A Reinforcement Learning Based Task Offloading Scheme for Vehicular Edge Computing Network

Jie Zhang[1], Hongzhi Guo[2], and Jiajia Liu[2(✉)]

[1] School of Cyber Engineering, Xidian University, Xi'an 710071, Shaanxi, China
[2] School of Cybersecurity, Northwestern Polytechnical University, Xi'an 710072, Shaanxi, China
liujiajia@nwpu.edu.cn

Abstract. Recently, the trends of automation and intelligence in vehicular networks have led to the emergence of intelligent connected vehicles (ICVs), and various intelligent applications like autonomous driving have also rapidly developed. Usually, these applications are compute-intensive, and require large amounts of computation resources, which conflicts with resource-limited vehicles. This contradiction becomes a bottleneck in the development of vehicular networks. To address this challenge, the researchers combined mobile edge computing (MEC) with vehicular networks, and proposed vehicular edge computing networks (VECNs). The deploying of MEC servers near the vehicles allows compute-intensive applications to be offloaded to MEC servers for execution, so as to alleviate vehicles' computational pressure. However, the high dynamic feature which makes traditional optimization algorithms like convex/non-convex optimization less suitable for vehicular networks, often lacks adequate consideration in the existing task offloading schemes. Toward this end, we propose a reinforcement learning based task offloading scheme, i.e., a deep Q learning algorithm, to solve the delay minimization problem in VECNs. Extensive numerical results corroborate the superior performance of our proposed scheme on reducing the processing delay of vehicles' computation tasks.

Keywords: Vehicular edge computing networks · Mobile edge computing · Reinforcement learning

1 Introduction

In recent years, automation and intelligence have caused extensive discussion in vehicular networks, which has led automakers such as Tesla and Mercedes-Benz

Supported by the National Natural Science Foundation of China (61771374, 61771373, 61801360, and 61601357), in part by China 111 Project (B16037), and in part by the Fundamental Research Fund for the Central Universities (JB171501, JB181506, JB181507, and JB181508).

S. Han et al. (Eds.): AICON 2019, LNICST 287, pp. 438–449, 2019.
https://doi.org/10.1007/978-3-030-22971-9_38

to focus on the research of intelligent connected vehicles (ICVs) [1]. By deploying various sensors (cameras, lidars, millimeter-wave radars, etc.) on the vehicles, an ICV can collect massive data about the information on road traffic and other ICVs. With these information, the ICV can perform a variety of intelligent applications, including autonomous driving, intelligent path planning, in-vehicle remote services, etc. However, the operations of these compute-intensive applications are often accompanied with enormous computation resources consumption, and correspondingly, most of the current vehicles are limited in computation resources, which makes them unable to guarantee the quality of service (QoS) requirements in terms of low latency. This contradiction between resources-limited vehicles and compute-intensive applications becomes a bottleneck in the development of vehicular networks [2].

To meet this challenge, cloud-based vehicular networks which introduce cloud computing into vehicular networks have been seen as an effective solution [3]. In such networks, vehicles' compute-intensive applications can be executed locally on the vehicles' CPUs or offloaded to a remote cloud server, so as to reduce the processing delay. It is undeniable that the introduction of cloud computing has alleviated the computational pressure of vehicles. However, considering that cloud servers are usually located away from vehicles, the data transmission delay over wide area network may result in severely reduced offloading efficiency. In order to address this deficiency, mobile edge computing offloading (MECO) has been considered as a new promising network paradigm [4,5], and has prompted the emergence of vehicular edge computing networks (VECNs). In VECNs, we can provide additional computation resources for vehicles by deploying MEC servers in road side units (RSUs) close to the road. Specifically, a vehicle moving on the road can offload its computation task which is not suitable for local execution to the MEC servers via a road side unit, so as to achieve lower processing delay.

Since its appearance, VECNs have attracted a lot of researchers' attention, and there has been many research works [6–8]. For example, Zhang *et al.* [6] studied the task offloading problem under delay constraints in cloud-enabled vehicular networks, and adopted contract theory to reduce the energy consumption of computation tasks. Further, in [9], they proposed a cloud-based task offloading framework in order to reduce offloading latency and transmission cost of the computation tasks. In their framework, they considered the effectiveness of vehicle-to-infrastructure (V2I) communications and vehicle-to-vehicle (V2V) communications, and proposed a predictive offloading scheme to solve the problem. Liu *et al.* [7] proposed a distributed computing offloading algorithm in order to reduce the processing delay of the computation tasks in vehicular networks. The study conducted by Dai *et al.* [8] investigated the multi-user and multi-server task offloading problem in vehicular networks, and they described it as a mixed integer nonlinear programming problem. To solve this problem, they divided the problem into two sub-problems, and proposed an optimization algorithm to jointly optimize the options of MEC servers and computing offloading.

It is obvious that the existing MECO schemes in VECNs are usually to formalize task offloading problem as an optimization problem and adopt appropriate optimization algorithms, i.e., game-based algorithm and contract-based algorithm, to obtain optimal or near-optimal solution [10]. However, due to fast-moving vehicles and constantly changing communications environment, it is difficult to obtain time-varying optimal solution using traditional optimization algorithms in high dynamic vehicular networks. In such case, machine learning, especially reinforcement learning which can interact with an unknown environment, adapt to environmental changes and take appropriate actions, seems to provide a good solution to this problem.

Recently, a lot of research works have focused on the application of reinforcement learning in vehicular networks [11–13]. For example, Zheng *et al.* [11] proposed a two-stage delay-optimal dynamic virtual radio scheduling scheme, which took channel and queue status information into account. In this paper, the virtual radio resource management was formulated as a partially observed Markov Decision Process (MDP) and then solved by an online distributed learning method. However, for task offloading problem in VECNs, there are currently not much research works using reinforcement learning. In this paper, we propose a reinforcement learning based task offloading scheme to solve the task offloading problem in VECNs, with the goal of minimizing processing delay of vehicles' computation tasks. In particular, the main contributions of this paper are as follows:

- In this paper, we propose a cloud enabled task offloading architecture in VECNs, in which remote cloud server acts as a backup server. In such scenario, we study the task offloading problem with the goal of minimizing processing delay of all the tasks.
- To solve the task offloading problem in our scenario, a reinforcement learning based scheme, i.e., a deep Q learning algorithm, is proposed.
- To verify the performance of our proposed schemes, we conduct a series of simulation experiments, and the numerical results we present demonstrate that our scheme can achieve superior performance on processing delay reduction.

The rest of this paper is organized as follows. In Sect. 2, we describe our proposed cloud enabled task offloading architecture and the system model. Section 3 gives the definition of the task offloading problem in our scenario and our proposed solution. After that, we present our numerical results in Sect. 4. Finally, Sect. 5 concludes the whole paper.

2 System Model

As shown in Fig. 1, there is an unidirectional straight road with N vehicles moving at a constant speed v. Moreover, M road side units (RSUs) are placed next to the road. and each RSU covers a certain range and is connected to a MEC server in order to provide edge computing service to the vehicles. Meanwhile, the remote cloud server thousands of miles away acts as a backup server so as to provide

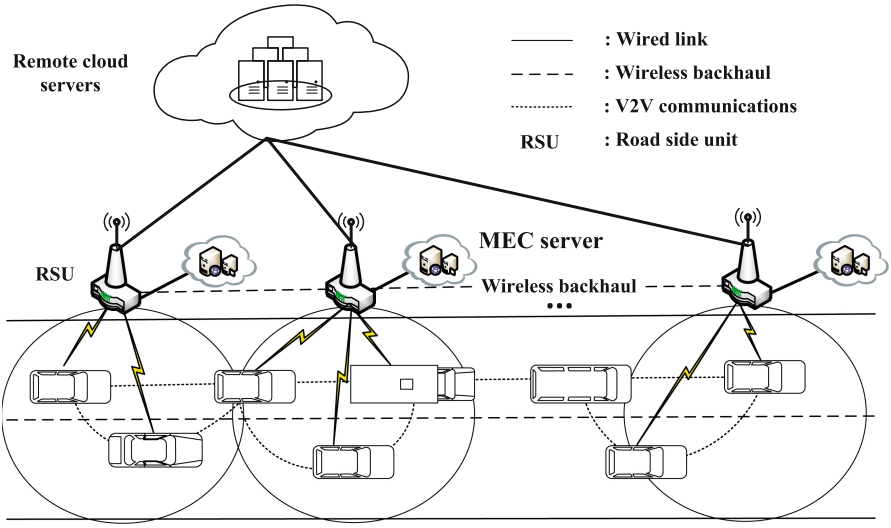

Fig. 1. Cloud enabled task offloading architecture in VECNs.

more computation resources for the vehicles when neither the MEC servers nor the vehicles themselves guarantee the processing delay requirements of the computation tasks. Therefore, for each vehicle, its computation task can be offloaded to the MEC servers or the remote cloud server for lower processing delay. Moreover, there are two modes for the data transmission, i.e., vehicle-to-vehicle (V2V) communications and vehicle-to-infrastructure (V2I) communications.

In order to facilitate the specific description of the problem, we use $\mathcal{N} = \{1, 2, 3, ..., N\}$ and $\mathcal{M} = \{1, 2, 3, ..., M\}$ denote the sets of the vehicles and the MEC servers connected to the RSUs, respectively. At the same time, we assume that only one computation task needs to be accomplished for each vehicle, and its characteristics can be described by four variables: the input data size m_i^{in}, the output data size m_i^{out}, the required CPU cycles to accomplish the task C_i, and the maximum permissible processing delay of the task D_i. So we denote the task of vehicle i as $T_i = \{m_i^{in}, m_i^{out}, C_i, D_i\}$.

According to the foregoing, there are $M+2$ task offloading decisions for each vehicle, i.e, executing its computation task locally on the vehicle's own CPU, offloading its task to the MEC server connected to RSU j, and offloading its task to the remote cloud server. The task offloading decisions of vehicle i can be denoted as $d_i \in \{0, -1, 1, 2, 3, ..., M\}$, where $d_i = 0$ means that vehicle i decides to execute its computation task locally on its own CPU, $d_i = -1$ is that vehicle i decides to offload its computation task to the remote cloud server, and $d_i = j\,(1 \leq j \leq M)$ means that vehicle i decides to offload its computation task to the MEC server connected to RSU j. After that, for different task offloading decisions, the specific system models are given by the following subsections.

2.1 Executing Locally

While $d_i = 0$, vehicle i decides to accomplish its computation task locally on its own CPU. In such case, the processing time of the computation task can be calculated as

$$t_i^L = C_i / F_L, \tag{1}$$

where F_L is the computation capacity of vehicle i, and we assume that all vehicles have the same computation capacities for simplicity.

2.2 Offloading to the MEC Servers Connected to the RSUs

If vehicle i decides to offload its computation task to the MEC server connected to RSU j, i.e., $d_i = j\,(1 \leq j \leq M)$, the processing time of the computation task mainly consist of the following parts: the execution time of the task on the MEC server, the transmitting time of the input data from the vehicle to the MEC server, and the transmitting time of the output data from the MEC server to the vehicle. Then the processing time in such case can be given as

$$t_i^{MEC} = C_i / F_M + t_i^{in,trans} + t_i^{out,trans}. \tag{2}$$

Here, the first item is the execution time of the computation task on the MEC server, and F_M is the computation capacity of the MEC server. Also, we assume that all MEC servers have the same computation capacities for simplicity. $t_i^{in,trans}$ and $t_i^{out,trans}$ are the transmitting time of the input data and output data, respectively.

For $t_i^{in,trans}$, since the vehicle chooses the MEC server connected to the RSU within its current communication range, its value can be given by

$$t_i^{in,trans} = m_i^{in} / r_{i,j}^{V2I}, \tag{3}$$

where $r_{i,j}^{V2I}$ is the data transmitting rate between vehicle i and RSU j, and it can be calculated as

$$r_{i,j}^{V2I} = \omega_1 \cdot log_2(1 + \frac{p_{i,j} \cdot g_{i,j}^{V2I}}{\sigma^2 + I_{i,j}}). \tag{4}$$

Here, ω_1 is the channel bandwidth between vehicle i and RSU j, $p_{i,j}$ is the transmit power of vehicle i, $g_{i,j}^{V2I}$ is the channel gain between vehicle i and RSU j, σ^2 is the background noise power, and $I_{i,j}$ denotes the interference at RSU j.

For $t_i^{out,trans}$, due to the different communications modes, it has different expression equations:

$$t_i^{out,trans} = \begin{cases} m_i^{out}/c + m_i^{out}/r_{k,i}^{V2I}, & V2I, \\ m_i^{out}/r_{j,l}^{V2I} + m_i^{out}/r_{l,l+1}^{V2V} + ... + m_i^{out}/r_{l+n,i}^{V2V}, & V2V, \end{cases} \tag{5}$$

where c is the data transmitting rate between the RSUs, and $r_{i,j}^{V2V}$ is the data transmitting rate of V2V communications, which can be calculated as

$$r_{i,j}^{V2V} = \omega_2 \cdot log2(1 + \frac{p_{i,j} \cdot g_{i,j}^{V2V}}{\sigma^2 + A_0 \cdot L^{-2}}). \tag{6}$$

Here, ω_2 and $g_{i,j}^{V2V}$ are the channel bandwidth and channel gain between vehicle i and vehicle j respectively, σ^2 is the background power noise, A_0 is a constant parameter, and L is the distance between vehicle i and vehicle j.

2.3 Offloading to the Remote Cloud Server

If $d_i = -1$, vehicle i decides to offload its computation task to the remote cloud server. In such case, the processing time of the computation task mainly consists the following parts: the transmitting time of the input data from vehicle i to RSU j, the transmitting time of the input data from RSU j to the remote cloud server, and the transmitting time of the output data from RSU k to vehicle i. We ignore the execution time of the computation task because of the powerful computing power of cloud server. Then the processing time of the task can be calculated as

$$t_i^{RCS} = m_i^{in}/r_{i,j}^{V2I} + m_i^{in}/c^{'} + m_i^{out}/r_{k,i}^{V2I}, \tag{7}$$

where $c^{'}$ is the data transmission rate between the RSUs and the remote cloud server.

According to the system model in this section, the processing time of vehicle i's computation task can be given as

$$t_i(d_i) = \begin{cases} t_i^L, & if\ d_i = 0, \\ t_i^{MEC}, & if\ d_i = j,\ 1 \le j \le M, \\ t_i^{RCS} & if\ d_i = -1. \end{cases} \tag{8}$$

3 Problem Formulation and Solution

In this section, we present the definition and formal description of the processing time minimization problem, and propose our solution to this problem.

3.1 Problem Formulation

As discussed in Sect. 2, there are $M+2$ task offloading decisions for each vehicle, i.e., executing its computation task locally on its own CPU, offloading its task to the MEC server connected to RSU j, and offloading its task to the remote cloud server. Due to different computation capacities and transmission efficiency, each type of offloading decisions for the vehicle results in different processing time of its task. Our goal is to minimize the total processing time of all the computation tasks while satisfying their maximum permissible processing time. Specifically, the problem can be given by the following definition.

Definition 1. ***Processing Time Minimization under Delay Constraints (PTMDC):*** *In vehicular edge computing networks, given the status of the vehicles (driving speed, information of computation tasks, wireless channel status, etc.), computation capacities of the MEC servers, and maximum permissible processing*

time of the tasks. The PTMDC problem is to find an optimal offloading decisions profile so as to minimize the total processing time of all the computation tasks while satisfying the maximum permissible processing time constraints.

After that, the PTMDC problem can be formulated as

$$\begin{aligned}
&\mathbf{P1}: \min_{\{d_i\}} \sum_{i=1}^{N} t_i(d_i) \\
&s.t.\ \ C1: t_i^L \cdot I_{\{d_i=0\}} + t_i^{MEC} \cdot I_{\{d_i=j\}} + t_i^{RCS} \cdot I_{\{d_i=-1\}} \\
&\qquad \leq t_i^{max},\ \forall\, i \in \mathcal{N}, j \in \mathcal{M}, \\
&\qquad C2: \sum_{i=1}^{N} I_{\{d_i=j\}} \leq K, \forall\, i \in \mathcal{N}, j \in \mathcal{M}, \\
&\qquad C3: d_i \in \{0, -1, 1, 2, ..., M\},\ \forall\, i \in \mathcal{N},
\end{aligned}$$

where $I_{\{*\}}$ is an indicator function, and $I_{\{*\}} = 1$ while $*$ is true. K is the number constraint of the wireless channels in RSU j.

3.2 Solution

To solve the problem of PTMDC, we first adopt an enumeration algorithm proposed in [14], in which we enumerate all possible task offloading decisions profiles and choose the profile with minimum processing time. This algorithm can obtain the optimal solution to the PTMDC problem. However, the computational complexity of this algorithm is exponential, so it is not suitable for complex scenes with a large number of vehicles. Therefore, we further propose a reinforcement learning based task offloading scheme, i.e., deep Q learning algorithm, to solve the problem.

Reinforcement learning is a large class in the family of machine learning, and Deepmind's achievements in AlphaGo bring it to the mainstream of artificial intelligence. In the framework of reinforcement learning, agents interact with environment to learn what actions can be taken to maximize long-term rewards in a given environment. This mechanism of reinforcement learning makes its good performance in the dynamic environment of the vehicular networks. In order to solve the PTMDC problem, we propose a deep Q learning algorithm. Next, we will give the details of the algorithm.

Deep Q learning algorithm is an extension of Q learning algorithm. For Q learning algorithm, there are three main elements, i.e., environment state, action, and reward. Specifically, we define these three elements of our algorithm as follows:

– State: we define the system state as the total processing time of the computation tasks $S(t) = \sum_i t_i$, where t_i is the processing time of vehicle i's computation task.

Algorithm 1. Deep Q learning algorithm (DQL).

Input: the initial state information of the environment.
Output: the total processing time and offloading decisions profile.
Initialize: the replay memory D;
the main Q network with random weight δ;
the target Q network with $\delta^- = \delta$;

for each episode **do**
  get the initial state S_1;
  for $t = 1, 2, 3, ..., T$ **do**
    choose a random probability p;
    if $p \leq \epsilon$ **then**
      randomly select an action A_t;
    else
      $A_t = arg\max_a Q(S, A, \delta)$;
    end if
    execute action A_t, obtain the reward R_t, and the next state S_{t+1};
    store the experience (S_t, A_t, R_t, S_{t+1}) into the replay memory D;
    get a batch of U samples S_i, A_i, R_i, S_{i+1} from the replay memory;
    calculate the target Q value Q_i^- from the target deep Q network:
      $Q_i^- = R_i + \epsilon Q(S_{i+1}, arg\max_{A'} Q(S_{i+1}, A'; \delta); \delta^-)$;
    update the main deep Q network by minimizing the loss:
      $L_i(\delta) = \frac{1}{U} \cdot \sum [Q_i^- - Q(S_i, A_i; \delta)]^2$;
    perform a gradient descent step on $L(\delta)$;
    every G steps, update the target deep Q network weight with rate α:
      $\delta^- = \alpha\delta + (1 - \alpha)\delta^-$;
  end for
end for

- Action: the set of tasks' offloading decisions $A(t) = \{d_1(t), d_2(t), ..., d_i(t)\}$.
- Reward: the action taken by the agent in each step will receive certain rewards or punishments, we use a reward function to represent it. Specifically, it can be calculated as $(T_L - T)/T_L$, where T_L is the total processing time while all vehicles decide to execute their tasks on their own CPUs, and T is the total processing time by taking selected action in the current state.

The core of Q learning is the matrix Q-table, and the rows and columns of Q-table represent the values of states and actions, respectively. The value of Q-table $Q(S, A)$ measures how well the action taken by the current state is. In the training process of Q learning, we use the Bellman Equation to update the Q-table [15]. However, the states may be infinite in real situations, which causes the Q-table to be very large. To deal with this problem, we implement Q-table via neural networks. Specifically, we use the states and actions as the input of neural networks, and then estimate Q value via neural networks. The combination of neural networks and Q learning is called deep Q learning, and we design our scheme based on this. The neural networks in our scheme contain the main Q network and the target Q network, with weights δ and δ^-, and the main Q network is used to get the current Q value $Q(S, A, \delta)$ while the role of the

target Q network is to obtain the target Q value $Q(S, A, \delta^{-})$. We update the main Q network by minimizing the loss $L_i(\delta)$ and approximate the current Q value. Meanwhile, we calculate the target Q value from the target deep Q network, and update the target deep Q network weight every G steps. Moreover, due to the stability problem of the correlation between states, we store the experience (S_t, A_t, R_t, S_{t+1}) of the current training episode to the replay memory D, and randomly sample the mini-batch from D to update the parameters of the neural networks. The details of the deep Q learning algorithm are shown in Algorithm 1.

4 Performance Evaluation

In this section, we conduct a series of simulation experiments and get some numerical results to evaluate the performance of our proposed algorithm. Without loss of generality, we consider a scenario where there is an unidirectional road, and ten RSUs are equidistantly placed on the road. Moreover, the remote cloud server is assumed to be placed thousands of miles away from the road. For the computation tasks, we assume that the input date sizes are randomly chosen between 500 KB and 1000 KB, and the output data sizes are between 50 KB and 300 KB. The CPU cycles required to accomplish the computation tasks of vehicles are set between 200 Megacycles and 1500 Megacycles, and the maximum permissible processing delays of the computation tasks are in the interval [0.5 s, 2 s]. For communications modes, the wireless bandwidth of the RSUs and the vehicles are set to 40 MHz and 20 MHz, respectively. Meanwhile, the background noise power is −100 dBm. Moreover, the computation capacities of the MEC servers connected to the RSUs are set to 4 GHz, and the computation capacities of the vehicles are 2 GHz.

First, in order to verify that our DQL algorithm can reach a near-optimal solution to the PTMDC problem, we compare the total processing time by adopting two different offloading schemes, i.e., an enumeration algorithm which can achieve an optimal solution to the problem and our DQL algorithm. The experimental results are shown in Fig. 2. We can see that our proposed DQL algorithm can achieve a near-optimal solution.

Figure 3 illustrates the numerical results of the total processing time obtained by three different offloading schemes, i.e., DQL algorithm, DQL algorithm without cloud server and executing all tasks locally, as the number of the computation tasks changes from 20 to 120. From the figure, one can notice that the total processing time obtained by DQL algorithm without cloud server are much less than that while executing all tasks locally. this result shows that MECO has a huge impact on reducing the processing time of vehicles' tasks. In addition, when we take the remote cloud server into account, the total processing time will be twenty to thirty percent less than DQL algorithm without cloud server. Therefore, it is very necessary to introduce the remote cloud server into VECNs.

Meanwhile, we have the experimental results shown in Fig. 4, which illustrate the trends in the number of vehicles choosing different communications modes, i.e., V2I communications and V2V communications, as the number of vehicles

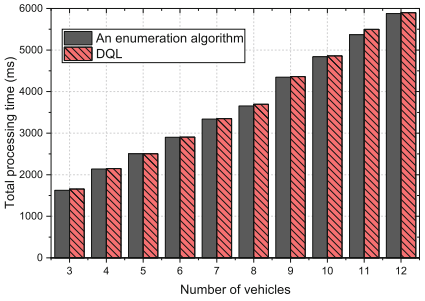

Fig. 2. Comparisons of the total processing time obtained by two algorithms, i.e., an enumeration algorithm and DQL algorithm, as the number of vehicles changes from 3 to 12.

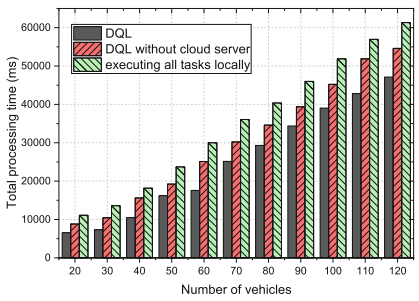

Fig. 3. Comparisons of the total processing time obtained by three schemes, i.e., DQL, DQL without cloud server, and executing all tasks locally, as the number of vehicles changes from 20 to 120.

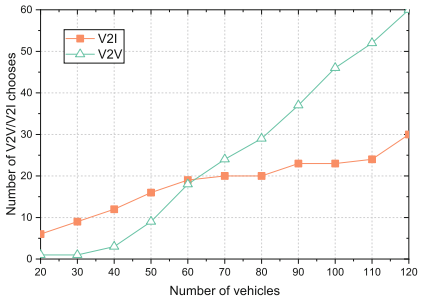

Fig. 4. Trends in the number of vehicles choosing different communications modes, i.e., V2I communications and V2V communications, as the number of vehicles changes from 20 to 120.

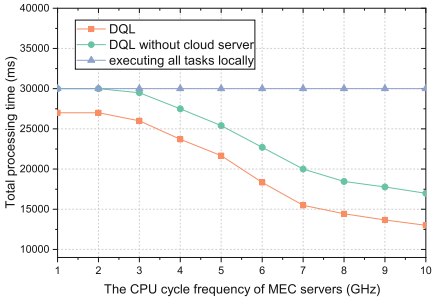

Fig. 5. Comparisons of the total processing time obtained by three schemes, i.e., DQL, DQL without cloud server, and executing all tasks locally, as the computation capacities of MEC servers change.

changes from 20 to 120. As can be seen from the figure, both the number of vehicles that choose V2V communications and that of V2I communications increase with the number of vehicles, and the growth rate of the former is larger than that of the latter. In the end, the number of vehicles that choose V2V communications will exceed that of V2I communications. These numerical results show that the number of vehicles has a more significant influence on V2V communications.

Figure 5 shows the changes in the total processing time with the increase in the CPU cycle frequency of the MEC servers connected to the RSUs. Numerical results in Fig. 5 demonstrate that our proposed DQL algorithm can achieve the minimum total processing time of the computation tasks. Specifically, the locally execution time does not change with the CPU cycle frequency of the MEC servers, because local computing does not need the computation capacities of the MEC servers. Also, we can see that the total processing time obtained by

DQL algorithm decrease with the increasing CPU cycle frequency of the MEC servers. However, the rate of decrease in the total processing time increases first and then decreases. This phenomenon indicates that the CPU cycle frequency of the MEC servers has a decisive influence on the result of DQL algorithm when it is not too large. But when it is big enough, the influence of the CPU cycle frequency of the MEC servers becomes small.

5 Conclusions

In this paper, we first presented a cloud enabled task offloading architecture in VECNs, in which the remote cloud server acts as a backup server considering its powerful computation capacities. Then we studied the task offloading problem in such scenario, and we defined the processing time minimization problem under delay constraints. After that, we proposed a reinforcement learning based task offloading scheme, i.e., a deep Q learning algorithm, as our solution. The numerical results corroborated the good performance of our proposed scheme on processing delay reduction.

References

1. Zhu, H., Yuen, K., Mihaylova, L., Leung, H.: Overview of environment perception for intelligent vehicles. IEEE Trans. Intell. Transp. Syst. **18**(10), 2584–2601 (2017). https://doi.org/10.1109/TITS.2017.2658662
2. Vegni, A.M., Loscrí, V.: A survey on vehicular social networks. IEEE Commun. Surv. Tutor. **17**(4), 2397–2419 (2015). https://doi.org/10.1109/COMST.2015.2453481
3. Shojafar, M., Cordeschi, N., Baccarelli, E.: Energy-efficient adaptive resource management for real-time vehicular cloud services. IEEE Trans. Cloud Comput. **7**(1), 196–209 (2019). https://doi.org/10.1109/TCC.2016.2551747
4. Guo, H., Liu, J., Zhang, J., Sun, W., Kato, N.: Mobile-edge computation offloading for ultradense iot networks. IEEE Internet Things J. **5**(6), 4977–4988 (2018). https://doi.org/10.1109/JIOT.2018.2838584
5. Wang, S., Zhang, X., Zhang, Y., Wang, L., Yang, J., Wang, W.: A survey on mobile edge networks: convergence of computing, caching and communications. IEEE Access **5**, 6757–6779 (2017). https://doi.org/10.1109/ACCESS.2017.2685434
6. Zhang, K., Mao, Y., Leng, S., Vinel, A., Zhang, Y.: Delay constrained offloading for mobile edge computing in cloud-enabled vehicular networks. In: 2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM), pp. 288–294, September 2016. https://doi.org/10.1109/RNDM.2016.7608300
7. Liu, Y., Wang, S., Huang, J., Yang, F.: A computation offloading algorithm based on game theory for vehicular edge networks. In: 2018 IEEE International Conference on Communications (ICC), pp. 1–6, May 2018. https://doi.org/10.1109/ICC.2018.8422240
8. Dai, Y., Xu, D., Maharjan, S., Zhang, Y.: Joint load balancing and offloading in vehicular edge computing and networks. IEEE Internet Things J. 1 (2019). https://doi.org/10.1109/JIOT.2018.2876298

9. Zhang, K., Mao, Y., Leng, S., He, Y., Zhang, Y.: Mobile-edge computing for vehicular networks: a promising network paradigm with predictive off-loading. IEEE Veh. Technol. Mag. **12**(2), 36–44 (2017). https://doi.org/10.1109/MVT.2017.2668838
10. Guo, H., Zhang, J., Liu, J.: Fiwi-enhanced vehicular edge computing networks: collaborative task offloading. IEEE Veh. Technol. Mag. **14**(1), 45–53 (2019). https://doi.org/10.1109/MVT.2018.2879537
11. Zheng, Q., Zheng, K., Zhang, H., Leung, V.C.M.: Delay-optimal virtualized radio resource scheduling in software-defined vehicular networks via stochastic learning. IEEE Trans. Veh. Technol. **65**(10), 7857–7867 (2016). https://doi.org/10.1109/TVT.2016.2538461
12. He, Y., Zhao, N., Yin, H.: Integrated networking, caching, and computing for connected vehicles: a deep reinforcement learning approach. IEEE Trans. Veh. Technol. **67**(1), 44–55 (2018). https://doi.org/10.1109/TVT.2017.2760281
13. Ye, H., Li, G.Y.: Deep reinforcement learning for resource allocation in V2V communications. In: 2018 IEEE International Conference on Communications (ICC), pp. 1–6, May 2018. https://doi.org/10.1109/ICC.2018.8422586
14. Guo, H., Liu, J.: Collaborative computation offloading for multiaccess edge computing over fiber-wireless networks. IEEE Trans. Veh. Technol. **67**(5), 4514–4526 (2018). https://doi.org/10.1109/TVT.2018.2790421
15. Li, J., Gao, H., Lv, T., Lu, Y.: Deep reinforcement learning based computation offloading and resource allocation for mec. In: 2018 IEEE Wireless Communications and Networking Conference (WCNC), pp. 1–6, April 2018. https://doi.org/10.1109/WCNC.2018.8377343