Engineering Applications of Artificial Intelligence 139 (2025) 109662

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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

Workload-based adaptive decision-making for edge server layout with deep reinforcement learning

Shihua Li [a], Yanjie Zhou [b,*], Bing Zhou [a], Zongmin Wang [a]

[a] School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou, 450001, China
[b] School of Management, Zhengzhou University, Zhengzhou, 450001, China

ARTICLE INFO

Keywords:
Server placement
Mobile edge computing
Deep reinforcement learning

ABSTRACT

Mobile edge computing (MEC) is crucial in applications such as intelligent transportation, innovative healthcare, and smart cities. By deploying servers with computing and storage capabilities at the network edge, MEC enables low-latency services close to end users. However, the configuration of edge servers needs to meet the low-latency requirements and effectively balance the servers' workloads. This paper proposes an adaptive layout and dynamic optimization method, modeling the edge server layout problem as a Markov decision process. It introduces a workload-based server placement rule that adjusts the locations of edge servers according to the load of base stations, enabling the learning of low-latency and load-balanced server layout strategies. Experimental validation on a real dataset from Shanghai Telecom shows that the proposed algorithm improves average latency performance by about 40% compared to existing technologies, and enhances workload balancing performance by about 17%.

1. Introduction

The proliferation of smart healthcare devices, intelligent driving vehicles, and various innovative mobile devices has led to denser data exchanges between the Internet of Things (IoT) devices (Ning et al., 2023; Haiyan et al., 2021). However, the limited storage and computational resources of mobile devices themselves cannot meet the real-time processing demands of specific tasks, necessitating network providers to offer faster response solutions (Sun and He, 2023). Mobile Edge Computing (MEC) has recently been regarded a promising solution. It places storage and computational resources on the demand side, reducing communication costs and addressing network congestion issues (Chai et al., 2023).

In recent years, there has been a substantial body of research addressing challenges in the field of MEC. However, most of these studies primarily concentrate on problems such as task migration (Dai et al., 2022; Moon et al., 2022; Liao et al., 2023), task offloading (Carvalho et al., 2020; Coito et al., 2022; Abbasi and Hadi, 2024), and task prediction (Ferreira et al., 2023; Wang et al., 2023a), without addressing the placement of edge servers. They focus exclusively on resource scheduling after determining server locations, aiming to provide high-quality services to users. Nevertheless, the placement of edge servers is equally critical for delivering high-quality services. A well-structured edge server placement scheme can reduce latency and optimize load, while a suboptimal placement strategy can lead to network congestion and resource wastage (Karasakal and Karasakal, 2023).

Existing research on edge server placement primarily formulates the problem as a multi-objective optimization problem and then addresses it using conventional methods such as mixed integer programming algorithms (Jasim and Al-Raweshidy, 2024), heuristic algorithms (Poularakis et al., 2020), etc. However, these algorithms are only suitable for limited edge server configurations. As the number of edge servers increases, the computational burden of mixed integer programming algorithms grows exponentially, rendering it infeasible to find the optimal solution within a limited time. Heuristic algorithms require parameter reconfiguration and exhibit limited adaptability. Solutions based on Deep Reinforcement Learning (DRL) frame the edge server placement challenge as a Markov decision process, where agents can iteratively enhance their placement strategy through online learning, providing real-time and flexible solutions (Lu et al., 2022).

Based on the reasons mentioned above, this paper presents an adaptive placement and dynamic optimization framework. This method is based on the Deep Q-Network (DQN) algorithm, which facilitates the optimization of strategies within high-dimensional state and action environments. Additionally, we present a workload-driven location selection criterion to dynamically modify the positions of edge servers. We develop reward and penalty metrics derived from server load

* Corresponding author.Yanjie Zhou (ieyjzhou@zzu.edu.cn)
E-mail addresses: lishihua@gs.zzu.edu.cn (S. Li), ieyjzhou@zzu.edu.cn (Y. Zhou), iebzhou@zzu.edu.cn (B. Zhou), zmwang@zzu.edu.cn (Z. Wang).

https://doi.org/10.1016/j.engappai.2024.109662
Received 24 June 2024; Received in revised form 4 November 2024; Accepted 10 November 2024
Available online 22 November 2024

variability and system average latency, iteratively refining the edge server placement strategy. The main contributions of this paper are summarized as follows:

- We formulate the edge server placement challenge as a Markov decision process and introduced an innovative adaptive placement and dynamic optimization (APD) strategy to reduce system average latency and enhance workload balancing performance;
- We develop a novel workload-based placement rule (LSD) that adjusts the locations of edge servers according to the load of base stations. This effectively prevents certain base stations from becoming overloaded, thereby enhancing the overall stability and reliability of the system;
- We perform comprehensive experiments demonstrating that the APD algorithm significantly outperforms multiple baseline methods, highlighting its superior effectiveness and adaptability in real-world applications.

The remaining sections of the paper are organized as follows: Section 2 provides an overview of related work in the field. Section 3 defines the edge server placement problem and formalizes it as a mathematical model. Section 4 introduces the APD framework utilized for solving the problem. Extensive experiments, performance evaluations, and discussions based on real data are presented in Section 5. Finally, Section 6 concludes the paper and outlines future research directions.

2. Literature review

2.1. Mathematical methods

Bhatta et al. formulated the problem into a multi-objective integer programming model and proved its computational NP-hardness. Subsequently, they proposed a dual-factor approximation algorithm to address the complexity of the problem (Bhatta and Mashayekhy, 2022). Tero et al. proposed a block coordinate descent algorithm that optimizes the server locations and workload distribution between the minimum access points while satisfying capacity constraints (Lähderanta et al., 2021). Xu et al. proposed a collaborative method for quantifying and deploying edge servers. This method first initializes population strategies through canopy and k-medoid clustering algorithms to estimate the approximate number of edge servers. Then, it utilizes nondominated sorting genetic algorithm III to obtain solutions with higher quality of service (Xu et al., 2021). Cao et al. adopted an offline and online two-phase method to study the placement of heterogeneous edge servers to optimize the overall system and individual base stations' expected response time. In the offline stage, the best placement strategy for heterogeneous edge servers was generated using integer linear programming techniques. In the online stage, a game theory approach based on mobility awareness was employed to handle the dynamic nature of user mobility (Cao et al., 2021). Zhang et al. proposed a two-step method involving clustering algorithms and nonlinear programming. They design a joint edge server placement and service placement model aimed at maximizing the total profit of all edge servers under constraints such as the number of edge servers, relationships between edge servers and base stations, storage capacity, and computing capability of each edge server (Zhang et al., 2022). Do et al. used the mixed-integer linear programming technique to model and solve the joint user association, service function chain placement, and resource allocation problems in 5G networks composed of decentralized units, centralized units, and the core network (Do and Kim, 2018). K. Balaji et al. employed an improved discrete firefly algorithm to effectively explore a large search space and find virtual machine placement schemes with minimal power consumption in data centers (Balaji et al., 2022). Cui et al. modeled the k-edge server placement problem as a constrained optimization problem involving joint user coverage and network robustness optimization. They proposed an optimal method based on integer programming to determine the optimal solution for the small-scale k-edge server placement problem (Cui et al., 2022). Song et al. transformed a nonlinear system into a linear system using a Takagi–Sugeno (T-S) fuzzy model. They then introduced the concepts of point measurement and point control to reduce the use of sensors and actuators, thereby conserving communication resources and lowering control costs (Song et al., 2023). Song et al. employed integral techniques and the Wirtinger inequality to derive sufficient conditions for system stability. Their work provides an effective research methodology for nonlinear systems' stability analysis and controller design (Song et al., 2024). Zhang et al. employed Lyapunov stability theory to develop an adaptive dynamic programming control framework. By designing a preset-time controller, they ensured the system state stabilizes within a predetermined time (Zhang et al., 2024).

These methods define the server placement problem as a multi-objective integer programming model and pursue the optimal solution via constraints. However, such methods are suitable solely for small-scale server positioning, as they involve significant computational complexity for large-scale server positioning challenges. Although specific studies have utilized approximation techniques to identify suboptimal solutions rapidly, this generally necessitates prior knowledge, and various challenges may demand the development of new approximation techniques, leading to restricted algorithm flexibility.

2.2. Reinforcement learning methods

Yuan et al. proposed a dynamic virtual edge node deployment scheme based on deep learning, which utilizes an on-demand computing model to acquire resources for virtual edge nodes from different clouds, enabling them to be deployed across clouds worldwide (Yuan et al., 2022). Wang et al. proposed a fault-tolerant control strategy based on Iterative Learning Control (ILC) that effectively addresses actuator failures. Their approach enables continuous optimization of system performance by employing real-time fault estimation and dynamically adjusting the ILC update law (Wang et al., 2023b). Furthermore, to minimize the long-term average response time of video analysis tasks, Zhu et al. utilized a Markov decision process to model the deployment process and employed deep reinforcement learning to explore the optimal strategy (Zhu et al., 2023). Moreover, Jiang et al. transformed the edge server deployment problem into a sequential decision problem based on Markov decision processes and used heatmaps and grayscale images to convert network states into inputs directly learnable by agents (Jiang et al., 2023). Additionally, Xue et al. introduced a deep reinforcement learning-based algorithm to tackle computation offloading and service caching issues in vehicle edge computing systems. This method effectively utilizes the limited cache and computing resources at each node through intelligent decision-making and resource management, achieving low-complexity decision-making and adaptive resource management (Xue et al., 2023). Kasi et al. also presented a multi-agent reinforcement learning solution, minimizing network latency and balancing edge server loads by learning the dynamic characteristics of the environment and adopting joint action strategies (Kasi et al., 2021). Lastly, Luo et al. formalized the state space, action space, and reward function of the deployment problem using deep Q-networks and reinforcement learning methods (DQN-ESPA), aiming to find the optimal solution by maximizing cumulative long-term rewards (Luo et al., 2022).

While these studies successfully tackle the edge server placement problem in large-scale MEC systems, they fall short in modeling dynamic and adaptive server deployment. Additionally, to fully exploit the capabilities of edge computing, all tasks should be optimally processed as close to the user as possible, taking into account server mobility distance. Therefore, this paper proposes an adaptive edge server placement and dynamic optimization approach and formulates a workload-based location selection strategy to adjust the positioning of edge servers dynamically.

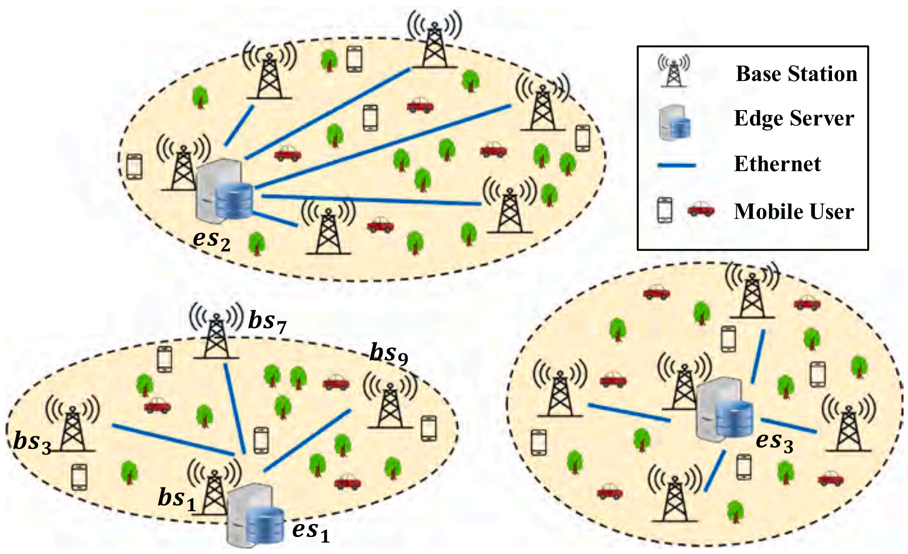


Fig. 1. The diagram of the MEC.

3. Problem definition

In this section, we introduce the server placement problem in MEC architecture, simplified as a network topology consisting of users, base stations, and servers. We then formalize server load balancing and user access delay, which are used to evaluate the efficacy of server placement algorithms. Ultimately, we introduce the constraints that the server placement problem needs to meet.

3.1. Problem description

To overcome the challenge of limited computational resources at the user end, MEC deploys servers with ample computing resources closer to users at the network edge. MEC can be represented by a network topology containing n base stations, m edge servers, and x users, denoted as $BS = \{bs_1, bs_2, \dots, bs_i, \dots, bs_n\}$ for the set of base stations, $ES = \{es_1, es_2, \dots, es_j, \dots, es_m\}$ for the set of edge servers, and $US = \{us_1, us_2, \dots, us_k, \dots, us_x\}$ for the set of users. Users transmit their computation demands through base stations to edge servers, and the edge servers return computation results to users through base stations. Therefore, the load of a server is the total load aggregated by all base stations. As shown in Fig. 1, edge server es_1 covers base stations bs_1, bs_3, bs_7 and bs_9, and the workload of es_1 is the sum of the loads of bs_1, bs_3, bs_7 and bs_9. To achieve optimal MEC architecture through the strategic placement of edge servers, we propose the following four assumptions (Zhao et al., 2021):

- Each edge server has the same computational capacity;
- Each edge server can only coexist with one existing base station at a location, and edge servers cannot share a location with other edge servers;
- Each base station can only be served by one edge server;
- The distance between base stations and edge servers represents delay.

Server placement refers to finding the optimal locations for edge servers based on historical data of user connections to base stations over several days, aiming to achieve workload balancing among edge servers and minimize the average access delay of base stations. The details of delay and workload balance are described as follows.

3.2. Average delay

Base stations establish communication links with edge servers through the communication network, and as mentioned earlier, we represent the distance between base stations and edge servers as delay. Let $L_n = [(lat_1, lon_1), (lat_2, lon_2), \dots, (lat_i, lon_i), \dots, (lat_n, lon_n)]$ represent the set of base station locations, and let $L_m = [(lat_1, lon_1), (lat_2, lon_2), \dots, (lat_j, lon_j), \dots, (lat_m, lon_m)]$ represent the set of edge server locations, where lat represents latitude and lon represents longitude, respectively. Then, the distance d_{ij} (km) between base station bs_i and edge server es_j can be calculated by the following equation:

$$p = \frac{\pi}{180}, \tag{1}$$

$$\Omega = \frac{1 - cos((lat_i - lat_j) \cdot p)}{2} + cos(lat_i \cdot p) \cdot cos(lat_j \cdot p) \cdot \frac{1 - cos((lon_i - lon_j) \cdot p)}{2}, \tag{2}$$

$$d_{ij} = 2 \cdot R \cdot \arcsin(\sqrt{\Omega}), \tag{3}$$

where R is the radius of the earth and $R = 6378.137$ km.

We calculate the average D (km) from each base station to its corresponding edge server to represent the overall system delay. The smaller the average distance, the smaller the overall system delay (Luo et al., 2022). This calculation is expressed by the following equation:

$$D = \frac{1}{n} \cdot \sum_{i=1}^{n} \sum_{j=1}^{m} d_{ij}. \tag{4}$$

3.3. Workload balance

All users' computation demands are relayed to edge servers through base stations. Therefore, the workload of server bs_i is the total workload of the base stations it covers. Let BS_j represent the set of base stations covered by edge server es_j, where $BS_j \subset BS$, and w_i denotes the workload of base station bs_i, where $bs_i \in BS_j$. Then, the workload W_j of edge server es_j is expressed as shown in Eq. (5):

$$W_j = \sum_{bs_i \in BS_j} w_i, \tag{5}$$

We define the time a base station serves users as the workload on the base station (Luo et al., 2022). To demonstrate the performance of workload balancing among servers, we calculate the standard deviation $W_{sd}(s)$ of the workloads of all edge servers. A smaller standard deviation indicates more balanced workloads across servers. The calculation of W_{sd} is given by Eq. (6):

$$W_{sd} = \sqrt{\frac{\sum_{j=1}^{m}(W_j - \overline{W})}{m}}. \tag{6}$$

3.4. Computational problem

The mathematical model of the studied problem is similar to a facility location problem (Zhou and Lee, 2020). We have two optimization goals: one is to achieve a more balanced server load, i.e., minimizing W_{sd}; the other is to minimize the average latency of base station access to the server, i.e., minimizing D. We calculate W_{sd} and D based on the placement of the servers. As mentioned earlier, each base station can only be served by one server, and all base stations must be covered. We model the server placement rules as follows:

$$\sum_{i=1}^{n} \alpha_i = m, \tag{7}$$

$$\sum_{j=1}^{n} \beta_{ij} = 1, \forall 1 \le i \le n, \tag{8}$$

$$\beta_{ij} \le \alpha_j, \forall 1 \le i \le n, 1 \le j \le n, \tag{9}$$

$$\alpha_i, \beta_{ij} \in \{0, 1\}, \forall 1 \le i \le n, \forall 1 \le j \le n, \tag{10}$$

indicates whether the base station bs_i is served by the server deployed at base station bs_j. If yes, then $\beta_{ij} = 1$; otherwise, $\beta_{ij} = 0$. Notably, there might be no server deployed at bs_j; this representation is for simplicity. Eq. (7) indicates that the number of edge servers is fixed. Eq. (8) states that each base station is served by only one server, and all base stations are ensured coverage. Eq. (9) signifies that no server is deployed at base station bs_j. Eq. (10) represents the range of values of α_i and β_{ij}.

4. Edge server placement method

The server placement problem aims to identify the optimal configuration of servers based on the load distribution of base stations within the system, ensuring equitable server utilization and minimal latency. This aligns with the interactivity framework between the agent and the environment in a Markov Decision Process (MDP). In this section, we formulate the server placement problem as a DRL problem. We first introduce the environment, state representations, and action space involved in this problem and then delineate the overall workflow of the APD algorithm.

4.1. MDP model

State:

The problem addressed in this paper is to find m locations to place edge servers among n base stations. Each placement strategy constitutes a solution to this problem, where different schemes result in varying server workloads and system delay. Therefore, the state space comprises the set of all server locations. The initial state involves randomly selecting m locations from the n base station positions, formalized as follows:

$$s = [(lat_1, lon_1), (lat_2, lon_2), \ldots, (lat_m, lon_m)]. \tag{11}$$

Action:

The initial state determines the positions of m edge servers, but this placement scheme is not optimal at this stage and requires relocating the edge servers to achieve the optimal solution. If the action space is modeled as all possible locations where edge servers can be moved, it would be too large, increasing computational complexity. To standardize the action space, we move only one edge server to a reasonable location at a time. Let $A_1 = (0, 1, 2, \ldots, m)$ represent the action space for selecting the candidate edge servers, where m denotes the number of servers. Let $A_2 = (0, 1, 2, 3)$ represent the action space for directions, where 0 represents moving east, 1 represents moving west, 2 represents moving south, and 3 represents moving north. We combine these two actions into one, i.e., the Cartesian product of A_1 and A_2, resulting in an action space $A = (0, 1, 2, \ldots, 4m - 1)$. When the network selects the current action a from the action space, we compute the server that needs to be moved, denoted as es_j, using Eq. (12). We then determine the direction in which server es_j should move using Eq. (13). It is important to note that we determine movement directions by comparing latitude and longitude values. For example, if the longitude of base station bs_i is greater than the longitude of server es_j, regardless of the latitude of base station bs_i, we consider base station bs_i to be east of server es_j. Additionally, if there are no available movable positions in the determined direction, no action is executed.

$$es_j = \frac{a}{\mathcal{X}}, \tag{12}$$

$$o = a\%\mathcal{X}. \tag{13}$$

where $\mathcal{X} = 4$ indicates the four directions, the sign '%' is used as a modulus operator.

While determining the direction o for the edge server es_j to move, there may be more than one movable position along direction o. Therefore, we propose a workload-based location selection rule (LSD). Firstly, we compute all feasible positions along direction o for the edge server es_j based on (es_j, o), and then we read the workload data of the movable positions' base stations to construct the workload matrix E. Simultaneously, we establish a binary feasibility position matrix H, with the same dimensions as the workload matrix E. If a position already has a server, it is 0; otherwise, it is 1. The Hadamard product of the workload matrix E and the movable position matrix H represents the potential moving domain for es_j. Finally, we sort the non-zero elements in the moving space to obtain the position with the highest workload as the next move for the edge server es_j.

For example, if the edge server es_j has five base stations in the east direction as candidate positions, and the workload matrix E obtained by reading the base station workload is $[4, 7, 10, 5, 2]$. Based on the current positions of all servers, the movable position matrix H for these five base stations is $[1, 1, 0, 1, 0]$. The Hadamard product of the two matrices is $[4, 7, 0, 5, 0]$. After sorting the non-zero elements, we obtain the base station in the east direction with a workload of 4 units as the next move for the edge server es_j.

Reward:

We compute two metrics, the average delay D and the standard deviation of edge server workload W_{sd}, to measure the performance of server placement and calculate reward r. A smaller average delay and a smaller standard deviation of edge server workload indicate better server placement performance. Each movement of an edge server to a new position represents a new placement scenario. Therefore, after each action execution, we store the current average delay D into a delay pool B_l and the current scenario's edge server workload standard deviation W_{sd} into a workload pool B_w. We then utilize the min–max normalization method to standardize the average delay and workload standard deviation to the same order of magnitude. For example, let X represent the normalized sequence of the workload pool, where x is an element in the workload pool B_w and $\overline{x}$ is the normalized value of x.

$$X = \{\overline{x} | \overline{x} = \frac{x - min(B_w)}{max(B_w) - min(B_w)}, \forall x \in B_w\}, \tag{14}$$

$$r = -(\mu\overline{D} + \eta\overline{W_{sd}}), \tag{15}$$

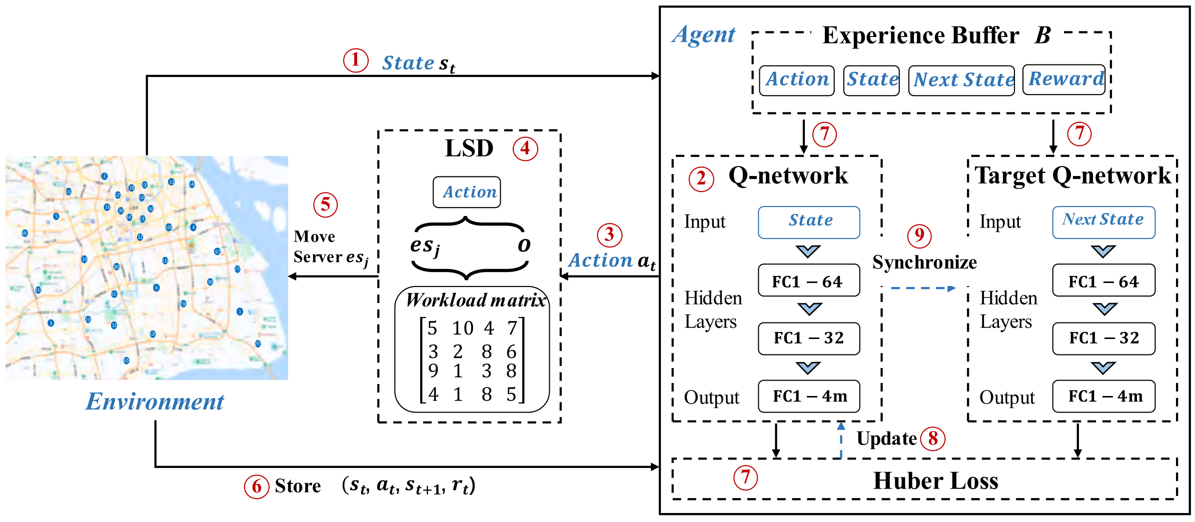


Fig. 2. The diagram of the APD algorithm.

where $\overline{D}$ and $\overline{W_{sd}}$ represent the average delay and the normalized values of the server workload standard deviation, respectively. μ and η denote the coefficients of the two factors, which sum up to 1. Since this work does not focus on analyzing the weights of different objectives, here μ and η are each set to 0.5.

4.2. APD algorithm

In this section, we provide a comprehensive description of the APD algorithm, as illustrated in Fig. 2. The specific procedural steps are outlined in Algorithm 1.

Firstly, we initialize the main Q-network, target Q-network, and necessary parameters. Due to the large state and action spaces in the problem addressed in this paper, we utilize a Q-network for decision-making to mitigate the curse of dimensionality. Neural networks are used to estimate Q-values, and the action with the maximum Q-value is selected to obtain the optimal edge server layout. The Q-network structure of our main Q-network is described in the Q-network section of Fig. 2. The dimensions of the states are consistent. In our experiments, the input layer has a specific dimension of m, which equals the number of servers, with each dimension representing the position of an edge server. The two hidden layers have 64 and 32 neurons, respectively, both using ReLU activation functions. The output layer consists of $4m$ neurons, representing the action space A. Finally, the Q-values for each action are computed, and the argmax function is used to map the Q-values to the action a. The target Q-network mainly calculates the loss, updates the main Q-network, and improves the convergence of the algorithm. The parameters of the main Q-network are synchronized with those of the target Q-network every C steps.

Then, from the n positions of the base stations, m positions are randomly selected to place the edge servers, and the positions of the m servers, $s_1 = [(lat_1, lon_1), (lat_2, lon_2), \dots, (lat_m, lon_m)]$, are initialized as the initial state. At the t-th iteration, the agent selects action a_t based on the ε-greedy policy, as shown in lines 6–10. From a_t, the server es_j to be relocated and its movement direction o are determined, and then the server's movement position pos is calculated using the LSD method. The server is then moved, and the reward r_t, and next state s_{t+1} are observed. Next, we employ the experience replay mechanism of the DQN algorithm, sampling a batch from the experience pool for training, as shown in lines 14–20. The loss is then computed using the Huber loss function, and the gradient descent method is utilized to update the main Q-network. The specific method for updating ε is as follows: if ε is greater than ε_{min}, $\varepsilon = 0.99\varepsilon$. If the current step is a multiple of C, the parameters of the main Q-network are synchronized with those of the target Q-network.

Algorithm 1 APD algorithm

```
1:  Initialize $Q(s;a;\theta)$, $\overline{Q}(s;a;\overline{\theta})$.
2:  Initialize the experience replay buffer $B$ and the policy parameter $\varepsilon$.
3:  Randomly select $m$ locations to place edge servers.
4:  t=1. Initialize the state $s_1 = [(lat_1, lon_1), (lat_2, lon_2), ..., (lat_m, lon_m)]$ .
5:  While $t \le T$ do
6:    Generate a random number $z \in [0, 1]$.
7:    If $z \le \varepsilon$ then
8:      Randomly select an action $a_t$.
9:    Else
10:     Select $a_t = argmax_{a\in A} Q(s_t; a; \theta)$.
11:   Calculate $a_t \to \{es_j, pos\}$ by LSD method.
12:   Move the server $es_j$ and observe $r_t$, $s_{t+1}$.
13:   Store the experience $(s_t, a_t, r_t, s_{t+1})$ in experience replay buffer $B$.
14:   Sample a mini batch from experience replay buffer $B$.
15:   For each sampled transition do
16:     If $t + 1 = T$ then
17:       $y = r_t$
18:     Else
19:       $y = r_t + \gamma\overline{Q}(s_{t+1}, Q(s_{t+1}, a, \theta), \overline{\theta})$
20:     End For
21:   $\delta = 1$. Calculate $Loss = \begin{cases} \frac{1}{2}(y - Q(s_t, a_t, \theta))^2, & |y - Q(s_t, a_t, \theta)| \le \delta \\ \delta \cdot (|y - Q(s_t, a_t, \theta)| - \frac{1}{2}\delta), & \text{otherwise} \end{cases}$
22:   Update $Q(s;a;\theta)$ by Adam optimizer and Update $\varepsilon$.
23:   If $t\%C = 0$ then
24:     Synchronize $\overline{Q}(s;a;\overline{\theta}) = Q(s;a;\theta)$.
25:   $t = t + 1$.
26: End While
```

4.3. Complexity analysis

The time complexity of the APD algorithm primarily encompasses two facets: the training complexity of the Q-network during the training phase, and the computational complexity of the Q-network during the testing phase. We will first analyze the time complexity of the training process. As illustrated in Fig. 2, the Q-network comprises an input, output, and three fully connected layers. The number of neurons in

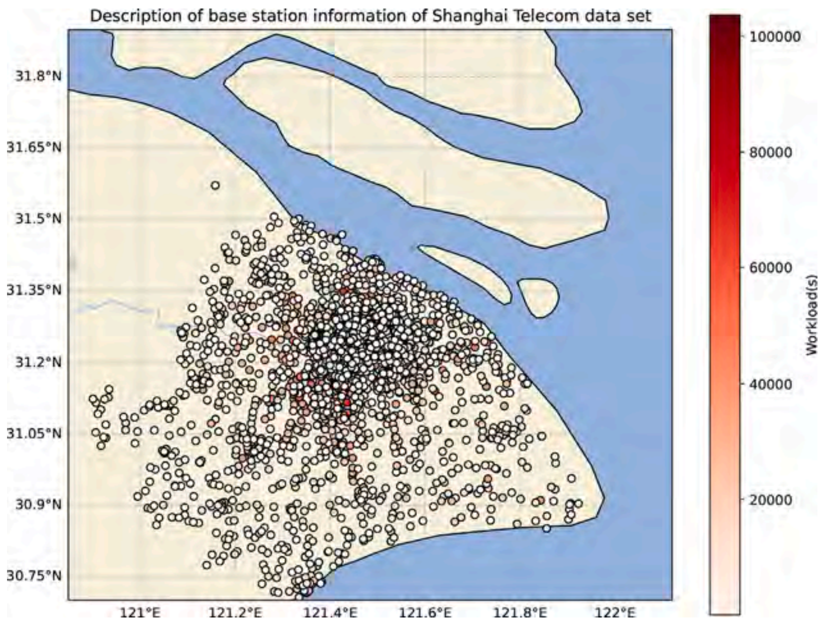


Fig. 3. Description of base station information of Shanghai Telecom data set.

the input layer is m, where m denotes the number of servers, and the number of neurons in the output layer is $4 \times m$. Let f_i denote the number of neurons in the ith layer of the Q-network; thus, $f_1 = m$, $f_2 = 64$, $f_3 = 32$, and $f_4 = 4 \times m$. Let T represent the maximum number of iterations. The time complexity for the training process is $O(m \times T \times \sum_{i=1}^{4} f_{i-1} \times f_i)$.

When training is completed, the trained Q-network makes scheduling decisions for each task at every time step during the testing phase. Therefore, the time complexity of the APD testing phase is $O(m \times \sum_{i=1}^{4} f_{i-1} \times f_i)$.

5. Experimental results and discussions

This section presents the experimental results. We first compare the training results with different core parameters. Then, we conduct ablation experiments using DQN-ESPA as the baseline algorithm. Finally, we test the performance of several comparative algorithms. All experimental results were obtained by training with randomly generated seeds and averaging the results.

5.1. Experimental environments

In this section, we outline the experimental setup and results. We use the standard deviation of workload balance and system average delay as evaluation metrics to verify the performance of the APD algorithm and compare it with several advanced algorithms. All experiments are implemented in Python 3.7 and executed on a computer with an Intel i7-12700KF CPU and 32 GB of memory.

The dataset used in the experiments is sourced from Shanghai Telecom.[1] It comprises over 7.2 million records of 9481 mobile phones accessing the internet through base stations over a period of 6 months. The dataset provides detailed geographic locations of the base stations, from which we selected valid data from 3000 base stations. Fig. 3 shows the base station location information and load information of the Shanghai TV data set. The darker the color, the higher the complexity. All other parameters are presented in Table 1.

[1] The dataset set can be accessed by http://sguangwang.com/TelecomDataset.html.

Table 1
Parameters.

Definition	Value
The number of base stations n	3000
The number of servers m	[100,200,300,400,500]
The number of layers in Q-network	4
The sample size b	32
The memory size M	10 000
The learning rate l_r	0.001
The discount rate γ	0.9
The initial value of ε	1
The minimum value of ε	0.01
The iteration coefficient of ε	0.99
The synchronization period of the target network C	100

5.2. Baselines

Before discussing the experimental results, we provide a brief overview of several comparative algorithms.

K-means (Lhderanta et al., 2021): A classical unsupervised method that categorizes base stations into m clusters based on their distance similarities, with each cluster representing the coverage area of an edge server. Here, m denotes the number of edge servers.

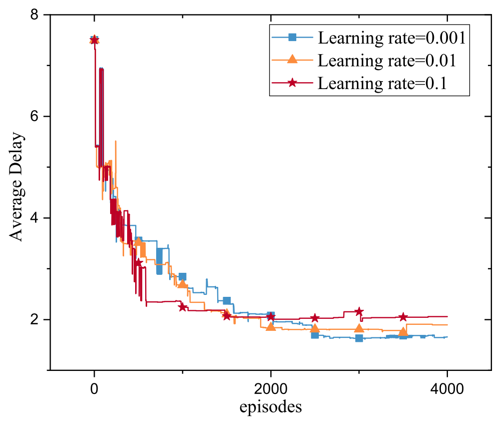
Top-k (Wang et al., 2019): This algorithm ranks the workload of each base station and selects the position with the highest workload as the server's location. Then, it determines the coverage of this server based on distance and removes the covered base stations from the sorted queue. This process is repeated until m server positions are found.

Random (Wang et al., 2019): This method randomly selects m positions from n available base station locations to place servers.

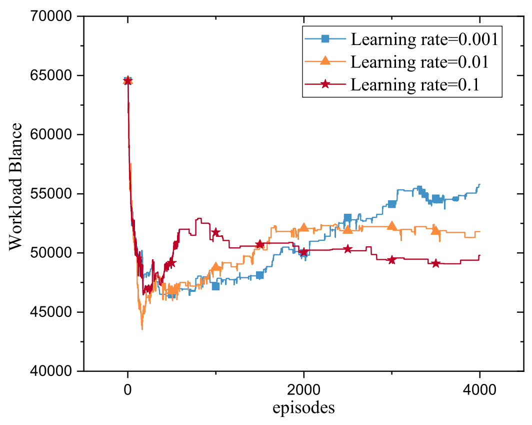
QMC (Mazloomi et al., 2023): This method discretizes the continuous state space, allowing the algorithm to learn and make decisions effectively within a finite state space.

TDMC (Mazloomi et al., 2023): This method leverages the concept of Temporal Difference Learning (TD Learning) to update the value estimation of the current state by predicting the difference between future rewards and actual rewards, thereby optimizing the strategy.

DQN-ESPA (Luo et al., 2022): This approach utilizes reinforcement learning, considering only the positions with the smallest changes in latitude and longitude when moving servers.

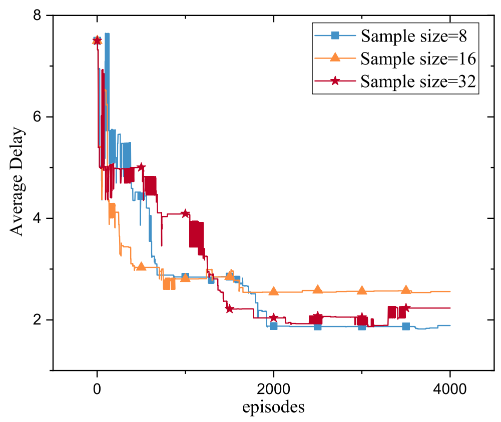


(a) Average delay.

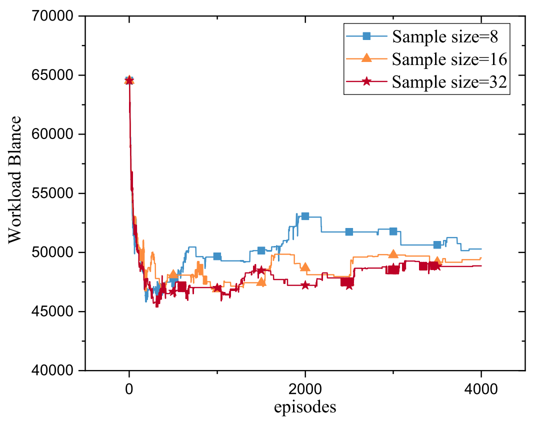


(b) Workload balance.

Fig. 4. Comparison of different learning rates.

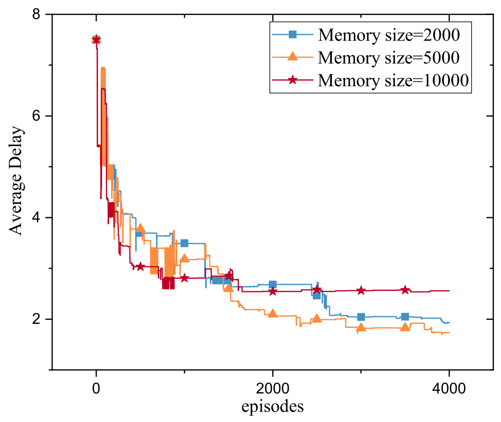


(a) Average delay.

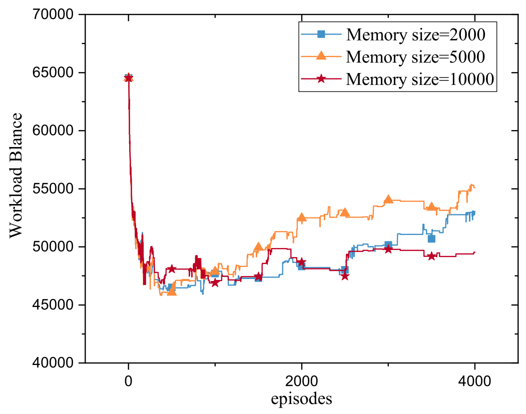


(b) Workload balance.

Fig. 5. Comparison of different sample sizes.



(a) Average delay.



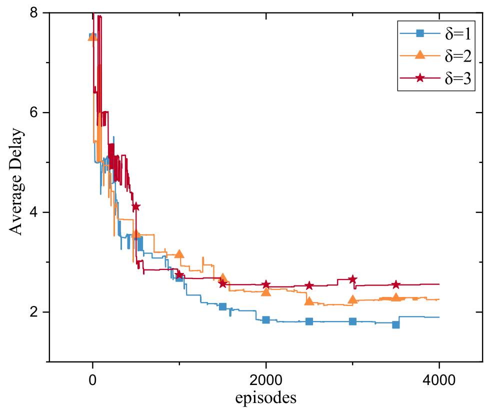
(b) Workload balance.

Fig. 6. Comparison of different memory sizes.

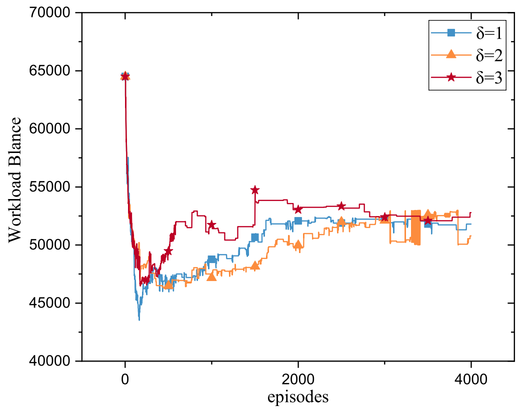
5.3. *Sensitivity analysis*

As shown in Figs. 4–8, to test the performance of the APD algorithm under different parameters, we set up an instance with 300 servers placed among 3000 base stations. We evaluated various configurations, including different learning rates l_r, sample sizes b, memory sizes M, the Huber loss parameter δ, and the target Q-network synchronization period C.

Figs. 4(a) and 4(b) correspond to the average delay and workload balance under different learning rates, respectively. It can be observed that a smaller learning rate leads to slower convergence but also minimizes the average delay, whereas larger learning rates exhibit the opposite behavior. Fig. 4(b) indicates that smaller learning rates result in slower convergence and may lead to local optima without finding the true optimal solution, while larger learning rates, although making

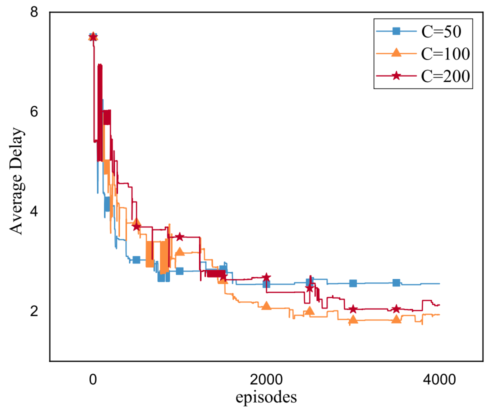


(a) Average delay.

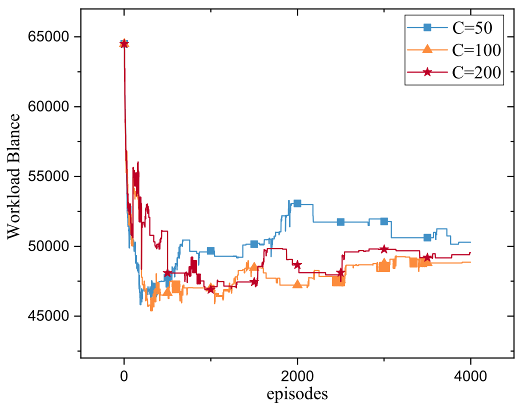


(b) Workload balance.

Fig. 7. Comparison of different δ.



(a) Average delay.



(b) Workload balance.

Fig. 8. Comparison of different C.

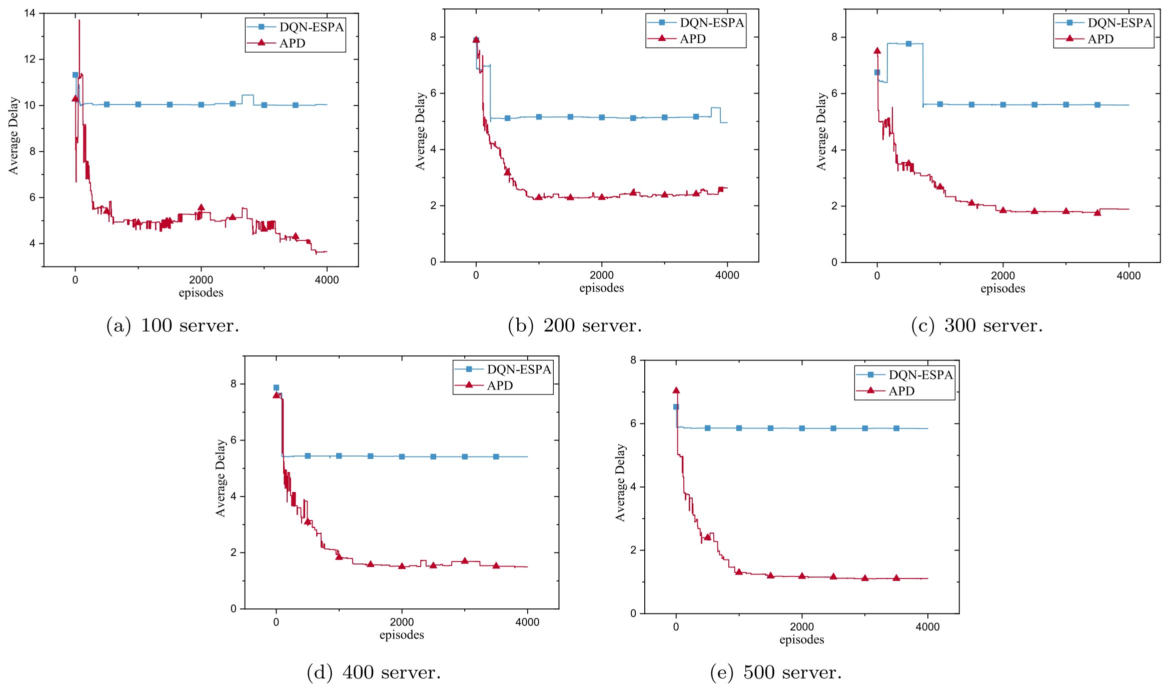
the workload more balanced, perform poorly in terms of average delay. Considering these factors, we adopt $l_r = 0.01$ in subsequent experiments.

Figs. 5(a) and 5(b) correspond to the average delay and workload balance under different sample sizes, respectively. From Fig. 5(a), it can be observed that both a large and a small sample size affect the convergence speed. A large sample size includes redundant data that can influence the agent's decision-making, while a small sample size lacks comprehensive information, similarly affecting the agent's decisions and thus the convergence speed. However, the final average delay converged to a satisfactory level. From Fig. 5(b), it can be seen that a smaller sample size results in a larger and more fluctuating workload balance, whereas a larger sample size leads to a smaller workload balance. Considering all factors into account, we adopted a sample size of 32 for the subsequent experiments.

Figs. 6(a) and 6(b) correspond to the average delay and workload balance under different sizes of experience pools, respectively. From Fig. 6(a), it can be seen that a larger experience pool, while speeding up convergence, causes the algorithm's average delay to get trapped in a local optimum. In contrast, a smaller experience pool, although resulting in slower convergence, achieves a lower average delay. Fig. 6(b) shows that with a smaller experience pool, the algorithm performs poorly in terms of workload. This is because the two optimization objectives — average delay and workload balance — are conflicting. With a smaller experience pool, it is difficult to comprehensively observe past training experiences, leading to the optimization of one objective at the expense of the other. Although a better average delay is achieved, it results in severe load imbalance. When the experience pool size is set to 10,000, it slightly sacrifices average delay but achieves more balanced workloads. Taking all factors into account, we adopted a memory size of 10,000 for the subsequent experiments.

Figs. 7(a) and 7(b) correspond to the average latency and workload balancing performance for different parameter values of δ, respectively. It is evident that varying d has little impact on load balancing performance, but the average latency is minimized when $\delta = 1$. This is because, as d increases, the characteristics of the Huber loss cause the loss to be calculated using Mean Squared Error (MSE), which amplifies the impact of outliers. Therefore, we opted to use $\delta = 1$ in subsequent experiments.

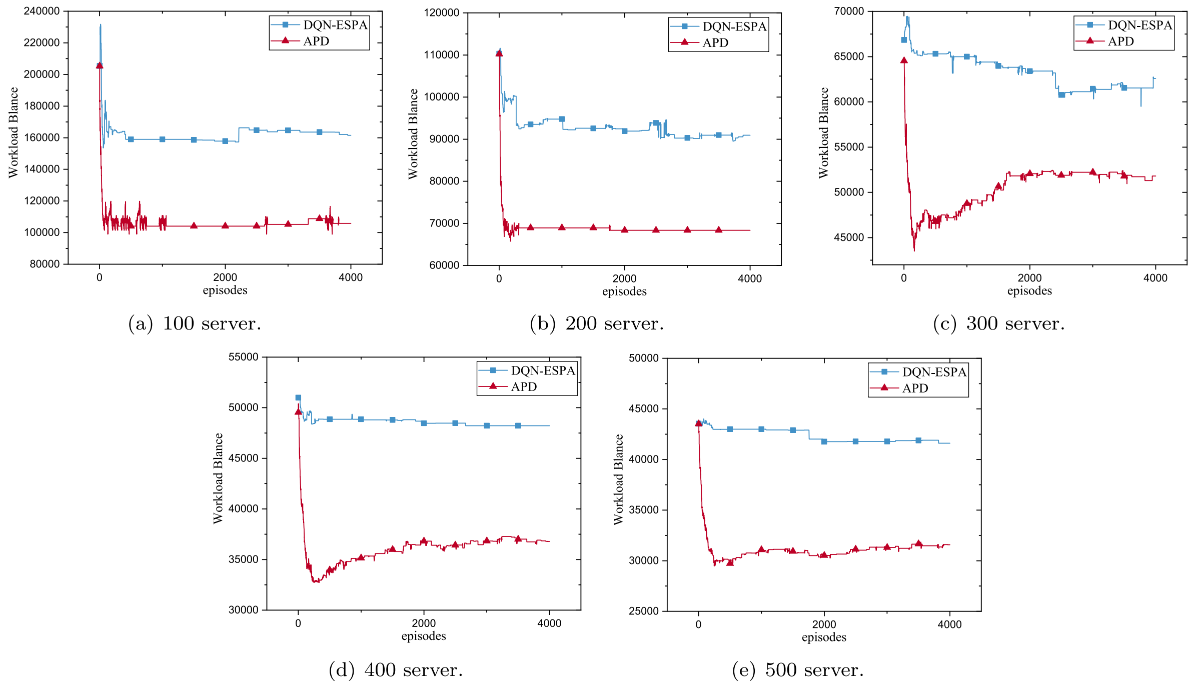
Figs. 8(a) and 8(b) correspond to the average latency and workload balancing performance for different synchronization periods C, respectively. It is observed that with $C = 50$, the convergence speed is relatively fast, but the average latency and workload balancing performance are poor. When $C = 100$, although the convergence speed is slower, the average latency and workload balancing performance are the best. When $C = 200$, while average latency and workload balancing performance improve, the convergence speed is the slowest. A more minor synchronization period results in less knowledge being learned in each cycle, leading to insufficient comprehensiveness. Conversely, a larger learning period introduces excessive redundant information, causing slower convergence and deteriorating decision-making performance. Therefore, we opted to use $C = 100$ in subsequent experiments.



(a) 100 server. (b) 200 server. (c) 300 server.

(d) 400 server. (e) 500 server.

Fig. 9. Comparison of average delay.



(a) 100 server. (b) 200 server. (c) 300 server.

(d) 400 server. (e) 500 server.

Fig. 10. Comparison of workload balance.

5.4. Ablation experiments

This subsection presents ablation experiments to validate the effectiveness of the proposed LSD method. Since this work is based on DQN-ESPA, we use DQN-ESPA as the baseline for our ablation experiments. Both algorithms are implemented using the parameters outlined in Table 1. Figs. 9 and 10 show the comparison of the average delay and workload balance between APD and DQN-ESPA, with 100, 200, 300, 400, and 500 servers placed across 3000 base stations, respectively. It is evident that the APD algorithm outperforms DQN-ESPA in both average delay and workload balance across all instances. This is because DQN-ESPA only considers the position with the least latitude and longitude movement when relocating servers, neglecting the impact on workload. Our proposed LSD method not only selects the servers that need to be moved and their direction of movement but also identifies the optimal position in that direction to further reduce average delay and workload balance.

Additionally, we find that both the APD and DQN-ESPA algorithms reduce the average delay and workload balance standard deviation across all instances. When the number of servers is small, the performance improvement of the APD algorithm over the DQN-ESPA algorithm is less pronounced. However, as the number of servers increases, the optimization performance of the DQN-ESPA algorithm for both metrics significantly declines, while the APD algorithm maintains strong performance. This is because, with fewer servers, the DQN-ESPA algorithm has fewer positions to move to in different directions when making decisions, making it more likely to choose the position with the least latitude and longitude movement to balance the server load. As the number of servers increases, the number of potential positions in different directions also increases, greatly impacting the decision accuracy of the DQN-ESPA algorithm. The LSD method proposed by the APD algorithm is not influenced by the number of servers; regardless of the instance or the number of possible positions in different directions during decision-making, it can consistently select strategies that reduce both average delay and workload balance standard deviation.

5.5. Comparison of different servers

Table 2 describes the system delay performance of different algorithms when placing different numbers of servers, measured by the average delay of servers as shown in Eq. (4). Table 3 describes the workload balance performance of different algorithms when placing different numbers of servers, measured by the variance of server workload as shown in Eq. (5). The data in boldface indicates the top two rankings. The results show that the K-means algorithm has the lowest average delay because it considers the similarity of distances between base stations when placing servers. However, this also neglects workload balancing, resulting in the largest variance in server workload for this algorithm. In most instances, the Top-k algorithm achieves more balanced server workloads because it sorts server workloads to balance them, but it overlooks the spatial relationships of servers, leading to a higher average delay. Although the Random algorithm makes decisions easily, its randomness prevents it from accurately finding better solutions. Therefore, in several tests with different random seeds, both the average delay and workload balance performance of this algorithm are poor. The QMC algorithm quantizes the state space to simplify the problem, while the TDMC utilizes temporal difference learning to dynamically adjust the value estimates of states. Both of them enhance the performance of the algorithms, but they do not consider the load on the base station. DQN-ESPA outperforms K-means, Top-k, Random, QMC and TDMC overall, especially with fewer servers. This is because DQN-ESPA considers moving servers only when there are minimal changes in latitude and longitude. With fewer servers, there are fewer servers in the moving direction, and the probability of the nearest server being the appropriate one is higher. However, as the number of servers increases, this strategy cannot consider the workload of servers in the moving direction, leading to a decrease in performance. The APD algorithm not only considers moving servers in four directions but also utilizes the LSD algorithm to thoroughly evaluate the workload of servers in the moving direction. Therefore, it exhibits better performance even as the number of servers increases.

5.6. Comparison of different base stations

To verify the robustness of the APD algorithm, we tested instances where the number of servers was fixed, but the number of base stations varied. The results are shown in Tables 4 to 7, with the data in boldface indicating the top two performances in each instance. From these tables, it is evident that, with a constant number of servers, the average latency and server load variance also increase as the number of base stations increases. Tables 4 and 5 respectively, present the average latency and load balancing performance of placing 100 servers in instances with varying numbers of base stations. Tables 6 and 7 respectively show the average latency and load balancing performance of placing 300 servers in instances with varying numbers of base stations.

Table 2
Average delay of placing different numbers of servers with 3000 base stations.

m	K-means	Top-k	Random	QMC	TDMC	DQN-ESPA	APD
100	**2.32**	12.24	11.69	11.06	10.42	10.04	**3.66**
200	**1.38**	6.78	8.69	7.23	6.33	4.95	**2.28**
300	**1.04**	5.67	8.92	7.15	6.01	5.59	**1.89**
400	**0.84**	4.45	11.65	6.75	5.83	5.41	**1.44**
500	**0.70**	3.96	8.41	6.13	5.42	5.84	**1.10**

These four tables show that the K-means algorithm improves average latency performance at the expense of load balancing performance, regardless of the instance. This also confirms our previous discussion that this algorithm focuses on the similarity of distances between base stations when placing servers, ignoring the server load. In contrast, the Top-k algorithm sacrifices average latency performance to improve load balancing performance because it focuses on server load, ignoring server distances. Although the Random algorithm performs well in load balancing when placing 100 servers among 2000 base stations, it performs poorly in both metrics in most instances. The DQN-ESPA algorithm's mobile server strategy outperforms K-means, Top-k, and random algorithms. The QMC and TDMC algorithms perform reasonably with fewer base stations, but as the number of base stations increases, their overall performance surpasses that of DQN-ESPA. Additionally, the temporal difference learning concept introduced by TDMC adjusts the value of the current state at each step based on the value estimates of the current and next states, optimizing the policy. Although the APD algorithm does not achieve the best performance in every instance, it consistently ranks in the top two for each metric across different instances, indicating the best overall performance of the APD algorithm.

5.7. Comprehensive evaluation

Furthermore, we introduce the Comprehensive Evaluation Index (CEI) to compare the overall performance of various algorithms (Luo et al., 2022). A lower CEI value signifies better overall performance for an algorithm. Due to the magnitude differences between the two evaluation metrics, average delay and load balancing, we use two different methods to standardize these metrics. Since the average delay typically remains below 10, we standardize it using the method outlined in Eq. (16). The load balancing data varies significantly, so we standardize it using the method described in Eq. (14). The comprehensive evaluation index is then obtained using Eq. (17). X represents the set of standardized average delays for each algorithm, $\overline{x}$ represents the standardized average delay value for each algorithm, $\mathcal{D}$ represents the set of average delays for each algorithm, $\overline{D}$ and $\overline{W_{sd}}$ represent the standardized average delay and load balancing standard deviation, respectively, and both μ and η are set to 0.5.

$$X = \{\overline{x}|\overline{x} = log_{10}(x)/log_{10}(max(\mathcal{D})), \forall x \in \mathcal{D}\}, \tag{16}$$

$$CEI = \mu\overline{D} + \eta\overline{W_{sd}}, \tag{17}$$

The results of the comprehensive evaluation are shown in Fig. 11. The characteristics of the K-means and Top-k algorithms enable them to excel in one specific metric, but their overall CEI is relatively poor. The Random algorithm, which randomly selects servers for placement, has the worst overall CEI. The DQN-ESPA algorithm performs well with a small number of servers, but its overall performance declines as the number of servers increases, further indicating that DQN-ESPA only considers the nearest servers and ignores the impact of load. The QMC and TDMC algorithms perform slightly worse than DQN-ESPA, although they map state–action pairs to a Q-table, this nonlinear approximation in larger spaces can lead to poorer results. The CEI of the APD algorithm consistently maintains the lowest in most cases.

Table 3
Workload balance of placing different numbers of servers with 3000 base stations.

m	K-means	Top-k	Random	QMC	TDMC	DQN-ESPA	APD
100	231 507.04	171 129.49	194 943.24	172 238.21	166 723.36	**161 508.46**	**105 801.9**
200	103 164.61	77 735.22	95 904.44	93 678.75	92 852.76	**90 922.88**	**68 352.64**
300	65 563.04	**44 420.76**	65 659.84	64 984.18	62 372.25	61 538.99	**51 805.95**
400	51 110.23	**28 897.75**	50 707.16	49 218.52	47 218.98	48 216.18	**35765.9**
500	41 279.23	**21 390.24**	40 566.48	43 231.54	42 218.38	41 609.61	**32 079.09**

Table 4
Average delay of placing 100 servers with different numbers of base stations.

n	K-means	Top-k	Random	QMC	TDMC	DQN-ESPA	APD
500	**1.07**	16.20	16.25	10.25	9.24	10.74	**3.93**
1000	**1.71**	8.86	13.99	9.25	8.23	9.96	**4.48**
1500	**1.82**	10.88	13.15	10.43	9.35	10.16	**3.80**
2000	**2.31**	13.54	14.28	9.23	8.18	8.72	**4.83**
2500	**2.31**	12.71	13.39	8.83	7.24	6.69	**5.94**
3000	**2.32**	12.24	13.03	10.31	9.27	10.04	**1.10**

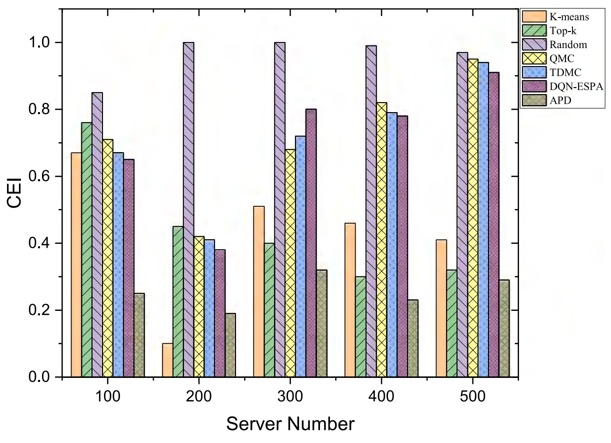


Fig. 11. CEI comparison for different numbers of servers placed at 3000 base stations.

Combined with previous conclusions, the LSD method proposed by the APD algorithm comprehensively considers both average delay and load balancing and effectively mitigates the service.

5.8. Server distribution

Finally, we visualize the server locations and workload information to further compare the performance of several algorithms. Fig. 12 shows the locations and workload information of servers placed among 100 server instances for 3000 base stations. In the figure, dots represent servers, and the darker the color, the higher the server's workload.

From Fig. 12(a), we can see that the servers placed by the K-means algorithm are more evenly distributed, covering all areas. This allows base stations in all areas to connect to the nearest server, resulting in lower average latency for the K-means algorithm. However, this approach causes servers in high workload areas to bear a significant load, leading to poor workload balance. From Fig. 12(b), it can be seen that the Top-k algorithm concentrates servers in high workload areas, achieving better load balancing but also causing higher access latency for base stations in peripheral areas. From Fig. 12(c), it can be seen that the distribution characteristics of servers chosen by the Random algorithm align with those of the base stations, with more servers placed in denser base station areas. However, the randomly selected locations do not effectively reduce average latency and workload balance standard deviation. Fig. 12(d) illustrates that the QMC algorithm can concentrate servers in load-intensive areas to reduce latency; however, this approach compromises load balancing performance. Fig. 12(e) demonstrates that the TDMC algorithm optimizes load balancing performance based on QMC, but the disparity in server load remains significant. Figs. 12(f) and 12(g) demonstrate that the DQN-ESPA and APD algorithms can place more servers in high workload areas to reduce the server workload balance standard deviation and fewer servers in peripheral areas to reduce average latency. Nevertheless, their performance varies. It is evident that the server workloads placed by the DQN-ESPA algorithm are mainly concentrated on a few key servers, while the APD algorithm distributes the servers more evenly in high workload areas, resulting in more balanced server loads.

6. Conclusions

This paper presents an adaptive placement and dynamic optimization (APD) method to effectively address the server placement issue in mobile edge computing systems. Firstly, we model the server placement challenge as a Markov decision process and formalize the state space, action space, and reward function. Then, we propose a workload-based location selection rule to adjust edge servers' positions, significantly reducing the average delay and server workload variance. We use real datasets from Shanghai Telecom to compare the APD algorithm with classic algorithms such as K-means, Top-k, Random, QMC, TDMC and DQN-ESPA. Experimental results demonstrate that the APD algorithm outperforms several baseline algorithms. The APD method proposed in this paper enhances user experience in communication networks while meeting the load-balancing requirements of operator servers. By employing reinforcement learning-based decision-making and the proposed LSD scheme to adjust server locations, the method adapts to dynamically changing network environments. However, our study has certain limitations. In our scenarios, servers are homogeneous, but in practical applications, server capabilities and resources can be tailored to different regional needs. Future research could explore the performance of APD algorithms with heterogeneous servers. Additionally, with the high-density base station layouts anticipated for future 6G networks, exploring base station sleep technologies to improve energy efficiency is essential. Under the context where base stations can enter sleep mode, optimizing server placement could be a potential area for improvement.

CRediT authorship contribution statement

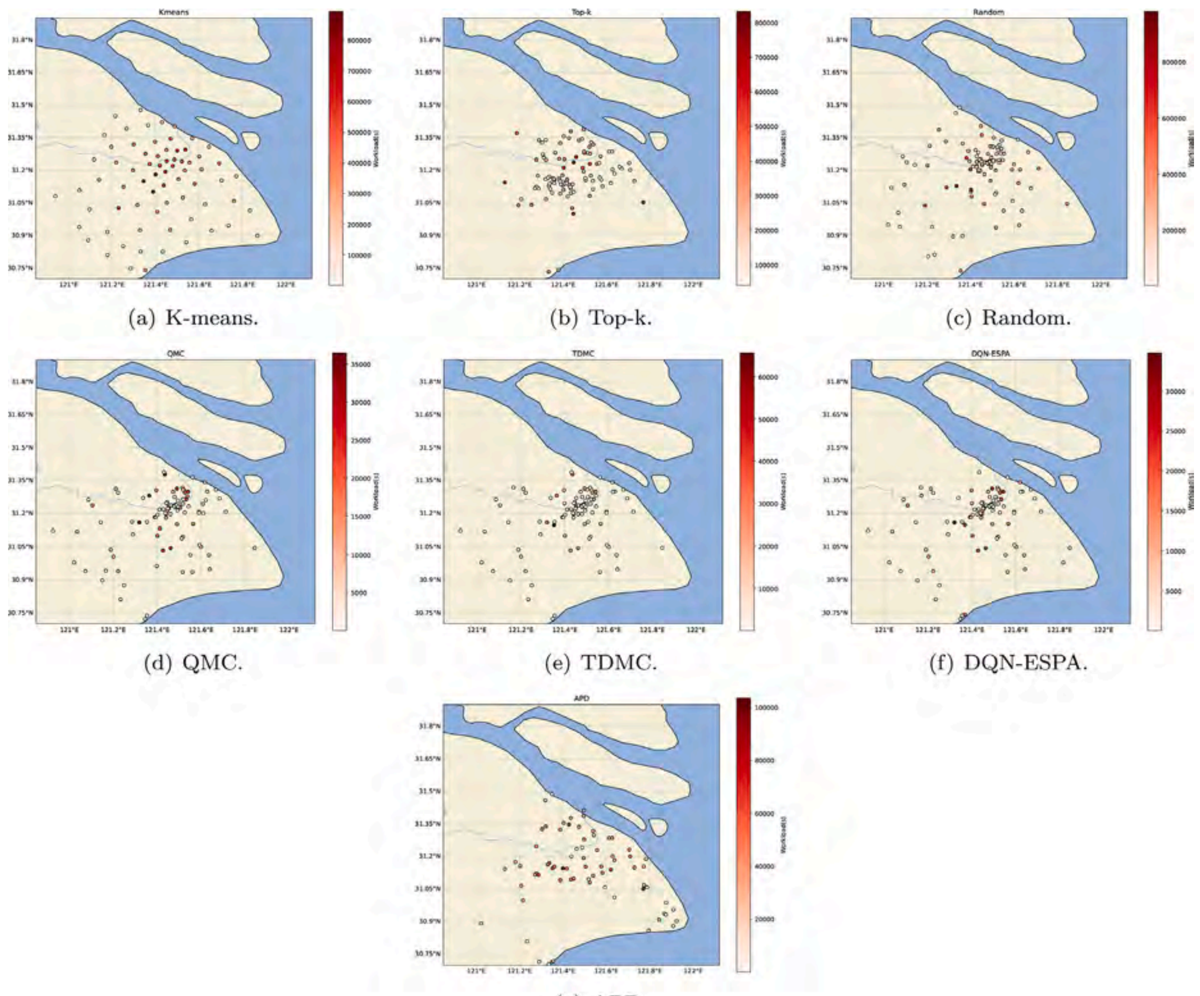
Shihua Li: Writing – review & editing, Resources, Methodology, Conceptualization. **Yanjie Zhou:** Writing – review & editing, Supervision, Project administration, Funding acquisition, Conceptualization. **Bing Zhou:** Writing – review & editing, Project administration, Investigation, Conceptualization. **Zongmin Wang:** Writing – review & editing, Resources, Investigation, Conceptualization.

Declaration of competing interest

The author(s) declared no potential conflicts of interest concerning this article's research, authorship, and/or publication.

Acknowledgments

The work described in this paper was financially supported by the National Key R&D Program of China (No. 2022YFC3803203), National Natural Science Foundation of China (72201252), and Henan



(a) K-means. (b) Top-k. (c) Random.

(d) QMC. (e) TDMC. (f) DQN-ESPA.

(g) APD.

Fig. 12. Distribution of 100 servers.

Table 5
Workload balance of placing 100 servers with different numbers of base stations.

n	K-means	Top-k	Random	QMC	TDMC	DQN-ESPA	APD
500	66 346.03	**34 832.25**	123 386.91	752 458.59	73 328.57	85 547.68	**54 695.12**
1000	105 006.94	**98 643.32**	146 996.29	113 358.24	110 923.38	125 343.51	**103 242.37**
1500	162 346.29	**118 340.78**	151 952.22	147 295.83	140 753.59	148 571.35	**138 193.26**
2000	205 807.21	163 866.80	**160 655.75**	170 284.50	168 673.29	168 048.12	**158 260.58**
2500	212 781.71	170 316.27	162 679.68	168 295.65	159 294.26	162 502.78	**157 420.07**
3000	231 507.04	171 129.49	194 943.24	150 834.67	148 267.85	**161 508.46**	**105 801.9**

Table 6
Average delay of placing 300 servers with different numbers of base stations.

n	K-means	Top-k	Random	QMC	TDMC	DQN-ESPA	APD
500	**0.20**	10.58	12.07	3.16	2.25	3.19	**1.08**
1000	**0.72**	6.40	11.27	4.73	4.38	5.16	**1.39**
1500	**0.83**	5.99	10.68	6.47	7.26	6.28	**2.02**
2000	**1.06**	5.98	10.39	9.02	8.54	9.62	**2.97**
2500	**1.03**	5.60	9.33	7.98	8.51	8.73	**2.94**
3000	**1.04**	5.67	8.92	5.26	4.25	5.59	**1.89**

Table 7
Workload balance of placing 300 servers with different numbers of base stations.

n	K-means	Top-k	Random	QMC	TDMC	DQN-ESPA	APD
500	**20 784.69**	**11 635.96**	43 107.40	36 148.27	306 475.79	316 045.23	25 388.25
1000	39 298.73	**17 003.52**	51 977.36	47 182.49	45 247.22	48 944.25	**35 751.15**
1500	54 452.83	**28 472.85**	55 718.32	53 874.25	52 936.24	50 473.56	**41 801.81**
2000	65 576.59	**42 519.02**	60 722.18	54 290.37	57 826.04	58 839.51	**49 945.91**
2500	64 176.84	**44 222.96**	61 148.65	58 356.93	56 832.14	60 888.16	**50 280.63**
3000	65 563.04	**44 420.76**	65 659.84	62 589.26	58 457.26	61 538.99	**51 805.95**

Zhongyuan Medical Science and Technology Innovation and Development Foundation (24YCG2006).

Data availability

Data will be made available on request.

References

Abbasi, Ahmed Bashir, Hadi, Muhammad Usman, 2024. Optimizing UAV computation offloading via MEC with deep deterministic policy gradient. Trans. Emerg. Telecommun. Technol. 35 (1), 4847. http://dx.doi.org/10.1002/ett.4874.

Balaji, K., Sai Kiran, P., Sunil Kumar, M., 2022. Power aware virtual machine placement in iaas cloud using discrete firefly algorithm. Appl. Nanosci. 13 (12), 2003–2011. http://dx.doi.org/10.1007/s13204-021-02337-x.

Bhatta, Dixit, Mashayekhy, Lena, 2022. A bifactor approximation algorithm for cloudlet placement in edge computing. IEEE Trans. Parallel Distrib. Syst. 33 (8), 1787–1798. http://dx.doi.org/10.1109/TPDS.2021.3126256.

Cao, Kun, Li, Liying, Cui, Yangguang, Wei, Tongquan, Hu, Shiyan, 2021. Exploring placement of heterogeneous edge servers for response time minimization in mobile edge-cloud computing. IEEE Trans. Ind. Inform. 17 (1), 494–503. http://dx.doi.org/10.1109/TII.2020.2975897.

Carvalho, Gonçalo, Cabral, Bruno, Pereira, Vasco, Bernardino, Jorge, 2020. Computation offloading in edge computing environments using artificial intelligence techniques. Eng. Appl. Artif. Intell. 95, 103840. http://dx.doi.org/10.1016/j.engappai.2020.103840.

Chai, ZhengYi, Liu, Xu, Li, Ya-Lun, 2023. A computation offloading algorithm based on multi-objective evolutionary optimization in mobile edge computing. Eng. Appl. Artif. Intell. 121, 105966. http://dx.doi.org/10.1016/j.engappai.2023.105966.

Coito, Tiago, Firme, Bernardo, Martins, Miguel S.E., Costigliola, Andrea, Lucas, Rafael, Figueiredo, João, Vieira, Susana M., Sousa, João M.C., 2022. Integration of industrial IoT architectures for dynamic scheduling. Comput. Ind. Eng. 171, 108387. http://dx.doi.org/10.1016/j.cie.2022.108387.

Cui, Guangming, He, Qiang, Chen, Feifei, Jin, Hai, Yang, Yun, 2022. Trading off between user coverage and network robustness for edge server placement. IEEE Trans. Cloud Comput. 10 (3), 2178–2189. http://dx.doi.org/10.1109/TCC.2020.3008440.

Dai, Penglin, Hu, Kaiwen, Wu, Xiao, Xing, Huanlai, Teng, Fei, Yu, Zhaofei, 2022. A probabilistic approach for cooperative computation offloading in MEC-assisted vehicular networks. IEEE Trans. Intell. Transp. Syst. 23 (2), 899–911. http://dx.doi.org/10.1109/TITS.2020.3017172.

Do, Truong-Xuan, Kim, Younghan, 2018. Latency-aware placement for state management functions in service-based 5G mobile core network. In: 2018 IEEE Seventh International Conference on Communications and Electronics. pp. 102–106.

Ferreira, Luis, Silva, Leopoldo, Morais, Francisco, Martins, Carlos Manuel, Pires, Pedro Miguel, Rodrigues, Helena, Cortez, Paulo, Pilastri, Andre, 2023. International revenue share fraud prediction on the 5G edge using federated learning. Computing 105 (9), 1907–1932. http://dx.doi.org/10.1007/s00607-023-01174-w.

Haiyan, Wang, Dai, Honghua, Zhou, Yanjie, Zhou, Bing, Lu, Peng, Zhang, Hongpo, Wang, Zongmin, 2021. An effective feature extraction method based on GDS for atrial fibrillation detection. J. Biomed. Inform. 119, 103819. http://dx.doi.org/10.1016/j.jbi.2021.103819.

Jasim, Ahmed M., Al-Raweshidy, Hamed, 2024. Optimal intelligent edge-servers placement in the healthcare field. IET Netw. 13 (1), 13–27. http://dx.doi.org/10.1049/ntw2.12097.

Jiang, Xiaohan, Hou, Peng, Zhu, Hongbin, Li, Bo, Wang, Zongshan, Ding, Hongwei, 2023. Dynamic and intelligent edge server placement based on deep reinforcement learning in mobile edge computing. AD Hoc Netw. 145 (1), 135–148. http://dx.doi.org/10.1016/j.adhoc.2023.103172.

Karasakal, O., Karasakal, E., 2023. A partial coverage hierarchical location allocation model for health services. Eur. J. Ind. Eng. 17 (1), 115–147. http://dx.doi.org/10.1504/EJIE.2023.127742.

Kasi, Mumraiz Khan, Abu Ghazalah, Sarah, Akram, Raja Naeem, Sauveron, Damien, 2021. Secure mobile edge server placement using multi-agent reinforcement learning. Electronics 10 (17), 2098–2112. http://dx.doi.org/10.3390/electronics10172098.

Lähderanta, Tero, Leppänen, Teemu, Ruha, Leena, Lovén, Lauri, Harjula, Erkki, Ylianttila, Mika, Riekki, Jukka, Sillanpää, Mikko J., 2021. Edge computing server placement with capacitated location allocation. IEEE J. Parallel Distrib. Comput. 153 (8), 130–149. http://dx.doi.org/10.1016/j.jpdc.2021.03.007.

Lhderanta, T., Leppnen, T., Ruha, L., Lovén, L., Harjula, E., Ylianttila, M., Riekki, J., Sillanp, M.J., 2021. Edge computing server placement with capacitated location allocation. Electronics 153 (1), 130–149. http://dx.doi.org/10.1016/j.jpdc.2021.03.007.

Liao, Zhuofan, Xu, Shuangle, Huang, Jiawei, Wang, Jianxin, 2023. Task migration and resource allocation scheme in IoV with roadside unit. IEEE Trans. Netw. Serv. Manag. 20 (4), 4528–4541. http://dx.doi.org/10.1109/TNSM.2023.3262878.

Lu, Jiawei, Jiang, Jielin, Balasubramanian, Venki, Khosravi, Mohammad R., Xu, Xiaolong, 2022. Deep reinforcement learning-based multi-objective edge server placement in internet of vehicles. Comput. Commun. 187 (1), 172–180. http://dx.doi.org/10.1016/j.comcom.2022.02.011.

Luo, Fei, Zheng, Shuai, Ding, Weichao, Fuentes, Joel, Li, Yong, 2022. An edge server placement method based on reinforcement learning. Electronics 24 (3), 1099–1113. http://dx.doi.org/10.3390/e24030317.

Mazloomi, A., Sami, H., Bentahar, J., Otrok, H., Mourad, A., 2023. Reinforcement learning framework for server placement and workload allocation in multiaccess edge computing. IEEE Internet Things J. 10 (2), 1376–1390. http://dx.doi.org/10.1109/JIOT.2022.3205051.

Moon, Sungwon, Lim, Yujin, 2022. Task migration with partitioning for load balancing in collaborative edge computing. Appl. Sci.-Basel 12 (3), 1168–1183. http://dx.doi.org/10.3390/app12031168.

Ning, Wang, Feng, Panpan, Ge, Zhaoyang, Zhou, Yanjie, Zhou, Bing, Wang, Zongmin, 2023. Adversarial spatiotemporal contrastive learning for electrocardiogram signals. IEEE Trans. Neural Netw. Learn. Syst. 35 (10), 13845–13859. http://dx.doi.org/10.1109/TNNLS.2023.3272153.

Poularakis, Konstantinos, Llorca, Jaime, Tulino, Antonia M., Taylor, Ian, Tassiulas, Leandros, 2020. Service placement and request routing in MEC networks with storage, computation, and communication constraints. IEEE-ACM Trans. Netw. 28 (3), 1047–1060. http://dx.doi.org/10.1109/TNET.2020.2980175.

Song, Xiaona, Peng, Zenglong, Song, Shuai, Stojanovic, Vladimir, 2023. Improved dynamic event triggered security control for T–s fuzzy LPV-pde systems via pointwise measurements and point control. Fuzzy Syst. 25, 3177–3192. http://dx.doi.org/10.1007/s40815-023-01563-5.

Song, Xiaona, Peng, Zenglong, Song, Shuai, Stojanovic, Vladimir, 2024. Anti-disturbance state estimation for PDT-switched RDNNs utilizing time-sampling and space-splitting measurements. Commun. Nonlinear Sci. Numer. Simul. 132, 107945. http://dx.doi.org/10.1016/j.cnsns.2024.107945.

Sun, Yu, He, Qijie, 2023. Joint task offloading and resource allocation for multi-user and multi-server MEC networks: A deep reinforcement learning approach with multi-branch architecture. Eng. Appl. Artif. Intell. 126, http://dx.doi.org/10.1016/j.engappai.2023.106790.

Wang, Xiong, Yang, Zhijun, Ding, Hongwei, Guan, Zheng, 2023a. Analysis and prediction of UAV-assisted mobile edge computing systems. Math. Biosci. Eng. 20 (12), 21267–21291. http://dx.doi.org/10.3934/mbe.2023941.

Wang, Shangguang, Zhao, Yali, Xu, Jinlinag, Yuan, Jie, Hsu, Ching-Hsien, 2019. Edge server placement in mobile edge computing. J. Parallel Distrib. Comput. 127 (6), 160–168. http://dx.doi.org/10.1016/j.jpdc.2018.06.008.

Wang, Rui, Zhuang, Zhihe, Tao, Hongfeng, Paszke, Wojciech, Stojanovic, Vladimir, 2023b. Q-learning based fault estimation and fault tolerant iterative learning control for MIMO systems. ISA Trans. 142 (3), 123–135. http://dx.doi.org/10.1016/j.isatra.2023.07.043.

Xu, Xiaolong, Shen, Bowen, Yin, Xiaochun, Khosravi, Mohammad R., Wu, Huaming, Qi, Lianyong, Wan, Shaohua, 2021. Edge server quantification and placement for offloading social media services in industrial cognitive iov. IEEE Trans. Ind. Inform. 17 (4), 2910–2918. http://dx.doi.org/10.1109/TII.2020.2987994.

Xue, Zheng, Liu, Chang, Liao, Canliang, Han, Guojun, Sheng, Zhengguo, 2023. Joint service caching and computation offloading scheme based on deep reinforcement learning in vehicular edge computing systems. IEEE Trans. Veh. Technol. 72 (5), 6709–6722. http://dx.doi.org/10.1109/TVT.2023.3234336.

Yuan, Xiaoqun, Sun, Mengting, Lou, Wenjing, 2022. A dynamic deep-learning-based virtual edge node placement scheme for edge cloud systems in mobile environment. IEEE Trans. Cloud Comput. 10 (2), 1317–1328. http://dx.doi.org/10.1109/TCC.2020.2974948.

Zhang, Xinglin, Li, Zhenjiang, Lai, Chang, Zhang, Junna, 2022. Joint edge server placement and service placement in mobile edge computing. IEEE Internet Things J. 9 (13), 11261–11274. http://dx.doi.org/10.1109/JIOT.2021.3125957.

Zhang, Z., Zhang, K., Xie, X., Stojanovic, V., 2024. Adp-based prescribed time control for nonlinear time-varying delay systems with uncertain parameters. IEEE Trans. Autom. Sci. Eng. 11, http://dx.doi.org/10.1109/TASE.2024.3389020.

Zhao, Xingbing, Zeng, Yu, Ding, Hongwei, Li, Bo, Yang, Zhijun, 2021. Optimize the placement of edge server between workload balancing and system delay in smart city. J. Parallel Distrib. Comput. 127 (2), 160–168. http://dx.doi.org/10.1007/s12083-021-01208-0.

Zhou, Y., Lee, G., 2020. A bi-objective medical relief shelter location problem considering coverage ratios. Int. J. Ind. Eng. Theory Appl. Pract. 27 (6), 971–988. http://dx.doi.org/10.23055/ijietap.2020.27.6.7603.

Zhu, Qijun, Wang, Sichen, Huang, Hualong, Lei, Yuchuan, Zhan, Wenhan, Duan, Hancong, 2023. Deep-reinforcement-learning-based service placement for video analysis in edge computing. In: 2023 8th International Conference on Cloud Computing and Big Data Analytics. Vol. 35, pp. 5–359.