard deviation of the compared algorithms are shown in Figure 4c,d. The results show that the DQN-ESPA is able to obtain both the lowest average delay and the minimum workload standard deviation.

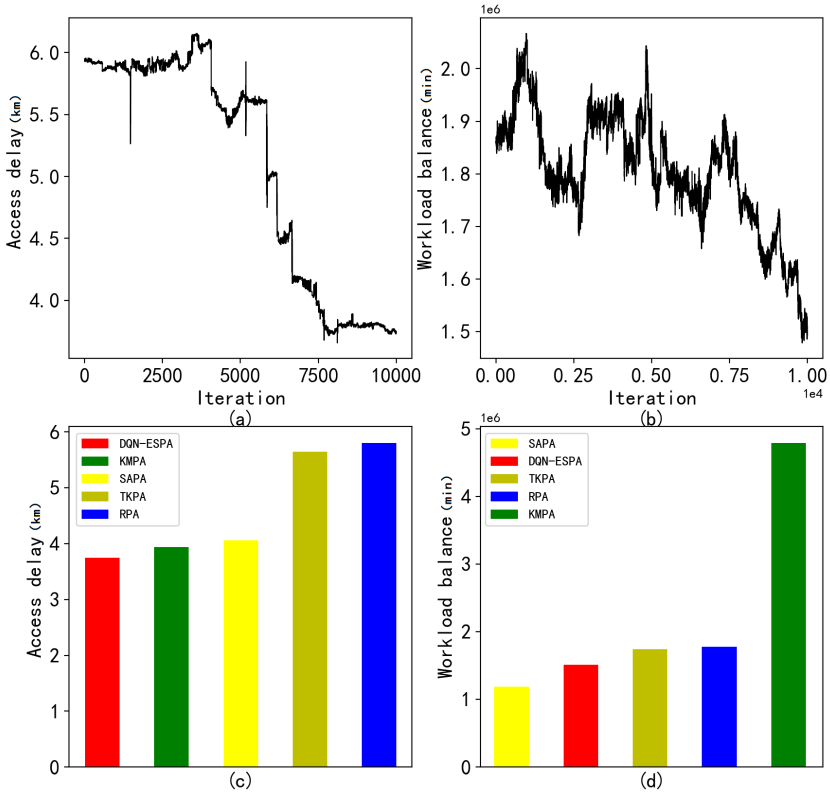

Figure 4. Results with the placement of 100 edge servers at 3000 base station locations. (**a**) Access delay of DQN-ESPA with the variety of iterations; (**b**) Workload balance of DQN-ESPA with the variety of iterations; (**c**) Average access delay of the compared algorithms; (**d**) Average workload standard deviation of the compared algorithms.

Figure 5 shows the experimental results when placing 300 edge servers. Therein, Figure 5a,b correspond to the DQN-ESPA delay and workload balance with the variety of the number of iterations. Similar to the previous experiment, the average delay and the workload standard deviation decreases with the rise of the number of iterations. The average delay and the standard deviation of the workload balance of all the compared algorithms are presented in Figure 5c,d. It shows that the DQN-ESPA obtains the lowest average delay. It also shows that KMPA can obtain relative low average delay when placing edge servers, but the workload standard deviation obtained by KMPA is much bigger than the other four algorithms. Meanwhile, TKPA obtains the minimum workload balance standard deviation, but it gets the worse average delay comparing to DQN-ESPA and KMPA.

It can also been seen that DQN-ESPA does not always achieve the best performance on the indicator of workload balance standard deviation, e.g., TKPA and SAPA obtain less

workload balance standard deviation placing 300 edge servers in Figure 5d. Therefore, to further compare the algorithms' performance, we utilize the overall evaluation indicator presented in Formula (16) to evaluate their comprehensive performance, and the results are shown in Table 1. It shows that the comprehensive performance ranking from high to low of the compared algorithms is DQN-ESPA > SAPA > KMPA > TKPA > RPA.

We also introduce the relative comprehensive performance improvement ratio RP to evaluate the performance, as shown in Formula (17). Therein, $index_{DQN\_ESPA}$ is the comprehensive performance of DQN-ESPA, while $index_{alg}$ is the comprehensive performance of the compared algorithms, and $alg \in \{SAPA, KMPA, TKPA, RPA\}$. When placing 100 edge servers, the overall indicator of DQN-ESPA outperforms SPPA, KMPA , TKPA, and RPA by 1.75%, 5.80%, 12.61%, and 13.40%, respectively. When placing 300 edge servers, the overall indicator of DQN-ESPA outperforms SAPA, KMPA, TKPA, and RPA by 2.39%, 5.22%, 13.26%, and 15.54%, respectively. The results indicate that DQN-ESPA comprehensively considers the average delay and workload balance, and subsequently achieves the best comprehensive performance among the compared algorithms considered in this evaluation.

$$RC = \frac{index_{alg} - index_{DQN\_ESPA}}{index_{alg}} \tag{17}$$

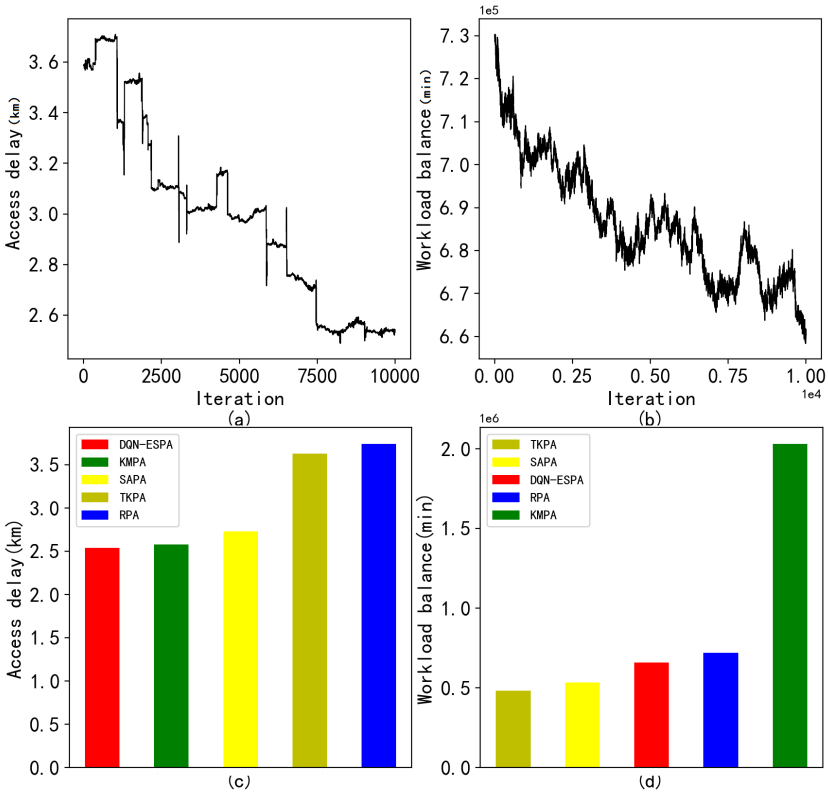

Figure 5. Results with the placement of 300 edge servers at 3000 base station locations. (**a**) Access delay of DQN-ESPA with the variety of iterations; (**b**) Workload balance of DQN-ESPA with the variety of iterations; (**c**) Average access delay of the compared algorithms; (**d**) Average workload standard deviation of the compared algorithms.

Table 1. Overall performance indicators.

No.	DQN-ESPA	SAPA	KMPA	TKPA	RPA
100	0.8380	0.8529	0.8896	0.9589	0.9677
300	0.8144	0.8343	0.8593	0.9389	0.9643

6. Conclusions

In this paper we proposed a novel algorithm, DQN-ESPA, for the ESPP in MEC systems. We abstract ESPP in a Markov decision sequence, and the edge server placement problem is solved through deep reinforcement learning. Utilizing a real dataset from Shanghai Telecom, we compared DQN-ESPA to several state-of-the-art algorithms, such as SAPA, KMPA, TKPA and RPA, and use the average access delay and workload balance of the entire system as evaluation indicators. Experimental results show that in comparison to SAPA, KMPA, TKPA and RPA, DQN-ESPA is able to achieve up to 13.40% better performance with 100 edge servers, and up to 15.54% better performance with 300 edge servers.

Although DQN-ESPA surpasses typical comparison algorithms, its convergence speed is relatively slow. This matter will be further studied as future work. In terms of new applications for the proposed algorithm, in the field of edge computing task offloading is another hot topic, and DQN-ESPA can be further studied to solve comprehensive problems for task offloading and the ESPP.

Author Contributions: Conceptualization, Y.L.; Methodology, W.D.; Supervision, F.L.; Writing—original draft, S.Z.; Writing—review editing, J.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the project on Shanghai Science and Technology Innovation Action Plan (No. 20dz1201400, No. 22ZR1416500) and sponsored by Shanghai Sailing Program (20YF1410900).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The dataset used in the experiments was obtained from Shanghai Telecom: http://sguangwang.com/TelecomDataset.html, accessed on 17 February 2022.

Acknowledgments: Yongjun Luo and Chunhua Gu are thanked for expert advice and inspiring discussions.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. You, C.; Huang, K.; Chae, H.; Kim, B.H. Energy-efficient resource allocation for mobile-edge computation offloading. *IEEE Trans. Wirel. Commun.* **2016**, *16*, 1397–1411. [CrossRef]
2. Mao, Y.; Zhang, J.; Letaief, K.B. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE J. Sel. Areas Commun.* **2016**, *34*, 3590–3605. [CrossRef]
3. Arkian, H.R.; Diyanat, A.; Pourkhalili, A. MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications. *J. Netw. Comput. Appl.* **2017**, *82*, 152–165. [CrossRef]
4. Gu, L.; Zeng, D.; Guo, S.; Barnawi, A.; Xiang, Y. Cost efficient resource management in fog computing supported medical cyber-physical system. *IEEE Trans. Emerg. Top. Comput.* **2015**, *5*, 108–119. [CrossRef]
5. Taleb, T.; Ksentini, A. An analytical model for follow me cloud. In Proceedings of the 2013 IEEE Global Communications Conference (GLOBECOM), Atlanta, GA, USA, 9–13 December 2013; pp. 1291–1296.
6. Taleb, T.; Ksentini, A.; Frangoudis, P. Follow-me cloud: When cloud services follow mobile users. *IEEE Trans. Cloud Comput.* **2016**, *7*, 369–382. [CrossRef]
7. Wang, S.; Guo, Y.; Zhang, N.; Yang, P.; Zhou, A.; Shen, X.S. Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach. *IEEE Trans. Mob. Comput.* **2019**, *20*, 939–951. [CrossRef]
8. Wang, S.; Zhao, Y.; Huang, L.; Xu, J.; Hsu, C.H. QoS prediction for service recommendations in mobile edge computing. *J. Parallel Distrib. Comput.* **2019**, *127*, 134–144. [CrossRef]
9. Ma, L.; Wu, J.; Chen, L. DOTA: Delay bounded optimal cloudlet deployment and user association in WMANs. In Proceedings of the 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), Madrid, Spain, 14–17 May 2017; pp. 196–203.
10. Xu, Z.; Liang, W.; Xu, W.; Jia, M.; Guo, S. Capacitated cloudlet placements in wireless metropolitan area networks. In Proceedings of the 2015 IEEE 40th Conference on Local Computer Networks (LCN), Clearwater Beach, FL, USA, 26–29 October 2015; pp. 570–578.

11. Meng, J.; Shi, W.; Tan, H.; Li, X. Cloudlet placement and minimum-delay routing in cloudlet computing. In Proceedings of the 2017 3rd International Conference on Big Data Computing and Communications (BIGCOM), Chengdu, China, 10–11 August 2017; pp. 297–304.
12. Wang, S.; Zhao, Y.; Xu, J.; Yuan, J.; Hsu, C.H. Edge server placement in mobile edge computing. *J. Parallel Distrib. Comput.* **2019**, *127*, 160–168. [CrossRef]
13. Guo, Y.; Wang, S.; Zhou, A.; Xu, J.; Yuan, J.; Hsu, C.H. User allocation-aware edge cloud placement in mobile edge computing. *Software Pract. Exp.* **2020**, *50*, 489–502. [CrossRef]
14. Zhang, R.; Yan, F.; Xia, W.; Xing, S.; Wu, Y.; Shen, L. An optimal roadside unit placement method for vanet localization. In Proceedings of the GLOBECOM 2017-2017 IEEE Global Communications Conference, Singapore, 4–8 December 2017 ; pp. 1–6.
15. Lu, H.; Gu, C.; Luo, F.; Ding, W.; Liu, X. Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning. *Future Gener. Comput. Syst.* **2020**, *102*, 847–861. [CrossRef]
16. Curran, W.; Brys, T.; Taylor, M.; Smart, W. Using PCA to efficiently represent state spaces. *arXiv* **2015**, arXiv:1505.00322.
17. Dolui, K.; Datta, S.K. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In Proceedings of the 2017 Global Internet of Things Summit (GIoTS), Geneva, Switzerland, 6–9 June 2017; pp. 1–6.
18. Lhderanta, T.; Leppnen, T.; Ruha, L.; Lovén, L.; Harjula, E.; Ylianttila, M.; Riekki, J.; Sillanp, M.J. Edge computing server placement with capacitated location allocation. *J. Parallel Distrib. Comput.* **2021**, *153*, 130–149. [CrossRef]
19. Zhao, X.; Zeng, Y.; Ding, H.; Li, B.; Yang, Z. Optimize the placement of edge server between workload balancing and system delay in smart city. *Peer-to-Peer Netw. Appl.* **2021**, *14*, 3778–3792. [CrossRef]
20. Liang, Y.; Liu, H.; Rajan, D. Optimal placement and configuration of roadside units in vehicular networks. In Proceedings of the 2012 IEEE 75th Vehicular Technology Conference (VTC Spring), Yokohama, Japan, 6–9 May 2012; pp. 1–6.
21. Wang, Z.; Zheng, J.; Wu, Y.; Mitton, N. A centrality-based RSU deployment approach for vehicular ad hoc networks. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017.
22. Aslam, B.; Amjad, F.; Zou, C.C. Optimal roadside units placement in urban areas for vehicular networks. In Proceedings of the 2012 IEEE Symposium on Computers and Communications (ISCC), Cappadocia, Turkey, 1–4 July 2012; pp. 000423–000429.
23. Trullols, O.; Fiore, M.; Casetti, C.; Chiasserini, C.F.; Ordinas, J.B. Planning roadside infrastructure for information dissemination in intelligent transportation systems. *Comput. Commun.* **2010**, *33*, 432–442. [CrossRef]
24. Balouchzahi, N.M.; Fathy, M.; Akbari, A. Optimal road side units placement model based on binary integer programming for efficient traffic information advertisement and discovery in vehicular environment. *IET Intell. Transp. Syst.* **2015**, *9*, 851–861. [CrossRef]
25. Premsankar, G.; Ghaddar, B.; Di Francesco, M.; Verago, R. Efficient placement of edge computing devices for vehicular applications in smart cities. In Proceedings of the NOMS 2018—2018 IEEE/IFIP Network Operations and Management Symposium, Taipei, Taiwan, 23–27 April 2018; pp. 1–9.
26. Fan, Q.; Ansari, N. Cost aware cloudlet placement for big data processing at the edge. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017.
27. Jia, M.; Cao, J.; Liang, W. Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks. *IEEE Trans. Cloud Comput.* **2015**, *5*, 725–737. [CrossRef]
28. Lewis, G.; Echeverría, S.; Simanta, S.; Bradshaw, B.; Root, J. Tactical cloudlets: Moving cloud computing to the edge. In Proceedings of the 2014 IEEE Military Communications Conference, Baltimore, MD, USA, 6–8 October 2014; pp. 1440–1446.
29. Li, H.; Dong, M.; Liao, X.; Jin, H. Deduplication-based energy efficient storage system in cloud environment. *Comput. J.* **2015**, *58*, 1373–1383. [CrossRef]
30. Li, H.; Dong, M.; Ota, K.; Guo, M. Pricing and repurchasing for big data processing in multi-clouds. *IEEE Trans. Emerg. Top. Comput.* **2016**, *4*, 266–277. [CrossRef]
31. Zeng, F.; Ren, Y.; Deng, X.; Li, W. Cost-effective edge server placement in wireless metropolitan area networks. *Sensors* **2019**, *19*, 32. [CrossRef]
32. Mao, H.; Alizadeh, M.; Menache, I.; Kandula, S. Resource management with deep reinforcement learning. In Proceedings of the 15th ACM Workshop on Hot Topics in Networks, Atlanta, GA, USA, 9–10 November 2016; pp. 50–56.
33. Skarlat, O.; Nardelli, M.; Schulte, S.; Borkowski, M.; Leitner, P. Optimized IoT service placement in the fog. *Serv. Oriented Comput. Appl.* **2017**, *11*, 427–443. [CrossRef]
34. Yang, L.; Cao, J.; Liang, G.; Han, X. Cost aware service placement and load dispatching in mobile cloud systems. *IEEE Trans. Comput.* **2015**, *65*, 1440–1452. [CrossRef]
35. Messaoudi, F.; Ksentini, A.; Bertin, P. On using edge computing for computation offloading in mobile network. In Proceedings of the GLOBECOM 2017-2017 IEEE Global Communications Conference, Singapore, 4–8 December 2017; pp. 1–7.
36. Watkins, C.J. Technical note q-learning. *Reinf. Learn.* **1993**, *8*, 279–292.
37. Bertsekas, D.P. Stable optimal control and semicontractive dynamic programming. *SIAM J. Control. Optim.* **2018**, *56*, 231–252. [CrossRef]
38. Scheffler, K.; Young, S. Automatic learning of dialogue strategy using dialogue simulation and reinforcement learning. In Proceedings of the Second International Conference on Human Language Technology Research, Citeseer, San Francisco, CA, USA, 24–27 March 2002; pp. 12–19.

39. Yang, G.S.; Chen, E.K.; An, C.W. Mobile robot navigation using neural Q-learning. In Proceedings of the 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 04EX826), Shanghai, China, 26–29 August 2004; Volume 1, pp. 48–52.
40. Wang, Y.C.; Usher, J.M. Application of reinforcement learning for agent-based production scheduling. *Eng. Appl. Artif. Intell.* **2005**, *18*, 73–82. [CrossRef]
41. Djonin, D.V.; Krishnamurthy, V. Q-Learning Algorithms for Constrained Markov Decision Processes With Randomized Monotone Policies: Application to MIMO Transmission Control. *IEEE Trans. Signal Process.* **2007**, *55*, 2170–2181. [CrossRef]
42. Bellman, R. Dynamic programming. *Science* **1966**, *153*, 34–37. [CrossRef]
43. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef]