Imitation Learning Enabled Task Scheduling for Online Vehicular Edge Computing

Xiaojie Wang, Zhaolong Ning, *Senior Member, IEEE*, Song Guo, *Fellow, IEEE*, and Lei Wang

Abstract—Vehicular edge computing (VEC) is a promising paradigm based on the Internet of vehicles to provide computing resources for end users and relieve heavy traffic burden for cellular networks. In this paper, we consider a VEC network with dynamic topologies, unstable connections and unpredictable movements. Vehicles inside can offload computation tasks to available neighboring VEC clusters formed by onboard resources, with the purpose of both minimizing system energy consumption and satisfying task latency constraints. For online task scheduling, existing researches either design heuristic algorithms or leverage machine learning, e.g., deep reinforcement learning (DRL). However, these algorithms are not efficient enough because of their low searching efficiency and slow convergence speeds for large-scale networks. Instead, we propose an imitation learning enabled online task scheduling algorithm with near-optimal performance from the initial stage. Specially, an expert can obtain the optimal scheduling policy by solving the formulated optimization problem with a few samples offline. For online learning, we train agent policies by following the expert's demonstration with an acceptable performance gap in theory. Performance results show that our solution has a significant advantage with more than 50 percent improvement compared with the benchmark.

Index Terms—Vehicular edge computing, task scheduling, imitation learning, online training

1 Introduction

Owing to the establishment of smart cities and the development of networking technologies, services and applications with the characters of location and latency requirements, such as smart transportation, entertainment resource sharing as well as public safety and emergency, are increasing rapidly. Seas of data are generated daily, with the demand of an equilibrium on flexible network management and computing resources [1]. Vehicular Edge Computing (VEC) is promising to cope with challenges in the Internet of Vehicles (IoV) by migrating computing-intensive tasks to network edges [2]. Road Side Units (RSUs) are a kind of infrastructure deployed along roads to provide network accesses for various kinds of vehicles.

In order to construct green cities and effectively utilize energy to provide seamless services for passing vehicles, energy-efficient policies are necessary for VEC. According to the research report named SMARTer2030 [3], the carbon dioxide emissions caused by the utilization of information and communication technologies are growing at an average annual rate of 6 percent. By the end of 2020, they will dominate 12 percent of global emissions. Furthermore, pure and hybrid electric vehicles will dominate the automotive market in the very near future. Therefore, the huge energy demands of both network operators and vehicles motivate us to study energy-efficient policies in VEC networks.

Currently, the framework of VEC can be classified into two kinds, i.e., infrastructure-based VEC and infrastructure-free VEC. For the former one, VEC servers are always merged with RSUs to cache contents or provide computing resources [4]. For the latter one, the idle resources in moving and parked vehicles can be explored to support VEC [5]. Since infrastructure-free VEC does not require additional network deployments and can effectively integrate idle network resources, it has drawn growing attention recently.

Though infrastructure-free VEC can offer offloading services for users on roads, the task scheduling policy should be carefully designed in highly-dynamic vehicular networks. This is because vehicular applications always require real-time responses, and how to satisfy the latency-sensitive tasks in VEC networks, as well as schedule them among dynamic servers with limited service time, becomes a major issue. Nevertheless, it is rather difficult to design solutions for infrastructure-free VEC by considering the limited service time of dynamic formulated servers. Generally, the participated vehicles in the network can be categorized into two kinds, i.e., Service Providing Vehicles (SPVs) and Service Demanding Vehicles (SDVs). The former acts as a resource provider to offer computation services; the latter can access network services and offload tasks to SPVs. We mention the terms of ”SPVs” and ”VEC servers” interchangeably, since the VEC server is formed by SPVs in our designed algorithm. The challenges for infrastructure-free VEC including multiple SPVs and SDVs can be summarized as follows:

- It is difficult to efficiently utilize available resources on SPVs. Existing researches always allow SDVs to

- *Xiaojie Wang and Song Guo are with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China. E-mail: {xiaojie.wang, song.guo}@polyu.edu.hk.*
- *Zhaolong Ning is with the School of Software, Dalian University of Technology, Dalian 116024, China, and also with the Chongqing Key Laboratory of Mobile Communications Technology, Chongqing University of Posts and Telecommunications, Chongqing 400065, China. E-mail: zhaolongning@dlut.edu.cn.*
- *Lei Wang is with the School of Software, Dalian University of Technology, Dalian 116024, and also with the Peng Cheng Laboratory, 518055, China. E-mail: lei.wang@ieee.org.*

Manuscript received 7 Jan. 2020; revised 10 July 2020; accepted 24 July 2020. Date of publication 28 July 2020; date of current version 7 Jan. 2022.
(Corresponding authors: Zhaolong Ning and Song Guo.)
Recommended for acceptance by K. R. Chowdhury.
Digital Object Identifier no. 10.1109/TMC.2020.3012509

offload their tasks to neighboring SPVs [6], or send multiple copies to SPVs for processing to guarantee task delay constraints [7]. However, on one hand, merely offloading tasks to neighbors may cause unbalanced burden of SPVs and low resource utilization. On the other hand, there is a large waste of network resources by sending multiple copies. Thus, how to effectively manage the idle resources on SPVs desires to be well investigated.

- In infrastructure-free VEC, SPVs periodically broadcast their information to the surrounding SDVs, including locations, speeds and moving directions, based on which SDVs schedule their tasks. However, the frequent information exchanges can cause large communication overheads, which should be reduced to improve communication efficiency.
- For online task scheduling, traditional algorithms always adopt heuristic searching strategies. Nevertheless, they may have low searching efficiency and high computation complexity in such highly-dynamic network topologies. In addition, the limited service time, unstable VEC services and dynamic vehicle movements make online task scheduling rather challenging. Thus, it is necessary to find a flexible strategy to tackle the task scheduling issue by considering those varying parameters.
- Generally, Deep Reinforcement Learning (DRL) [8] is utilized to solve the task scheduling issue in the complex environment. However, DRL-based algorithms usually have a slow convergence speed. In addition, the frequently evolved network status caused by volatile roles, uncertain movement and a large population of vehicles, lead to super large state and action spaces for DRL. The large computation delay of the learning algorithm can result in extremely poor network performance for millions of steps during the learning process, which is not acceptable for online scheduling. Thus, it is necessary to design a learning-based algorithm that has a fast convergence speed to guarantee the online system performance.

To resolve the above challenges, this paper proposes an imitation learning enabled task scheduling algorithm for online VEC, named IELTS. For the first challenge, we establish the VEC model and formulate the task scheduling issue as an optimization problem. RSUs are leveraged to help schedule tasks among different servers, acting as the role of routers or gateways. Our solution is still infrastructure-free, since it does not require RSUs to cache or process computational tasks for SDVs. After that, we analyze the service abilities of SPVs in the coverage of RSUs, and cluster them by integrating their idle resources. To tackle with the second challenge, each RSU is utilized to maintain the cluster-level information, and the SPV-level information is maintained inside its cluster. Thus, the communication overhead between SPVs and RSUs can be largely reduced.

To resolve the last two challenges, we design an imitation learning-based method to conquer the disadvantages brought by traditional algorithms, including the low searching efficiency and the slow convergence speed. Actually, imitation learning is a kind of machine learning methods, allowing the agent to imitate reference policies (or the expert's demonstration) that are effective solutions for the original problem. Due to high time complexity, the reference policies cannot be directly executed in an online manner. Hence, a learning policy should be designed to enable imitation via training processes. To the best of our knowledge, this paper is an early effort to investigate the online task scheduling issue by boundary enabled imitation learning in highly-dynamic VEC networks. Our contributions can be summarized as follows:

- We first establish the system model by considering communication and computation resources, and formulate the task scheduling issue as an optimization problem. To solve the formulated problem, we decompose it into two sub-problems, i.e., resource aggregation and task scheduling, with the purpose of assigning tasks to suitable SPV clusters.
- To solve the first sub-problem, we analyze the service abilities of local SPVs in the coverage of RSUs, by which we reveal that the clustered SPVs can be modeled as an $M/G/K/N$ queueing system during their service time. Based on that, the formulated optimization problem can be transformed by considering the formed clusters. Then, the second subproblem can be solved by imitating the expert's policy for cluster-based task scheduling, in which the convergence time can be largely reduced.
- We analyze the algorithm performance in a theoretical manner, and prove that our algorithm can achieve an acceptable performance gap with that provided by the expert. In addition, we also derive the upper bound of energy consumption of our designed task scheduling algorithm. To the best of our knowledge, this work makes an early effort on solving task scheduling issues in VEC networks by imitation learning.
- We conduct experiments based on a dataset containing real-world taxi trajectories. Performance results show that our algorithm has superiority on the average consumed energy, task processing ratios by local SPVs, and average task execution delay with more than 50 percent improvement compared with the benchmark, which heuristically schedules tasks between VEC servers and the remote cloud.

The rest of this paper is structured as follows: in Section 2, we review the related work; we present the VEC model and formulate the studied problem in Section 3; in Section 4, we design an imitation learning enabled task scheduling algorithm for VEC, followed by performance evaluation in Section 5; finally, we conclude our work in Section 6.

2 Related Work

In this section, we review the state-of-the-art researches about VEC and imitation learning.

2.1 Vehicular Edge Computing

Recent researches on VEC focus on how to realize different purposes of network optimization by enabling terminal traffic or task offloading. According to their purposes, we can classify the studies into three categories, i.e., how to minimize the

TABLE 1
The Comparison Among Different VEC Solutions

Purposes	References	Methods	Advantages	Disadvantages
Minimizing offloading delay	[6], [7], [9]	Multi-copies, multi-armed bandit theory	Efficient utilization of idle resources, latency-sensitive	High computation and communication overhead
	[5]	One copy, heuristic algorithms		Low search efficiency, high computation complexity
Minimizing energy consumption	[10]	One copy, DRL	Energy efficient, online scheduling	Additional delay can be caused
	[11], [12]	One copy, heuristic algorithms		Limited service time of edge servers is not considered
Minimizing system costs	[13], [14]	One copy, game theory	Joint communication, caching, and computation scheduling	Do not consider user mobilities
	[15], [16], [17]	One copy, DRL		Slow convergence speed, poor initial performance

offloading delay, how to minimize energy consumption and how to minimize the system cost. Different technologies, such as heuristic algorithms, game theory and machine learning, have been employed in recent studies to improve system performance. In addition, one copy or multi-copies of tasks are generated by different algorithms, aiming to improve resource utilization or reduce system delay. We summarize representative researches and provide comprehensive comparisons among them, as shown in Table 1.

Although there are many researches considering different aspects of VEC, few studies focus on how to efficiently utilize the clustered idle resources of vehicles on roads. In [5], the moving vehicle-based fog nodes are modeled as an $M/M/1$ queueing system, which is too idealistic. In the $M/M/1$ queueing model, the arriving flow follows a Poisson distribution and is processed by a server with the service time following a negative exponential distribution. However, in real-world scenarios, there can be more than one server, and the service time of each task mainly depends on the required CPU and the processing abilities of servers. By contract, this paper models the clustered SPVs as an $M/G/K/N$ queueing system, in which there are K servers (representing K SPVs in the cluster) and their service time follows a general distribution with maximum queueing length N. Thus, it is more realistic in general.

2.2 Imitation Learning

Imitation learning is traditionally utilized in the robotic control area, e.g., automatic driving and robotic motion planning [18]. Generally, it concerns on matching the performance of the expert. An efficient learning policy is presented in [19], which iteratively generates new policies according to the demonstrator by providing additional feedback to the learning agent. Robotic motion planning problems are investigated in [20], where imitation learning is leveraged to obtain a heuristic policy that can largely alleviate the search effort compared with traditional search-tree based methods. Policies for automatic end-to-end driving are trained via imitation learning in [21], significantly improving the performance in both realistic simulations and a physical robotic vehicle.

Regrettably, the applications of imitation learning in wireless networks are almost vacant except the scheme designed in [22]. Compared with it, the improvement made in this paper can be summarized as: 1) We utilize imitation learning to resolve the task scheduling issue in highly-dynamic networks, where vehicle mobility, vehicle quantities and network topologies are all dynamic. However, the algorithm in [22] considers a static resource allocation scenario; 2) We improve the performance of imitation learning by guaranteeing an acceptable theoretical gap, which is ignored in [22]; 3) We consider a heterogeneous network architecture, including not only the core networks and RSUs, but also vehicles. Specially, the resources of vehicles are integrated for vehicle-as-a-service VEC.

3 System Model and Problem Formulation

In this section, we first overview the designed system, and then illustrate service delay and energy consumption models. At last, we formulate the problem.

3.1 System Overview

As shown in Fig. 1, when an SDV moves into the wireless communication coverage of an RSU, it can upload tasks to the RSU, and then that RSU distributes those tasks to SPVs for processing. Without loss of generality, it is assumed that the vehicle flow entering the wireless communication coverage of an RSU follows a Poisson distribution with arrival rate λ [23]. Symbols λ_d and λ_p represent the arrival rates of SDVs and SPVs, respectively, satisfying $\lambda_d + \lambda_p = \lambda$.

We consider that a set of RSUs are deployed along a road, denoted by $R = \{R_1, \ldots, R_j, \ldots, R_J\}$, where J is the total number of RSUs. All of them are connected to a back-

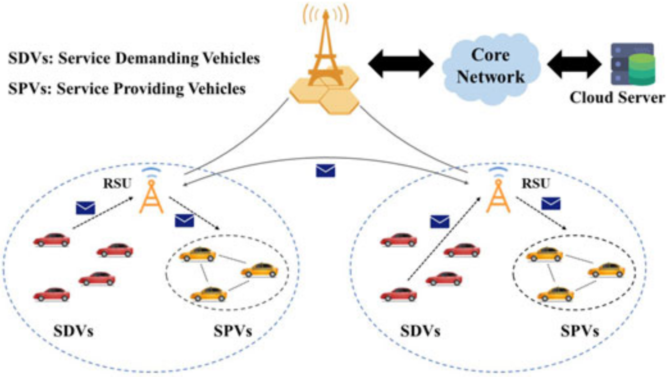

SDVs: Service Demanding Vehicles
SPVs: Service Providing Vehicles
Core Network
Cloud Server
RSU
RSU
SDVs
SPVs
SDVs
SPVs

Fig. 1. Illustrative system model.

end intelligent transportation system center by cellular communications or fiber links, and the center can be leveraged as a learning agent. In addition, neighboring RSUs are also connected, and broadcast their current status periodically, e.g., computational abilities of local clusters. The SPVs in the wireless coverage of RSU R_j can be represented by $\mathbb{S} = \{\mathbb{S}_1, \ldots, \mathbb{S}_k, \ldots, \mathbb{S}_{K_j}\}$, and K_j is the number of SPVs in $\mathbb{S}$.

Due to the instability of SPVs, we consider a remote cloud server exists, which can be integrated with a macro cell to provide highly computational capacities. Tasks are processed by SPVs preferentially since they are close to SDVs, while they are uploaded to the cloud server if their delay constraints cannot be satisfied by SPVs. The reasons for the delay caused by SPVs are: 1) The inapposite number of available SPVs; 2) Their insufficient computation capabilities; and 3) The large number of offloaded tasks. A vehicle may change its role into either an SPV or SDV when moves into the coverage of another RSU, and keeps its role until leaving. A learning agent can be deployed on an edge server near vehicles and RSUs. For example, the base station connecting the core network and RSUs in Fig. 1 can be utilized as the learning agent similar to [24]. Since the base station always has powerful computation capacities compared with RSUs and vehicles, the latency of decision-making process can be neglected [25]. Meanwhile, the main concern of this paper is to realize online task scheduling, thus we do not consider link disconnection happens during task processing. For the fault-tolerance issues in vehicular networks, detailed solutions can be found in [26].

3.2 Service Delay and Energy Consumption Models

For task i, it can be represented by a tuple $(c_i, d_i, r_i, t_i^{max})$, where c_i is its required number of CPU cycles, and d_i is its size. Symbol r_i is the result size of task i, and t_i^{max} is its latency constraint. It should be noted that SDVs are not aware of the network status, and they directly upload their tasks to the nearest RSU. Then, the RSU sends the related information of task i to the learning agent. The agent returns back the scheduling decision by following the expert's policy. Therefore, when task i is processed by the cloud, its delay can be computed by

$$t_i^c = \sum_{j=1}^{J} (t_{ij}^{v2r} + t_{ij}^{c,t})\alpha_{ij} + t_i^{c,p} + \sum_{j=1}^{J} t_{ij}^{c2v}\beta_{ij}, i \in \mathcal{I}, \tag{1}$$

where α_{ij} and β_{ij} are two binary values. Variable $\mathcal{I} = \{1, \ldots, i, \ldots, I\}$, and I is the total number of tasks in the network. If task i is uploaded to RSU R_j, $\alpha_{ij} = 1$. Otherwise, $\alpha_{ij} = 0$. If the result of task i is returned back to the SDV via RSU R_j, $\beta_{ij} = 1$, and vice versa. Symbol t_{ij}^{v2r} represents the delay of task i transmitted from an SDV to RSU R_j; $t_{ij}^{c,t}$ refers to the delay for RSU R_j to upload task i to the cloud; $t_i^{c,p}$ is the processing delay; t_{ij}^{c2v} is the delay for returning back the computation result from the cloud to the SDV. The delay caused by wireless communications can be computed by the task size divided by the achievable transmission rate of the wireless channel [27]. The computation delay can be obtained based on required CPU cycle c_i divided by the CPU-cycle frequency of the server.

Similarly, the consumed energy of task i processed by the cloud can be calculated as follows:

$$e_i^c = \sum_{j=1}^{J} (e_{ij}^{v2r} + e_{ij}^{c,t})\alpha_{ij} + e_i^{c,p} + \sum_{j=1}^{J} e_{ij}^{c2v}\beta_{ij}, i \in \mathcal{I}, \tag{2}$$

where e_{ij}^{v2r} is the consumed transmission energy from the SDV to RSU R_j; $e_{ij}^{c,t}$ represents the consumed transmission energy from RSU R_j to the cloud; $e_i^{c,p}$ is the processing energy of task i; and e_{ij}^{c2v} is the consumed energy for transmitting the result from the cloud to the SDV with the assist of RSU R_j.

When task i is processed by an SPV, its delay can be computed by

$$\begin{aligned} t_i^m = & \sum_{j=1}^{J} t_{ij}^{v2r}\alpha_{ij} + \sum_{j=1}^{J}\sum_{k=1}^{K_j} (t_{ijk}^{m,t} + t_{ijk}^{w,m})\epsilon_{ijk}\mu_{ij} \\ & + \sum_{j=1}^{J}\sum_{k=1}^{K_j} (t_{ijk}^{m,p} + t_{ijk}^{m,b})\epsilon_{ijk}\beta_{ij}, i \in \mathcal{I}, \end{aligned} \tag{3}$$

where $t_{ijk}^{m,t}$ is the transmission delay from the SDV to SPV $\mathbb{S}_k$ in the wireless communication coverage of RSU R_j; $t_{ijk}^{w,m}$ denotes the time that task i waits in the queue of SPV $\mathbb{S}_k$ for processing; $t_{ijk}^{m,p}$ refers to the consumed time for processing task i by $\mathbb{S}_k$; and $t_{ijk}^{m,b}$ is the delay caused by feeding back the result of task i from SPV $\mathbb{S}_k$ to the SDV; If task i is processed by the k_{th} SPV in the coverage of RSU r_j, $\epsilon_{ijk} = 1$; otherwise, $\epsilon_{ijk} = 0$; If task i is transmitted to RSU R_j for processing, $\mu_{ij} = 1$; otherwise, $\mu_{ij} = 0$.

The consumed energy of task i processed by SPVs can be computed based on V2R uploading energy e_{ij}^{v2r}, transmission energy $e_{ijk}^{m,t}$, processing energy $e_{ijk}^{m,p}$, and feedback energy $e_{ijk}^{m,b}$, i.e.,

$$\begin{aligned} e_i^m = & \sum_{j=1}^{J} e_{ij}^{v2r}\alpha_{ij} + \sum_{j=1}^{J}\sum_{k=1}^{K_j} e_{ijk}^{m,t}\epsilon_{ijk}\mu_{ij} \\ & + \sum_{j=1}^{J}\sum_{k=1}^{K_j} (e_{ijk}^{m,p} + e_{ijk}^{m,b})\epsilon_{ijk}\beta_{ij}, i \in \mathcal{I}. \end{aligned} \tag{4}$$

Therefore, the total execution delay for task i can be obtained by

$$t_i = \gamma_i t_i^c + (1 - \gamma_i) t_i^m, \tag{5}$$

where γ_i is a binary value. If task i is processed by the cloud server, $\gamma_i = 1$; otherwise, $\gamma_i = 0$. The consumed energy for task i can be obtained by

$$e_i = \gamma_i e_i^c + (1 - \gamma_i) e_i^m. \tag{6}$$

3.3 Problem Formulation

When an SDV arrives into the communication coverage of an RSU, it uploads computation tasks to the RSU. For task i, it can be either processed by the cloud or the clusters formulated by SPVs. The purpose of SDVs is to guarantee their task execution latency, while that of the network operator is to minimize the system energy consumption. To coordinate their distinct purposes, we formulate a task scheduling problem by minimizing the average consumed energy of offloaded computation tasks while guaranteeing their execution latency as follows:

$$\text{P1:} \quad \min_{\alpha_{ij},\beta_{ij},\mu_{ij},\epsilon_{ijk},\gamma_i,t^{w,m}_{ijk},t^{m,p}_{ijk}} \frac{1}{I}\sum_{i=1}^{I} e_i, j \in \mathcal{J}, k \in \mathcal{K}, \tag{7}$$

$$\text{s.t.} \quad t_i \leqslant t_i^{max}, i \in \mathcal{I}, \tag{8}$$

$$t_k^{ser} \geq t_{ijk}^{m,p} + t_{ijk}^{w,m} + t_{ijk}^{m,b}, k \in \mathcal{K}, \tag{9}$$

$$\sum_{j=1}^{J} \alpha_{ij} = 1, \alpha_{ij} \in \{0,1\}, i \in \mathcal{I}, \tag{10}$$

$$\sum_{j=1}^{J} \beta_{ij} = 1, \beta_{ij} \in \{0,1\}, i \in \mathcal{I}, \tag{11}$$

$$\sum_{j=1}^{J} \mu_{ij} = 1, \mu_{ij} \in \{0,1\}, i \in \mathcal{I}, \tag{12}$$

$$\sum_{j=1}^{J}\sum_{k=1}^{K_j} \epsilon_{ijk} = 1, \epsilon_{ijk} \in \{0,1\}, i \in \mathcal{I}, \tag{13}$$

where $\mathcal{J} = \{1, \ldots j, \ldots, J\}$ and $\mathcal{K} = \{1, 2, \ldots, K_j\}$. Constraints (8) and (9) guarantee the latency of task i should be less than its delay constraint, and the remaining service time of SPV $\mathbb{S}_k$ is no less than the required delay for executing task i, respectively. Constraints (10) and (11) guarantee that task i and its computation result can only be uploaded and returned by one RSU, respectively. Similarly, Constraints (12) and (13) make sure that task i is merely processed by one SPV. The important notations can be found in Table 3 in Appendix I of Supplemental File, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TMC.2020.3012509.

4 An Imitation Learning Enabled Task Scheduling Algorithm

In this section, we present the designed imitation-learning enabled task scheduling algorithm. We first provide the algorithm overview, and then specify the approaches, i.e., SPV clustering and imitation learning for task scheduling, to solve the formulated problem. At last, we elaborate the algorithm analysis.

4.1 Algorithm Overview

The problem in P1 is a Mixed Integer Non-Linear Program (MINLP) problem, and its parameters are coupled and inter-dependent, which cannot be resolved directly. This is because the movements of SDVs and SPVs lead to dynamic network topologies, unstable VEC services and limited service time. A task processing request accepted by an RSU cannot be directly scheduled to underloaded SPVs, since SPVs may move out of the RSU coverage and sufficient service time cannot be guaranteed. To design a feasible scheduling approach, we settle Problem P1 by two steps. First, we model the SPVs in the wireless communication coverage of each RSU, and estimate the waiting delay of tasks and the processing delay of SPVs. Then, we propose an imitation learning enabled task scheduling algorithm by following the expert's demonstration with a few samples.

For the first issue, we reveal the service abilities by aggregating SPVs into clusters. Clustering is a promising method to aggregate idle on-board resources to provide services for offloaded tasks. With formed clusters, the mobility of SPVs can be recorded and managed locally in each cluster by the cluster head, so that RSUs do not need to update frequently to record the states of SPVs. The cluster head is an SPV in the cluster, and can be determined by the existing clustering algorithm, such as [28], [29], [30]. At the beginning of each time slot, RSUs update their local information of the formed clusters, and estimate their future status. In Section 4.2, we prove that the clustered SPVs can be modeled as an $M/G/K/N$ queueing system. Then, the formulated problem in P1 can be relaxed and transferred into P3.

For the second issue, we first solve the relaxed problem in P3 by the branch-and-bound algorithm with a few iterations to get the expert's demonstration. Then, in each time slot h, when task i arrives, the learning agent solves the formulated problem in P3 by imitation learning. If a suitable solution can be found, task i can be assigned to the corresponding cluster according to the binary values of $\alpha_{ij}, \beta_{ij}, \mu_{ij}, \epsilon_{ijk}^{'}, \gamma_i$, and the objective value can be obtained by F^*. Otherwise, task i is transmitted to the cloud server for processing. The detailed description can be found in Section 4.3, and the pseudo-code of our presented IELTS is illustrated in Algorithm 1.

Algorithm 1. Pseudo-Code of IELTS

```
Input: Status of RSU group $R$, the cloud server, vehicle flow
    with arrival rate $\lambda$.
Output: Task allocation policy $\alpha_{ij}^*$, $\beta_{ij}^*$, $\mu_{ij}^*$, $\epsilon_{ijk}^{*'}$, $\gamma_i^*$.
Initialize queueing buffers and training dataset
$D \leftarrow \emptyset$;
Initialize initial policy $\hat{\pi}_1$ to any policy in $\Pi$;
for each time slot $h = 1, 2, \ldots, H$ do
  Update local VEC clusters in each RSU coverage;
  for task $i = 1, 2, \ldots, I^h$ do
    $r = r + 1$;
    if $r\%m = 0$ then
      $D \leftarrow D \cup D_l$;
      $l = l + 1$;
      Train classifer $\hat{\pi}_l$ on $D$;
      Initialize sub-dataset $D_l \leftarrow \emptyset$;
      Set policy $\pi_{mix} = \eta_l \pi^* + (1 - \eta_l)\hat{\pi}_l$;
    end
    Form current Problem P3;
    $F_q, \alpha_{ij}, \beta_{ij}, \mu_{ij}, \epsilon_{ijk}^{'}, \gamma_i \leftarrow$ solve the relaxed Problem P3';
    while Subproblem-set $L \neq \emptyset$ do
      $F_q, \alpha_{ij}, \beta_{ij}, \mu_{ij}, \epsilon_{ijk}^{'}, \gamma_i \leftarrow$ solve current sub-problem $L(0)$;
      $F^* = \min(F^*, F_q)$;
      $a_q = \pi_{mix}(L(0))$;
      if $a_q = A_2$ then
        Add two sub-problems to $L$ by $L \leftarrow \{L(0,0), L(0,1)\}$
      end
      $L.pop(0)$ ▷ Delete the solved sub-problem from $L$;
    end
    $\alpha_{ij}^*$, $\beta_{ij}^*$, $\mu_{ij}^* \leftarrow \alpha_{ij}, \beta_{ij}, \mu_{ij} | F^*$;
    $D_l \leftarrow D_l \cup < s_q, \pi^*(s_q) >$ .
  end
end
```

4.2 SPV Clustering

The vehicle flows entering the wireless communication coverage of an RSU are considered to be steady during a short time period [31]. The task scheduling problem in time slot h can be treated with fixed arrival rate λ^h, and $\lambda_d^h + \lambda_p^h = \lambda^h$ holds. The location of SPV $\mathbb{S}_i$ in the wireless communication coverage of RSU R_j at time instant t is represented by $p_i(t)$,[1] and the maximum communication range of two SPVs is d_0. Since many researches have investigated the clustering algorithm [28], [29], [30], we do not intend to specify it here. Our designed method is compatible with those clustering algorithms. As a result, we merely consider how can an SPV join in an existing cluster, and estimate the service time and abilities of clusters. When an SPV prepares to join in a cluster, it should satisfy the following condition:

Theorem 1. *When SPV $\mathbb{S}_i$ moves in the wireless communication coverage of RSU R_j, it can join in cluster $\mathbb{C}_k$ if the connection time of $\mathbb{S}_i$ and $\mathbb{S}_k$ in $\mathbb{C}_k$ satisfies $t_{i,k}^{link} \geq t_{i,k}^{in}$. Herein, $t_{i,k}^{link} = (d_0 - |p_i(t) - p_k(t)|)/|v_i - v_k|$, $t_{i,k}^{in} = min\{(p_{r_j} + r - p_i(t))/v_i, (p_{r_j} + r - p_k(t))/v_k\}$, $|p_i(t) - p_k(t)| \leq d_0$, and v_i is the speed of $\mathbb{S}_i$.*[2]

The proof can be found in Appendix A of Supplementary File, available online.

Based on Theorem 1, we can obtain the condition whether an SPV can join in a cluster to form VEC clusters, i.e., the SPV preparing to join in a cluster should have a close distance with a member in the cluster before they leave the coverage of the RSU. In other words, if the above condition cannot be satisfied, the SPV cannot join in the cluster. After that, the service time of local clusters should be learned to satisfy task latency constraints. The service time of each cluster can be estimated by the following corollary:

Corollary 1. *In time slot h, the average service time of cluster $\mathbb{C}_k$, the size of which at beginning time τ_0 of time slot h is n_k^h, can be computed by $E(t_k^{service}) = \sum_{i=1}^{n_k^h} t_i^{h,in}/n_k^h$, where $t_i^{h,in} = min\{\tau^h - (t - \tau_0), t_i^{in}\}$, and $\tau_0 \leq t < \tau_0 + \tau^h$.*

The proof can be found in Appendix B of Supplementary File, available online.

After getting the condition for an SPV to join in an existing cluster and the service time of a cluster, the service ability of each cluster should be also estimated to reasonably assign tasks. As a result, we derive the following theorem to estimate the service ability of a cluster:

Theorem 2. *The average number of SPVs in cluster $\mathbb{C}_k$ in time slot h can be computed by $E(n_k^h) = n_k^{h-1} + E(n_k^{h,j})$, where n_k^{h-1} is the number of SPVs in cluster $\mathbb{C}_k$ during time slot $h - 1$. Symbol $E(n_k^{h,j})$ is the number of newly joined SPVs of $\mathbb{C}_k$ in time slot h, where $E(n_k^{h,j}) = \sum_{i=1}^{\Delta n_k^h} i\mathbb{P}_{i,k}$. Variable $\mathbb{P}_{i,k}$ is the probability that vehicle v_i joins in cluster $\mathbb{C}_k$, and Δn_k^h is the average number of vehicles arriving in the coverage of RSU R_j during the period that cluster $\mathbb{C}_k$ is also in that coverage in time slot h.*

The proof can be found in Appendix C of Supplementary File, available online.

Based on the above theorem, we can acquire the service time, the service ability, and the condition for an SPV to join in a cluster. We assume that each SPV can provide equal idle resource c^s to form clusters. Then, the clustered SPVs can be modeled by the following theorem:

Theorem 3. *Cluster $\mathbb{C}_k$ in the coverage of RSU R_j can be modeled as an $M/G/K/N$ queueing system in time slot h, where $K = E(n_k^h)$, and $N = c^s E(t_k^{service} n_k^h / c_i)$.*

The proof can be found in Appendix D of Supplementary File, available online.

Based on Theorem 3, the aggregated computing resources of clusters can be modeled as an $M/G/K/N$ queueing system. Thus, the cluster information, including its size and service time, merely needs to be updated at the beginning of each time slot. RSUs can estimate their service abilities based on the above theorems, which can largely reduce the communication overhead and the maintenance cost of RSUs. Then, the RSU can assign tasks to each cluster based on its formulated queueing model. The delay for task i waiting in the queue of cluster $\mathbb{C}_k$ can be computed by Equation (8) in [32]. Thus, Problem P1 can be transferred into

$$\text{P2:} \quad \min_{\alpha_{ij}, \beta_{ij}, \mu_{ij}, \epsilon_{ijk}^{'}, \gamma_i} \frac{1}{I^h} \sum_{i=1}^{I^h} e_i, j \in \mathcal{J}, k \in \mathcal{K}', \tag{14}$$

$$\begin{aligned} \text{s.t.} \quad & \text{Constraints } (8), (9), (10), (11), (12), \\ & \sum_{j=1}^{M} \sum_{k=1}^{n_j} \epsilon_{ijk}^{'} = 1, \epsilon_{ijk}^{'} \in (0, 1), n_j \in \mathcal{N}, i \in \mathcal{I}, \end{aligned} \tag{15}$$

where I^h denotes the total number of computation tasks in time slot h. We use $\epsilon_{ijk}^{'}$ to replace ϵ_{ijk}, representing whether cluster $\mathbb{C}_k$ in the coverage of RSU R_j can be selected for task i. Variable $\mathcal{K}' = \{1, \ldots, C\}$ is the group of clusters, where C is the total cluster number. When task i arrives, Problem P3 needs to be solved for task scheduling

$$\begin{aligned} \text{P3:} \quad & \min F(\alpha_{ij}, \beta_{ij}, \mu_{ij}, \epsilon_{ijk}^{'}, \gamma_i), j \in \mathcal{J}, k \in \mathcal{K}', \\ \text{s.t.} \quad & \text{Constraints } (8), (9), (10), (11), (12), (15), \end{aligned} \tag{16}$$

where $F(\alpha_{ij}, \beta_{ij}, \mu_{ij}, \epsilon_{ijk}^{'}, \gamma_i) = e_i$. In our system, computation tasks can be always scheduled among the clustered SPVs if their delay constraints can be satisfied. Otherwise, the cloud server is selected. Thus, when task i arrives, we first set $\gamma_i = 0$ and solve Problem P3 by imitation learning. If no suitable result can be found, we set $\gamma_i = 1$ and task i is transmitted to the cloud server.

4.3 Imitation Learning With Experts

For Problem P3, the optimal solution is utilizing the branch-and-bound algorithm to get the minimum value of e_i for each arrival task i. Nevertheless, the multiple unknown variables and their sizes make the standard branch-and-bound algorithm time-consuming, whose time complexity is exponential growth with the number of branches. In our algorithm, the time complexity of employing the standard branch-and-bound algorithm for scheduling task i is $O(2^{J^2 C})$. For example, if a road has a length of 2,000 meters, and RSUs are deployed average 200 meters. Then, the time complexity for scheduling

[1] Note that the location can be represented by a point in a two-dimensional coordinate system.

[2] Note that the locations, speeds and distances are all vectors, thus their operations are vector operations.

one task is $O(2^{100})$, even if only one SPV cluster is in the coverage of each RSU. Consequently, we utilize the branch-and-bound algorithm at the beginning with a few iterations, the result of which can be utilized as trajectories of the expert. After that, we utilize imitation learning for online learning, which can help the learning agent make proper task scheduling decisions by mimicking the expert's demonstrations.

We model the task scheduling problem as a sequential decision process, consisting of state space S, action space A and policy space Π. The learning agent adopts policy $\pi \in \Pi$ to determine which action in A should be taken for given state s. After that, one-step loss $L(s,a) \in [0,1]$ can be obtained. Then, the state can be transferred into $s^{'}$ according to transition probability $P(s^{'}|s,a)$. Symbol d_π^t can be utilized to denote the state distribution at time t when policy π is executed from steps 1 to $t-1$, while $d_\pi = \sum_{t=1}^{T} d_\pi^t$ represents the average state distribution over total T steps. According to [19], the corresponding expected cost $\mathbb{J}(\pi)$ for executing policy π over T-steps is

$$\mathbb{J}(\pi) = \sum_{t=1}^{T} E_{s\sim d_\pi^t}[L(s,\pi(s))]. \tag{17}$$

Definition 1. *For an expert, its policy π^* (α_{ij}^*, β_{ij}^*, μ_{ij}^*, $\epsilon_{ijk}^{'*}$, γ_i^*) can be obtained by solving Problem P3 based on offline learning, which satisfies $\pi^* = \arg\min_{\pi\in\Pi} \mathbb{J}(\pi)$.*

For *offline learning*, we can conduct this process in a branch-and-bound manner. The process is as follows:

1) *Solve the relaxed problem*: To solve Problem P3, we first relax its integer constraints to non-integer ones, i.e.,

$$\text{P3}': \quad \min F(\alpha_{ij}, \beta_{ij}, \mu_{ij}, \epsilon_{ijk}^{'}, \gamma_i), j \in \mathcal{J}, k \in \mathcal{K}', \tag{18}$$

$$\text{s.t.} \quad \text{Constraints (8) and (9)},$$

$$\sum_{j=1}^{J} \alpha_{ij} = 1, \alpha_{ij} \in [0,1], i \in \mathcal{I}, \tag{19}$$

$$\sum_{j=1}^{J} \beta_{ij} = 1, \beta_{ij} \in [0,1], i \in \mathcal{I}, \tag{20}$$

$$\sum_{j=1}^{J} \mu_{ij} = 1, \mu_{ij} \in [0,1], i \in \mathcal{I}, \tag{21}$$

$$\sum_{j=1}^{J} \sum_{k=1}^{n_j} \epsilon_{ijk} = 1, n_j \in \mathcal{N}, \epsilon_{ijk} \in [0,1], i \in \mathcal{I}. \tag{22}$$

An interior point method [33] can be leveraged here to solve the above problem. If integer solutions for α_{ij}, β_{ij}, μ_{ij}, $\epsilon_{ijk}^{'}$ and γ_i can be found, the process is over. Otherwise, the original problem in P3 is branched.

2) *Branch and mark states*: We select a non-integer solution, e.g., α_{ij} with value $\bar{a}$. Then, we can generate two additional conditions, i.e., $0 \le \alpha_{ij} \le \lfloor \bar{a} \rfloor$ and $\lceil \bar{a} \rceil \le \alpha_{ij} \le 1$. Since $0 \le \alpha_{ij} \le 1$, $\lfloor \bar{a} \rfloor = 0$ and $\lceil \bar{a} \rceil = 1$ hold. Thus, two subproblems $L(0,0)$ and $L(0,1)$ can be obtained based on P3$'$. Problem $L(0,0)$ is with additional constraint $\lfloor \bar{a} \rfloor = 0$, and problem $L(0,1)$ is with $\lfloor \bar{a} \rfloor = 1$. In addition, features can be extracted to represent status s_q of node N_q in the search tree by

$$s_q = \Big\{ D_q^N, D_q^P, F_q, \alpha_{ij}, \beta_{ij}, \mu_{ij}, \epsilon_{ijk}^{'}, \gamma_i, \alpha_{ij}^*, \beta_{ij}^*, \mu_{ij}^*, \epsilon_{ijk}^{'*}, \gamma_i^*, F^* \Big\}, \tag{23}$$

where D_q^N and D_q^P are the depth and plunge depth of N_q, respectively, and F_q is the optimal objective value of the relaxed problem of P3 at N_q. Variables α_{ij}, β_{ij}, μ_{ij}, $\epsilon_{ijk}^{'}$ and γ_i are the relaxed solutions obtained at N_q, while α_{ij}^*, β_{ij}^*, μ_{ij}^*, $\epsilon_{ijk}^{'*}$, and γ_i^* are the optimal solutions obtained by all the searching processes. Variable F^* is the optimal objective value.

3) *Bound and mark actions*: By solving subproblems $L(0,0)$ and $L(0,1)$, we can find whether an integer solution exists. If integer solution F can be obtained, we compare it with minimum solution F^* ever found. If $F < F^*$, then $F^* = F$. If a non-integer solution Z exists to make $Z < F^*$, a new branch should be added based on the current subproblem. Thus, a searching tree can be formed, and its node can be recorded by each branched subproblem as well as its solution. Node N_q in the searching tree can be represented by state s_q. Action space is represented by $A = \{A_1, A_2\}$, where A_1 represents the prune action, and A_2 denotes the preservation action. The above process continues until the minimum integer solution for Problem P3 can be found. We can obtain dataset D that contains states and actions to describe the best choice for each decision, which can be utilized for expert trajectories. Thus, we can draw the following corollary:

Corollary 2. *The expert policy obtained by the above offline learning process is the best and its performance is theoretically optimal for the clustered VEC networks in an offline manner.*

For *online learning*, the learning agent imitates the expert's policies by learning which node should be pruned to accelerate the searching process, instead of exploring all possible nodes in the search tree. Thus, the learning agent can learn a policy by the following equation according to [19]:

$$\hat{\pi} = \arg\min_{\pi\in\Pi} E_{s_q\sim d_\pi} \big[L\big(s_q, \pi, \pi^*(s)\big)\big], q \in \{1, \ldots, |Q|I\}, \tag{24}$$

where $\pi^*(s)$ is the action taken by the expert in state s, and $L\big(s_q, \pi, \pi^*(s)\big)$ is the surrogate loss of policy π conducted in state s_q. Expression $|Q|I$ is the total number of nodes for all the searching processes of arrival tasks.

Definition 2. *For loss function $L\big(s_q, \pi, \pi^*(s)\big)$, we define it equals to $E_\pi\Big[(\pi_\epsilon(s_q) - a_q)^2\Big]$, where ϵ is the parameter the neural network prepares to optimize; $\pi_\epsilon(s_q)$ represents the output of the neural network; and a_q is the real action taken for node N_q in the searching tree.*

The online learning process is similar with that of offline learning. The main difference is that after computing the solution for each subproblem, the learning agent can utilize the trained policy to decide whether to branch or prune the current node in the searching tree, which can largely reduce computation delay and resources. For the imitation learning enabled task scheduling, we iteratively train policies $\hat{\pi}_1$, $\hat{\pi}_2 \ldots, \hat{\pi}_{\lceil u/m \rceil}$ as follows: m trajectories are collected by m tasks for training dataset D_r. After that, we update dataset D by adding D_r and train policy $\hat{\pi}_{r+1}$. To solve the formulated problem in P3 for each task i, we improve the standard branch-and-bound algorithm as below:

For each node in the searching process, the predicted action a_q can be obtained by a mixed policy $\pi_{mix} = \eta_l \pi^* + (1-\eta_l)\hat{\pi}_l$ at each tree node N_q, where η_l is a blending parameter. As a result, we can speed up the process of branch-and-bound by pruning unnecessary branches based on the demonstration of the expert.

4.4 Algorithm Analysis

In this subsection, we provide comprehensive analyses for our designed imitation learning enabled task scheduling algorithm.

We aggregate SPVs into different clusters to provide computing resources for SDVs, which can largely reduce the communication overhead between RSUs and SPVs. This is because the status of SPVs is maintained in each cluster, and RSUs only need to record the status of clusters periodically instead of managing the status of each single SPV. Thus, we can draw the following theorem:

Theorem 4. *In time slot h, the communication overhead reduced by clustered SPVs can be estimated by $(E(n_k^h)-1)\lambda_p^h \tau^h \mathbb{O}/E(n_k^h)$, where $\mathbb{O}$ is the communication frequency in each time slot between a pair of RSUs and SPVs, and τ^h is the time duration of time slot h.*

The proof can be found in Appendix E of Supplementary File, available online.

For imitation learning, we leverage it for task scheduling that determines which SPV cluster can be selected to process the offloaded tasks. For the expert policy, we can obtain it by offline training within a few time slots. We discover that the branch-and-bound based expert policy is the best and its performance is optimal based on the clustered SPVs in theory. For the online learning agent, it always mimics the behaviors of the expert and has a theoretical performance gap with that of the expert. The analysis is as follows:

Theorem 5. *Let $E_{\hat{\pi}}(\mathcal{F})$ be the average consumed energy for tasks over the long run under policy $\hat{\pi}$, and $E(\mathcal{F}_i|F^* \notin \mathcal{F}_i) \leq u$ represents the average consumed energy for all the possible VEC solutions except the optimal solution F^* for scheduling task i. The expected average consumed energy for executing tasks on the cloud is less than κ. Then, $E_{\hat{\pi}}(\mathcal{F}) \leq E_{\pi^*}(\mathcal{F}) + \mathcal{E}^s\kappa + \mathcal{E}^n u$ holds.*

The proof can be found in Appendix F of Supplementary File, available online. Herein, symbols κ and u are two upper bounds for average energy consumption, when policy $\hat{\pi}$ makes at least one mistake leading to no solution and a suboptimal solution of the branch-and-bound algorithm, respectively.

For our considered task scheduling issue, the optimal solution can be found by global searching, i.e., to search all available SPVs and find the most suitable SPV to process the task. However, it is time-consuming and computing-intensive since a global searching should be taken for each task, which is not suitable for online scheduling. Therefore, we propose the imitation learning based task scheduling algorithm, which is efficient and can make timely scheduling decisions by the learning agent. Since the expert policy has been proved to be optimal for the clustered VEC networks, we compare it with the theoretical optimal solution reached by the global searching for the formulated task scheduling problem. The following theorem can be obtained:

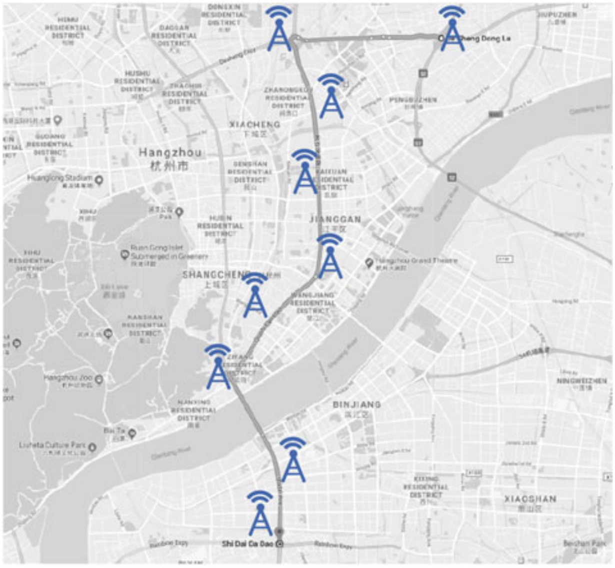


Fig. 2. The selected main road in Hangzhou (China).

Theorem 6. *The expected average consumed energy for expert policy $E_{\pi^*}(\mathcal{F})$ is bounded by $E_{\pi^*}(\mathcal{F}) \leq \frac{2}{3}E_{opt}(\mathcal{F}) + \frac{1}{3}E_b(\mathcal{F})$, where $E_{opt}(\mathcal{F})$ is the average consumed energy for the optimal solution achieved by the global searching, and $E_b(\mathcal{F})$ is the average consumed energy for scheduling tasks to the most distant SPV from the RSU that satisfies the constraints of Problem P1.*

The proof can be found in Appendix G of Supplementary File, available online.

Based on the above theorem for the expert policy, we can obtain the following theorem for the learning policy in our designed algorithm:

Theorem 7. *The expected average consumed energy for the learning agent in our imitation learning enabled task scheduling approach satisfies $E_{\hat{\pi}}(\mathcal{F}) \leq \frac{2}{3}E_{opt}(\mathcal{F}) + \frac{1}{3}E_b(\mathcal{F}) + \mathcal{E}^s\kappa + \mathcal{E}^n u$.*

The proof can be found in Appendix H of Supplementary File, available online.

5 Performance Evaluation

In order to validate the network performance of IELTS, we conduct extensive simulations based on the map of Hangzhou (China) and its real-world traces of taxies by Python 3.7 and Tensorflow 1.13.1.

5.1 Simulation Setup

The real-world dataset is collected from September 1, 2018 to September 30, 2018, in Hangzhou, China, where speeds and GPS locations of taxies are recorded. For simplicity, we only consider the task scheduling issue on one main road, and it is easy to extend the solution to other roads. As shown in Fig. 2, a road over 15 km, from the Rainbow overpass to the Siqiao Road, is selected. More than 10,000 vehicles move across the road each day. We extract the traffic flows on this road and compute the average speed of each

TABLE 2
Simulation Parameters

Parameter description	Value
The distance between two RSUs:	150 m
The wireless communication coverage r of each RSU:	150 m
The size d_i of each task:	$[100, 300, 500, 700, 900] KB$ [27]
The required CPU c_i of each task:	$[50, 60, 70, 80, 90]$ Magacycles [27]
The uplink bandwidth from vehicles to RSUs:	10 MHz
The transmission power of vehicles:	10 dBm [34]
Noise power:	-172 dBm
The downlink transmission rate from an RSU to a vehicle:	3 Mbps [7]
Task delay constraint:	[0.05, 0.1, 0.15, 0.2, 0.25] s
Task generation period:	[0.2, 0.4, 0.6, 0.8, 1] s
The computational capability of the cloud:	30 GHz [35]
The computational capability of SPVs:	[1.5, 2.5] GHz [35]

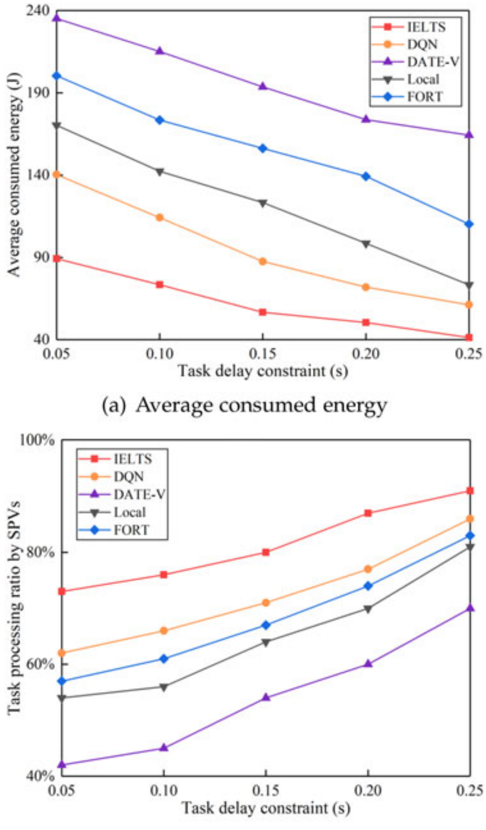
vehicle from the dataset. According to the real-world map, we virtually deploy RSUs along that road, and set the corresponding parameters according to representative references, which are summarized in Table 2. For vehicles on the road extracted from the real dataset, when they enter into the wireless coverage of an RSU, they randomly act as SDVs or SPVs. For SDVs, they generate tasks periodically. We consider that when the time interval is reached, each SDV generates one task. The total number of tasks increases when the generation period becomes small. For SPVs, they form clusters according to the clustering algorithm in [28]. The processing energy can be computed by a function of c_i and the CPU-cycle frequency f^c of the cloud [27], i.e., $e_i^p = \zeta (f^c)^2 c_i$ and $\zeta = 10^{-26}$. According to [36], the consumed transmit power from sender r_i to receiver r_j can be computed by

$$P(x_{ij}) = P_i^t (1 - \Upsilon_{ij}^S \Upsilon_{ij}^F (x_{ij})^{-\omega}), \tag{25}$$

where x_{ij} is the distance between sender r_i and receiver r_j; P_i^t is the transmit power of sender r_i; Υ_{ij}^S and Υ_{ij}^F are the channel gains caused by small-time-scale channel fading and large-time-scale shadowing, respectively; ω is the path loss factor. The consumed transmission energy between a sender and a receiver equals to the consumed transmit power multiples its transmission delay. In addition, we generate 2000 examples for training the expert's policy.

Four representative algorithms are compared with the designed algorithm:

- Deep Q Network (DQN)-based algorithm [37]: It is utilized to schedule tasks among different SPVs;
- DATE-V [7]: It sends copies of tasks to several SPVs formed by the idle on-board vehicular resources and selects servers by a learning based method;
- Local optimization [38]: First, the available SPV that satisfies the constraint of Problem P1 in the coverage of RSUs should be found. Once a server is found, the task offloading process is the same as that in [38]. That is, the task is processed by the available SPVs, and it should be uploaded to the cloud server for processing when no available SPV exists;
- FORT [5]: It is a task offloading algorithm in vehicular networks with the purpose of minimizing the task execution delay by using a heuristic algorithm.



(a) Average consumed energy

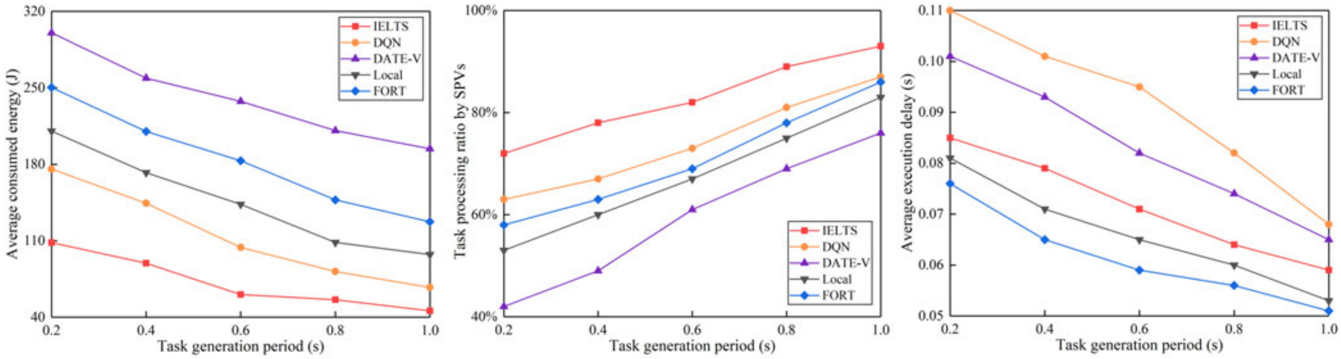
(b) Task processing ratio by SPVs

Fig. 3. Performance with different task delay constraints.

5.2 Simulation Results

5.2.1 Impacts of Task Delay Constraints

Fig. 3 illustrates the network performance with different task delay constraints. The trend of average consumed energy is shown in Fig. 3a. We notice that the average consumed energy of IELTS is much lower than that of other algorithms. For example, when the task delay constraint is 0.1s, the average consumed energy of IELTS, the DQN-based solution, DATE-V, local optimization and FORT is 73J, 114J, 215J, 142J and 173J, respectively. The reason is that IELTS employs imitation learning to imitate the policies of the expert, with an acceptable performance gap. The DQN-based algorithm learns in a trial-and-error manner, resulting in low performance in the training process and cannot always find a good solution. DATE-V transmits several copies to SPVs for processing, largely increasing the energy consumption of computation and communication. The local optimization cannot always find a suitable SPV, and has to upload its local tasks to the cloud for computation with additional energy consumption. FORT is a heuristic algorithm to minimize the task execution delay by scheduling tasks among SPVs, cloudlet and cloud servers. Thus, task execution delay is its most concerned purpose while neglecting energy consumption. When the task delay constraint becomes large, the performance of these four algorithms becomes better. For example, when the task delay constraint is 0.15s, the average consumed energy of IELTS, the DQN-based solution, DATE-V, local optimization and FORT is 57J, 88J, 193J, 123J and 156J, respectively. When the task delay constraint becomes 0.2s, the corresponding average consumed energy of the five algorithms drops to 50J, 71J, 174J, 99J



(a) Average consumed energy

(b) Task processing ratio by SPVs

(c) Average execution delay

Fig. 4. Performance with different task generation periods.

and 139J, respectively. This is because more available SPVs can be found within the constrained time, and more choices can be selected to lower the energy consumption.

Fig. 3b illustrates the task processing ratio by SPVs, indicating the ratio of tasks processed by SPVs to those generated totally in the network. It is obvious that the task processing ratio of IELTS is higher than those of the other four algorithms. For example, when the task delay constraint is 0.1s, the ratio of IELTS is 76 percent, while those of the other four algorithms are 66, 45, 56 and 61 percent, respectively. That is to say, IELTS can find a feasible policy to schedule more tasks among SPVs, which largely reduces the consumed energy compared with the tasks processed by the cloud. DATE-V delivers several copies of tasks to SPVs, which consumes more resources, and more tasks are sent to the cloud. The task processing ratios of the five algorithms become large with the increase of task delay constraint. The reason is that tasks have more available time for processing before their delay constraints, and they have more chances to be scheduled among SPVs.

5.2.2 Impacts of Task Generation Periods

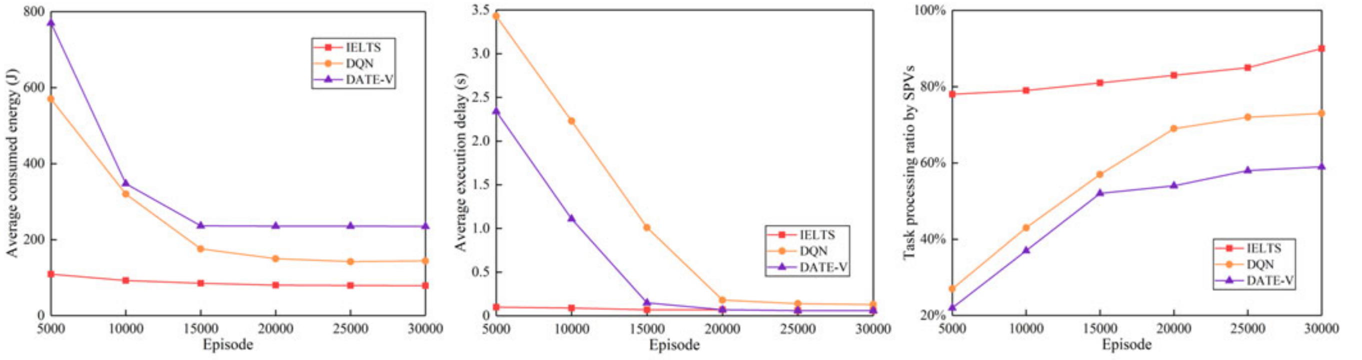
The performance impacted by different task generation periods is shown in Fig. 4. The performance trend of the average consumed energy along with the change of task generation period is shown in Fig. 4a. IELTS consumes less energy compared with the other solutions. For instance, IELTS saves 38, 48, 57 and 65 percent energy compared with those of the DQN-based solution, local optimization, FORT and DATE-V, respectively, when the task generation period is 0.4s. This is because more tasks can be processed locally by SPVs in IELTS compared with others. When the task generation period becomes large, the total number of tasks generated by each vehicle in each time slot becomes small. The reason is that there are relatively sufficient VEC resources for task offloading, so that tasks do not need to be uploaded to the cloud. For example, the energy consumption is 108J when the task generation period is 0.2s for IELTS, while dropping to 56J when the period is 0.8s.

Fig. 4b shows the performance of task processing ratio by SPVs with different task generation periods. IELTS has the highest task processing ratio compared with the other three algorithms, since it can schedule more tasks among different SPVs with the help of the learning agent to imitate the optimal policy. The trend of task processing ratio by SPVs rises with the increase of task generation period. When the task generation period is 0.4s, the task processing ratios of IELTS, the DQN-based solution, local optimization, DATE-V and FORT are 78, 67, 49, 60 and 63 percent, respectively. When the task generation period increases to 0.6s, the task processing ratios of the five algorithms become 82, 73, 61, 67 and 69 percent, respectively. This is because a smaller number of tasks are generated when the task generation period becomes large. Thus, there are more available resources of SPVs to process tasks.

Average task delay with different task generation periods is captured in Fig. 4c. When the task generation period becomes large, the average task delay decreases. This is because when the task generation period becomes large, there are a smaller number of tasks in one time slot compared with that when the generation period is small. Therefore, there are a smaller number of tasks for SPVs to process and the waiting time for tasks also becomes shorter. Correspondingly, when the task generation period becomes large, the value of average task delay drops. The algorithm of local optimization merely checks the local available SPVs in the coverage of the RSU, and does not allow tasks to be scheduled to SPVs in the coverage of other RSUs. Then, it will upload the task to the cloud, if local SPVs cannot satisfy the task latency requirement. Therefore, the average task delay of local optimization is lower than that of IELTS with the cost of more consumed energy. DATE-V sends several copies to SPVs and its delay depends on the earliest returned result, which can guarantee a relative low delay. However, multiple copies can increase the waiting delay of tasks in each SPV conversely. The average task delay of FORT is the minimum, since its purpose is to minimize the task execution delay as much as possible and it can balance tasks among SPVs, cloudlets and remote clouds by feasible task scheduling.

5.2.3 Impacts of Training Episodes

The performance of average consumed energy, average task delay and task processing ratio by SPVs is also evaluated from the aspect of training episodes, as shown in Fig. 5. The average consumed energy is illustrated in Fig. 5a, from which we can observe that the performance of IELTS is more stable and better than its counterparts. At the first 10,000 episodes, the performance of the DQN-based solution and DATE-V is extremely poor, since they are both



(a) Average consumed energy (b) Average execution delay (c) Task processing ratio by SPVs

Fig. 5. Performance with different training episodes.

based on deep reinforcement learning, which has no prior knowledge of the system and learns policies based on the interaction with the environment with a long convergence time. DATE-V begins to converge after around 15,000 episodes, while the DQN-based solution does not converge until around 20,000 episodes. Contrastly, IELTS always has low and stable performance by following the expert's policies.

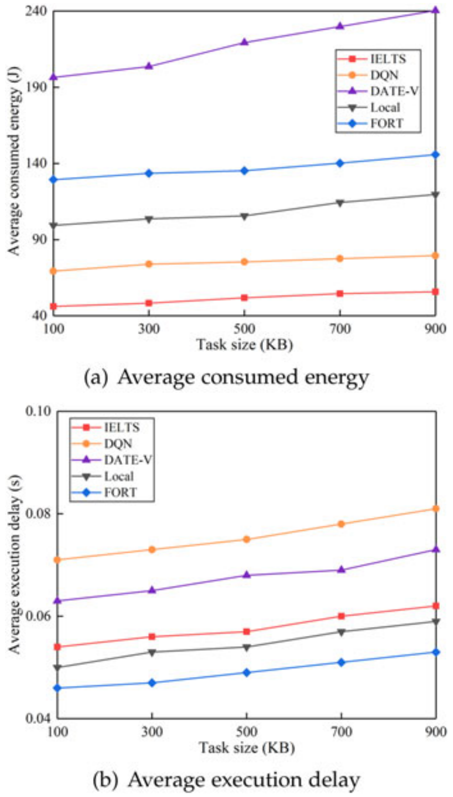
Fig. 5b is the performance trend of average task delay with different training episodes. It is obvious that IELTS has a lower average task delay compared with other algorithms at the beginning of the training process. This advantage is brought by imitation learning, which allows the learning agent to mimic the behaviors of experts from the beginning of training episodes. That is to say, the learning agent has some prior experiences about the system obtained from the expert, while other learning-based methods do not have and need to learn the environment by themselves with many training iterations. Therefore, the performance of IELTS is better than that of other learning-based algorithms. With enough training episodes, the performance of the DQN-based solution and DATE-V becomes better. This is because the delay of DATE-V depends on the earliest returned results of all the copies. The performance trend of the task processing ratio by SPVs is illustrated in Fig. 5c. IELTS has a better performance from the beginning of the experiment than its counterparts.

5.2.4 Impacts of task sizes

Fig. 6 illustrates the performance of the five algorithms based on the change of task sizes. From Fig. 6a, we can observe that when the task size becomes large, the average consumed energy of the five algorithms also increases. This is because the transmission delay for each task becomes large with the increase of task sizes. The performance of the designed IELTS is better than that of the other four algorithms, since it can find better scheduling policies for tasks. Fig. 6b illustrates the performance of average task delay based on different task sizes. When the task size becomes large, the average task delay also increases since the large size causes the increase of transmission delay. The average task delay of the designed IELTS is lower than those of the DQN-based solution and DATE-V, while higher than those of local optimization and FORT. The reason is that the purpose of FORT is to minimize the task execution delay, and it tries to find reasonable solutions by task scheduling. Local optimization utilizes single SPVs in the RSU coverage. It always uploads tasks to the cloud when no reasonable local SPV can be found. However, the DQN-based solution and DATE-V are both based on model-free learning, which has poor performance before reaching convergence.

5.2.5 Impacts of Required CPU Cycles

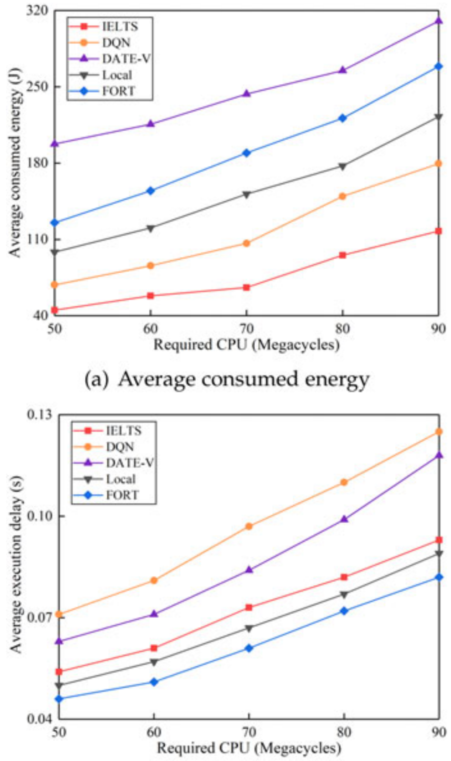
Fig. 7 is the performance trend with the change of required CPU cycles of tasks for the five algorithms. From Fig. 7a, we can observe that when the required CPU cycle of tasks becomes large, the average consumed energy of the five algorithms also increases. For example, when the required CPU of tasks is 50 Megacycles, the average consumed energy of IELTS, DQN, DATE-V, local optimization and FORT is 45.13J, 68.32J, 197.50J, 98.31J and 125.36J, respec-



(a) Average consumed energy

(b) Average execution delay

Fig. 6. Performance with different task sizes.



(a) Average consumed energy

(b) Average execution delay

Fig. 7. Performance with different required CPU cycles.

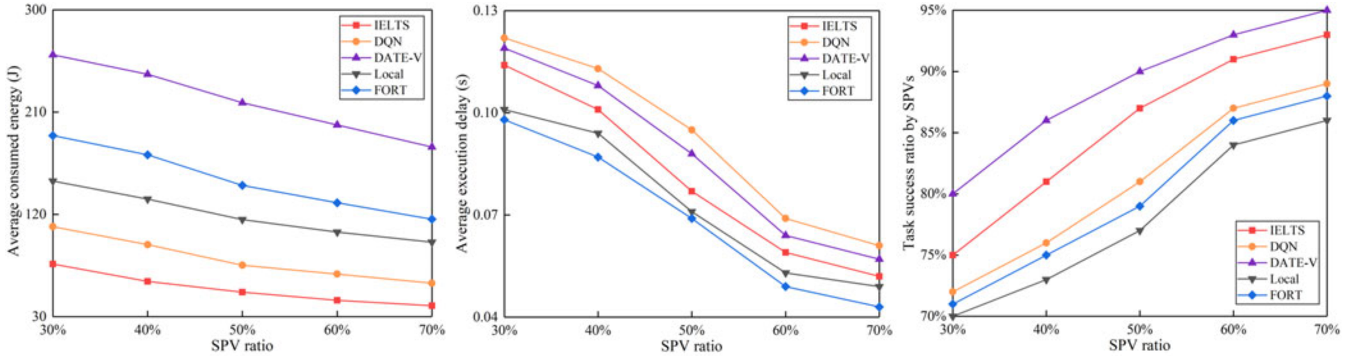
tively. When the required CPU cycle of tasks increases to 70 Megacycles, the average consumed energy of the five algorithms is 65.81J, 106.37J, 243.36J, 151.56J and 189.26J, respectively. This is because the computation delay for each task becomes large with the increase of required CPU cycles. The performance of the designed IELTS is better than that of the other four algorithms, since it can find better scheduling policies for tasks with the purpose of minimizing the average consumed energy.

Fig. 7b illustrates the performance of average execution delay of tasks based on different required CPU cycles. When the required CPU cycle becomes large, the average task delay also increases since the large required CPU cycle causes the increase of computation delay. The trends of the five algorithms are similar with Fig. 6b, but steeper than those of Fig. 6b. This is because the computation delay is the dominated factor to affect the total execution delay of tasks.

5.2.6 Impacts of SPV Ratios

The performance results based on the changes of SPV ratios are shown in Fig. 8. The SPV ratio refers to the ratio of the SPV number to the total number of vehicles in the system. Fig. 8a is the trends of average consumed energy based on different SPV ratios. We can observe that when the SPV ratio increases, the average consumed energy decreases. For example, when the SPV ratio is 0.3, the average consumed energy of IELTS, DQN, DATE-V, local optimization and FORT is 76.31J, 109.22J, 260.52J, 149.34J and 189.37J, respectively. When the SPV ratio increases to 0.5, the average consumed energy of the five algorithms is 51.61J, 75.34J, 218.24J, 115.42J and 145.58J, respectively. This is because when there are more SPVs in the system, tasks have more opportunities to be offloaded to the SPVs instead of the remote cloud. The energy consumption of the designed IELTS algorithm is the minimum in the five algorithms with similar reasons as described in the above subsection. Fig. 8b illustrates the curves of average execution delay with different SPV ratios. When the SPV ratio becomes big, the average execution delay decreases. The reason is that more SPVs provide more available resources for task offloading, and tasks can be scheduled to the most suitable SPV for processing.

Fig. 8c is the trends of task success ratio by SPVs. Herein, the task success ratio by SPVs refers to the tasks processed by SPVs and the corresponding results received by the original SDVs before the latency deadline. Due to the unstable network topology and dynamic movements of vehicles, some tasks may not be processed or their results cannot be received in time. When the SPV ratio becomes large, the task success ratio by SPVs also increases. This is because when the SPV ratio becomes large, more available resources on SPVs can be scheduled to process the offloaded tasks and their delay constraints can be met. The task success ratio by SPVs of IELTS is smaller than that of DATE-V, while larger than those of the other three algorithms. This is because DATE-V sends task copies to multiple SPVs for processing to guarantee the task success ratio at the cost of the consumed energy. However, the designed IELTS merely sends each task to one SPV for processing with the purpose of minimizing the consumed energy. In addition, our imitation-learning enabled task scheduling algorithm can mimic the expert policies to make more appropriate decisions than DQN, local optimization and FORT schemes, thus the success ratio of IELTS is bigger than those three algorithms.



(a) Average consumed energy

(b) Average execution delay

(c) Task success ratio by SPVs

Fig. 8. Performance with different SPV ratios.

6 Conclusion

In this paper, we proposed an imitation learning enabled task scheduling algorithm for online VEC, with the purpose of minimizing the system energy consumption while satisfying the task latency requirement. We first established the VEC model and formulated the service time limited task scheduling issue as an optimization problem. Specifically, we demonstrated that the clustered SPVs can be modeled as an $M/G/K/N$ queueing system during their limited service time. To the best of our knowledge, this paper was a prior attempt to leverage imitation learning for online task scheduling in VEC networks, allowing the learning agent always to follow the expert's policy with an acceptable theoretical performance gap. At last, real-world traces of taxies in Hangzhou, China were utilized to demonstrate the effectiveness of the designed IELTS method. In addition, our imitation learning enabled task scheduling algorithm can also be used in other sequential decision making problems given a few expert's demonstrations.

Acknowledgments

This research was supported in part by the General Research Fund of the Research Grants Council of Hong Kong (PolyU 152221/19E), the National Natural Science Foundation of China (Grants 61872310 and 61971084), the National Key Research and Development Plan (2017YFC0821003-2), and Dalian Science and Technology Innovation Fund (2019J11CY004).

References

[1] C. Huang, R. Lu, and K.-K. R. Choo, "Vehicular fog computing: Architecture, use case, and security and forensic challenges," *IEEE Commun. Mag.*, vol. 55, no. 11, pp. 105–111, Nov. 2017.

[2] M. Sookhak *et al.*, "Fog vehicular computing: Augmentation of fog computing using vehicular cloud computing," *IEEE Veh. Technol. Mag.*, vol. 12, no. 3, pp. 55–64, Sep. 2017.

[3] G. e-Sustainability Initiative, "SMARTer2030, ICT solutions for 21st century challenges," *Belgium, GeSI, Accenture Strategy*, c/o Scotland House Rond Point Schuman 6 B-1040 Brussels, 2015. [Online]. Available: http://smarter2030.gesi.org/downloads/Full_report.pdf

[4] K. Zhang, Y. Mao, S. Leng, Y. He, and Y. Zhang, "Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading," *IEEE Veh. Technol. Mag.*, vol. 12, no. 2, pp. 36–44, Jun. 2017.

[5] X. Wang, Z. Ning, and L. Wang, "Offloading in Internet of vehicles: A fog-enabled real-time traffic management system," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4568–4578, Oct. 2018.

[6] Y. Sun *et al.*, "Adaptive learning-based task offloading for vehicular edge computing systems," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3061–3074, Apr. 2019.

[7] L. Chen and J. Xu, "Task replication for vehicular cloud: Contextual combinatorial bandit with delayed feedback," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 748–756.

[8] N. C. Luong *et al.*, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3133–3174, Fourthquarter 2019.

[9] K. Li, M. Tao, and Z. Chen, "Exploiting computation replication for mobile edge computing: A fundamental computation-communication tradeoff study," *IEEE Trans. Wireless Commun.*, vol. 19, no. 7, pp. 4563–4578, Jul. 2020.

[10] Y. Liu, C. Yang, L. Jiang, S. Xie, and Y. Zhang, "Intelligent edge computing for IoT-based energy management in smart cities," *IEEE Netw.*, vol. 33, no. 2, pp. 111–117, Mar./Apr. 2019.

[11] F. Wang, J. Xu, and S. Cui, "Optimal energy allocation and task offloading policy for wireless powered mobile edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 19, no. 4, pp. 2443–2459, Apr. 2020.

[12] M. Li, N. Cheng, J. Gao, Y. Wang, L. Zhao, and X. Shen, "Energy-efficient UAV-assisted mobile edge computing: Resource allocation and trajectory optimization," *IEEE Trans. Veh. Technol.*, vol. 69, no. 3, pp. 3424–3438, Mar. 2020.

[13] Z. Ning *et al.*, "Mobile edge computing enabled 5G health monitoring for Internet of medical things: A decentralized game theoretic approach," *IEEE J. Sel. Areas Commun.*, pp. 1–16, 2020. [Online]. Available: https://www.researchgate.net/publication/339874488_Mobile_Edge_Computing_Enabled_5G_Health_Monitoring_for_Internet_of_Medical_Things_A_Decentralized_Game_Theoretic_Approach_IEEE_Journal_on_Selected_Areas_in_Communications_2020

[14] Q. He *et al.*, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 3, pp. 515–529, Mar. 2020.

[15] Q. Luo, C. Li, T. H. Luan, and W. Shi, "Collaborative data scheduling for vehicular edge computing via deep reinforcement learning," *IEEE Internet Things J.*, early access, Mar. 26, 2020, doi: 10.1109/JIOT.2020.2983660.

[16] X. Liu, J. Yu, J. Wang, and Y. Gao, "Resource allocation with edge computing in IoT networks via machine learning," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 3415–3426, Apr. 2020.

[17] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11158–11168, Nov. 2019.

[18] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Comput. Surv.*, vol. 50, no. 2, 2017, Art. no. 21.

[19] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, 2011, pp. 627–635.

[20] M. Bhardwaj, S. Choudhury, and S. Scherer, "Learning heuristic search via imitation," in *Proc. Conf. Robot Learn.*, 2017, pp. 271–280.

[21] F. Codevilla, M. Miiller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 1–9.

[22] Y. Shen, Y. Shi, J. Zhang, and K. B. Letaief, "LORM: Learning to optimize for resource management in wireless networks with few training samples," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 665–679, Jan. 2020.

[23] J. He, L. Cai, P. Cheng, and J. Pan, "Delay minimization for data dissemination in large-scale VANETs with buses and taxis," *IEEE Trans. Mobile Comput.*, vol. 15, no. 8, pp. 1939–1950, Aug. 2016.

[24] Z. Li, C. Wang, and C.-J. Jiang, "User association for load balancing in vehicular networks: An online reinforcement learning approach," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 8, pp. 2217–2228, Aug. 2017.

[25] Z. Ning, P. Dong, X. Wang, J. Rodrigues, and F. Xia, "Deep reinforcement learning for vehicular edge computing: An intelligent offloading system," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 6, 2019, Art. no. 60.

[26] K. Wang, H. Yin, W. Quan, and G. Min, "Enabling collaborative edge computing for software defined vehicular networks," *IEEE Netw.*, vol. 32, no. 5, pp. 112–117, Sep./Oct. 2018.

[27] J. Zhang *et al.*, "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2633–2645, Aug. 2018.

[28] D. Zhang, H. Ge, T. Zhang, Y.-Y. Cui, X. Liu, and G. Mao, "New multi-hop clustering algorithm for vehicular ad hoc networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 4, pp. 1517–1530, Apr. 2019.

[29] L. Hu, Y. Tian, J. Yang, T. Taleb, L. Xiang, and Y. Hao, "Ready player one: UAV-clustering-based multi-task offloading for vehicular VR/AR gaming," *IEEE Netw.*, vol. 33, no. 3, pp. 42–48, May/Jun. 2019.

[30] H. Xiao, Y. Chen, Q. Zhang, A. T. Chronopoulos, Z. Zhang, and S. Ouyang, "Joint clustering and power allocation for the cross roads congestion scenarios in cooperative vehicular networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 6, pp. 2267–2277, Jun. 2019.

[31] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, "Vehicular fog computing: A viewpoint of vehicles as the infrastructures," *IEEE Trans. Veh. Technol.*, vol. 65, no. 6, pp. 3860–3873, Jun. 2016.

[32] S. A. Nozaki and S. M. Ross, "Approximations in finite-capacity multi-server queues by Poisson arrivals," *J. Appl. Probability*, vol. 15, no. 4, pp. 826–834, 1978.

[33] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Math. Program.*, vol. 106, no. 1, pp. 25–57, 2006.

[34] X. Wu *et al.*, "Cars talk to phones: A DSRC based vehicle-pedestrian safety system," in *Proc. IEEE 80th Veh. Technol. Conf.*, 2014, pp. 1–7.

[35] Z. Ning, X. Wang, J. J. Rodrigues, and F. Xia, "Joint computation offloading, power allocation, and channel assignment for 5G-enabled traffic management systems," *IEEE Trans. Ind. Informat.*, vol. 15, no. 5, pp. 3058–3067, May 2019.

[36] J. Zhang, S. C. Liew, and L. Fu, "On fast optimal STDMA scheduling over fading wireless channels," in *Proc. IEEE Conf. Comput. Commun.*, 2009, pp. 1710–1718.

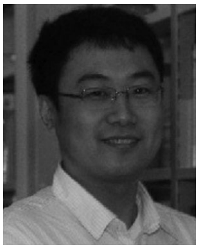
[37] D. Van Le and C.-K. Tham, "A deep reinforcement learning based offloading scheme in ad-hoc mobile clouds," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2018, pp. 760–765.

[38] 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.*, vol. 6, no. 3, pp. 4804–4814, Jun. 2019.

Xiaojie Wang received the MS degree from Northeastern University, China, in 2011, and the PhD degree from the Dalian University of Technology, Dalian, China, in 2019. From 2011 to 2015, she was a software engineer in NeuSoft Corporation, China. Currently, she is a postdoctor at the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. Her research interests include wireless networks, mobile edge computing, and machine learning.

Zhaolong Ning (Senior Member, IEEE) received the PhD degree from Northeastern University, China, in 2014. He was a research fellow at Kyushu University from 2013 to 2014, Japan. Currently, he is an associate professor at the Dalian University of Technology, China and a research fellow in The University of Hong Kong, Hong Kong. His research interests include Internet of Things, mobile edge computing, deep learning, and resource management. He has published more than 120 scientific papers in international journals and conferences. He serves as an associate editor or guest editor of several journals, such as the *IEEE Transactions on Industrial Informatics*, the *IEEE Transactions on Social Computational Systems*, the *The Computer Journal* and so on. He is the outstanding associate editor of the *IEEE Access* 2018. He is elected to be the Young Elite Scientists Sponsorship Program by CAST and Hong Kong Scholar.

Song Guo (Fellow, IEEE) received the PhD degree in computer science from the University of Ottawa, Canada and was a professor with the University of Aizu, Japan. He is a full professor with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. His research interests include Big Data, cloud computing and networking, and distributed systems with more than 400 papers published in major conferences and journals. His work was recognized by the 2016 Annual Best of Computing: Notable Books and Articles in Computing in ACM Computing Reviews. He is the recipient of the 2017 IEEE Systems Journal Annual Best Paper Award and other five Best Paper Awards from IEEE/ACM conferences. He was an associate editor of the *IEEE Transactions on Parallel and Distributed Systems* and an IEEE ComSoc distinguished lecturer. He is now on the editorial board of the *IEEE Transactions on Emerging Topics in Computing*, the *IEEE Transactions on Sustainable Computing*, the *IEEE Transactions on Green Communications and Networking*, and the *IEEE Communications*. He also served as general, TPC and symposium chair for numerous IEEE conferences. He currently serves as an officer for several IEEE ComSoc Technical Committees and a director in the ComSoc Board of Governors.



Lei Wang received the BS, MS, and PhD degrees from Tianjin University, China, in 1995, 1998, and 2001, respectively. He is currently a full professor of the School of Software, Dalian University of Technology, China. He was a member of Technical Staff with Bell Labs Research China (2001-2004), a senior researcher with Samsung, South Korea (2004-2006), a research scientist with Seoul National University, South Korea (2006-2007), and a research associate with Washington State University, Vancouver, Washington (2007-2008). His research interests include wireless ad hoc network, sensor network, social network, and network security. He has published more than 100 papers in the corresponding areas.