Latency Minimization for Proximity Detection in Road Networks based on Mobile Edge Computing

Yunlong Song, Yaqiong Liu*, *Member, IEEE*, and Guochu Shou

***Abstract*—In recent years, with the development of 5G and artificial intelligence, autonomous driving technology has made some breakthroughs. In the road networks, mobile users need to obtain the proximity relationship with other users in a real-time manner, which is referred to as the problem of proximity detection that has significant meaning to autonomous driving. However, in the case of limited computing resources and storage resources with real-time changes of road network status, how to calculate and update the proximity relationship between users in a short time becomes a difficult task. Therefore, in this paper, we first propose a computation offloading scheme for proximity detection in the dynamic road network based on mobile edge computing (MEC), and formulate the latency optimization problem for proximity detection in the dynamic road network. Then we use SLSQP algorithm to solve the latency optimization problem and obtain the optimal total latency. In addition, we also use the reinforcement learning approach, i.e., the DDPG algorithm to address the latency optimization problem. The simulation results prove that compared with the SLSQP algorithm, the DDPG algorithm can effectively and efficiently reduce the computational time of the latency each time by continuously adjusting the task allocation strategy, and the optimization time of the DDPG algorithm is two orders of magnitude less than the SLSQP algorithm.**

***Index Terms*—proximity detection, communication latency, mobile edge computing, convex optimization, deep reinforcement learning**

I. Introduction

In recent years, autonomous driving technology has attracted more and more researchers' attention. As a scenario of autonomous driving, the road network continuously and dynamically changes its state with time. In the road network, detecting the proximity relationship between mobile users (clients) accurately and efficiently is called the proximity detection, which finds applications in autonomous driving. To ensure the safety of users in the road network, each user should perform proximity detection in real time.

Under the traditional architecture of proximity detection based on Client-Server (C/S) [2], each client sends a location update message to the same central server, and the central server can send two types of messages to the client: an inquiry message and a notification message. The inquiry message and the notification message are used to inquire users their information and notify which pairs of users have a proximity relationship. Since each client must communicate with a central server that is far away from them, a large communication latency is caused. At the same time, because the central server is responsible for calculating the proximity detection tasks of all users, it suffers a great computing pressure. In the architecture based on Peer-to-Peer (P2P) [3] proximity detection, mobile users communicate with each other. Adopting the P2P communication mode can make a user know quickly about which other users are close to them. However, in the P2P architecture, mobile users communicate with each other in pairs, which incurs a high communication cost. Since each user has to calculate which of other users is close to it, it causes repeated and redundant calculations. Moreover, the P2P architecture cannot detect, from a global perspective, which users among all users given in the road network have a proximity relationship.

In addition, both of the above two architectures use the physical distance, i.e., Euclidean distance to measure the distance between users. Assuming that two mobile users are facing each other on two parallel roads, they will not collide when they are physically close; however, when the two users are facing each other on the same road at a great speed, then a collision may occur within a short time. Therefore, in the proximity detection task, the Euclidean distance does not fully reflect the proximity between users. Therefore, we believe that time distance is more suitable for measuring the proximity of users than the Euclidean distance. Time distance refers to how much time needed for the two mobile users to meet each other.

Compared to the abovementioned two architectures, the mobile edge computing (MEC) architecture is more efficient in handling communication latency. MEC was firstly proposed by ETSI, an emerging organization jointly supported by Nokia Networks, Vodafone, IBM, Intel, NTT DoCoMo, and Huawei [4-8]. In the MEC architecture, all edge servers are distributed in the edge clouds closer to the user. MEC has the characteristics of high throughput and low latency. Edge servers have strong computing and storage capabilities, and they are much closer to end users than the central server, so they can effectively reduce the network latency.

Literature [9-10] combines MEC framework with NOMA technology to study the optimization of offloading latency. The deep reinforcement learning DQN algorithm is used to choose users in pair according to the user's task volume. Without knowing other users' operation, the best user combination status is obtained for signal transmission which leads to the minimum offloading latency.

Reference [11] studies the offloading of vehicle network tasks based on MEC architecture. It uses the computing resources of the mobile vehicle itself and edge server resources to perform task calculations, and proposes a V2I (Vehicle-to-Infrastructure) task offloading model. In the V2I model, computing tasks are divided into three parts: local computing tasks, computing tasks offloaded to the edge server, and computing tasks offloaded to adjacent vehicles. Finally, the three parts of the calculation latency are jointly optimized to reduce communication latency and save resources.

In Reference [1], the authors propose a task query

This article is an extension of our conference paper [44].

mechanism, and put the task query, transmission and calculation into three places for execution: mobile user, edge server and central server. The central server is responsible for executing the proximity detection of the mobile users which are located at the border of the coverage area of each edge server, and most of computing tasks are still offloaded to the edge servers to reduce the transmission latency.

Due to the advantages of the MEC architecture in the above literature, this article reduces the total latency of the road network proximity detection task based on the MEC architecture. In this article, multiple edge cloud servers and central servers with powerful computing capabilities are set up to process user inquiry tasks and computing tasks. Based on the MEC enhanced proximity detection architecture proposed in [1], the latency at each moment t is optimized by SLSQP (Sequential Least Squares Programming) algorithm that can find the optimal solution [12], and deep reinforcement approach, i.e., DDPG (Deep Deterministic Policy Gradient) algorithm [19] that continuously optimizes the total latency at any time moment t in the dynamic road network.

The contributions of this article can be summarized as follows.

- We propose a computation offloading scheme for offloading computing tasks under the proximity detection architecture based on MEC in the dynamic road network.
- We formulate a latency optimization problem which aims to obtain the minimum proximity detection latency under some constraints.
- We adopt the SLSQP algorithm to solve the latency optimization problem in dynamic road networks.
- We also utilize the DDPG algorithm, i.e., a deep reinforcement learning approach, to obtain the minimum proximity detection latency.
- We conduct experiments to evaluate the performance of our proposed solutions. As the SLSQP algorithm can find the optimal solution [12], we use it as the baseline optimal solution to explore the optimization effect of the DDPG algorithm. We find that after multiple rounds of online training with the DDPG algorithm, the gap between the optimized value of the total latency and the theoretical optimal value at the corresponding time is basically within $0.2s$.

The rest of this article is structured as follows: Section II proposes a computation offloading scheme based on the proximity detection architecture enhanced by MEC and model the dynamic road networks. Section III formulates the total latency calculation process at any moment for the dynamic road network. Section IV solves the optimization problem using the convex optimization algorithm SLSQP and the deep reinforcement learning approach, i.e., DDPG algorithm, respectively. Section V evaluates the total latency optimization model established in Section IV through experiments. Section VI draws conclusions.

II. Computation Offloading Scheme and Dynamic Road Network Update Model

In this section, we introduce the computation offloading scheme in the scenario of proximity detection based on MEC and model the dynamic road networks.

A. Offloading Scheme for MEC-Enhanced Proximity Detection

In the road network, every mobile user needs to know the proximity relationship between him/her and other mobile users at each time epoch. As shown in Fig. 1, in the MEC enhanced proximity detection architecture [1], each mobile user communicates with the nearest edge server. Each mobile user can report their motion information to the nearest edge server and the edge server can share other user's information with the mobile user as well. In this process, each mobile user receives the information of other mobile users in the road network from the nearest edge server that is much closer than the central server to mobile users, and generates their own proximity detection calculation tasks. To reduce the computation burden of mobile users, each user can offload part of the computing tasks to the edge server for computation.

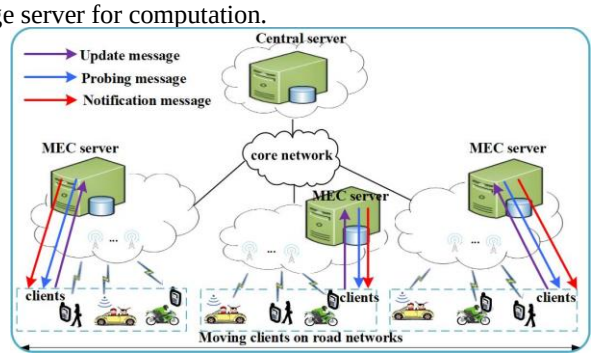

Fig. 1: Proximity detection architecture based on MEC [1]

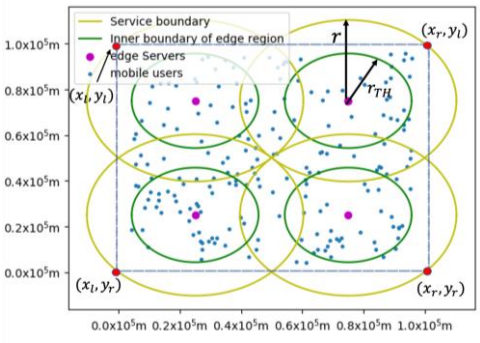

Fig. 2: Schematic diagram of serving area of each edge server at current epoch

As shown in Fig. 2, all edge servers have the same serving radius r. In the process of proximity detection, most mobile users can get aware of other mobile users which are in proximity with them through the calculation of an edge server. However, for a small number of mobile users which are located in the border region of the serving area of an edge server, some of the mobile users which are in proximity with them may not be located in the serving area of this edge server, but may be located in the serving area of other edge servers. In this case, the central server needs to be involved in the calculation. As shown in Fig. 2, if the distance between the edge server to the inner boundary of the border region of its serving area is r_{TH}, then the distance

This article is an extension of our conference paper [44].

from the edge server to the border region which is a circular ring falls into interval $[r_{TH}, r]$.

Fig. 3 is the schematic diagram of computation offloading for the MEC-enhanced proximity detection. As shown by the blue dotted line in Fig. 3, mobile users who are located in the non-border region of the serving area of the edge server in the road network send information to the edge server that communicates with them. For example, for User 2 and User 3 in Fig. 3, their proximity detection tasks only require the Edge Server 1 and Edge Server 2 to participate in the calculation. In terms of User 2, as shown by the solid blue line in Fig. 3, the edge server transmits user information to each mobile user. Then the mobile user generates computing tasks, offloads part of the computing tasks to the edge server for execution, and leaves the rest to be executed locally. After the edge server completes the calculation, it sends the calculation result back to the mobile user.

For a few mobile users which are in the border region of the edge server, for example, User 1 in Fig. 3, its proximity detection tasks require both the Edge Server 1 and the Central Server to cooperate in the calculation. As shown by the red dotted line in Fig. 3, all edge servers send mobile users' information within their serving areas to the central server. Thus, the central server is aware of all the mobile users in the road network, and each edge server is aware of the mobile users which are inside its own serving area. Without loss of generality, we take User 1 which locates in the border region of Edge Server 1 as an example. If we use v to denote the speed of User 1, the proximity time threshold is T_ε and the maximum speed of mobile users is V_{max}, then the Central Server uses a circle with User 1 as the center and $(|v| + |V_{max}|) \cdot T_\varepsilon$ as the radius, to select mobile users inside the circle, and sends their information to Edge Server 1. Then the Edge Server 1 which serves as a relay sends the information of these selected mobile users to User 1. Afterwards, User 1 generates computing tasks locally and offloads some computation tasks to the Edge Server 1. Finally, after calculation, this Edge Server 1 sends the calculation results back to User 1.

Therefore, the proximity detection tasks of the majority of users are computed by the edge servers whereas the proximity detection tasks of only a minority of users who locate in the border region of each edge server needs the central server to be involved in the calculation.

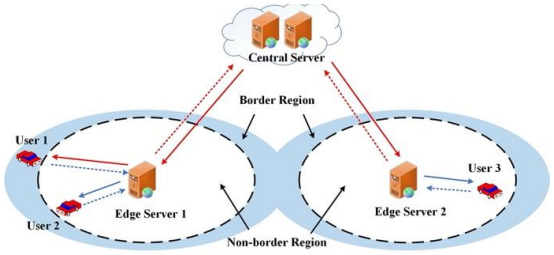

Fig. 3: Schematic diagram of computation offloading in the scenario of MEC-enhanced proximity detection

B. Dynamic Road Network Status Update Model

Section II.A describes the process of offloading, transmitting and calculating the task at a certain moment t. In this part, we introduce the status update process of the dynamic road network. In this and subsequent sections, we set N mobile users moving in the road network and the environmental state changes in every Δt time. All of them select the edge server with which he/she directly communicates according to the method in Section II.A. Since the central server does not execute the calculation task of the proximity detection of mobile users, changes in the calculation rate and available memory value of the central server can be ignored. In addition, changes of wireless channel transmission rate, the transmitting power changes and noise signal changes during transmission can also be ignored.

The State update algorithm, i.e., the RNDU algorithm, is shown in Algorithm 1. In the RNDU algorithm, we set the speed matrix and position matrix of mobile clients in the road network at time t as $\boldsymbol{V}_c^t$ and $\boldsymbol{P}_c^t$, and the dimensions of these two matrices are both $N \times 2$. The acceleration matrix is required to be more suitable for the dynamic environment. We use the Ornstein-Uhlenbeck (OU) process [13-14] to update acceleration matrix to make it time-dependent and more inertial. At time $t+1$, the acceleration of the mobile user, $\boldsymbol{acc}^{t+1}$, is updated by:

$$(\boldsymbol{acc}^{t+1})_{i,j} = (\boldsymbol{acc}^{t})_{i,j} + \theta_u\left(\mu - (\boldsymbol{acc}^{t})_{i,j}\right) + \sigma N(0, \Delta t) \quad (1)$$

where θ_u is the attenuation coefficient, μ is the mean and σ is the variance. The two columns of $\boldsymbol{acc}^{t+1}$ respectively represent the acceleration component. As shown in Fig. 2, to simplify the model, the coordinates of the four vertices of the rectangular road network boundary are (x_l, y_l), (x_l, y_r), (x_r, y_l), (x_r, y_r) where $x_l \le x_r, y_l \le y_r$. As shown in Fig. 4, if the physical distance S between a mobile user, say, user A, and the boundary is less than a threshold τ $(r_{TH} \le \tau \le r)$, it needs to decelerate with a larger acceleration $\boldsymbol{acc}_{max}$, and in the RNDU, $\boldsymbol{acc}_{max}$ will be superimposed on the $\boldsymbol{acc}^{t+1}$ update process, so that the acceleration updated through the OU random process has good braking performance in the boundary area of the road network (Lines 2 and 3 of Algorithm 1). The update of $\boldsymbol{V}_c^t$ follows the kinematic formula $\boldsymbol{V}_c^{t+1} = \boldsymbol{V}_c^t + \boldsymbol{acc}^{t+1} \cdot \Delta t$, but the speed of mobile users is constrained by the maximum speed threshold V_{max} (Line 4 and 5). Finally, because the acceleration and speed update process has ensured that all of the mobile users will not leave the road network, the update of $\boldsymbol{P}_c^t$ follows $\boldsymbol{P}_c^{t+1} = \boldsymbol{P}_c^t + \boldsymbol{V}_c^{t+1} \cdot \Delta t$ (Line 7). The matrix composed of the calculation rate and available memory values of all users are $\boldsymbol{R}_{cv}^t$ and $\boldsymbol{Q}_{cv}^t$, and the calculation rates and available memory values of all edge servers are $\boldsymbol{R}_{mv}^t$ and $\boldsymbol{Q}_{mv}^t$. In RNDU, both the mobile users' local computing rate update and the servers' computing rate update follow the normal distribution $(R_{cv}^{t+1})_i \sim N((R_{cv}^t)_i, \sigma_{cv})$ (Line 8), $(R_{mv}^{t+1})_i \sim N((R_{mv}^t)_i, \sigma_{Rm})$ (Line 9) and do not exceed their respective maximum thresholds R_{cmax} and R_{mmax}, where we use the value of the previous moment as the expectation, and σ_{cv} and σ_{Rm} as the variance. The mobile users' local and server-side available memory values are also updated with normal distribution $(Q_{cv}^{t+1})_i \sim N((Q_{cv}^t)_i - \eta(D_{local}^t)_i, \sigma_c)$ (Line 10), $(Q_{mv}^{t+1})_i \sim N((Q_{mv}^t)_i - \eta(D_{mec}^t)_i, \sigma_m)$ (Line 11) and do not exceed their respective maximum thresholds Q_{cmax} and Q_{mmax}, where we use the value of the previous moment as the expectation, and σ_c and σ_m as the variance. In order to

This article is an extension of our conference paper [44].

Algorithm 1: RNDU (Road Network Dynamic Update Algorithm)

Input: $\boldsymbol{V}_c^t, \boldsymbol{P}_c^t, \boldsymbol{acc}^t, \boldsymbol{R}_{cv}^t, \boldsymbol{Q}_{cv}^t, \boldsymbol{R}_{mv}^t, \boldsymbol{Q}_{mv}^t, \boldsymbol{\Sigma}_c, \boldsymbol{\Sigma}_m, D_{mec}^t, D_{local}^t, x_l, x_r, y_l, y_r, \boldsymbol{acc}_{max} = (acc_x, acc_y), V_{max}, \tau, \mu, \sigma, \eta, \theta_u, \Delta t$

Output: $\boldsymbol{V}_c^{t+1}, \boldsymbol{P}_c^{t+1}, \boldsymbol{acc}^{t+1}, \boldsymbol{R}_{cv}^{t+1}, \boldsymbol{Q}_{cv}^{t+1}, \boldsymbol{R}_{mv}^{t+1}, \boldsymbol{Q}_{mv}^{t+1}$

1: **For** $i = 1: \boldsymbol{acc}^t.size[0]$ ***do***

2: $(acc^{t+1})_{i,1} =$

$$\begin{cases} (acc^{t+1})_{i,1} = (acc^t)_{i,1} + \theta_u(\mu - (acc^t)_{i,1}) + \sigma N(0,\Delta t), x_l + \tau < (P_c^t)_{i,1} < x_r - \tau \\ (acc^{t+1})_{i,1} = (acc^t)_{i,1} + \theta_u(\mu - (acc^t)_{i,1}) + \sigma N(0,\Delta t) + acc_x \cdot \frac{(P_C^t)_{i,1} - x_l}{\left\|(P_C^t)_{i,1} - x_l\right\|}, (P_c^t)_{i1} \le x_l + \tau \\ (acc^{t+1})_{i,1} = (acc^t)_{i,1} + \theta_u(\mu - (acc^t)_{i,1}) + \sigma N(0,\Delta t) + acc_x \cdot \frac{(P_C^t)_{i,1} - x_r}{\left\|(P_C^t)_{i,1} - x_r\right\|}, (P_c^t)_{i,1} \ge x_r - \tau \end{cases}$$

3: $(acc^{t+1})_{i,2} =$

$$\begin{cases} (acc^{t+1})_{i,2} = (acc^t)_{i,2} + \theta_u(\mu - (acc^t)_{i,2}) + \sigma N(0,\Delta t), y_l + \tau < (P_c^t)_{i,2} < y_r - \tau \\ (acc^{t+1})_{i,2} = (acc^t)_{i,2} + \theta_u(\mu - (acc^t)_{i,2}) + \sigma N(0,\Delta t) + acc_y \cdot \frac{(P_C^t)_{i,2} - y_l}{\left\|(P_C^t)_{i,2} - y_l\right\|}, (P_c^t)_{i,2} \le y_l + \tau \\ (acc^{t+1})_{i,2} = (acc^t)_{i,2} + \theta_u(\mu - (acc^t)_{i,2}) + \sigma N(0,\Delta t) + acc_y \cdot \frac{(P_C^t)_{i,2} - y_r}{\left\|(P_C^t)_{i,2} - y_r\right\|}, (P_c^t)_{i,2} \ge y_r - \tau \end{cases}$$

4: $$(V_c^{t+1})_{i,1} = \begin{cases} (V_c^t)_{i,1} + (acc^{t+1})_{i,1}\Delta t, \left|(V_c^t)_{i,1} + (acc^{t+1})_{i,1}\Delta t\right| < V_{max} \\ V_{max} \cdot \frac{(V_C^t)_{i,1}}{\|(V_C^t)_{i,1}\|}, \left|(V_c^t)_{i,1} + (acc^{t+1})_{i,1}\Delta t\right| \ge V_{max} \end{cases}$$

5: $$(V_c^{t+1})_{i,2} = \begin{cases} (V_c^t)_{i,2} + (acc^{t+1})_{i,2}\Delta t, \left|(V_c^t)_{i,2} + (acc^{t+1})_{i,2}\Delta t\right| < V_{max} \\ V_{max} \cdot \frac{(V_C^t)_{i,2}}{\|(V_C^t)_{i,2}\|}, \left|(V_c^t)_{i,2} + (acc^{t+1})_{i,2}\Delta t\right| \ge V_{max} \end{cases}$$

6: **End for**

7: $\boldsymbol{P}_c^{t+1} = \boldsymbol{P}_c^t + \boldsymbol{V}_c^{t+1}\Delta t$

8: $\boldsymbol{R}_{cv}^{t+1}$ is sampled from $(\min\{R_{cmax}, N((R_{cv}^t)_i, \sigma_{cv})\})_i, 1 \le i \le N_c$

9: $\boldsymbol{R}_{mv}^{t+1}$ is sampled from $(\min\{R_{mmax}, N((R_{mv}^t)_i, \sigma_{Rm})\})_i, 1 \le i \le N_m$

10: $\boldsymbol{Q}_{cv}^{t+1}$ is sampled from $(\min\{Q_{cmax}, N((Q_{cv}^t)_i - \eta(D_{local}^t)_i, \sigma_c)\})_i, 1 \le i \le N_c$

11: $\boldsymbol{Q}_{mv}^{t+1}$ is sampled from $(\min\{Q_{mmax}, N((Q_{mv}^t)_i - \eta(D_{mec}^t)_i, \sigma_m)\})_i, 1 \le i \le N_m$

reflect the impact of the amount of uncalculated tasks at time t on the memory at time $t+1$, the expectations of the two normal distributions need to subtract the calculating task $(D_{local}^t)_i$ and $(D_{mec}^t)_i$ at time t, and the coefficient η is used to weigh the importance of $(D_{local}^t)_i$ and $(D_{mec}^t)_i$ (Lines 10 and 11), where $(D_{local}^t)_i$ represents the amount of local computing tasks of user *i* at time *t* and $(D_{mec}^t)_i$ represents the amount of computing tasks that users have offloaded to the edge server *i* at time *t*.

When a mobile user moves out of the serving area of the edge server during the calculation process of an edge server, the edge server needs to upload the calculation result to the central server first and the central server transmits the calculation result back to the mobile user through the edge server serving it at this time. For instance, as shown in Fig. 4, user B's offloaded task is executed on Edge Server 1. However, it moves to the serving area of Edge Server 2 at time $t+1$. The result is first uploaded to the Center Server, and the Center Server transmits the calculation result to the user B through the Edge Server 2. Because the task volume of the results sequence is much smaller than that of the proximity detection task, the communication latency of the process of transmitting the calculation results sequence can be ignored.

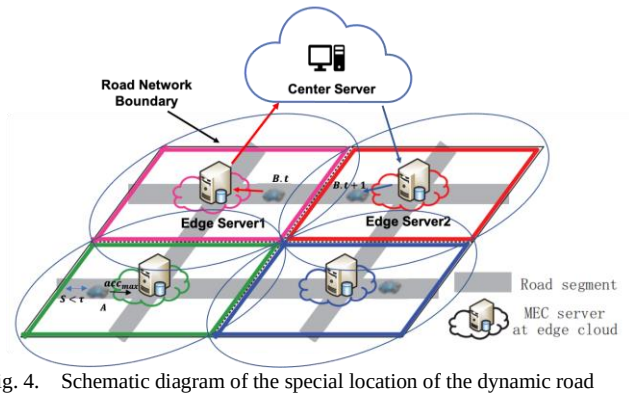

Fig. 4. Schematic diagram of the special location of the dynamic road network

This article is an extension of our conference paper [44].

III. Problem Formulation

In this section, we formulate our latency optimization problem at any moment t (Superscripts 't' are omitted for all state quantities at time t appearing in this section), based on the computation offloading scheme described in Section II. The information transmission and computation offload execution strategies of all users in the road network are the same. In order to facilitate the introduction and without loss of generality, for the rest of this article, we take a single user as an example. We name this user as *ob*, and the server that communicates with it is called Mo.

We assume that the information volume of the user's speed, location and proximity detection result sequence is much smaller than the information volume of the user's proximity detection computation task. Therefore, the transmission latency of the user's speed, location and proximity detection result sequence can be ignored [11, 15-16]. Therefore, the transmission and computation latency of the computation task will be the main part of the total latency. As we know, the *ob* generates computing task sequence. Suppose the number of mobile users in *Mo*'s serving area is N_{mc} (including the target user itself). Then the computing task of *ob* can be divided into $N_{mc}-1$ subtasks. Then the computing task of the *ob* can be divided into $N_{mc}-1$ subtasks $\boldsymbol{D}_e = \left[D_{1e}, D_{2e}, \dots, D_{ie}, \dots, D_{(N_{mc}-1)e}\right]^T$ (each subtask is the calculation task between the *ob* and the i-th user). The *ob* assigns weights α_i to each subtask, $0 \le \alpha_i \le 1, 1 \le i \le N_{mc}-1$, where α_i refers to the proportion ratio of subtask i that should be offloaded to Mo. Then the total amount of offloaded tasks is:

$$(D_{mec})_{ob} = \sum_{i=1}^{\mathrm{N_{mc}-1}} \alpha_i \times D_{ie} \tag{2}$$

The amount of local computation tasks is:

$$(D_{local})_{ob} = \sum_{i=1}^{\mathrm{N_{mc}-1}} (1-\alpha_i) \times D_{ie} \tag{3}$$

Mobile users in the road network communicate with edge servers through wireless channels. The wireless channel transmission power is related to the distance $d(t)$ between the user and the edge server, where $d(t)$ denotes the Euclidean distance function that changes with time t [11].

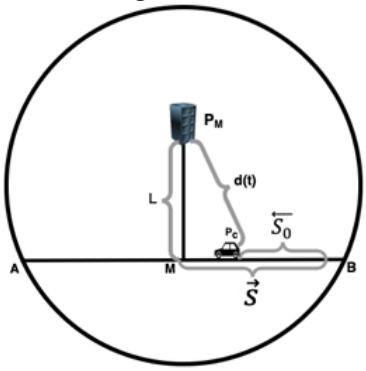

Fig. 5. Location relationship between the *ob* and the edge server

The *ob* makes a curved motion with time t in the road network, within the time period of $t\sim(t+\Delta t)$ (before the road network state changes), because this time interval is small, the *ob*'s trajectory can be approximated as a straight line. According to the analytic geometry theory, as shown in Fig. 5, the *ob*'s position coordinate $P_0(x_0, y_0)$ and its speed $v_o(v_{0x}, v_{0y})$ can determine a straight line $\overline{AB}$:

$$\frac{x-x_0}{v_{0x}} = \frac{y-y_0}{v_{0y}} \tag{4}$$

The edge server *Mo*'s position coordinates $P_M(x_{mec}, y_{mec})$ and its serving area's radius r can determine the boundary curve of the serving area:

$$(x-x_{mec})^2 + (y-y_{mec})^2 = r^2 \tag{5}$$

The above two equations can be used to obtain two intersection points $A(x_A, y_A)$, $B(x_B, y_B)$ of the boundary curve of the serving area and the straight line $\overline{AB}$. Then the coordinates of the middle points of the straight line $\overline{AB}$ is $M(x_m, y_m)$, where $x_m = \frac{x_A+x_B}{2}, y_m = \frac{y_A+y_B}{2}$.

Then the Euclidean distance from Mo to the straight line $\overline{AB}$ is:

$$L = \sqrt{(x_m-x_{mec})^2 + (y_m-y_{mec})^2} \tag{6}$$

As shown in Fig. 5, let the vector $\vec{s}$ be:

$$\vec{s} = (x_0, y_0) - (x_m, y_m) \tag{7}$$

and let the vector $\overrightarrow{s_0}$ be:

$$\overrightarrow{s_0} = (v_{0x}, v_{0y})\Delta t \tag{8}$$

The distance $d(t)$ between *ob* and Mo at time t can be denoted as:

$$d(t) = \sqrt{|\vec{s} + \overrightarrow{s_0}|^2 + L^2} \tag{9}$$

Let t_s denote the maximum time during which time period the target user *ob* is within the serving area of the edge server Mo. Then t_s can be calculated as:

$$(t_s)_{\mathrm{ob}} = \begin{cases} \frac{x_B-x_0}{v_{0x}}, \frac{x_B-x_0}{v_{0x}} \ge 0 \\ \frac{x_A-x_0}{v_{0x}}, \frac{x_A-x_0}{v_{0x}} \ge 0 \end{cases} \tag{10}$$

We set the wireless channel bandwidth as B and the orthogonal frequency is normally adopted for all communications between users and edge servers. Suppose the noise power spectral density is N_0, the transmission power is P, the channel gain is h, the factor δ is the path loss exponent. We ignore the influence of other factors and simplify the transmitted power to be $Pd(t)^{-\delta}h^2$ [11]. $d(t)$ changes continuously with t when *ob* moves along the straight line $\overline{AB}$ in the period of $t\sim(t+\Delta t)$. From Shannon's formula, the transmission rate R_u also changes with t, so the average transmission rate is used to calculate the transmission latency of the computation task. The wireless channel uplink and downlink transmission rates are both equal to R_u:

$$R_u = \frac{1}{t_s}\int_0^{t_s} B\log_2\left(1+\frac{Pd(t)^{-\delta}h^2}{N_0 B}\right)dt \tag{11}$$

The offloading latency from *ob* to *Mo* can be expressed as:

$$(t_{m1})_{ob} = \frac{(D_{mec})_{ob}}{R_u} \tag{12}$$

This article is an extension of our conference paper [44].

We set ob's local computing rate as $(R_{cv})_{ob}$ and Mo's computing rate as $(R_{mv})_{Mo}$. Then the local task computing latency t_{local} and the edge server computation latency t_{m2} can be obtained as below respectively:

$$(t_{local})_{ob} = \frac{(D_{local})_{ob}}{(R_{cv})_{ob}} \tag{13}$$

$$(t_{m2})_{Mo} = \frac{(D_{mec})_{ob}}{(R_{mv})_{Mo}} \tag{14}$$

The ob offloads part of the calculation task to the Mo for execution while executing the calculation task locally. The Mo sends the result sequence back to the ob. We assume that the result sequence records the proximity information between other users and the ob. In this article, the proximity between users is described by proximity information which is mainly the time distance between users. We also assume that the information amount of the result sequence is much smaller than that of the computation task, so the transmission latency of the result sequence is negligible. Therefore, the total latency of ob executing proximity detection at any time can be expressed as:

$$T = \max\{(t_{m1})_{ob} + (t_{m2})_{Mo}, (t_{local})_{ob}\} \tag{15}$$

Our objective is to minimize the latency T in Equation (15), subject to the following constraints:

1) The weight α_i assigned by the ob for each subtask needs to satisfy $N_{mc}-1$ double-side constraints, as given in Equation (16), which is equivalent to $2N_{mc}-2$ unilateral constraints:
$$C1: 0 \le \alpha_i \le 1, 1 \le i \le N_{mc}-1 \tag{16}$$
2) The sum of the offloading latency and computing latency of the edge server is not greater than the maximum time duration when the ob is in the serving area of the edge server Mo:
$$C2: (t_{m1})_{ob} + (t_{m2})_{Mo} \le (t_s)_{ob} \tag{17}$$
3) The amount of computation tasks executed locally by the ob must be within the user's local computation task threshold $(Q_{cv})_{ob}$:
$$C3: (D_{local})_{ob} \le (Q_{cv})_{ob} \tag{18}$$
4) The amount of computation tasks executed by the edge server Mo must be within the threshold $(Q_{mv})_{ob}$ of the tasks storage capacity of the edge server:
$$C4: (D_{mec})_{ob} \le (Q_{mv})_{Mo} \tag{19}$$

Hence, we finally formulate the proximity detection latency minimization problem with several constraints as Equation (20), which is a non-linear programming problem.

$$\begin{aligned} &min_{\{\alpha_i\}_{i=1}^{N_{mc}-1}} T, \\ s.t. \quad & C1 - C4 \end{aligned} \tag{20}$$

IV. Solutions To the Latency Optimization Problem

A. Solution based on SLSQP

From Equation (20), we know that our goal is to obtain an optimal ratio sequence $\{\alpha_i\}_{i=1}^{N_{mc}-1}$, which minimizes the total latency of proximity detection. This is a quadratic programming (QP) problem, and has a number of variables. In this subsection, we use the SLSQP algorithm [12][17-18] to combine the equality constraints and inequality constraints with the original objective function, construct a Lagrange function, convert it into a series of quadratic programming subproblems, and finally get the optimal solution of the original problem. The Lagrange function constructed by Equation (20) can be expressed as:

$$\begin{aligned} L(\boldsymbol{\alpha}, \boldsymbol{\lambda}) = T &+ \left(\sum_{i=1}^{N_{mc}-1} -\lambda_i \alpha_i\right) + \left(\sum_{i=N_{mc}}^{2N_{mc}-2} \lambda_i(\alpha_i - 1)\right) \\ &+ \lambda_{2N_{mc}-1}((t_{m1})_{ob} + (t_{m2})_{Mo} - (t_s)_{\mathrm{ob}}) \\ &+ \lambda_{2N_{mc}}((D_{local})_{ob} - (Q_{cv})_{ob}) \\ &+ \lambda_{2N_{mc}+1}((D_{mec})_{ob} - (Q_{mv})_{Mo}), \\ &s.t. \lambda_i \ge 0, 1 \le i \le 2N_{mc}+1 \end{aligned} \tag{21}$$

Where $\boldsymbol{\alpha} = \left(\alpha_1, \alpha_2, \dots, \alpha_{N_{mc}-1}\right)^T$ and Lagrange multiplier is $\boldsymbol{\lambda} = \left(\lambda_1, \lambda_2, \dots, \lambda_{N_{mc}-1}\right)^T$. Then, the SLSQP algorithm linearizes it to obtain the QP subproblem [12] and the objective function of the k-th subproblem is:

$$\begin{aligned} &\min\left(\frac{1}{2}\boldsymbol{d}_k^T \boldsymbol{H}_k \boldsymbol{d}_k + \nabla T(\boldsymbol{\alpha}^k)^T \boldsymbol{d}_k\right) \\ &s.t. \nabla g_i(\boldsymbol{\alpha}^k)^T \boldsymbol{d}_k + g_i(\boldsymbol{\alpha}^k) \le 0, i = 1, 2, \dots, 2N_{mc}+1 \end{aligned} \tag{22}$$

where $\boldsymbol{\alpha}^k = \left(\alpha_1^k, \alpha_2^k, \dots, \alpha_{N_{mc}-1}^k\right)^T$, $\boldsymbol{d}_k = \boldsymbol{\alpha}^{k+1} - \boldsymbol{\alpha}^k$,and

$$g_i(\boldsymbol{\alpha}^k) = -\alpha_i^k, 1 \le i \le N_{mc}-1 \tag{23}$$

$$g_i(\boldsymbol{\alpha}^k) = \alpha_i^k - 1, N_{mc} \le i \le 2N_{mc}-2 \tag{24}$$

$$g_{2N_{mc}-1}(\boldsymbol{\alpha}^k) = (t_{m1})_{ob} + (t_{m2})_{Mo} - (t_s)_{\mathrm{ob}} \tag{25}$$

$$g_{2N_{mc}}(\boldsymbol{\alpha}^k) = (D_{local})_{ob} - (Q_{cv})_{ob} \tag{26}$$

$$g_{2N_{mc}+1}(\boldsymbol{\alpha}^k) = (D_{mec})_{ob} - (Q_{mv})_{Mo} \tag{27}$$

According to Equations (23)-(27), $g_i(\cdot)$ are all linear function of independent variables, so $\nabla g_i(\boldsymbol{\alpha}^{k+1}) = \nabla g_i(\boldsymbol{\alpha}^k)$. The $\boldsymbol{H}_k$ is a Hessian positive definite quasi-Newton matrix of $L(\boldsymbol{\alpha}, \boldsymbol{\lambda})$, which can be calculated by BFGS [12] algorithm:

$$\boldsymbol{H}_{k+1} = \boldsymbol{H}_k + \frac{\boldsymbol{q}_k \boldsymbol{q}_k^T}{\boldsymbol{q}_k^T \boldsymbol{s}_k} - \frac{\boldsymbol{H}_k^T \boldsymbol{H}_k}{\boldsymbol{s}_k^T \boldsymbol{H}_k \boldsymbol{s}_k} \tag{28}$$

$$\begin{aligned} \boldsymbol{q}_k &= \nabla T(\boldsymbol{\alpha}^{k+1}) + \sum_{i=1}^{2N_{mc}+1} \lambda_i \nabla g_i(\boldsymbol{\alpha}^{k+1}) \\ &\quad - \left[\nabla T(\boldsymbol{\alpha}^k) + \sum_{i=1}^{2N_{mc}+1} \lambda_i \nabla g_i(\boldsymbol{\alpha}^k)\right] \\ &= \nabla T(\boldsymbol{\alpha}^{k+1}) - \nabla T(\boldsymbol{\alpha}^k) \end{aligned} \tag{29}$$

$$\boldsymbol{s}_k = \boldsymbol{\alpha}^{k+1} - \boldsymbol{\alpha}^k \tag{30}$$

Where:

$$\nabla T(\cdot)_i = \begin{cases} D_{ie}\left(\frac{1}{R_u} + \frac{1}{(R_{mv})_{Mo}}\right), (t_{m1})_{ob} + (t_{m2})_{Mo} \ge (t_{local})_{ob} \\ D_{ie}\frac{1}{(R_{cv})_{ob}}, (t_{m1})_{ob} + (t_{m2})_{Mo} < (t_{local})_{ob} \end{cases}, \quad 1 \le i \le N_{mc} \tag{31}$$

The SLSQP uses the active set strategy algorithm to solve Equation (22) at each iteration [12]. The Active set strategy algorithm directly converts inequality constraints into a series of equality constraints. We use $\boldsymbol{I}$ to represent the identity matrix. According to the $2N_{mc}+1$ unilateral constraints, we get the coefficient matrix $\boldsymbol{C}$ and constant term matrix $\boldsymbol{b}$ composed of all constraint coefficients by KKT (Karush-Kuhn-Tucker) conditions:

This article is an extension of our conference paper [44].

$$\boldsymbol{C} = \left[\boldsymbol{I}_{N_{mc}-1}, \boldsymbol{I}_{N_{mc}-1}, \left(\frac{1}{(R_{mv})_{Mo}}+\frac{1}{R_u}\right)\boldsymbol{D}_e, \boldsymbol{D}_e, \boldsymbol{D}_e\right]^T \quad (32)$$

$$\boldsymbol{b} = \left[\boldsymbol{0}^T_{(N_{mc}-1)\times 1}, \boldsymbol{1}^T_{(N_{mc}-1)\times 1}, (t_s)_{ob}, \boldsymbol{1}^T_{(N_{mc}-1)\times 1}\boldsymbol{D}_e - (Q_{cv})_{ob}, (Q_{mv})_{Mo}\right]^T \quad (33)$$

In Active set strategy shown in Algorithm 2, $A(\boldsymbol{\alpha}) = \{i \in [1, 2N_{mc}+1] | \boldsymbol{C}_{i:}\boldsymbol{\alpha} = \boldsymbol{b}\}$ is called the effective set at $\boldsymbol{\alpha}$. We define $U(0,1)$ to be uniformly distributed between 0 and 1, the initial value of $\boldsymbol{\alpha}$ is $\boldsymbol{\alpha}_0$, and initialize $\boldsymbol{\alpha}_0$ by sampling from $U(0,1)$. A_k is an iterative set where k starts from 0, initially $A_0 = A(\boldsymbol{\alpha}_0)$, and is called the working set. The goal of the algorithm is to make the working set approximate the effective set of the minimum latency parameter $\boldsymbol{\alpha}^*$ through iterating constantly $A_k \to A(\boldsymbol{\alpha}^*)$.

Algorithm 2: Active Set Strategy

Input: R_u , $(R_{mv})_{Mo}$, $(R_{cv})_{ob}$, $\boldsymbol{D}_e$, T , $(t_s)_{ob}$, $(D_{local})_{ob}$, $(D_{mec})_{ob}$,$(Q_{cv})_{ob}$,$(D_{mec})_{Mo}$

Output: The minimum latency T_{min} and corresponding $\boldsymbol{\alpha}^*$

1: Initialize: feasible point $\boldsymbol{\alpha}_0 \sim U(0,1)$, and set $A_0 = A(\boldsymbol{\alpha}_0)$, $k = 0, I = \{i | 1 \le i \le 2N_{mc}+1\}$

2: Create sub-objective function (22) according to (23)-(33), and converts inequality constraints into a series of quadratic constraints.

3: Get $\boldsymbol{d}_k$ *and* λ_i^k by KKT conditions:

$$\begin{bmatrix} \boldsymbol{H}_k & -\boldsymbol{C}_{i:}^T \\ \boldsymbol{C}_{i:} & \boldsymbol{0} \end{bmatrix}\begin{bmatrix} \boldsymbol{d}_k \\ \lambda_i^k \end{bmatrix} = \begin{bmatrix} -T(\boldsymbol{\alpha}^k) \\ 0 \end{bmatrix}$$

4: If $\boldsymbol{d}_k = 0$, **then**

5: **If** $\lambda_i^k \ge 0, \forall i \in A_k \cap I$, **then**

6: $\boldsymbol{\alpha}^* = \boldsymbol{\alpha}^k$, STOP

7: **Else**

8: $\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k, A_{k+1} = A_k \backslash \{i\}$,

9: Where $i = argmin\{\lambda_j^k | \lambda_j^k < 0, j \in A_k \cap I\}$

10: $k = k+1$, back to line 2.

11: Else

12: $\varepsilon^k = min\left\{1, \frac{b_i - \boldsymbol{C}_{i:}^T\varepsilon^k}{\boldsymbol{C}_{i:}^T\boldsymbol{d}_k} \middle| i \in I \backslash A_k, \boldsymbol{C}_{i:}^T\boldsymbol{d}_k < 0\right\}$,

13: $\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k + \varepsilon^k \boldsymbol{d}_k$

14: **If** $\exists i \in I \backslash A_k$, make $\boldsymbol{C}_{i:}^T\boldsymbol{\alpha}^{k+1} = b_i$, **then**

15: $A_{k+1} = A_k \cup \{i\}$

16: **Else**

17: $A_{k+1} = A_k$, $k = k+1$, back to line 2

B. Solution based on DDPG

In this subsection, we propose the second solution, i.e., DDPG algorithm.

From the dynamic environment presented in Section II.B, we can see that the state of the edge server in the previous moment has an impact on the state and task offloading strategy of the mobile user at a later moment, and the proximity detection tasks of the mobile users in the road network need to be completed before the state update of the mobile user and the edge server. The characteristic of the SLSQP algorithm is memoryless [17], that is, the algorithm only uses the current state of the mobile user for latency optimization, and the state of the previous time is not fully utilized. In addition, the SLSQP algorithm also needs long iterative optimization time for the problem [18], because if the number of calculation tasks is large, this algorithm cannot guarantee that the optimization is completed within Δt interval.

Therefore, we propose a solution that uses the DDPG algorithm to optimize the latency of proximity detection in dynamic road networks, and get the optimal offloading ratio of computing tasks at different time moments. Mathematically, in DDPG algorithm [19-20], $\boldsymbol{S}$ is the state space of the agent and $\boldsymbol{s} \in \boldsymbol{S}$ is the State value vector of the agent at a certain moment. $\boldsymbol{A}$ is the action space, and $\boldsymbol{a} \in \boldsymbol{A}$ is the action vector of the agent at a certain moment. r is a reward function, which represents the potential reward obtained by an agent in a certain state after performing a certain action. $\pi(\boldsymbol{a}|\boldsymbol{s})$ is a strategy function that represents the mapping from a state to an action. Finally, $Q(\boldsymbol{s}, \boldsymbol{a})$ is the state-action value function.

The DDPG algorithm integrates the advantages of the PG (Policy Gradient) algorithm [21] that can handle continuous action space decision problems, and the DQN (Deep Q-Network) framework [22-24] that can use the greedy strategy to handle the continuous state space, through the actor-critic architecture [25]. $Q(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{w})$ is used as the critic network, and $\pi(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{\theta})$ is used as the actor network. $\boldsymbol{w}$ and $\boldsymbol{\theta}$ are used as parameters in the critical network and the actor network, respectively. The DDPG algorithm adopts the idea of DDQN (Double DQN) [26], making the actor and critic use double network with the same structure (one as the target network with parameters $\boldsymbol{\theta}'$ and $\boldsymbol{w}'$, another for training network with parameters $\boldsymbol{\theta}$ and $\boldsymbol{w}$) to update the parameters in the target network through soft target updates [19].

The rest of this subsection will introduce the establishment of $\boldsymbol{S}, \boldsymbol{A}, r, \pi, Q$ in detail in the optimization process of the DDPG algorithm under the dynamic environment of the road network.

1) State vector $\boldsymbol{s}$ and action vector $\boldsymbol{a}$ design

Taking the target user *ob* as an example, in the road network, although the speed, location, calculation rate and available memory of all mobile users and the calculation rate and available memory of all edge servers are dynamically changing, what is relevant to the detection task of the proximity to *ob* is the *ob* itself and the edge server directly communicating with it. Therefore, in order to avoid inputting features into the state vector $\boldsymbol{s}$, which are not directly related to *ob*, the state vector should only include the state parameters of *ob* and this edge server at each time. Given *ob's* speed v_0, position P_0, calculation rate $(R_{cv})_{ob}$ and available memory value $(Q_{cv})_{ob}$, the calculation rate $(R_{mv})_{Mo}$ and the available memory

This article is an extension of our conference paper [44].

$(Q_{mv})_{Mo}$ of the edge server communicating with it, the state vector $\boldsymbol{s}_1^t$ at t can be expressed as:

$$\boldsymbol{s}_1^t = \left[v_{0x}^t, v_{0y}^t, x_0^t, y_0^t, (R_{cv})_{ob}^t, (Q_{cv})_{ob}^t, (R_{mv})_{Mo}^t, (Q_{mv})_{Mo}^t\right]^T \quad (34)$$

Because the goal of the DDPG algorithm is to calculate the optimal offloading ratio of the subtasks that minimizes the latency, the value of the subtasks generated by *ob* at all time moments should be added to the current state. According to $\boldsymbol{D}_e$ in Section II, the number of *ob's* subtasks $N_{mc}-1$ depends on the number of users in the serving area of the edge server or the number of users selected by the central server, so the number of subtasks generated by *ob* at each time moment may be different. However, the state vector is used as the input layer of the two neural networks $\pi(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{\theta})$ and $Q(\boldsymbol{s},\boldsymbol{a},\boldsymbol{w})$, and its dimensions must be determined, otherwise the error back propagation algorithm cannot be used to perform gradient calculations on the neural network [27-29]. For the above problem, it is assumed that the CPU in the edge server can generate at most N_{Thm} threads, and the maximum number of threads generated by the mobile user's local CPU is N_{THl}, $N_{THl} \le N_{Thm}$. As shown in Fig. 6, in this article, we divide all computing subtasks into N_{Thm} shares in order to make full use of all thread computing in the edge server. The mobile user may not be able to generate N_{Thm} threads locally, but the CPU scheduling algorithm can be used to complete the scheduling of N_{Thm} computing tasks [30-31].

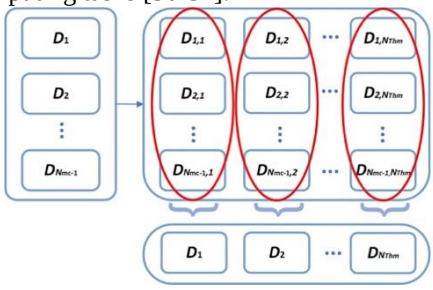

Fig. 6. Subdivision graph

Finally, N_{Thm} computing subtasks will be generated as new computing subtasks. The subtask vector at time t is $\boldsymbol{D}_v^{t\,T} = \left[D_1^t, D_2^t, \dots, D_{N_{Thm}}^t\right]^T$. We define the action vector at time t is $\boldsymbol{a}^t$, which is equal to the offloading proportion vector of *ob's* N_{Thm} subtasks at time t plus explore ingredients $\boldsymbol{\epsilon}^t$ that follows the Ornstein–Uhlenbeck process [13]:

$$\boldsymbol{a}^t = \left[f(\alpha_1^t), f(\alpha_2^t), \dots, f(\alpha_{N_{Thm}}^t)\right]^T + \boldsymbol{\epsilon}^t, 1 \le i \le N_{Thm} \quad (35)$$

$$\boldsymbol{\epsilon}^t = \boldsymbol{\epsilon}^{t-1} + \theta_a(\mu_a - \boldsymbol{\epsilon}^{t-1}) + \sigma_a \boldsymbol{N}(0, \Delta t) \quad (36)$$

Where θ_a is also the attenuation coefficient, μ_a is mean and σ_a is variance. Then the action space is $\boldsymbol{A}: \boldsymbol{R}^{N_{Thm}}$. We define the function $f(x) = \min\{1, \max\{0, x\}\}$, where f is the constraint on each element in $\boldsymbol{a}^t$, $0 \le f(\alpha_i^t) \le 1$, then the state vector at time t $\boldsymbol{s}_2^t$ is:

$$\boldsymbol{s}_2^t = \left[\alpha_1^{t-1} D_1^t, (1-\alpha_1^{t-1}) D_1^t, \alpha_2^{t-1} D_2^t, (1 - \alpha_2^{t-1}) D_2^t, \dots \alpha_{N_{Thm}}^{t-1} D_{N_{Thm}}^t, (1-\alpha_{N_{Thm}}^{t-1}) D_{N_{Thm}}^t\right]^T \quad (37)$$

Finally, by combining $\boldsymbol{s}_1^t$ and $\boldsymbol{s}_2^t$, *ob*'s state value vector can be represented as

$$\boldsymbol{s}^t = \left[\boldsymbol{s}_1^{t\,T}, \boldsymbol{s}_2^{t\,T}\right]^T. \quad (38)$$

2) Reward function $\boldsymbol{r}$ design

The reward function $r^t(\boldsymbol{s}^t, \boldsymbol{a}^t)$ obtained by the mobile user at time t directly reflects the pros and cons of the user's current decision. If the user's local decision-making $\boldsymbol{a}^t$ makes the latency greater, the user gets a smaller reward at this time. Therefore, the reward function is inversely related to the latency. The user-generated $\boldsymbol{a}^t$ must satisfy the constraints $C1-C4$ in Equation (20). If any of the constraints are not satisfied, the user's reward will be reduced. Because the $f(x)$ function has restricted the value range of the elements in $\boldsymbol{a}^t$ to [0,1], only the constraints $C2-C4$ need to be introduced in the reward function. Combined with Equations (17)-(19), the reward function is as follows:

$$r^t = -T - \beta_1 g(relu((D_{local})_{ob}^t) - (Q_{cv})_{ob}^t)) - \beta_2 g(relu((D_{mec})_{ob}^t - (Q_{mv})_{Mo}^t)) - \beta_3 g(relu((t_{m1})_{ob}^t + (t_{m2})_{Mo}^t - (t_s)_{ob}^t))$$
$$s.t. \quad \beta_1, \beta_2, \beta_3 > 0 \quad (39)$$

In Equation (39), the reward is linearly and negatively correlated with the latency. If the action at the current moment satisfies the constraints $C2-C4$, then these constraints have no influence on the reward currently received by the user. If any of the constraints is not satisfied, the reward currently received by the user will be reduced. As expressed in Equation (39), we first apply activation function $relu(x) = \begin{cases} x, x \ge 0 \\ 0, x < 0 \end{cases}$ [32] to constraints $C2-C4$, which are then introduced to the reward function. $\beta_1, \beta_2, \beta_3$ are used as weight coefficients in the reward function, which can reflect the importance attached to the violation of constraints in the reward function. In Equation (39), the order of magnitude of latency T is 10^{-1}, while the order of magnitude of other terms is $10^{-1} \sim 10^5$. In order to avoid the value generated by the constraints $C2-C4$ is too large, we use a mathematically compressing mapping function $g(\cdot)$ to limit the value of the constraint part in the reward function to the interval $[0,1]$. In this article, the unit step function, sigmoid function and tanh function are used to compress the image of the constraint part. Below we will introduce the characteristics of the three functions.

The analytical formula of the step function is $\varepsilon(x) = \begin{cases} 0, x \le 0 \\ 1, x > 0 \end{cases}$. It can be seen from the analytical formula that the value of the independent variable after the step function can be compressed into the discrete interval [0,1]. However, the value of the constraint term is continuous, and the range of the step function is discrete, i.e., only 0, 1 two values, so different values of constraint items cannot be distinguished after the image is compressed. Therefore, no matter how great the difference between the values of the constraints, even if the values of the constraints differ by several orders of magnitude, the difference between the values after the step function is only 0 or 1.

The analytical formula of sigmoid function is $sigmoid(x) = \frac{1}{1+e^{-x}}$ [33]. The independent variable is compressed into the continuous interval [0,1] through the sigmoid function, so the difference between the different values of the constraints still exists after the image is compressed. However, the sigmoid function also has a vanishing gradient

This article is an extension of our conference paper [44].

situation, which often occurs in the parameter update process of the front neural network layer [27].

The analytical formula of the tanh function is $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ [34]. In the interval [-1,1], the tanh function can be approximately equal to the proportional function of independent variables by Taylor expansion. Compared with the sigmoid function, the vanishing gradient problem can be solved in this interval. However, on the real number axis outside this interval, as the independent variable changes, the function value will quickly saturate around the value of -1 or 1, resulting in the slowdown of the neural network parameter training [35].

3) Strategy function $\boldsymbol{\pi(a|s,\theta)}$ design

In this subsection, we will introduce the design method of the strategy function $\pi(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{\theta})$ in detail. In the DDPG algorithm, $\pi(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{\theta})$ is expressed through a neural network. At time t, the state vector $\boldsymbol{s}^t$ is input into the strategy function $\pi(\boldsymbol{a}^t|\boldsymbol{s}^t,\boldsymbol{\theta})$ to obtain the deterministic action vector $\boldsymbol{a}^t$ at the current time. It can be seen from the expression of $\boldsymbol{s}^t$ in Equation (38), the elements in $\boldsymbol{s}_1^t$ are physical quantities that describe the basic properties of *ob* and the edge server, while $\boldsymbol{s}_2^t$ is composed of the local computing task volume and the edge server-side computing task volume. Therefore, $\boldsymbol{s}_1^t$ and $\boldsymbol{s}_2^t$ are two vectors with different properties. In this article, the two are separately input into the neural network of the strategy function. The specific structure of $\pi(\boldsymbol{a}^t|\boldsymbol{s}_1^t,\boldsymbol{s}_2^t,\boldsymbol{\theta})$ neural network is shown in Fig. 7.

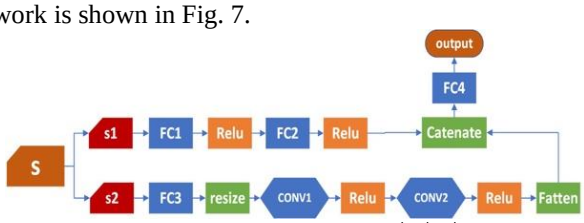

Fig. 7. The strategy function neural network $\pi(\boldsymbol{a}^t|\boldsymbol{s}_1^t,\boldsymbol{s}_2^t,\boldsymbol{\theta})$ structure

First, the state $\boldsymbol{s}^t$ at time t is divided into $\boldsymbol{s}_1^t$ and $\boldsymbol{s}_2^t$ and input into different neural networks respectively. As shown in Fig. 7, $\boldsymbol{s}_1^t$ is input into the fully connected layers FC1 and FC2 to extract features in turn and input into the Relu layer to get the nonlinear output, and the output feature dimensions are 28. The features of $\boldsymbol{s}_2^t$ are extracted through the fully connected layer FC3, with the output dimension of the FC3 layer being 100, then the dimension is converted to 1 × 10 × 10, and sparse interaction between features is performed through the convolution layers CONV1 and CONV2. The convolution kernel sizes of CONV1 and CONV2 are 7 × 7 and 5 × 5, respectively, and both of them use the Relu layer to perform nonlinear mapping [36, 37]. After the steps above, the output dimension is still 1 × 10 × 10. Then the feature is converted into a vector with a dimension of 100, and stitched with the output of the $\boldsymbol{s}_1^t$ network to get a 128-dimensional vector, which is the Catenate operation in Fig. 11. Finally, 64-dimensional motion vectors are output through the fully connected network FC4.

4) State-action value function $\boldsymbol{Q(s,a,w)}$ design

In this subsection, we introduce the design method of the state-action value function $Q(\boldsymbol{s},\boldsymbol{a},\boldsymbol{w})$ in detail. In the DDPG algorithm, $Q(\boldsymbol{s},\boldsymbol{a},\boldsymbol{w})$ is still expressed through the neural network. Unlike the strategy function, at time t, the state vector $\boldsymbol{s}^t$ and the action vector $\boldsymbol{a}^t$ are simultaneously input into the state-action function to obtain the current state action function value [19]. The specific structure of the $Q(\boldsymbol{s}_1^t,\boldsymbol{s}_2^t,\boldsymbol{a}^t,\boldsymbol{w})$ neural network is shown in Fig. 8.

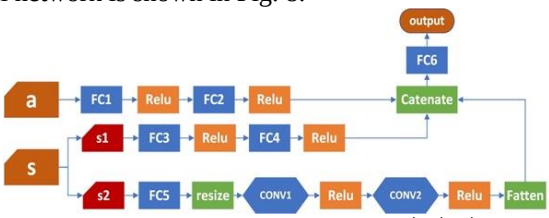

Fig. 8. The state-action function neural network $Q(\boldsymbol{s}_1^t,\boldsymbol{s}_2^t,\boldsymbol{a}^t,\boldsymbol{w})$ structure

In addition to the input state vector $\boldsymbol{s}^t$, the state-action function network at time t also needs to input the action vector $\boldsymbol{a}^t$. $\boldsymbol{s}^t$ and $\boldsymbol{a}^t$ are input to two different neural networks. As shown in Fig. 8, the network structure in which $\boldsymbol{s}^t$ is input is the same as the strategy function, which makes the input $\boldsymbol{s}^t$ divided into $\boldsymbol{s}_1^t,\boldsymbol{s}_2^t$, and then gets two different intermediate results through two different neural networks. In addition, the features of $\boldsymbol{a}^t$ must be further extracted through two fully connected layers FC1 and FC2 in turn. The dimensions of the output feature vector of the action vector through FC1 and FC2 are 64 and 128, respectively, and Relu layers are used as the activation function of the result of the fully connected layer. The feature vectors obtained from the output of $\boldsymbol{s}_1^t,\boldsymbol{s}_2^t$ and $\boldsymbol{a}^t$ networks are spliced into a feature vector with a length of 256, as shown in the *Catenate* operation in Fig. 8. Finally, one-dimensional state value output is obtained through the fully connected layer FC6.

5) Loss function and parameter update of DDPG algorithm

Sections IV.B.1), IV.B.2), IV.B.3) and IV.B.4) have introduced the 4-tuples used in the DDPG algorithm respectively. In this subsection, the parameter update method in the DDPG algorithm will be presented.

It can be seen from literature [38] that the DDPG algorithm puts the $(\boldsymbol{s}^t,\boldsymbol{a}^t,r^t,\boldsymbol{s}^{t+1})$ tuple obtained by sampling into the experience replay pool *Buffer*. In the case, the size of the experience replay pool is M, the number of data batches sampled from the experience replay pool *Buffer* when training the action-state function is m, and $m < M$, the length of a single sampling chain is T, $Q(\boldsymbol{s}_1^T,\boldsymbol{s}_2^T,\boldsymbol{a}^T,\boldsymbol{w}') = 0$, and the attenuation coefficient is γ [19][39], then the current critical network target y_i is:

$$y_i = r_i^t + \gamma\, Q(\boldsymbol{s}_1^{t+1},\boldsymbol{s}_2^{t+1},\boldsymbol{a}^{t+1},\boldsymbol{w}')_i, 1 \le i \le m \quad (40)$$

The loss function of the critic network is:

$$J(w) = \frac{1}{m}\sum_{i=1}^{m}(y_i - Q(\boldsymbol{s}_1^t,\boldsymbol{s}_2^t,\boldsymbol{a}^t,\boldsymbol{w})_i)^2 \quad (41)$$

The loss function of the actor network is:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} Q(\boldsymbol{s}_1^t,\boldsymbol{s}_2^t,\boldsymbol{a}^t,\boldsymbol{w})_i \quad (42)$$

Equations (41) and (42) update the parameters $\boldsymbol{w}$ and $\boldsymbol{\theta}$ through gradient back propagation, respectively. The parameter $\boldsymbol{w}'$ of the target actor network and the parameter $\boldsymbol{\theta}'$ of the target critic network are softly updated by the parameters $\boldsymbol{w}$

This article is an extension of our conference paper [44].

Algorithm 3: RoadNetwork-DDPG Algorithm

Input: $\boldsymbol{V}_c^0$, $\boldsymbol{P}_c^0$, $\boldsymbol{acc}^0$, $\boldsymbol{R}_{cv}^0$, $\boldsymbol{Q}_{cv}^0$, $\boldsymbol{R}_{mv}^0$, $\boldsymbol{Q}_{mv}^0$, $\boldsymbol{\Sigma}_c$, $\boldsymbol{\Sigma}_m$, $\boldsymbol{D}_v^{0^T}$, D_{mec}^0 , D_{local}^0 , x_l , x_r , y_l , y_r , $\boldsymbol{acc}_{max} = (acc_x, acc_y)$, V_{max} , ob_i , τ , μ , σ , η , Δt , θ_u , $\boldsymbol{\theta}$, $\boldsymbol{w}$, $\boldsymbol{\theta}' = \boldsymbol{\theta}$, $\boldsymbol{w}' = \boldsymbol{w}$, $\boldsymbol{a}^0 = \left[a_1^0, a_2^0, \dots, a_{N_{Thm}}^0\right]^T + \boldsymbol{\epsilon^0}$,$\boldsymbol{\epsilon^0} \sim \boldsymbol{N}(\boldsymbol{0}, \boldsymbol{I})$,$\boldsymbol{M}$,$\boldsymbol{m}$,$\boldsymbol{\beta_1}$,$\boldsymbol{\beta_2}$,$\boldsymbol{\beta_3}$,$r^0 = 0$

1: Initialize: $\boldsymbol{s}_1^0 = [(\boldsymbol{V}_c^0)_{ob_i,1}, (\boldsymbol{V}_c^0)_{ob_i,2}, (\boldsymbol{P}_c^0)_{ob_i,1}, (\boldsymbol{P}_c^0)_{ob_i,2}, (\boldsymbol{R}_{cv}^0)_{ob_i,1}, (\boldsymbol{Q}_{cv}^0)_{ob_i,1}, (\boldsymbol{R}_{mv}^0)_{ob_i,1}, (\boldsymbol{Q}_{mv}^0)_{ob_i,1,}]^T$, $\boldsymbol{s}_2^0 = \left[a_1^0 D_1^0, (1-a_1^0) D_1^0, a_2^0 D_2^0, (1-a_2^0) D_2^0, \dots, a_{N_{Thm}}^0 D_{N_{Thm}}^0, \left(1-a_{N_{Thm}}^0\right) D_{N_{Thm}}^0\right]^T$,$\boldsymbol{s}^0 = \left[\boldsymbol{s_1^{0T}}, \boldsymbol{s_2^{0T}}\right]^T$,$\pi(\boldsymbol{a}^0 | \boldsymbol{s}_1^0, \boldsymbol{s}_2^0, \boldsymbol{\theta}), Q(\boldsymbol{s}_1^0, \boldsymbol{s}_2^0, \boldsymbol{a}^0, \boldsymbol{w}), \pi(\boldsymbol{a}^0 | \boldsymbol{s}_1^0, \boldsymbol{s}_2^0, \boldsymbol{\theta}'), Q(\boldsymbol{s}_1^0, \boldsymbol{s}_2^0, \boldsymbol{a}^0, \boldsymbol{w}')$

2: For $episode = 1:T$ **do**

3: **For** $t = 1:T$ **do**

4: $\epsilon^t = \epsilon^{t-1} + \theta_a(\mu - \epsilon^{t-1}) + \sigma_a \boldsymbol{N}(0, \Delta t)$

5: Calculate the action vector $\boldsymbol{a}^t$ at time t by equation (35) and (36)

6: Calculate the state vector $\boldsymbol{s}^t = \left[\boldsymbol{s_1^t}^{\boldsymbol{T}}, \boldsymbol{s_2^t}^{\boldsymbol{T}}\right]^T$ at time t by equation (34) and (37)

7: Calculate the reward r^t at time t by equation (38)

8: Get the D_{mec}^t,D_{local}^t from equation (1) and (2)

9: Get the $\boldsymbol{V}_c^{t+1}$,$\boldsymbol{P}_c^{t+1}$,$\boldsymbol{acc}^{t+1}$,$\boldsymbol{R}_{cv}^{t+1}$,$\boldsymbol{Q}_{cv}^{t+1}$,$\boldsymbol{R}_{mv}^{t+1}$,$\boldsymbol{Q}_{mv}^{t+1}$ through **RNDU**($\boldsymbol{V}_c^t$,$\boldsymbol{P}_c^t$,$\boldsymbol{acc}^t$,$\boldsymbol{R}_{cv}^t$,$\boldsymbol{Q}_{cv}^t$,R_{mv}^t,$\boldsymbol{Q}_{mv}^t$,$\boldsymbol{\Sigma}_c$,$\boldsymbol{\Sigma}_m$,D_{mec}^t,D_{local}^t,x_l,x_r,y_l,y_r,$\boldsymbol{acc}_{max}$,V_{max},τ,μ,σ,η,θ_u,Δt)

10: Store tuple $(\boldsymbol{s}^t, \boldsymbol{a}^t, r^t, \boldsymbol{s}^{t+1})$ in $Buffer$

11: **If** $Buffer.size > M$ **then**

12: $Buffer = Buffer[0:M-1]$

13: **End if**

14: Sample a random batch of m tuples from $Buffer$ by uniform distribution $U(1, M)$

15: Calculate the target value y_i by equation (39)

16: Update critic network by minimizing the loss equation (40)

17: Update actor network by minimizing the loss equation (41)

18: Update the target actor network and target critic network by equation (42)

19: **End for**

20: End for

and $\boldsymbol{\theta}$ and the state-keeping proportional parameter τ respectively:

$$\boldsymbol{w}' \leftarrow \tau \boldsymbol{w} + (1-\tau)\boldsymbol{w}', \quad \boldsymbol{\theta}' \leftarrow \tau \boldsymbol{\theta} + (1-\tau)\boldsymbol{\theta}' \quad (43)$$

Finally, the DDPG algorithm is applied to the road network dynamic model, as described in Algorithm 3.

V. Performance Evaluation

The initial values of the state parameters and control parameters of the road network simulation scenario in our experiments are shown in TABLE I. Next, we will investigate the impact of some parameters on the optimization of the total latency of the *ob*'s proximity detection task in the road network using the SLSQP algorithm and DDPG algorithm respectively. Our simulation environment is Linux workstation with Intel(R) Xeon(R) Gold 6240 CPU @ 2.60GHz and 125G memory. The GPU resource used by the DDPG algorithm is Tesla V100 SXM2 32GB. We use python 3.7 with numpy and pytorch for simulation.

TABLE I
Parameters for Dynamic Road Network Environment

Parameter	Value	Parameter	Value
N	200	N_{Thm}	64
x_l	0	h	0.95
x_r	$1 \times 10^5 m$	T_ε	$1s$
y_l	0	V_{maxx},V_{maxy}	$60km/h$
y_r	$1 \times 10^5 m$	V_{maxy}	$120km/h$
σ	0.2	$\boldsymbol{acc}_{\max}$	$(2.m/s^2, 2.m/s^2)$
τ	$50m$	R_{cmax}	$1 \times 10^3 Hz$

This article is an extension of our conference paper [44].

θ_u	0.15	R_{mmax}	$1 \times 10^5 Hz$
η	0.01	Q_{cmax}	$128G$
B	$6.3 \times 10^6 MHz$	Q_{mmax}	$1000T$
N_0	$1 \times 10^{-10} W/Hz$	σ_m	10000
P	$1 \times 10^{-6} W$	σ_{Rc}	100
δ	0.9	σ_c	15
σ_{Rm}	6000	Δt	$1s$
$\boldsymbol{V_c^0}$	$(V_c^0)_{i,1}$ is sampled from a uniform distribution $U(-V_{maxx}, V_{maxx})$ and $(V_c^0)_{i,2}$ is sampled from a uniform distribution $U(-V_{maxy}, V_{maxy})$, where $1 \le i \le N$		
$\boldsymbol{P_c^0}$	$(P_c^0)_{i,1}$ is sampled from a uniform distribution $U(x_l + \tau, x_r - \tau)$ and $(P_c^0)_{i,2}$ is sampled from a uniform distribution $U(y_l + \tau, y_r - \tau)$, where $1 \le i \le N$		
$\boldsymbol{D_e^0}$	$(D_e^0)_i$ is sampled from a normal distribution with an expectation of 1×10^3 bit and a variance of 10		
$\boldsymbol{R_c^0}$	$(R_c^0)_i$ is sampled from a normal distribution with an expectation of 1×10^3 Hz and a variance of 1		
$\boldsymbol{R_m^0}$	$(R_m^0)_i$ is sampled from a normal distribution with an expectation of 1×10^5 Hz and a variance of 1		
$\boldsymbol{Q_c^0}$	$(Q_c^0)_i$ is sampled from a normal distribution with an expectation of 128 G and a variance of 10		
$\boldsymbol{Q_m^0}$	$(Q_m^0)_i$ is sampled from a normal distribution with an expectation of 1000 T and a variance of 100		
$\boldsymbol{acc^0}$	$(acc^0)_i$ is sampled from a normal distribution with an expectation of 2 m/s^2 and a variance of 0.1		

A. *Experiments of The Optimization By SLSQP*

In this section, we evaluate the first solution to the latency minimization problem described in Equation (20). To investigate the optimization effect of the SLSQP algorithm on the proximity detection latency in road networks, we use the Algorithm 1 RNDU to continuously update the road network status for 100 times, and use the SLSQP algorithm to optimize *ob*'s total proximity detection latency at each time moment t. In addition, we explore the effect of some parameter values on the latency optimization results at a single time moment (e.g., the time we randomly select is $t = 0$).

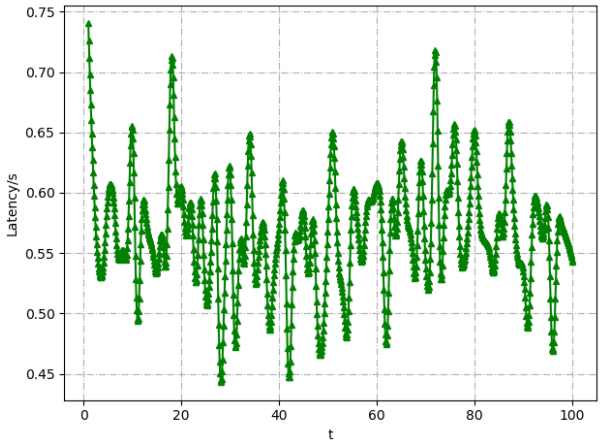

Fig. 9. The total latency optimized by SLSQP w.r.t. time t

Fig. 9 shows the total latency optimized by SLSQP with respect to the time t. It can be seen that during the 100 moments, the total latency optimized by the SLSQP algorithm at each moment fluctuates between 0.43s and 0.74s. The reason for this fluctuation is that although there are a total of 200 users in the road network, the number of users in the same edge server's serving area as *ob*, may be different at each moment, so *ob*'s proximity detection calculation task vector $\boldsymbol{D_e}$ is different, resulting in the optimal value of the total execution latency is different at each moment.

Fig. 10 shows the relationship between the total latency and the number of iterations of the SLSQP algorithm when the number of mobile users is 100, 150, 200, 250, and 300, respectively, at time $t = 0$. It can be seen from Fig. 10 that under the scenario of different number of users, the total latency has dropped significantly in the first several iterations, and in the next few iterations, the total latency decreases slowly, and finally converges to the optimal value. In addition, the convergence speed of the total latency in different mobile user scenarios is different. The convergence speed is the fastest when the number of users is 100 and 150, whose two curves tend to be flat after 5 iterations of optimization. The convergence speed is the slowest when the number of users is 200, whose curve tends to be flat after 10 iterations of optimization. Therefore, in general, SLSQP algorithm optimizes the total latency quickly, and can efficiently solve this optimization problem.

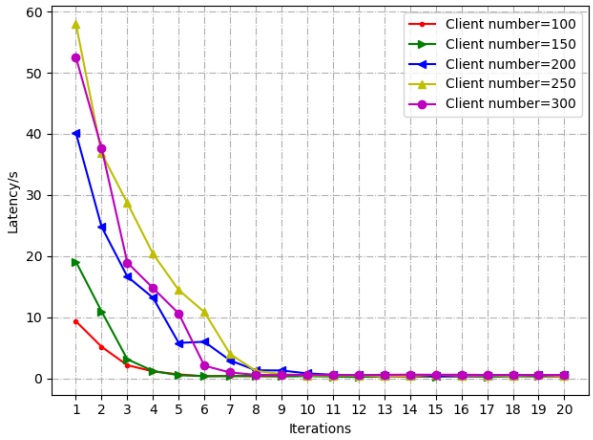

Fig. 10. The latency vs. the number of iterations of the SLSQP algorithm optimization when the number of mobile users' is 100, 150, 200, 250 and 300, respectively at $t = 0$

In the discussion of the last paragraph of Section II.A, we mentioned that when *ob* is in the border serving area of the edge server, the central server needs to be involved. The selection range of the central server is related to V_{max} and T_ε. Therefore, we also study the number of users selected by the central server with respect to these parameters. Fig. 11 is a line chart of the number of mobile users selected by the central server with respect to V_{max} and T_ε when the number of mobile users is 200. From Fig. 11, we can see that the number of users screened out with V_{max}=120km/h, 90km/h and 60km/h first increases and then becomes stable. Because the larger the T_ε and V_{max}, the larger the screening radius of the central server. While T_ε continues increasing, there are no more users, the Euclidean distance between whom and *ob* is within this radius, so the

This article is an extension of our conference paper [44].

second half of each of the three curves is parallel to the horizontal axis. It can be concluded from this that the greater the threshold of user movement speed V_{max}, the more users to be screened by the central server to detect proximity.

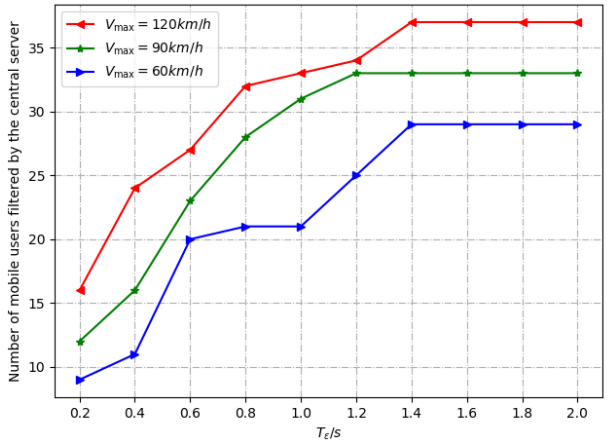

Fig. 11. The number of mobile users selected by the central server w.r.t. V_{max} and T_ε

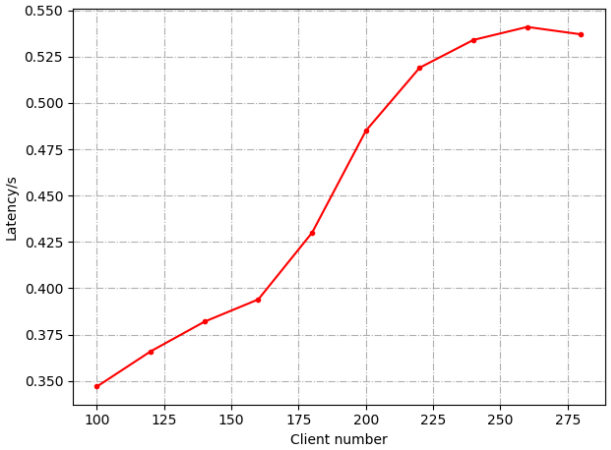

Fig. 12. A line chart of the total time latency of the model and the total number of mobile users

We also explore the relationship between the total latency and the number of users in the road network. Fig. 12 is a line chart of the total optimal latency obtained by SLSQP with respect to the total number of mobile users. As shown in this figure, the total optimal latency increases as the number of users increases. We can also see that the total optimal latency curve increases slowly first, and then increases rapidly when the number of users is around 180, and when the number of users is 260, the total optimal latency starts to decrease.

B. Experiments of The Optimization By DDPG

In this subsection, we evaluate the performance of our DDPG algorithm. Besides TABLE I, some other initial parameter values used by the DDPG algorithm are shown in TABLE II.

TABLE II
PARAMETERS FOR DDPG ALGORITHM

Parameter	Value
$\boldsymbol{a}^0$	**0**
σ_a	0.2
θ_a	0.15
T	$100s$
$episode$	100
M	1000
m	500
lr	10^{-2}
$\beta_1, \beta_2, \beta_3$	0.99

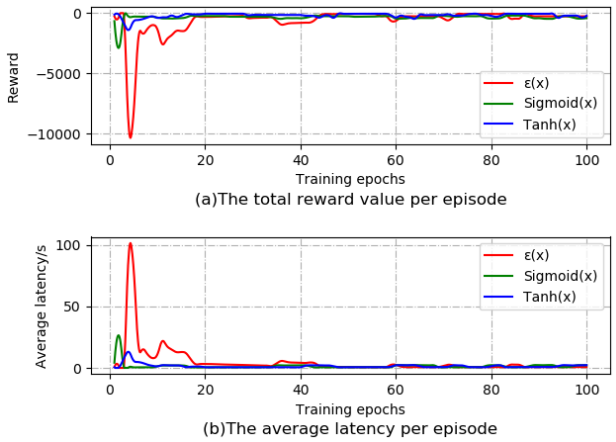

Fig. 13. The change of the total reward value and the average latency at T times with different compressing mapping functions

First, the step function, sigmoid, and tanh are applied to Equation (39) as the compressing mapping function. Fig. 13 shows the experimental results obtained by applying these three kinds of compressing mapping functions, respectively. In Fig. 13, It can be seen from the two subfigures that the three reward functions increase and the latency values decrease with the number of sampling rounds, and finally both of them reach the state of convergence. From the two subfigures in Fig. 13, we find that the step function produces larger oscillations in the earlier stage than the sigmoid and tanh functions, and the oscillation amplitude in the earlier stage of tanh is the smallest. As the number of sampling rounds continues to increase, both their reward function and the later stage of the latency show small amplitude oscillations, and the oscillation amplitude of the step function becomes smaller and smaller. The latency values obtained by applying the three functions finally converge to within 1s, which is the time interval of dynamic environment status update in the road network. In the simulation process, we find that the gradient of the neural network disappears during the training process of applying both the sigmoid function and the tanh function, which causes the local oscillation phenomenon of the neural network training in advance. Therefore, in the subsequent simulation experiments of this article, the step function is used as the compressing mapping function.

The learning rate lr determines the training speed of the actor network and the critic network [40]. A smaller lr causes the network to converge more slowly. The neural network

This article is an extension of our conference paper [44].

parameter update step size per iteration is small, which is easy to fall into the local optimal solution [22][42]. A larger lr makes the network converge fast, and the single update step size of the neural network parameters is large, which may lead to the failure to converge to one of the local optimal solution. Fig. 14 explores the effects of different learning rates on the reward function and the average latency optimization process. The learning rates lr in the figure are set to $10^{-3}, 10^{-2}, 5\times10^{-3}$, respectively. It can be seen that the convergence rate of the reward function and the latency curve is relatively slow when $lr = 10^{-3}$, and the oscillation amplitude of the curve at the later stage is basically the same as that of the other two learning rate values. In contrast, for $lr = 10^{-2}$ and $lr = 5\times10^{-3}$, the latency converges faster in the early stage. When $lr = 10^{-2}$, not only the convergence speed of the reward function and the latency is fast, but also the oscillation amplitude after convergence is also smaller than that of the curve of $lr = 5\times10^{-3}$.

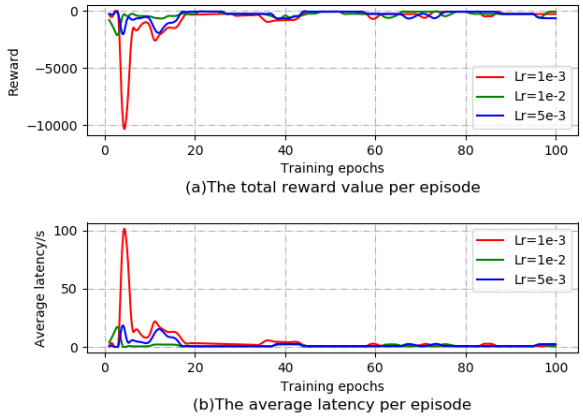

Fig. 14. The change of the total reward value and the average latency at T times with different learning rate

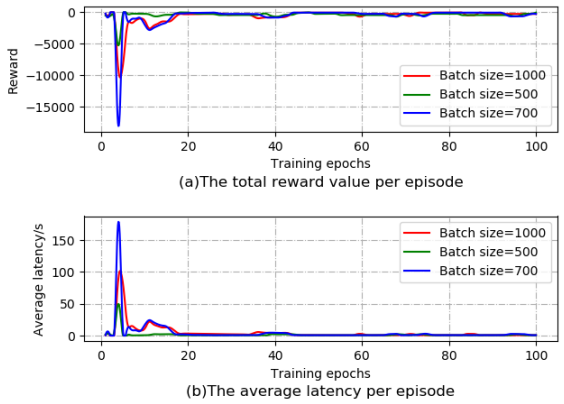

Fig. 15. The change of the total reward value and the average latency at T times with the number of samples with different batch size

As the input of the actor network and the critic network, the size of a single batch of data plays an important role in the training process of neural networks [42]. If the data distribution in a single batch is closer to the actual data distribution, it will produce better training results. If the data volume of a single batch is small, it will not fit the actual data distribution well. If the data volume of a single batch is large, it may lead to a long training time and insufficient computer memory [42]. Fig. 15 shows the convergence results of the reward function and the average latency when the batch size of a single batch of data is 1000, 500 and 700, respectively. It can be seen that the reward function and the average optimal latency have a large oscillation amplitude when the single data batch is 700, and the convergence speed is the fastest and the oscillation amplitude is the smallest when the batch size is 500. When the batch size is 500, the training time of each network is the shortest.

C. Experiments of The Running Time and Performance Comparison

In order to better show the latency optimization effects of the SLSQP algorithm and DDPG algorithm in the dynamic road network, this article uses the Gibbs Sampling algorithm [43] as a baseline method which has no optimization, to obtain the offloading proportional vector at each moment and calculate the latency without any optimization at each time moment. According to the constraints $C2-C4$ in Equation (20), the action vector $\boldsymbol{a}^t$ at time t belongs to the set which is the intersection of the following 4 sets:

$$\boldsymbol{a}^t \in S = S_1 \cap S_2 \cap S_3 \cap S_4 \tag{43}$$

where,

$$S_3: \{\boldsymbol{a}^t | 0 \le a_i^t \le 1, 1 \le i \le N_{Thm}\} \tag{44}$$

$$S_1: \left\{ \boldsymbol{a}^t \left| \sum_{i=1}^{N_{Thm}-1} a_i^t D_{ie} \le \frac{t_s^t}{\frac{1}{R_u^t}+\frac{1}{R_m^t}} \right. \right\} \tag{45}$$

$$S_2: \left\{ \boldsymbol{a}^t \left| \sum_{i=1}^{N_{Thm}-1} a_i^t D_{ie} \le Q_m^t \right. \right\} \tag{46}$$

$$S_4: \left\{ \boldsymbol{a}^t \left| \sum_{i=1}^{N_{Thm}-1} (1-a_i^t) D_{ie} \le Q_c^t \right. \right\} \tag{47}$$

In order to make the sampling space more complete, all sample points in the sampling space are given the same sampling probability. We suppose that the posterior probability follows a uniform distribution. By combining Equations (43) to (47), the Gibbs Sampling algorithm is shown in Algorithm 4. At time t, we use $\alpha_1^t, \ldots, \alpha_{i-1}^t$ which are generated by sampling at the current timestamp, and $\alpha_{i+1}^{t-1}, \ldots \alpha_{N_{Thm}}^{t-1}$ which are sampled at time $t-1$ to obtain α_i^t by solving all inequalities about α_i^t in Equations (43) to (47).

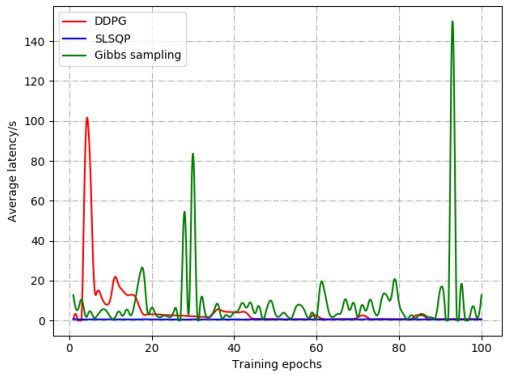

Fig. 16. The average latency of Gibbs Sampling, DDPG algorithm and SLSQP algorithm at each epoch

As shown in Fig. 16, the latency oscillation amplitude calculated by Gibbs sampling is the largest. The reason is that no optimization algorithm is applied during sampling, and the random vector is sampled only by a uniformly distributed

Algorithm 4: RoadNetwork Gibbs Sampling Algorithm

Input: $\boldsymbol{V}_c^0, \boldsymbol{P}_c^0, \boldsymbol{acc}^0, \boldsymbol{R}_{cv}^0, \boldsymbol{Q}_{cv}^0, \boldsymbol{R}_{mv}^0, \boldsymbol{Q}_{mv}^0, \boldsymbol{\Sigma}_c, \boldsymbol{\Sigma}_m, \boldsymbol{D}_v^{0\,T}, D_{mec}^0, D_{local}^0, x_l, x_r, y_l, y_r, \boldsymbol{acc}_{max} = (acc_x, acc_y), V_{max}, ob, \tau, \mu, \sigma, \eta, \Delta t, \theta_u, \boldsymbol{a}^0$

Output: $T_{avg}^1, T_{avg}^2, \dots, T_{avg}^T$

1: For $episode = 1{:}T$ **do**

2: $\boldsymbol{a}^t = []$

3: $Set\ the\ average\ latency\ of\ all\ moments\ in\ an\ episode\ T_{avg}^{episode} = 0$

4: For $t = 1{:}T$ **do**

5: **For** $i = 1{:}N_{Thm}$ **do**

6: $a_i^t\ is\ sampled\ from\ the\ uniform\ distribution$:

7: $a_i^t \sim U(\max\left\{0, 1 - \frac{Q_c^t - \sum_{k=1}^{i-1} a_k^t D_{ke}^t - \sum_{k=i+1}^{N_{Thm}} a_k^{t-1} D_{ke}^t}{D_{ie}^t}\right\}, \min\{1, \frac{\left(\frac{t_s^t}{\frac{1}{R_u^t} + \frac{1}{R_m^t}}\right) - \sum_{k=1}^{i-1} a_k^t D_{ke}^t - \sum_{k=i+1}^{N_{Thm}} a_k^{t-1} D_{ke}^t}{D_{ie}^t}, \frac{Q_m^t - \sum_{k=1}^{i-1} a_k^t D_{ke}^t - \sum_{k=i+1}^{N_{Thm}} a_k^{t-1} D_{ke}^t}{D_{ie}^t}\})$

8: Add a_i^t to $\boldsymbol{a}^t$

9: **End for**

10: Calculate the latency T_{per}^t at time t by $\boldsymbol{a}^t = \left[a_1^t, a_2^t, \dots, a_{N_{Thm}}^t\right]^T$

11: Update the average latency of all monments in an episode $T_{avg}^{episode} = \frac{(t-1)T_{avg}^{episode} + T_{per}^t}{t}$

12: Get the $\boldsymbol{V}_c^{t+1}, \boldsymbol{P}_c^{t+1}, \boldsymbol{acc}^{t+1}, \boldsymbol{R}_{cv}^{t+1}, \boldsymbol{Q}_{cv}^{t+1}, \boldsymbol{R}_{mv}^{t+1}, \boldsymbol{Q}_{mv}^{t+1}$ through

$\boldsymbol{RNDU}(\boldsymbol{V}_c^t, \boldsymbol{P}_c^t, \boldsymbol{acc}^t, \boldsymbol{R}_{cv}^t, \boldsymbol{Q}_{cv}^t, \boldsymbol{R}_{mv}^t, \boldsymbol{Q}_{mv}^t, \boldsymbol{\Sigma}_c, \boldsymbol{\Sigma}_m, D_{mec}^t, D_{local}^t, x_l, x_r, y_l, y_r, \boldsymbol{acc}_{max}, V_{max}, ob, \tau, \mu, \sigma, \eta, \theta_u, \Delta t)$

13: **End for**

14: End for

posterior. In the process of the DDPG algorithm optimization, due to *ob*'s incomplete early training, the latency is higher than the optimization result of the SLSQP algorithm in the early stage, but the latency of the DDPG algorithm optimization in the subsequent stage is not much different from that of the SLSQP algorithm optimization. The reason why latency optimization result by the DDPG algorithm in the later stage will fluctuate in some places is that *ob* performs asynchronous strategy exploration through Ornstein–Uhlenbeck process by Equation (36). When the strategy executed and the strategy exploring is not as good as the original strategy, it will continue to execute the original strategy. In addition, in the simulation process, the optimization speed of the DDPG algorithm is significantly faster than that of the SLSQP algorithm, because the action decisions made by the DDPG algorithm at a previous time epoch has a strong correlation with the decision made at a later time epoch.

Fig. 17 shows the average running time comparison of DDPG algorithm and SLSQP algorithm at each epoch. The average running time of the DDPG algorithm at each sampling episode is within 1s each time, and is relatively stable, while the running time of the SLSQP algorithm is between 40s and 60s, and fluctuates greatly. The reason is that the DDPG algorithm only performs gradient descent update once each time, and the SLSQP algorithm has different iteration times with different subtask vectors input each time. As can be seen from Fig. 16, after epoch 44, the effect of actor network decision-making in the DDPG calculation tends to be stable, and the latency occasionally increases due to exploration.

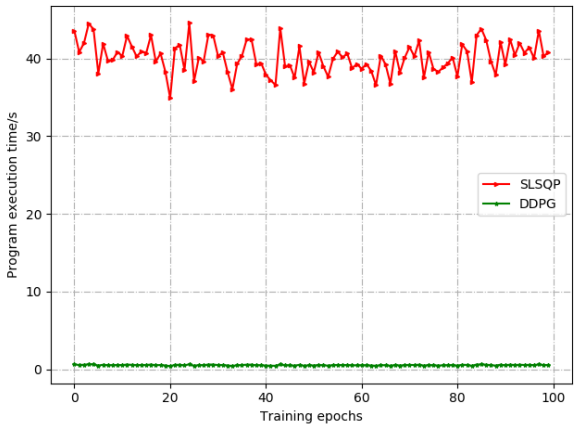

Fig.17. The average running time comparison of DDPG algorithm and SLSQP algorithm at each epoch

Fig. 18 shows the difference between the optimal latency obtained by the DDPG algorithm and SLSQP algorithm ($t_{DDPG} - t_{SLSQP}$) from epoch 44 to epoch 100. By comparing Fig. 17 and Fig. 18, it can be seen that the DDPG algorithm greatly reduces the algorithm execution time by reducing the number of optimization iterations in each epoch. However, the difference between the total latency after the optimization of the DDPG algorithm and the total latency after the optimization of the SLSQP algorithm in most epochs is basically within 0.2s.

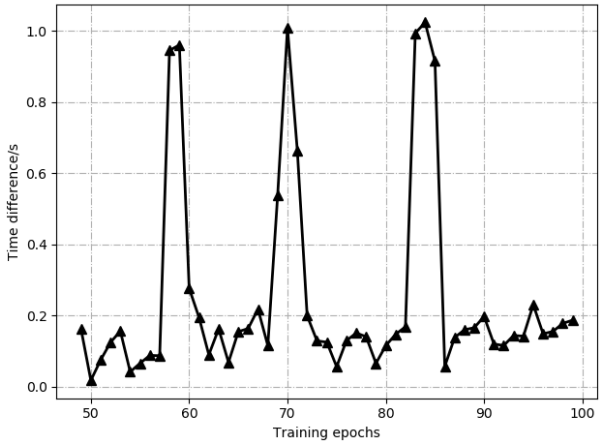

Fig.18. Difference between DDPG algorithm and SLSQP algorithm optimization results from epoch 44 to epoch 100

VI. CONCLUSIONS

In this article, we apply MEC to the latency optimization problem for proximity detection in dynamic road networks. We first propose a computation offloading scheme for this problem. Then, both the convex optimization method, i.e., the SLSQP algorithm, and the deep reinforcement learning method, i.e., DDPG algorithm, are used to solve the latency optimization problem. Experiments show that the SLSQP algorithm can effectively optimize the latency of each moment in the dynamic road network, but the optimization process consumes longer time than the DDPG algorithm. The DDPG algorithm is more efficient than the SLSQP algorithm because of its strong memory and high representation of the internal neural network structure, and after a certain number of optimizations, the optimized latency value of the DDPG algorithm can be very close to the optimized latency value of the SLSQP algorithm.

ACKNOWLEDGEMENT

This work is supported in part by the National Natural Science Foundation of China (Grant No. 61901052), in part by the 111 project (Grant No. B17007), and in part by the Director Funds of Beijing Key Laboratory of Network System Architecture and Convergence (Grant No. 2017BKL-NSAC-ZJ-02).

REFERENCES

[1] Y. Liu, M. Peng, G. Shou. "Mobile Edge Computing-Enhanced Proximity Detection in Time-Aware Road Networks." IEEE Access, 2019, 7: 167958-167972.

[2] A. Amir, A. Efrat, J. Myllymaki, et al. "Buddy tracking—efficient proximity detection among mobile friends." Pervasive and Mobile Computing, 2007, 3(5): 489-511.

[3] M. L. Yiu, S. Šaltenis, K. Tzoumas. "Efficient proximity detection among mobile users via self-tuning policies." Proceedings of the VLDB Endowment, 2010, 3(1-2): 985-996.

[4] A. Tzanakaki, M.P. Anastasopoulos, G.S. Zervas, B.R. Rofoee, R. Nejabati, and D.Simeonidou. "Virtualization of heterogeneous wireless optical network and IT infrastructures in support of cloud and mobile cloud services." IEEE Communications Magazine, vol. 51, no. 8, pp. 155-161, August 2013.

[5] H. T. Dinh, C. Lee, D. Niyato, and P. Wang. "A Survey of mobile cloud computing: Architecture, applications, and approaches." Wireless Communications and Mobile Computing, 13(18), pp. 1587-1611, 2013.

[6] A. Checko et al. "Cloud RAN for Mobile Networks—A Technology Overview." IEEE Communications Surveys & Tutorials, vol. 17, pp. 405-426, 2015.

[7] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya. "Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges." IEEE Communications Surveys & Tutorials, vol. 16, pp. 369-392, 2014.

[8] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. "The case for VM-based cloudlets in mobile computing." Pervasive Computing, IEEE, vol. 8, pp.14-23, 2009.

[9] B. P. Rimal, D. P. Van, and M. Maier. "Mobile-Edge Computing Empowered Fiber-Wireless Access Networks in the 5G Era." IEEE Communications Magazine, vol. 55, no. 2, pp. 192-200, February 2017.

[10] P. Yang, L. Li, W. Liang, et al. "Latency Optimization for Multi-user NOMA-MEC Offloading Using Reinforcement Learning."/2019 28th Wireless and Optical Communications Conference (WOCC). IEEE, 2019: 1-5.

[11] H. Wang, X. Li, H. Ji, et al. "Federated offloading scheme to minimize latency in MEC-enabled vehicular networks."2018 IEEE Globecom Workshops (GC Wkshps). IEEE, 2018: 1-6.

[12] D. Kraft. "A software package for sequential quadratic programming." Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt, 1988.

[13] Uhlenbeck G E, Ornstein L S. On the theory of the Brownian motion[J]. Physical review, 1930, 36(5): 823.

[14] Barndorff-Nielsen O E, Shephard N. Non-Gaussian Ornstein–Uhlenbeck-based models and some of their uses in financial economics[J]. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 2001, 63(2): 167-241.

[15] Wang, Yanting, et al. "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling." IEEE Transactions on Communications 64.10 (2016): 4268-4282.

[16] Mazza, Daniela, Daniele Tarchi, and Giovanni E. Corazza. "A partial offloading technique for wireless mobile cloud computing in smart cities." 2014 European Conference on Networks and Communications (EuCNC). IEEE, 2014.

[17] Woldemichael D E, Woldeyohannes A D. OPTIMIZATION OF PRESSURE VESSEL DESIGN USING PYOPT[J]. 2006.

[18] Zahery M, Maes H H, Neale M C. CSOLNP: Numerical Optimization Engine for Solving Non-linearly Constrained Problems[J]. Twin Research and Human Genetics, 2017, 20(4): 290-297.

[19] Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning[J]. arXiv preprint arXiv:1509.02971, 2015.

[20] Nachum O, Norouzi M, Xu K, et al. Trust-pcl: An off-policy trust region method for continuous control[J]. arXiv preprint arXiv:1707.01891, 2017.

[21] Silver D, Lever G, Heess N, et al. Deterministic policy gradient algorithms[C]. 2014.

[22] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. Nature, 2015, 518(7540): 529-533.

[23] Mohammadi M, Al-Fuqaha A, Guizani M, et al. Semisupervised deep reinforcement learning in support of IoT and smart city services[J]. IEEE Internet of Things Journal, 2017, 5(2): 624-635.

[24] Pan, Jie, et al. "Multisource transfer double DQN based on actor learning." IEEE transactions on neural networks and learning systems 29.6 (2018): 2227-2238.

[25] Konda V R, Tsitsiklis J N. Actor-critic algorithms[C]//Advances in neural information processing systems. 2000: 1008-1014.

[26] Van Hasselt H, Guez A, Silver D. Deep reinforcement learning with

This article is an extension of our conference paper [44].

double q-learning[C]//Thirtieth AAAI conference on artificial intelligence. 2016.
[27] Goodfellow I, Bengio Y, Courville A. Deep learning[M]. MIT press, 2016.
[28] Yan L C, Bengio Y S, Hinton G. Deep learning[J]. nature, 2015, 521(7553): 436-444.
[29] Wang, Yu, et al. "Data-driven deep learning for automatic modulation recognition in cognitive radios." IEEE Transactions on Vehicular Technology 68.4 (2019): 4074-4077.
[30] Amestoy P R, Guermouche A, L'Excellent J Y, et al. Hybrid scheduling for the parallel solution of linear systems[J]. Parallel computing, 2006, 32(2): 136-156.
[31] Ernemann C, Hamscher V, Schwiegelshohn U, et al. On advantages of grid computing for parallel job scheduling[C]//2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02). IEEE, 2002: 39-39.
[32] Wan L, Zeiler M, Zhang S, et al. Regularization of neural networks using dropconnect[C]//International conference on machine learning. 2013: 1058-1066.
[33] Ito Y. Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory[J]. Neural Networks, 1991, 4(3): 385-394.
[34] Fan E. Extended tanh-function method and its applications to nonlinear equations[J]. Physics Letters A, 2000, 277(4-5): 212-218.
[35] Sharma S. Activation functions in neural networks[J]. Towards Data Science, 2017, 6.
[36] LeCun Y, Boser B, Denker J S, et al. Backpropagation applied to handwritten zip code recognition[J]. Neural computation, 1989, 1(4): 541-551.
[37] LeCun Y, Bengio Y. Convolutional networks for images, speech, and time series[J]. The handbook of brain theory and neural networks, 1995, 3361(10): 1995.
[38] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv:1312.5602, 2013.
[39] Sutton R S, Barto A G. Reinforcement learning: An introduction[M]. MIT press, 2018.
[40] Jin W, Li Z J, Wei L S, et al. The improvements of BP neural network learning algorithm[C]//WCC 2000-ICSP 2000. 2000 5th international conference on signal processing proceedings. 16th world computer congress 2000. IEEE, 2000, 3: 1647-1649.
[41] Wang Z, Schaul T, Hessel M, et al. Dueling network architectures for deep reinforcement learning[J]. arXiv preprint arXiv:1511.06581, 2015.
[42] Li M, Zhang T, Chen Y, et al. Efficient mini-batch training for stochastic optimization[C]//Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. 2014: 661-670.
[43] Carter C K, Kohn R. On Gibbs sampling for state space models[J]. Biometrika, 1994, 81(3): 541-553.
[44] Song, Yunlong, et al. "Latency Optimization for Mobile Edge Computing Based Proximity Detection in Road Networks." 2020 IEEE/CIC International Conference on Communications in China (ICCC Workshops). IEEE, 2020.

This article is an extension of our conference paper [44].