Mobility-Aware Partial Computation Offloading in Vehicular Networks: A Deep Reinforcement Learning Based Scheme

Jianfei Wang[1], Tiejun Lv[1,*], Pingmu Huang[1,*], P. Takis Mathiopoulos[2]

[1] School of Information and Communication Engineering, Beijing University of Posts and Telecommunications (BUPT), Beijing 100876, China

[2] Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Athens 157 84, Greece

* The corresponding author, email: lvtiejun@bupt.edu.cn

Abstract: Encouraged by next-generation networks and autonomous vehicle systems, vehicular networks must employ advanced technologies to guarantee personal safety, reduce traffic accidents and ease traffic jams. By leveraging the computing ability at the network edge, multi-access edge computing (MEC) is a promising technique to tackle such challenges. Compared to traditional full offloading, partial offloading offers more flexibility in the perspective of application as well as deployment of such systems. Hence, in this paper, we investigate the application of partial computing offloading in-vehicle networks. In particular, by analyzing the structure of many emerging applications, e.g., AR and online games, we convert the application structure into a sequential multi-component model. Focusing on shortening the application execution delay, we extend the optimization problem from the single-vehicle computing offloading (SVCOP) scenario to the multi-vehicle computing offloading (MVCOP) by taking multiple constraints into account. A deep reinforcement learning (DRL) based algorithm is proposed as a solution to this problem. Various performance evaluation results have shown that the proposed algorithm achieves superior performance as compared to existing offloading mechanisms in deducing application execution delay.

Keywords: partial offloading; MEC; fog computing; vehicular networks; D2D; AR

I. INTRODUCTION

According to Cisco's forecast [1], global mobile data traffic will increase sevenfold between 2017 and 2022, reaching 77.5 exabytes per month by 2022. Plenty of computation-intensive and data-hungry applications have emerged and are currently used by millions of users, e.g., video streaming, online game, virtual reality (VR) and augmented reality (AR). Such applications require unprecedented computing capacity escalation in vehicular networks [2, 3]. In that sense, vehicular networks must employ abundant communication and computing technologies to ensure personal safety, reduce traffic accidents, ease traffic congestion and guarantee quality of service (QoS). Strict delay constraints have become an obstacle in order to effectively run such complex applications on vehicles. In general, vehicles cannot handle large amount of applications due to limited computing and hardware capabilities. According to existing studies [4], MEC is a promising technique to tackle such

Received: Dec. 5, 2019
Revised: Jan. 6, 2020
Editor: Fuhong Lin

A novel mobility model based on motion trajectory prediction is proposed to indicate the mobility intensity of vehicles.

challenges.

Different from mobile cloud computing (MCC), MEC moves communication, control, scheduling, computing and storage from the remote centralized cloud to the edge of mobile networks. MEC significantly reduces the transmission delay from the base station (BS) to the cloud server. Edge offloading and device-to-device (D2D) offloading are potential techniques which can be used within the multi-access edge computing (MEC) framework. D2D offloading [5, 6, 7, 8, 9, 10] further enhances the computing capacity by integrating the idle computing resources of nearby devices. Vehicles can share contents, report congestion and offload applications via vehicle-to-vehicle (V2V) links. In [11], D2D caching and offloading have been integrated for improving the computing and caching performance of vehicular networks.

Reinforcement learning (RL) [12] is an emerging tool to tackle scheduling problems in communication, caching and computing in vehicular networks[13, 14, 15]. RL is a reward-guided method where an agent interacts with the environment and learns in a "trial and error" manner how to maximize the long-term reward. It is noted that RL is not suitable for problems with high-dimensional action and state space[16]. To tackle this problem, researchers combine RL with deep learning and propose deep reinforcement learning (DRL). For example, Mnih *et al*. propose a deep Q-network (DQN) which achieves a superhuman level in playing Atari 2600 video games [16]. In 2017, Alpha Go [17] by Google DeepMind defeated Ke Jie, the world's number one Go player. DRL begins to play an increasingly important role in the computation offloading field [18]. D. Van Le *et al*. [19] have investigated the computation offloading problem in a multiple ad-hoc network and have proposed a DQN based algorithm. H. Ye *et al*. [20] focused on the resource allocation for V2V communications and have propose a decentralized DRL based mechanism. Existing studies reveal that DRL is an effective scheduling mechanism in the computation offloading scenario.

In this paper, we make further efforts to design, analyze and optimize the resource allocation and computation offloading problems in the vehicle and road side unit (RSU) levels considering the vehicle's mobility and hardware limitations. Specifically, the main contribution of our paper can be summarized as follows:

1. We model the heterogeneous networks (HetNets) with multiple computation offloading schemes. The local execution, edge offloading, and D2D offloading are all integrated to improve the computational capacity of vehicular networks. By analyzing the processing structure of the most emerging applications, such as AR and online games, we design a sequential application model to meet the actual requirements in application and deployment. In the problem formulation, the mobility intensity and hardware constraints of vehicles are comprehensively considered.
2. We formulate the joint computation offloading and resource allocation problem from a single-vehicle to a multi-vehicle scenario. We aim to minimize the system cost under the constraints of limited computation and communication resources of vehicles and MEC servers. Since the energy supply in vehicles is not a practical issue, we focus on the execution delay in application processing and transferring.
3. We develop an offloading algorithm based on DRL to configure the parameters of computation offloading and resource allocation as well as to determine the feasible sets of vehicles and RSUs for the requesting vehicles. To reduce the high complexity caused by the large action space, we propose a continuous-control based algorithm and make a variety of improvements in the action output.
4. We compare our algorithm with other existing DQN based algorithms [11,14] in their structure, and argue about its superiority. To demonstrate the performance of our algorithm in reducing execution delay,

we provide numerical results against the existing DRL based schemes as well as the traditional methods. The simulation results confirm the outstanding performance of our algorithm.

The organization of the paper is as follows. After this introduction, the system models, namely network architecture, application model, single and multi vehicle computation offloading, mobility model, as well as the execution delay optimization program formulation are presented. In Section III, the RL and DRL are introduced. Section IV details our proposed DRL based scheme. Numerical results and analysis are provided in Section V, while the conclusion of the paper can be found in Section VI.

II. System Models

2.1 Network architecture

We consider a vehicular network, composed of one BS with a system controller, M relays/RSUs integrating MEC servers and N vehicles. Note that MEC servers are installed at RSUs to help reduce the transmission delay and the workload of BS. The BS collects information from vehicles and RSUs. The arrangement for computation offloading and resource allocation is controlled by the system controller. We investigate the case that both edge offloading and D2D offloading are enabled. Let $K = 1, \ldots, k$ and $U = 1, \ldots, u$ be the sets of the RSUs (or the MEC servers) and the vehicles, respectively. Let us assume that the requesting vehicle can selectively offload applications to nearby vehicles or MEC servers. As illustrated in the network architecture of Figure 1, vehicle 2 firstly enters the coverage area of the RSU 2. The vehicle 2 can offload applications to the RSU 2. Then vehicle 3 enters the communication range of the vehicle 2, and D2D offloading from vehicle 2 to vehicle 3 is enabled. Vehicle 2 further moves out of the coverage area of the RSU 2. However, since in this case there are no available vehicles or RSUs, vehicle 2 must handle applications locally.

Due to the complexity of the network, there are particularly many variables in the article. For the convenience of the presentation and to make it easier for the readers to understand, the main system parameters used in this paper and some of which will be introduced later on, are summarized in Table 1.

Table I. *System parameters.*

Parameter	Meaning
K	the RSU/MEC server set
U	the vehicle set
M	the number of RSUs/MEC servers
N	the number of vehicles
ϖ	the mobility set of users
lt	the latitude
gt	the gratitude
d_i	the moving direction
v	the moving speed
$T_{i,j}$	the j-th component of the application of vehicle i
$I_{i,j}$	the input data size
$C_{i,j}$	the required computational resources
T_{ij}	the deadline time to accomplish application
$F_k^{\max}/F_u^{\max}$	the maximal CPU frequency of MEC servers/vehicles
$t_{i,j}$	the transmission time
$p_{i,j}$	the processing time
α	the indicator for local execution
β	the indicator for edge offloading
γ	the indicator for D2D offloading
J	the average execution delay

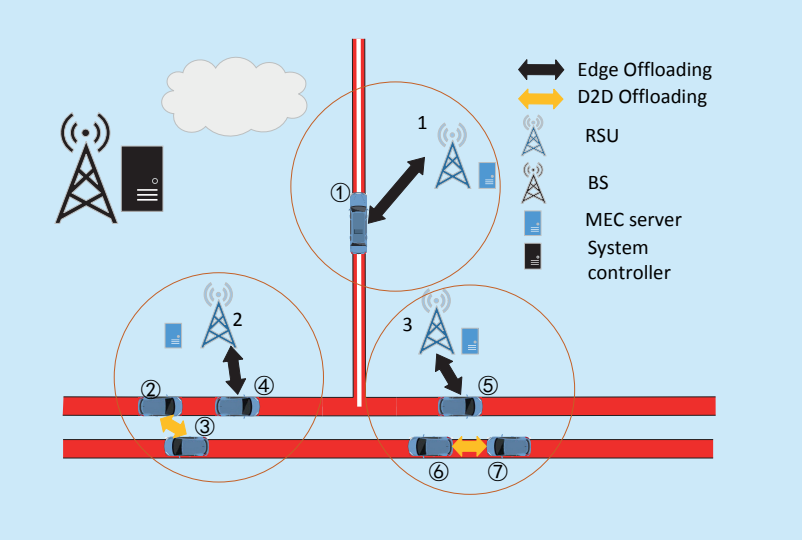


Fig. 1. *Illustration of the network architecture.*

2.2 Application model

There exist many application models that support the cloud service [21]. Compared to the full offloading application model, the partial offloading application model provides more flexibility, which could better match the application scenario. In general, the application for cloud service could be classified into the following groups:

- Data partitioned oriented applications, where they can be arbitrarily partitioned into several separate parts that could be processed in parallel at different platforms [22, 23, 24, 25].
- Code partitioned oriented applications, where they consist of several fixed components. These components can be processed in parallel or sequentially for some components rely on the output of others [26, 27, 28].
- Continuous execution applications. Since they require continuous execution, the corresponding data size and required computational resources are not known in advance. Cloud online gaming and some interactive applications belong to this group [29, 30, 31].

Since complex applications are composed of several fixed components which cannot be arbitrarily partitioned, we focus on code partitioned oriented applications and adopt a process based application model, in which the components of applications are processed sequentially. For the clarity of the presentation, we will illustrate some examples of this application model. At first, we introduce the emerging application ARCore. ARCore is a software development kit developed by Google that allows for augmented reality applications to be built. As Figure. 2 shows, the ARCore application is composed of several components, which can be converted to a multi-component and sequential architecture. The details of the ARCore application can be found in [32]. Figure 3 shows the architecture of the video gaming application. Both of these figures show the application model of the emerging ARCore application, and video gaming application. Clearly, the application consists of several separate components, with the current component relying on the output of the previous component.

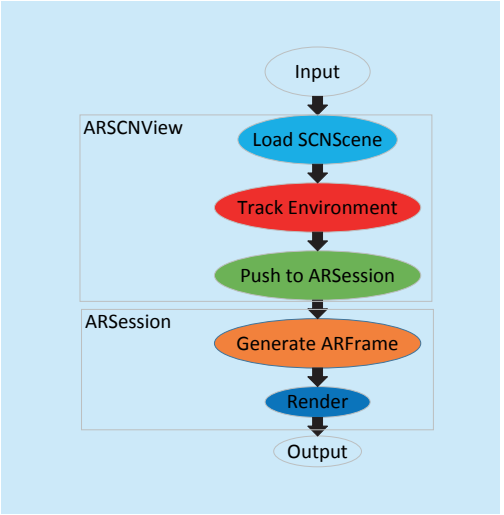


Fig. 2. *ARCore application.*

2.3 Single-vehicle computation offloading

Due to the complex task model and the changing network environment, the discussion of our offloading scenario is difficult to understand by the reader. Thus, we introduce the single-vehicle computation offloading problem (SVCOP) at first, where only a single vehicle makes computation offloading. As previously mentioned, the application in our vehicle networks is modeled as a linearly sequential processing program that consists of μ components. Each component can be executed locally, offloaded to MEC servers or nearby vehicles. Similar to [33, 34, 35, 36, 37], we define the computation task $\tau_j = \{I_j, T_j, C_j\}$, where I_j is the input data size of the j-th component, C_j is the required CPU cycles, and T_j stands for the deadline time to accomplish the component. The execution delay is composed of two parts, the transmission time t_j and the processing time p_j. The transmission time t_j can be ignored if the two adjacent components, respectively the (j-1)-th and j-th components, are processed in the same place.

Since the very first input/last output data must be from/to the local vehicle, we add two additional components 0 and $\mu+1$ to the head and tail of the sequential model as the virtual input and output components. As shown in Figure 2 for an ARCore application, an application consists of several components and each component except the virtual input/output module can be freely processed in different platforms. The current component relies on the output of the former component, thus these components must be processed sequentially.

According to the transmission time $t_j(1 \le j \le \mu+1)$ and processing time $p_j(1 \le j \le \mu+1)p_j(1 \le j \le \mu+1)$, the SVCOP decides which platform the component should be processed at. Therefore we formulate the problem under consideration here as

$$\min_{\alpha,\beta,\gamma}\left(\sum_{j=1}^{\mu+1} t_j + \sum_{j=1}^{\mu} p_j\right). \tag{1}$$

Let p_j^{L}, p_j^{E} and p_j^{V} represent the processing time for the j-th component that processed in the local vehicle, the MEC server and a nearby vehicle. The processing time p_j can be calculated by

$$p_j = \alpha p_j^{\mathrm{L}} + \beta p_j^{\mathrm{E}} + \gamma p_j^{\mathrm{V}}, \tag{2}$$

$$s.t. \quad \alpha+\beta+\gamma = 1, \tag{2a}$$

$$\alpha,\beta,\gamma \in \{0,1\}, \tag{2b}$$

where α, β, and γ are the control parameters for p_j^{L}, p_j^{E} and p_j^{V}. The processing time p_j is related to the required CPU cycles and the allocated CPU frequencies. The transmission time t_j is affected by the input data size and the communication environment, such as the signal transmission power, the channel bandwidth, and the channel noise. The details of p_j and t_j will be presented in the next subsection.

2.4 Multi-vehicle computation offloading

In this section, we study the multi-vehicle computation offloading problem (MVCOP). At first, we model our system as a time-slotted structure indexed by x. The computing and communication resources in vehicles and RSUs/MEC servers, and the mobility intensity of vehicles can be considered static in a small time frame η.

Clearly, MVCOP is more complicated than SVCOP, and vehicles in MVCOP compete for the limited computational and channel resources [36, 38, 39, 40]. Applications from different vehicles may be processed in the same vehicle or MEC server. Thus, the processing and transmission time for MVCOP is far longer than in the SVCOP case.

We assume the channel bandwidth is allocated in an average way to each vehicle. The vehicles within the vehicular network upload requests and data to the RSU by orthogonal frequency division multiple access. If the total available bandwidth is B, the allocated bandwidth for each vehicle is $W = B/N$. Similar to previous discussion, each computation task can be expressed as $\tau_{i,j}(x) = \{I_{i,j}(x), C_{i,j}(x), T_{i,j}(x)\}$, where $I_{i,j}(x)$ is the input data size of the j-th component of the i-th vehicle in the time slot x, $C_{i,j}(x)$ is the required CPU cycles to handle the computation task, and $T_{i,j}(x)$ denotes the deadline time to accomplish the task, and can be executed locally, offloaded to a MEC server or a nearby vehicle. The total processing time consists of two parts, i.e., the transmission time $t_{i,j}(x)$ and processing time $p_{i,j}(x)$.

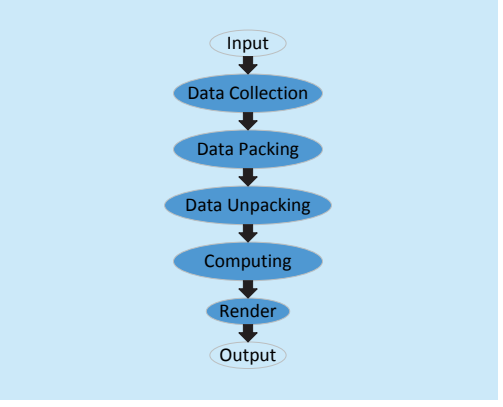


Fig. 3. *Example of a video gaming application.*

2.4.1 Processing time

It is assumed that a vehicle u (and a MEC server k) is equipped with a powerful processor with the constant computation capacity $F_u^{\max}$ (and $F_k^{\max}$). Let $f_i^{\mathrm{L}}(x)$, $f_{i,k}^{\mathrm{E}}(x)$, and $f_{i,u}^{\mathrm{V}}(x)$ represent the allocated CPU frequency for the vehicle i in different platforms, respectively, the vehicle itself, the associated MEC server and the allocated nearby vehicle. Generally, $f_i^{\mathrm{L}}(x)$ and $f_{i,u}^{\mathrm{V}}(x)$ are related to vehicle's computational capability, which can be considered fixed in a time slot. Since multiple vehicles could offload computation tasks to the same MEC server, $f_{i,k}^{\mathrm{E}}(x)$ is highly correlated to the spare computational resources in the k-th MEC server. If the task $\tau_{i,j}(x)$ is processed locally, the processing time $p_{i,j}^{\mathrm{L}}(x)$ can be written as

$$p_{i,j}^{\mathrm{L}}(x)=\frac{C_{i,j}(x)}{f_i^{\mathrm{L}}(x)}. \tag{3}$$

Meanwhile, we take into account that vehicles may run some personal tasks, e.g. running the operating system, thus we define $\iota_i(x)$ as the current CPU load of vehicle i. $f_i^{\mathrm{L}}(x)$ can be calculated as

$$f_i^{\mathrm{L}}(x)=\iota_i(x)F_i^{\max}. \tag{4}$$

When the task $\tau_{i,j}(x)$ is offloaded to the k-th MEC server, the processing time $p_{I,j,k}^{\mathrm{E}}(x)$ is given by

$$p_{i,j,k}^{\mathrm{E}}(x)=\frac{C_{i,j}(x)}{f_{i,k}^{\mathrm{E}}(x)}. \tag{5}$$

Now, if $\tau_{i,j}(x)$ is offloaded to vehicle u, the processing time $p_{i,j,u}^{\mathrm{V}}(x)$ can be expressed as

$$p_{i,j,u}^{\mathrm{V}}(x)=\frac{C_{i,j}(x)}{f_{i,u}^{\mathrm{V}}(x)}. \tag{6}$$

If $\alpha_{i,j}(x)$, $\beta_{i,j}(x)$ and $\gamma_{i,j}(x)$ are the offloading control parameters for the task $\tau_{i,j}(x)$, the processing time can be calculated as

$$\begin{aligned} p_{i,j}(x)= & \alpha_{i,j}(x)p_{i,j}^{\mathrm{L}}(x) \\ & +\sum_{k\in K}\beta_{i,j,k}(x)p_{i,j,k}^{\mathrm{E}}(x) \\ & +\sum_{u\in U,u\neq i}\gamma_{i,j,u}(x)p_{i,j,u}^{\mathrm{V}}(x), \end{aligned} \tag{7}$$

where $\alpha_{i,j}(x)+\sum_{k\in K}\beta_{i,j,k}(x)+\sum_{u\in U,u\neq i}\gamma_{i,j,u}(x)=1$ and $\alpha_{i,j}(x),\beta_{i,j,k}(x),\gamma_{i,j,u}(x)\in\{0,1\}$.

2.4.2 Transmission time

To maximize the use of spectrum resources, we assume that the D2D link and cellular link share the spectrum resources, and since the channel bandwidth is allocated in an average way to each vehicle, the channel interference can be neglected. Furthermore, the transmission rate from a vehicle to a RSU or a vehicle can be calculated as

$$r_{i,k}(x)=W\log_2\left(1+\frac{P_{i,k}(x)h_{i,k}(x)}{\sigma^2}\right),k\in K, \tag{8}$$

$$r_{i,u}(x)=W\log_2\left(1+\frac{P_{i,u}(x)h_{i,u}(x)}{\sigma^2}\right),u\in U,u\neq i, \tag{9}$$

where W is the allocated bandwidth to an vehicle, $P_{i,k}(x)/P_{i,u}(x)$ stands for the transmission power from the vehicle i to the RSU k / the vehicle u and $h_{i,k}(x)$/ $h_{i,u}(x)$ is the related channel gain. Note that, if the two adjacent task is processed in the same platform, there is no need to send the output of the previous component to another platform. Clearly, the related transmission time is 0. Meanwhile, the transmission rate of the downlink from RSUs to vehicles is far greater than the uplink. Thus, the transmission time from RSUs to vehicles can be ignored. Therefore, the transmission time can be simplified for two cases, i.e. from vehicles to vehicles and from vehicles to RSUs, respectively. Assuming $t_{i,j}(x)$ is the transmission time from the (j -1)-th platform to the j -th platform, it can be expressed as

$$\begin{aligned} t_{i,j}(x)= & \alpha_{i,j-1}(x)I_{i,j}(x)\left(\frac{\beta_{i,j,k}(x)}{r_{i,k}(x)}+\frac{\gamma_{i,j,u}(x)}{r_{i,u}(x)}\right) \\ & +\gamma_{i,j-1,u}(x)I_{i,j}(x)\left(\frac{\alpha_{i,j}(x)}{r_{u,i}(x)}+\frac{\beta_{i,j,k}(x)}{r_{u,k}(x)}+\frac{\gamma_{i,j,v}(x)}{r_{u,v}(x)}\right), \end{aligned} \tag{10}$$

where u, and v is the allocated vehicle for (j-1)-th, and j-th component.

2.5 Mobility model

Since the speed of vehicles is much high-

er than that of ordinary UEs, their location changes rapidly within a small time frame. In this paper, we assume that the vehicles move in a two-dimensional plane. Therefore, the parameter tuple $\varpi(x) = \{lt(x), gt(x), di(x), v(x)\}$ can perfectly indicate the mobility of vehicles, where the *lt* (and *gt*) is the longitude (and the latitude) of vehicles, and *di*|*v* stands for the moving direction| |speed of vehicles. Typically, the moving direction and speed can be considered constant within a small time frame. Thus the future location of vehicles can be calculated by

$$lt(x^t) = lt(x) + v(x)(x^t - x)\sin di(x), \quad (11)$$

$$gt(x^t) = gt(x) + v(x)(x^t - x)\cos di(x). \quad (12)$$

Define $D_k^{\max}$, and $D_u^{\max}$ as the maximum communication distance for RSUs, and vehicles. On the basis of equation (11) and equation (12), we could calculate the contact time between a vehicle and another vehicle or an RSU, respectively $T_{i,u}^{\text{con}}(x)$ and $T_{i,k}^{\text{con}}(x)$. In order to achieve stable task offloading, the contact time between two devices must be greater than or equal to the frame length. Therefore, we could establish the connectivity matrix as below:

$$G(x) = [g_{i,j}(x)]_{N\times(M+N)}, i \in U, j \in K \cup U, \quad (13)$$

where $g_{i,j}(x) = 1$ if $T_{i,j}^{\text{con}}(x) \geq \eta$, otherwise it is 0.

2.6 Execution delay optimization

The execution delay can be optimized in the MVCOP scenario by considering the impact factors of both transmission and processing delay, as follows:

$$\min_{\alpha,\beta,\gamma} J(x) = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{j=1}^{\mu+1} t_{i,j}(x) + \sum_{j=1}^{\mu} p_{i,j}(x)\right), \quad (14)$$

where

$$\begin{aligned}
s.t.\quad & C1: \alpha + \beta + \gamma = 1, \alpha, \beta, \gamma \in \{0,1\},\\
& C2: \alpha_{i,j}(x) + \sum_{k\in K}\beta_{i,j,k}(x)\\
& \quad + \sum_{u\in U, u\neq i}\gamma_{i,j,u}(x) = 1,\\
& C3: \alpha_{i,j}(x), \beta_{i,j,k}(x), \gamma_{i,j,u}(x) \in \{0,1\},\\
& C4: \sum_{i\in U} f_{i,k}^{\text{E}}(x) \leq F_k^{\max}, k \in K,\\
& C5: \sum_{i\in U} f_{i,u}^{\text{V}}(x) \leq F_u^{\max}, u \in U,\\
& C6: \sum_{k\in K}\beta_{i,j,k}(x) \leq 1,\\
& C7: \sum_{u\in U, u\neq i}\gamma_{i,j,u}(x) \leq 1,\\
& C8: t_{i,j}(x) + p_{i,j}(x) \leq T_{i,j}(x).
\end{aligned}$$

Constraints C1, C2, C3, C6, and C7 ensure that a component can be only processed in one platform, locally, in a MEC server or a nearby vehicle; C4 guarantees that the total allocated computing resources in MEC servers cannot exceed their computational capacity; C5 guarantees that the allocated computing resources in vehicles for D2D offloading must be less than their computational capabilities; C8 ensures that a component must be accomplished with its deadline time.

The multi-user computation offloading problem in MCC is proved to be NP-hard [41]. Taking the competition among vehicles and the resource constraints into consideration, clearly the proposed MVCOP is more complex and thus more difficult to be solved than other multi-user computation offloading problems. Since previous known methods, e.g. [23, 24] and [28] for solving such problems cannot be used to solve the proposed MVCOP, next a novel DRL based approach will be introduced and a solution to the MVCOP optimization problem will be presented in Section 4. To facilitate readers' understanding, in the next section a brief introduction into the deep reinforcement learning and how it can be used to solve our optimization problem will be presented.

III. Deep Reinforcement Learning

Since our work involves the comparison of different reinforcement learning algorithms, a

brief introduction to reinforcement learning is necessary. Next, we will give a brief introduction to the development history of reinforcement learning and some classical algorithms.

3.1 Reinforcement learning

Reinforcement learning is a sub-area of machine learning, where software agents interact with environments to maximize some accumulative rewards[12]. Reinforcement learning is one of three basic machine learning paradigms and the other two are supervised learning and unsupervised learning. Different from supervised learning where labeled input/output data need be presented and sub-optimal actions need be corrected, the reinforcement learning focus on balancing the current knowledge and uncharted territory. In general, the interactive environment is formulated as a Markov decision process (MDP).

3.2 Q-learning

Q-learning is a model-free reinforcement learning algorithm [42]. The goal of Q-learning is to learn a policy, which guides the software agent to make appropriate action according to current circumstances. For any finite Markov decision process, Q-learning finds an optimal policy which maximizes the expected total reward over all steps, starting from current state.

The process of Q-learning algorithm consists of the following steps.

1. Firstly, a Q table is formed which includes the transition relationship from a state to an another state after taking an action.
2. Before learning begins, the Q table is initialized with possibly arbitrary fixed value.
3. At each time slot x, the agent selects an action a(x) according to Q table and observes the obtained reward r(x), then enters a new state s(x+1).
4. Update Q table, according to r(x), a(x), s(x) and s(x+1). The update equation is listed as below:

$$\begin{aligned} &Q(s_x, a_x) = (1-\rho)Q(s_x, a_x) \\ &+\rho r_x + \varrho \max_a Q(s_{x+1}, a), \end{aligned} \tag{15}$$

where ρ is the learning rate that determines the rate that newly obtained information overrides the history information and ϱ denotes the discount factor that determines the importance of future reward.

3.3 DQN

In DQN, a neural network is used to approximate the Q-value function [16]. The state is the input of DQN and the Q-value of all possible actions is the output. It is interesting to make the following comparison between Q-learning and deep Q-learning:

1. All the historical experience is stored in a memory structure.
2. The Q-network chooses the action with the maximum Q-value as the output according to the following equation:

$$Q^*(s,a) = \mathrm{E}_s\left[\mathrm{r}+\rho \max_{\mathrm{a}'} \mathrm{Q}^*(\mathrm{s},\mathrm{a})|\mathrm{s},\mathrm{a}\right]. \tag{16}$$

3. The loss function here is mean squared error of the predicted Q-value and the target Q value Q^*, which is defined as below:

$$Loss(\theta) = \mathrm{E}\left[\left(\mathrm{y}\text{-}\mathrm{Q}(\mathrm{s},\mathrm{a},\theta)\right)^2\right], \tag{17}$$

where the target value y can be expressed as

$$y = r + \max_a Q(s,a,\theta^-). \tag{18}$$

Additional information about DQN can be found in [16].

3.4 Actor-critic

Actor-critic methods are Temporal-Difference (TD) methods that have a separate memory structure to accurately indicate the policy independent of the value function [12]. The policy structure is known as the actor, and the estimated value function is known as the critic. The actor is used to select actions, and the critic is used to evaluate the the action made by the actor. Learning can be viewed as an on-policy: The critic must learn about and evaluate the actions followed by the actor all the time. Actor-Critic methods are the extension of the reinforcement comparison methods to TD methods. In general, the critic is a state-value function that evaluates the state after the actor has taken an action. Such evalu-

ation has a TD error:

$$\delta(x) = r_{x+1} + \rho V(s_{x+1}) - V(s_x), \quad (19)$$

where r is the obtained reward, ρ denotes the discount factor, and V is the current value function applied in the critic. The TD error is used to evaluate the selected action. If the TD error is positive, it suggests that the tendency to select the action should be strengthened for the future, whereas if the TD error is negative, it suggests the tendency should be weakened. Suppose actions are generated by the Gibbs softmax method:

$$\pi_x(s,a) = Pr\{a_x = a \mid s_x = s\} = \frac{e^{p(s,a)}}{\sum_b e^{p(s,b)}}, \quad (20)$$

where the $p(s,a)$ indicates the tendency to select action a in state s. Then, the strengthening or weakening described above can be implemented by increasing or decreasing $p(s,a)$, for instance, by

$$p(s_x,a_x) \leftarrow p(s_x,a_x) + \phi\delta(x), \quad (21)$$

where ϕ is another positive step-size parameter. More details of actor-critic methods can be found in [43, 44].

IV. DRL Based Offloading Scheme

We now use DRL to formulate the MVCOP, where the parameters of computing and communication are jointly optimized. Recall that our vehicular network consists of M RSUs/MEC servers and N vehicles. The computing resources of vehicles and MEC servers as well as the task, mobility intensity and communication status of vehicles all change over time. We aim to allocate the set of RSUs and nearby vehicles as well as their computing resources for the requesting vehicle according to the current state. Due to the large state and action space, it is very difficult to solve such a problem using traditional methods. In order to tackle this issue, we use the DRL to determine the optimal action based on the large amount of input data.

The operation of the control system can be explained as follows. At first, the control system collects the information of task, computing, communication, and mobility intensity from each vehicle and each RSU/MEC server. Then, it constructs the system state based on the collected information. The neural network receives the system state and determine the optimal action a^*. Finally, the control system forwards the action a^* to vehicles and RSUs. In this paper, we utilize the recent asynchronous advantage actor-critic (A3C) algorithm [45] to determine the optimal action.

The A3C algorithm belongs to actor-critic methods and its structure is illustrated in Figure 4. Unlike recently popular DQN algorithm, A3C utilizes multiple agents to learn more efficiently. In A3C, there is a global net and a set of worker agents. Each worker agent interacts with its own copy of the global environment. The experience of each agent is independent of the experience of the others.

4.1 Mobility-aware offloading algorithm using A3C

Although some recent studies [11, 14] have been based on DQN, in our work we have chosen an Actor-critic-based algorithm referred to as asynchronous advantage actor-critic (A3C), as it turns out to be a better alternative. To

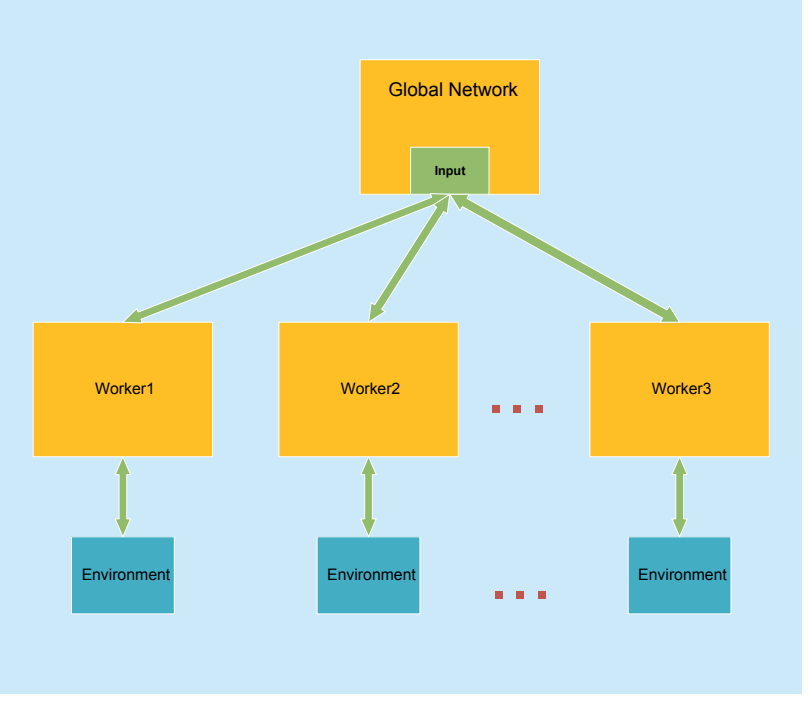


Fig. 4. *A3C structure.*

verify this, we choose a DQN-based algorithm as the comparison. The details of the A3C-based algorithm and the DQN-based algorithm are presented in Algorithm 1 and Algorithm 2. Next, the two algorithms will be compared from the perspective of the algorithm architecture. The related simulation results will be presented later on in Section 5.

By comparing the structures of Algorithm 1 and Algorithm 2, we present the advantages of A3C as below:

The A3C utilizes multi-threading techniques while the DQN is single-threading. The A3C deploys a global network and multiple worker networks. Theoretically speaking, the number of worker networks is linear with the number of threads. The worker agents asynchronously update the global network after an episode. Each of the worker agents interacts with its own copy of the environment at the same time as the other agents are interacting with their environments. The training speed of A3C is much faster than the DQN. Meanwhile, the A3C converges faster than the DQN with the help of worker agents. This means that the A3C is easier to deploy and has a better performance than the DQN.

Similar to [11, 14], the goal for the agents in Algorithm 1 and Algorithm 2 is to accomplish the requesting tasks within one time slot, while the optional goal is to perform the action according to the raw state. For this to happen, the system selects appropriate sets of available vehicles/RSUs for computing. Next, the agents select the optimal actions based on states, where the actions aim to maximize the immediate reward. However, the proposed mechanisms in [11, 14] are not suitable for complicated networks with massive computing nodes. The related proof is presented in subsection 4.5.

In the mobility scenario, because the resource allocation is very complicated, the action space can be very large and may lead to high complexity in deployment. To realize a low-complex algorithm, we amend Algorithm 1 to adapt to heterogeneous networks with massive nodes and use it for learning to obtain the optimal policy. Moreover, we need define the system states, perform the system actions, and calculate the reward functions. All these requirements are presented in the following subsections.

Algorithm 1. (A3C based algorithm).

Collect information from vehicles and RSUs
Construct initial state s_0
Initialize thread step counter $x \leftarrow 1$
Repeat {
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_N \leftarrow 0$;
  Sync thread-specific parameters $\theta' = \theta$ and $\theta_v' = \theta_v$;
  $x_{start} = x$;
  Collect information from vehicles and RSUs and build state s_x;
  Repeat {
    Obtain a_x according to policy $\pi(a_x \mid s_x : \theta')$;
    RSUs and vehicles perform action a_x;
    Collect new information from RSUs and vehicles;
    Construct new state s_{x+1} and obtain reward r_x;
    $x \leftarrow x+1$;
    $X \leftarrow X+1$;
  }
  Until terminal s_x or $x - x_{start} == x_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_x \\ V(s_x, \theta_x) & \text{for non-terminal } s_x \end{cases}$
  For $(i \in \{x-1, ..., x_{start}\})$ **do** {
    $R \leftarrow r_i + \varrho R$;
    Accumulate gradients θ, according to
    $d\theta \leftarrow d\theta + \nabla_\theta \log \pi(a_i \mid s_i; \theta)(R - V(s_i; \theta_v))$;
    Accumulate gradients θ_v according to
    $d\theta_v \leftarrow d\theta_v + \partial(\text{R-V}(s_i;\theta_v))^2 / \partial\theta_v$;
    Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$
  }
}

4.2 System state

State is a concrete and immediate situation in which the agent finds itself [12]. The state setting should sufficiently reflect the system environment including the communication, computing and mobility state of vehicles as well as the computing and communication state of RSUs/MEC servers. Let us define s as a variable set including the computing information of vehicles and MEC servers as well as the communication, mobility intensity and task

information of vehicles. We also define $s(x)$ as $s(x) = \{ s_U(x), s_K(x) \}$, where $s_U(x)$ denotes the information set for vehicles and $s_K(x)$ stands for the information set for MEC servers.

It is convenient to represent $s_U(x) = \{\tau(x), f^{\mathrm{L}}(x), \varpi(x), sig(x)\}$, where $\tau(x)$, $f^L(x)$, $\varpi(x)$, and $sig(x)$ are the task, computing, mobility intensity and communication information of vehicles. Here, $\tau(x)$ can be expressed as $\tau(x) = [\{C_{i,j}(x), I_{i,j}(x), T_{i,j}(x)\}]_{N\times\mu}$. $f^L(x)$ is given by $f^L(x) = [f_i(x)]_{N\times 1}$, $\varpi(x)$ is defined as $\varpi(x) = [\{lt_i(x), gt_i(x), di_i(x), v_i(x)\}]_{N\times 1}$, and $sig(x)$ is calculated by $sig(x) = [\{p_i(x), h_i(x)\}]_{N\times 1}$. Note that the definition of the relevant parameters has been given above and will not be described here.

$s^K(x)$ represents the information set of MEC servers. Define $M\times 1$ set $s_{\mathrm{K}}(x) = [F_k(x)]_{N\times 1}$, where $F_k(x)$ denotes the available computing resources of the k-th server and $F_k(x)$ follows $F_k(x) \le F_k^{\max}$.

4.3 System action

Action is the set of all possible moves the agent can make. An action is almost self-explanatory [16, 42]. In the system, the agent has to decide which vehicle or RSU is allocated to the requesting vehicle, how much computational resource is allocated to the computation task, and whether or not a computation task is offloaded to an RSU or a vehicle. Such composite action $A(x)$ is denoted by $A(x) = \{a(x), F^{\mathrm{E}}(x)\}$, where $a(x)$ decides the operational platforms for computation tasks and $F^{\mathrm{E}}(x)$ decides the allocated computational resources in MEC servers for requesting vehicles. Specifically, we define a(x) as $a(x) = \{\alpha(x), \beta(x), \gamma(x)\}$, where $\alpha(x)$, $\beta(x)$, and $\gamma(x)$ are the operational set for local execution, edge offloading, and D2D offloading. These control parameters will be discussed next.

$\alpha(x)$ is the local execution controller. Define $N\times\mu$ matrix $\alpha(x) = [\alpha_{i,j}(x)]_{N\times\mu}$, where $\alpha_{i,j}(x) \in \{0,1\}$ and each element $\alpha_{i,j}(x)$ denotes whether the computation task $\tau_{i,j}(x)$ is locally processed. Here, $\alpha_{i,j}(x) = 1$ indicates that the j-th task of the i-th vehicle in the time slot x is processed by the local processor and 0 else.

$\beta(x)$ decides whether or not a computation task is offloaded to an RSU/MEC server. Define $N\times M\times\mu$ matrix $\beta(x) = [\beta_{i,j,k}(x)]_{N\times M\times\mu}$, where $\beta_{i,j,k}(x) \in \{0,1\}$ and each element $\beta_{i,j,k}(x)$ denotes whether the computation task $\tau_{i,j}(x)$ is offloaded to the k-th RSU/MEC server. Here, $\beta_{i,j,k}(x) = 1$ means that the computation task $\tau_{i,j}(x)$ is offloaded to the k-th RSU/MEC server and 0 else.

$\gamma(x)$ determines whether or not a computation task is processed by a nearby vehicle. Define $N\times N\times\mu$ matrix $\gamma(x) = [\gamma_{i,j,u}(x)]_{N\times N\times\mu}$, where $\gamma_{i,j,u}(x) \in \{0,1\}$ and each element $\gamma_{i,j,u}(x)$ determine whether the i-th vehicle offloads the task $\tau_{i,j}(x)$ to the u-th vehicle.

Algorithm 2. (DQN based algorithm).

Initialize replay buffer R_d;
Initialize action value function Q with random weights θ;
Initialize target value function $Q^{\wedge}$ with weights $\theta^{-} = \theta$;
For (*episode* = 1, M_d) **do** {
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\varphi_1 = \varphi(s_1)$
  With Probability select a random action a_x otherwise select $a_x = \mathrm{argmax}_{\mathrm{a}} \mathrm{Q}(\phi(\mathrm{s}_{\mathrm{x}}), \mathrm{a}:\theta)$;
  Forward action a_{x} to RSUs and vehicles, then observe reward r_x and construct the new state s_{x+1};
  Preprocess $\varphi_{x+1} = \varphi(s_{x+1})$ and store transition $(\varphi_x, a_x, r_x, \varphi_{x+1})$ in R_d;
  Sample random mini batch of transitions $(\varphi_j, a_j, r_j, \varphi_{j+1})$ from R_d;
  If (episode terminates at $j+1$) {
    Set $y_j = r_j$;
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to the network parameters θ;
    Reset $Q^{\wedge} = Q$ after every C steps;
  }
}

$\gamma_{i,j,u}(x)=1$ means that the task $\tau_{i,j}(x)$ is processed by the vehicle u and 0 else. In order to avoid the conflicts with $\alpha(x)$, we force $\gamma_{i,j,i}=0$.

$F^{\mathrm{E}}(x)$ decides the allocated computational resources for vehicles in MEC servers. Define $N\times M$ matrix $F^{\mathrm{E}}(x)=[F_{i,k}(x)]_{N\times M}$. In general, allocating computational resources for each computation task of vehicles is very complicated. At the same time, it will greatly increase the complexity of the system [11]. To simplify this problem, we assume the allocated computing resources for the vehicle i in the k-th MEC server are proportional to the tasks that the vehicle i offloads to the MEC server k. $F_{i,k}(x)$ can be calculated as

$$F_{i,k}(x)=\frac{\sum_{j=1}^{\mu}\beta_{i,j,k}C_{i,j}F_k^{\max}}{\sum_{i=1}^{N}\sum_{j=1}^{\mu}\beta_{i,j,k}C_{i,j}}. \quad (22)$$

In a word, the direct output of the neural networks is $a(x)$ and $F^{\mathrm{E}}(x)$ is calculated on the basis of $a(x)$. Therefore, we define $a_{N\times\mu\times(N+M+1)}$ matrix to represent $a(x)$.

4.4 Reward Function

Reward is the feedback by which we measure the success or failure of an agent's actions in a given state. The reward setting plays a key role in training the neural networks[16, 46]. The main objective of the MVCOP of equations (14) is to minimize the average task execution delay, while the reward function aims to maximize the obtained reward. To this end, the reward should be inversely proportional to the task execution delay. Let us consider the following reward function

$$r(x)=\frac{J^t(x)-J_{s,a}(x)}{J^t(x)}, \quad (23)$$

where $J^t(x)$ is the cost obtained using existing offloading methods such as the greedy algorithm and local execution method and $J_{s,a}(x)$ denotes the obtained cost based on Algorithm 1. The most distinctive feature of the reward function is that it can constantly compare with existing algorithms. The agent will automatically search an advantageous policy compared to the comparison algorithm $J^t(x)$. Since different algorithms might perform differently when the network environment changes, we need to utilize multiple algorithms for comparison. Therefore, we further introduce $J^t(x)$ as

$$J^t(x)=\min\{J_1(x),J_2(x),...,J_i(x),...\}, \quad (24)$$

where $J_i(x)$ represents the obtained cost based on an offloading algorithm. This ensures the agent trains towards obtaining the cost less than the optimal comparison algorithm. In this paper, $J^t(x)$ is given by

$$J^t(x)=\min\{J_{\mathrm{local}}(x),J_{\mathrm{greedy}}(x),J_{\mathrm{DQN}}(x)\} \quad (25)$$

where $J_{\mathrm{local}(x)}$, $J_{\mathrm{greedy}(x)}$, and $J_{\mathrm{DQN}(x)}$ denote the cost by the local execution method, the greedy mechanism, and the DQN algorithm, respectively.

4.5 Action parsing

In this subsection, we will present the work we have made to improve existing researches in [11, 14]. The direct output of our A3C based algorithm is a matrix of floating point numbers, while the required action is a matrix consisted of 0-1 integers. This means that we need to do some preprocessing on the output.

We define the action output as

$$a(x)=[a_{i,j,l}]_{N\times\mu\times(N+M+1)}. \quad (26)$$

In order to convert $a_{i,j,l}$ to 0-1 integers, we do the following transform:

$$a_{i,j,l}=\begin{cases}0, & a_{i,j,l}<a_{\mathrm{Th}},\\ 1, & a_{i,j,l}\geq a_{\mathrm{Th}},\end{cases} \quad (27)$$

where a_{Th} is the threshold for the action output. Through the above operation, we successfully convert the direct action output of neural networks into the offloading matrix consisted of 0-1 integers.

4.6 Complexity analysis

The computational complexity of neural networks is affected by many factors, e.g., model

complexity, data size as well as algorithm structure [47, 48]. Analysis of computational complexity in neural networks is very complicated. Existing researches rarely address such issues [11, 14]. To simplify this problem, we focus on analyzing the computational complexity of generating the optimal action.

4.6.1 Time complexity analysis

Through analyzing the algorithm structure of DQN and A3C, we could find that the optimal action of the DQN is given by

$$a_x = \text{argmax}_a Q(\phi(s_x), a : \theta), \tag{28}$$

while the optimal action of the A3C is calculated by

$$a_x = \pi(a_x \mid s_x : \theta^t). \tag{29}$$

In each iteration, the agent in the DQN traverses all the actions to find the optimal action with the maximal Q value. In our system, there are $2^{N\times\mu\times(N+M+1)}$ available actions, therefore the corresponding time complexity is $O\left(2^{N\times\mu\times(N+M)}\right)$. In contrast to the DQN, the agent in the A3C directly generates the optimal action with respect to the policy. In order to parse the obtained action set to 0-1 integers, the agent would traverse the action matrix and perform the conversation. Thus the time complexity of the A3C is $O\left(N\times\mu\times\left(N+M\right)\right)$.

4.6.2 Space complexity analysis

Generally, the optimal action of the DQN algorithm is chosen from the fixed action set. In our system, we must maintain a fixed action set, the size of which is $2^{N\times\mu\times(N+M+1)}$. Therefore, the space complexity of the DQN algorithm is $O\left(2^{N\times\mu\times(N+M)}\right)$. In contrast to the DQN algorithm, the A3C algorithm directly outputs the offloading action matrix. The space complexity of the A3C algorithm is linear to the size of the action matrix. Thus the space complexity of the A3C algorithm is $O\left(N\times\mu\times\left(N+M\right)\right)$..

Through the above analysis, we conclude that our A3C based algorithm can significantly decrease the computational complexity of the system in the perspectives of both time and space complexity. Our A3C based algorithm can work perfectly in networks with massive computing nodes due to its low computational complexity. Meanwhile, it can reduce the computational burden on the network controller. Furthermore, it has higher economic benefits and can save operator's investment in network equipment.

V, Performance Evaluation and Discussion

This section presents the numerical results to illustrate the cost performance of our proposed computing and communication allocation algorithm. We assume that there are 40 vehicles and 3 RSUs/MEC servers in the network. The computing capability of the vehicle and the MEC server is respectively 1 GHz and 20 GHz. The total bandwidth of the network is 100 MHz, which is evenly allocated to each vehicle. The communication distance for V2V link and vehicle-to-RSU (V2R) link is respectively 20 m and 200 m. The moving speed of vehicles is 20 km/h and the moving direction of each vehicle changes from 0 to 2 π. To facilitate reader's understanding, the key parameters are listed in Table 2. Note that the simulation parameters are based on the real measurements in [49].

In order to comprehensively reveal the performance of our A3C based algorithm, we choose three additional offloading methods as simulation comparisons. They are the local execution method, greedy scheme, and the DQN based algorithm proposed in [11, 14]. For AR is the most emerging application in recent years, we choose the application ARCore mentioned in Figure 2 as the simulation application. Next, we will verify the performance from five perspectives, respectively the MEC computational capacity, the number of vehicles, the averaged task size, the moving speed of vehicles, and the CPU frequency of vehicles. Typically, vehicles generate computation tasks at a certain frequency. We set the task generating frequency $\vartheta = 0.5$, which means that the probability of generating

tasks in a time frame is 0.5. By analyzing the ARCore application, we find that the whole application must be accomplished within the deadline time while the execution delay of each component is not restricted. Therefore, we convert the constraint for the deadline time to $\sum_{j=1}^{\mu+1} t_{i,j}(x) + \sum_{j=1}^{\mu} p_{i,j}(x) \le \sum_{j=1}^{\mu} T_{i,j}(x)$.

In this paper, we adopt the average deadline time as a benchmark to indicate whether the computation tasks can be accomplished within the specified time. Next, we start to analyze the performance of our proposed A3C based algorithm on the basis of abundant simulation results.

Figure 5, and Figure 6 present the learning curve of the A3C based algorithm and the DQN based algorithm. In order to obtain stable experimental results, we run the two algorithms for 800 episodes with 400 iterations per episode. Let's look at Figure 5, and Figure 6, each picture consists of five sub pictures. Each sub picture represents the learning curves with the changing of one parameter. The results indicate that both the A3C based algorithm and DQN based algorithm converge after a period of training. The pictures also show that the rewards are eventually positive. According to the reward setting in subsection 4.4, this indicates that our A3C based algorithm achieves superior performance than the comparison algorithms. Meanwhile, the pictures also reveal that the learning curve of the A3C based scheme is more stable than the DQN based scheme. This is exactly the advantage of the A3C based scheme. By utilizing multi-process technology, the A3C based scheme converges faster and its performance is more stable [24].

Then let us discuss the performance of the A3C based scheme with the changing of simulation parameters. We classify the five perspectives into two groups: a) the vehicle hardware group; b) the network environment group. The vehicle hardware group includes the variables related to the vehicle's hardware, like the CPU frequency of vehicles, the task size, and the moving speed. While the network environment group consists of the variables related to network environments such as the computational capacity of MEC servers and the number of vehicles.

Figure 7, Figure 8, and Figure 9 present the simulation results belonging to the vehicle hardware group, which respectively show the average cost versus the task size, the moving speed of vehicles, and the CPU frequency of vehicles. Note that the yellow line denotes the average deadline time. In Figure 7, we increase task size from 50 kbit to 300 kbit to investigate the impact of task size. The simulation result indicates that the average cost increases as the task size increases. This is because the processing time and transmission time is proportional to the task size. Thus the execution delay increases with the task size. Due to lacking researches that investigate the influence of the moving speed of vehicles, we do related simulations in Figure 8. The result reveals that when the moving speed increases, the average costs of the greedy algorithm, the DQN based algorithm and the A3C based algorithm all increase and approach the average cost of the local execution. This issue is following the mobility model defined in subsec-

Table II. *Simulation parameters.*

Parameter	Value Range
μ	5
M	3
N	40
η	1 s
B	100 MHz
$F^{\max_k}$	20 GHz
$F_u^{\max}$	1 GHz
$I_{i,j}$	100 kbit
$C_{i,j}$	1000 cycle/bit
$T_{i,j}$	1.2 ms/kbit
P_i	23 dBm
σ^2	-114 dBm
$D_u^{\max}$	20 m
$D_k^{\max}$	200 m
v	20 km/h
di	[0,2 π]
ι	[0,0.7]

tion 2.5. The faster a vehicle moves, the more difficult it can establish a stable link with a nearby vehicle or an RSU and the more it relies on the local execution. Later, we research the impact of the CPU frequency of vehicles. As shown in Figure 9, the average cost decrease with the increasing of the CPU frequency. This is because a powerful CPU processor could significantly deduce the processing time in vehicles.

Figure 7, Figure 8, and Figure 9 also reveal that our A3C based algorithm performs better than the DQN based scheme and the greedy algorithm. On average, the execution delay of the A3C based algorithm is 20% ~ 30% less than the DQN based scheme and 30% ~ 40% less than the traditional greedy method. Based on the simulation results, we could infer that the more abundant the computing resources in vehicles compared to the MEC servers, the worse the offloading mechanisms perform. This is because there is no need to offload computation tasks from a resource-rich vehicle.

Figure 10, and Figure 11 shows the impact of network environments. As illustrated in Figure 10, the average cost decreases with the increasing of the computation capacity in MEC servers. The related reason is that if the computation capacity of MEC servers increases, the average assigned computing resources naturally rise, which deduces the processing time in MEC servers. The simulation result in Figure 11 is very relevant to Figure 10, the growing number of vehicles deduces the computing resources that vehicles obtain from MEC servers, which leads to the increase of the cost. Furthermore, Figure 10, and Figure 11 reveal that the impact of network environments is smaller than vehicle hardware, such as the cost changes slowly with the increase of the computational capacity in MEC servers and the number of vehicles. This is because despite the average computation resources in MEC servers decrease, vehicles can offload computation tasks to nearby vehicles.

Finally, let's make a conclusion. The simulation results in Figure 7, Figure 8, Figure 9, Figure 10, and Figure 11 verify the performance of our proposed A3C based algorithm. On average, the execution delay of the A3C based algorithm is significantly less than the DQN based scheme [36, 29] and the traditional greedy method. In the vast majority of cases, the average execution delay of our proposed algorithm is less than the deadline time, which guarantees the QoS of users. Moreover, by investigating the simulation results, we find that the deployment of D2D offloading

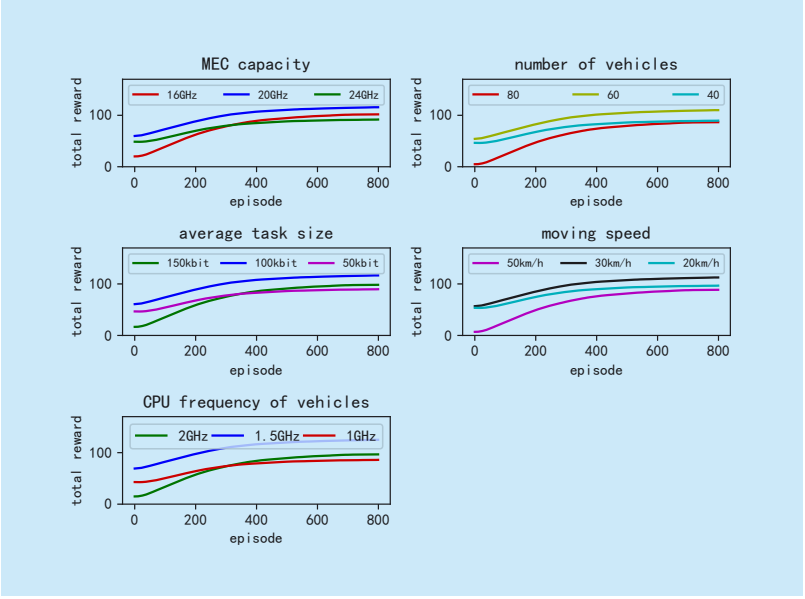


Fig. 5. *Learning curve of A3C.*

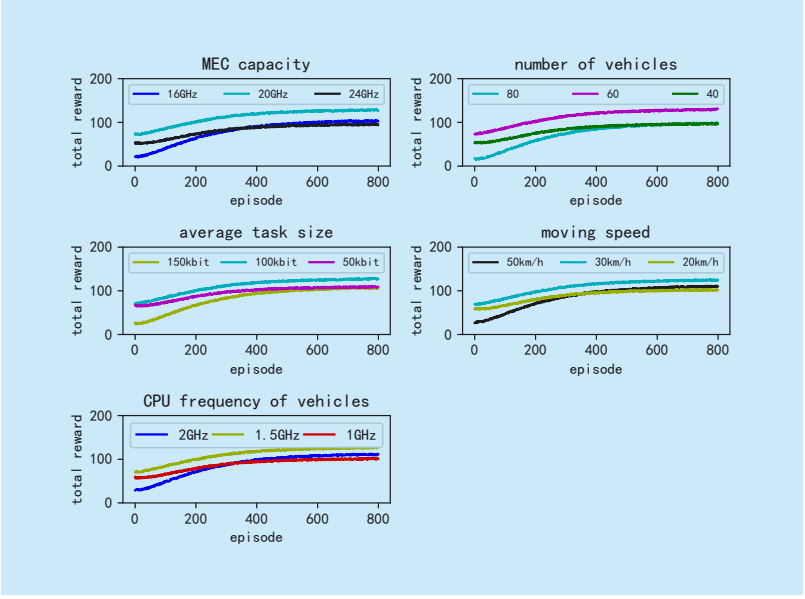


Fig. 6. *Learning curve of DQN.*

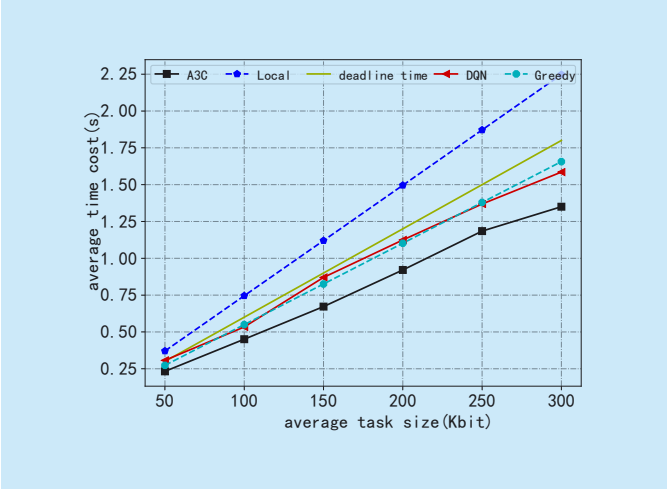


Fig. 7. *Average time cost vs the average task size.*

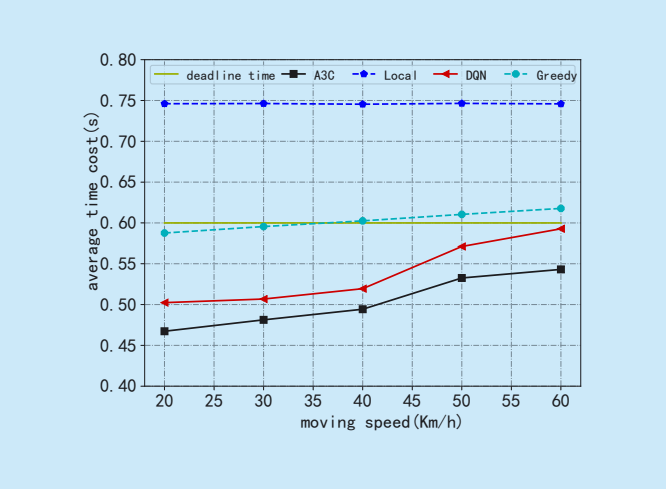


Fig. 8. *Average time cost vs the moving speed of vehicles.*

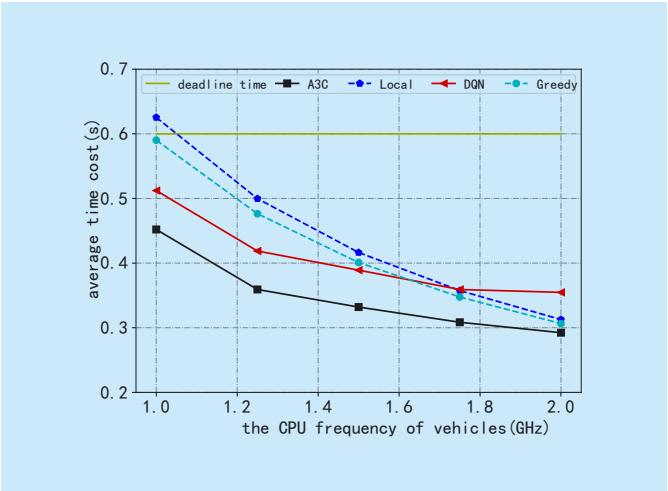


Fig. 9. *Average time cost vs the CPU frequency of vehicles.*

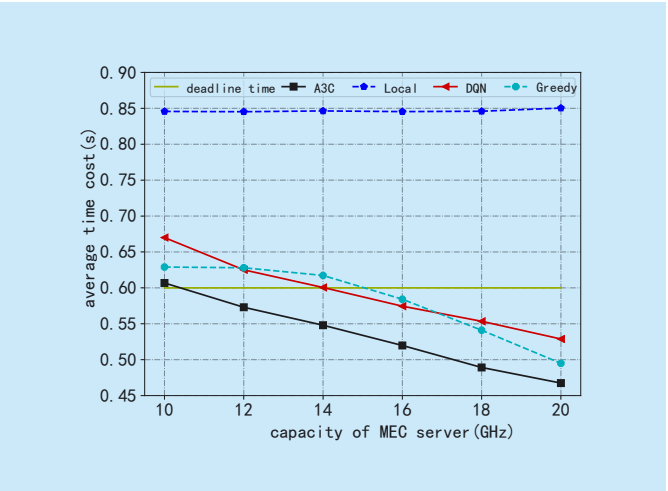


Fig. 10. *Average time cost vs the computation capacity of the MEC server.*

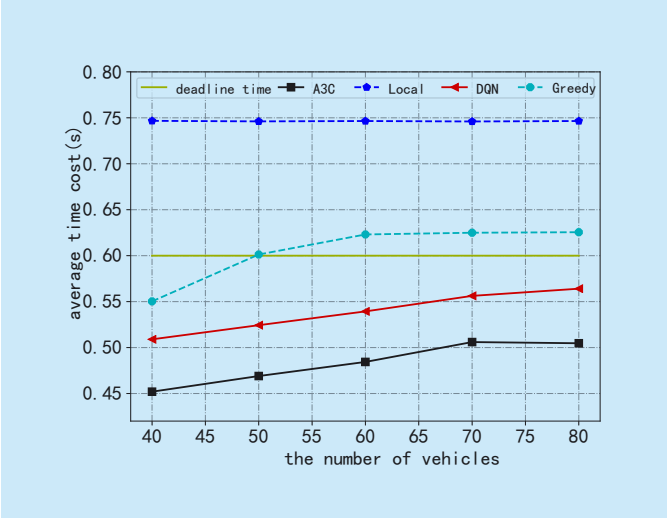


Fig. 11. *Average time cost vs the number of vehicles.*

can significantly enhance the computing capability of vehicular networks. By taking advantage of diverse task offloading methods, the task execution delay is significantly reduced. Meanwhile, the dependency on MEC servers is also reduced via integrating D2D offloading. Generally, the D2D offloading can help operators reduce investment in network hardware.

VI. Conclusion

In this paper, we research the partial computation offloading in vehicular networks. By investigating the architectures of emerging applications, we adopt a sequentially multi-component model to characterize the complex computation tasks. A novel mobility model based on motion trajectory prediction is proposed to indicate the mobility intensity of vehicles.

Then, we formulate the computation offloading and resource allocation problem from the single-vehicle scenario to the multi-vehicle scenario by taking multiple constraints into account. Due to the large action and state space of the optimization problem, we propose a A3C based scheme to solve the optimization problem. Compared with existing DQN based scheme, our A3C based scheme has lower time and space complexity. It can accommodate more users under the same hardware conditions. Meanwhile, it has higher economic benefits and can save operator's investment in network equipment. Finally, we investigate the performance of our proposed algorithm in multiple perspectives, which can perfectly reflect the complex network environment and the moving intensity of vehicles. Simulation results reveal that our proposed algorithm achieves better performance than the existing DQN based scheme as well as the traditional greedy method.

Acknowledgement

This work is financially supported by the National Natural Science Foundation of China (NSFC) (Grant No. 61671072).

References

[1] "MEC in 5G Networks", ETSI White Pages 2018.

[2] E. Ahmed and H. Gharavi, "Cooperative vehicular networking: A survey", *IEEE Trans. Intell. Transp. Syst. vo*l. 19, no. 3, 2018, pp. 996-1014.

[3] S. Barbarossa, P. Di Lorenzo, and S. Sardellitti, "Computation offloading strategies based on energy minimization under computational rate constraints", *in Proc. European Conference on Networks and Communications (EuCNC) 2014*, pp. 1-5.

[4] S. Bhatnagar, R. S Sutton, M. Ghavamzadeh, and M. Lee, "Natural actor-critic algorithms", *Automatica vo*l. 45, no. 11, 2009, pp. 2471-2482.

[5] V. D Blondel and John N Tsitsiklis, "A survey of computational complexity results in systems and control", *Automatica vo*l. 36, no. 9, 2000, pp. 1249-1274.

[6] Y. Cai, Y. Ni, J. Zhang, S. Zhao, and H. Zhu, "Energy efficiency and spectrum efficiency in underlay device-to-device communications enabled cellular networks", *China Communications vo*l. 16, no. 4, 2019, pp. 16-34.

[7] X. Chen, "Decentralized computation offloading game for mobile cloud computing", *IEEE Trans. Parallel Distrib. Syst. vo*l. 26, no. 4, 2015, pp. 974-983.

[8] X. Chen, J. Wu, Y. Cai, H. Zhang, and T. Chen, "Energy-efficiency oriented traffic offloading in wireless networks: A brief survey and a learning approach for heterogeneous cellular networks", *IEEE J. Sel. Areas Commun. vo*l. 33, no. 4, 2015, pp. 627-640.

[9] VNI Cisco, "Cisco visual networking index: forecast and trends, 2017-2022", White Paper 2018.

[10] X. Diao, J. Zheng, Y. Wu, and Y. Cai, "Joint computing resource, power, and channel allocations for D2D-assisted and NOMA-based mobile edge computing", *IEEE Access 7 2019*, pp. 9243-9257.

[11] S. Glass, I. Mahgoub, and M. Rathod, "Leveraging MANET-based cooperative cache discovery techniques in VANETs: A survey and analysis", *IEEE Commun. Surveys Tuts. vo*l. 19, no. 4, 2017, pp. 2640-2661.

[12] F. Guo, H. Zhang, H. Ji, X. Li, and V. C. M. Leung, "An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing", *IEEE/ACM Trans. Netw. vo*l. 26, no. 6, 2018, pp. 2651-2664.

[13] H. Hui, C. Zhou, S. Xu, and F. Lin, "A novel secure data transmission scheme in industrial internet of things", *China Communications 17 2020*, pp. 73-88.

[14] M. Kamoun, W. Labidi, and M. Sarkiss, "Joint resource allocation and offloading strategies in cloud enabled cellular networks", *in Proc. IEEE ICC 2015*, pp. 5529-5534.

[15] J. Kwak, Y. Kim, J. Lee, and S. Chong, "DREAM: dynamic resource and task allocation for energy minimization in mobile cloud systems", *IEEE J. Sel. Areas Commun. vo*l. 33, no. 12, 2015, pp. 2510-2523.

[16] T. P Lillicrap, J. J Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning", arXiv preprint arXiv:1509.02971 2015.

[17] J. Linowes and K. Babilinski, "Augmented reality for developers: Build practical augmented reality applications with Unity, ARCore, ARKit, and Vuforia ".

[18] M. Liu and Y. Liu, "Price-based distributed offloading for mobile-edge computing with computation capacity constraints", *IEEE Wireless Commun. Lett. vo*l. 7, no. 3, 2018, pp. 420-423.

[19] X. Liu, Y. Xu, Y. Cheng, Y. Li, L. Zhao, and X. Zhang, "A heterogeneous information fusion deep reinforcement learning for intelligent frequency selection of HF communication", *China Communications vo*l. 15, no. 9, 2018, pp. 73-84.

[20] M. Liwang, J. Wang, Z. Gao, X. Du, and M. Guizani, "Game theory based opportunistic computation offloading in cloud-enabled IoV", *IEEE*

Access 7 2019, pp. 32551-32561.

[21] S. E. Mahmoodi, R. N. Uma, and K. P. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications", *IEEE Trans. on Cloud Comput. v*ol. 7, no. 2, 2019, pp. 301-313.

[22] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: the communication perspective", *IEEE Commun. Surveys Tuts. v*ol. 19, no. 4, 2017, pp. 2322-2358.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A Rusu, J. Veness, M. G Bellemare, A. Graves, M. Riedmiller, A. K Fidjeland, G. Ostrovski, "Human-level control through deep reinforcement learning", 2015.

[24] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning", *in International Conference on Machine Learning 2016*, pp. 1928-1937.

[25] O. Munoz, A. Pascual-Iserte, and J. Vidal, "Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading", *IEEE Trans. Veh. Technol. v*ol. 64, no. 10, 2015, pp. 4738-4755.

[26] D. N. Nguyen, E. Dutkiewicz, and M. Krunz, "Harvesting short-lived white spaces via opportunistic traffic offloading between mobile service providers", *IEEE Trans. on Cogn. Commun. Netw. v*ol. 4, no. 3, 2018, pp. 635-647.

[27] Z. Ning, P. Dong, X. Kong, and F. Xia, "A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things", *IEEE Internet Things J. v*ol. 6, no. 3, 2019, pp. 4804-4814.

[28] J. Peters and S. Schaal, "Natural actor-critic", *Neurocomputing 71*, 7-9 2008, pp. 1180-1190.

[29] X. Qiu, L. Liu, W. Chen, Z. Hong, and Z. Zheng, "Online deep reinforcement learning for computation offloading in blockchain-empowered mobile edge computing", *IEEE Trans. Veh. Technol. v*ol. 68, no. 8, 2019, pp. 8050-8062.

[30] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing", *IEEE Trans. Signal Inf. Process. Netw. v*ol. 1, no. 2, 2015, pp. 89-103.

[31] Y. N. Shnaiwer, S. Sorour, T. Y. Al-Naffouri, and S. N. Al-Ghadhban, "Opportunistic network coding-assisted cloud offloading in heterogeneous fog radio access networks", *IEEE Access 7 2019*, pp. 56147-56162.

[32] S. Chen, Y. Wang, and M. Pedram, "A semi-markovian decision process based control method for offloading tasks from mobile devices to the cloud", *in Proc. IEEE Global Communications Conference (GLOBECOM) 2013*, pp. 2885-2890.

[33] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, "Mastering the game of go without human knowledge", *Nature vol*. 550, no. 7676, 2017, pp. 354.

[34] J. Síma and P. Orponen, "General-purpose computation with neural networks: A survey of complexity theoretic results", *Neural Computation 15 2003*, pp. 2727-2778.

[35] R. S. Sutton, A. G. Barto, Introduction to reinforcement learning vol. 135, (MIT press Cambridge, 1998).

[36] 36 L. T. Tan and R. Q. Hu, "Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning", *IEEE Trans. Veh. Technol. v*ol. 67, no. 11, 2018, pp. 10190-10203.

[37] 37 D. Van Le and C. Tham, "A deep reinforcement learning based offloading scheme in ad-hoc mobile clouds", *in Proc. IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) 2018*, pp. 760-765.

[38] C. J. Watkins and P. Dayan, "Q-learning", *Machine learning 8*, 3-4 1992, pp. 279-292.

[39] Z. Wei, B. Zhao, J. Su, and X. Lu, "Dynamic edge computation offloading for internet of things with energy harvesting: A learning method", *IEEE Internet Things J. v*ol. 6, no. 3, 2019, pp. 4436-4447.

[40] Y. Wu, K. Ni, C. Zhang, L. P. Qian, and D. H. K. Tsang, "NOMA-assisted multi-access mobile edge computing: A joint optimization of computation offloading and time allocation", *IEEE Trans. Veh. Technol. v*ol. 67, no. 12, 2018, pp. 12244-12258.

[41] Y. Wu, L. P. Qian, K. Ni, C. Zhang, and X. Shen, "Delay-minimization nonorthogonal multiple access enabled multi-user mobile edge computation offloading", *IEEE J. Sel. Topics Signal Process. v*ol. 13, no. 3, 2019, pp. 392-407.

[42] D. Xu, Y. Li, X. Chen, J. Li, P. Hui, S. Chen, and J. Crowcroft, "A survey of opportunistic offloading", *IEEE Commun. Surveys Tuts. v*ol. 20, no. 3, 2018, pp. 2198-2236.

[43] L. Yang, J. Cao, H. Cheng, and Y. Ji, "Multi-user computation partitioning for latency sensitive mobile cloud applications", *IEEE Trans. Comput. v*ol. 64, no. 8, 2015, pp. 2253-2266.

[44] H. Ye, G. Y. Li, and B. F. Juang, "Deep reinforcement learning based resource allocation for V2V communications", *IEEE Trans. Veh. Technol v*ol. 68, no. 4, 2019, pp. 3163-3173.

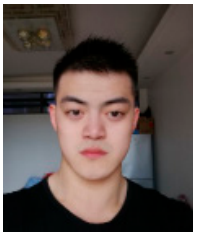
[45] W. Zhang, Y. Wen, and D. O. Wu, "Collaborative task execution in mobile cloud computing under a stochastic wireless channel", *IEEE Trans. Wireless Commun. v*ol. 14, no. 1, 2015, pp. 81-93.

[46] Y. Zhang, P. Du, J. Wang, T. Ba, R. Ding, and N. Xin, "Resource scheduling for delay minimization in multi-server cellular edge computing systems", *IEEE Access 7 2019*, pp. 86265-86273.

[47] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in ve-

hicular networks", *IEEE Trans. Veh. Technol* vol. 68, no. 8, 2019, pp. 7944-7956.
[48] J. Zheng, Y. Cai, Y. Wu, and X. Shen, "Dynamic computation offloading for mobile cloud computing: A stochastic game-theoretic approach", *IEEE Trans. Mobile Comput.* vol. 18, no. 4, 2019, pp. 771-786.
[49] H. Zhou, H. Wang, C. Zhu, and V. C. M. Leung, "Freshness-aware initial seed selection for traffic offloading through opportunistic mobile networks", *in Proc. IEEE Wireless Communications and Networking Conference (WCNC) 2018*, pp. 1-6.

Biographies



Jianfei Wang, received the B.E. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2017, and is currently pursuing the M.S. degree at the Beijing University of Posts and Telecommunications, Beijing. His research interests include mobile edge computing(MEC), Fog computing, vehicular networks, and reinforcement learning. E-mail: wangjianfei3344@bupt.edu.cn

Tiejun Lv (M'08-SM'12), received the M.S. and Ph.D. degrees in electronic engineering from the University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 1997 and 2000, respectively. From January 2001 to January 2003, he was a Postdoctoral Fellow with Tsinghua University, Beijing, China. In 2005, he became a Full Professor with the School of Information and Communication Engineering, Beijing University of Posts and Telecommunications (BUPT). From September 2008 to March 2009, he was a Visiting Professor with the Department of Electrical Engineering, Stanford University, Stanford, CA, USA. He is the author of two books and has published more than 70 published IEEE journal papers and 185 conference papers on the physical layer of wireless mobile communications. His current research interests include signal processing, communications theory and networking. He was the recipient of the Program for New Century Excellent Talents in University Award from the Ministry of Education, China, in 2006. He received the Nature Science Award in the Ministry of Education of China for the hierarchical cooperative communication theory and technologies in 2015. E-mail: lvtiejun@bupt.edu.cn

Pingmu Huang, received the M.S. degrees from Xi'an Jiaotong University, Xian, China, in 1996, and the Ph.D. degree in signal and information processing from the Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2009.,He is now a Lecturer with the School of Information and Communication Engineering, BUPT. His current research interests include speech synthesis and signal processing. He has authored or co-authored several journal papers and conference papers on speech synthesis and signal processing. E-mail: pmhuang@bupt.edu.cn

P. Takis Mathiopoulos (SM'94), received the Ph.D. degree in digital communications from the University of Ottawa, Ottawa, ON, Canada, in 1989. From 1982 to 1986, he was with Raytheon Canada Ltd., where he focused on the areas of air navigational and satellite communications. He is the TPC vice chair and the co-chair of several IEEE and other influential conferences. He has delivered numerous invited presentations, including plenary and keynote lectures, and has taught many short courses all over the world. E-mail: mathio@ieee.org