Hierarchical Reinforcement Learning Empowered Task Offloading for Vehicular Edge Computing

Xinyu You, Yuedong Xu, Liangui Dai

Abstract—Vehicular edge computing (VEC) plays a pivotal role in the acceleration of in-vehicle applications by offloading the intensive computation to road-side units. In this paper, we investigate the optimal task offloading policy for an in-vehicle user equipment (UE) with the purpose of balancing the tradeoff among processing delay, energy consumption and edge computing cost. Each task consists of some interdependent sub-tasks expressed by a directed acyclic graph (DAG). The decision of the UE is hierarchical: task assignment, task offloading, local computation resource control and transmission power allocation. The VEC environment is stochastic due to the random task DAGs, the random resource demand of tasks and the variable vehicle speed. In light of the inherent complexity of this problem, we propose the Deep Hierarchical Vehicular Offloading (DHVO) scheme based on deep reinforcement learning (DRL). The interdependency among the task DAGs is extracted using a graph neural network with attention mechanism that assigns different importance for the features of each sub-task. A novel parameterized DRL algorithm is proposed to deal with the hierarchical action space containing both discrete and continuous actions. Experimental results on a real-world car speed dataset demonstrate that DHVO can effectively reduce the system cost under various environment parameters.

Index Terms—Vehicular edge computing, Deep reinforcement learning, Hierarchical action space, Graph neural network.

1 Introduction

With the rapid development of vehicular networks, a variety of applications in in-vehicle user equipment (UE) such as travel assistance, augmented reality (AR), image processing and speech recognition are employed to improve the user experience for both drivers and passengers [1], [2]. These applications are usually delay sensitive and need huge computation resources to process a large volume of workload data [3]. However, the constrained computing capability and limited battery power of the UE are difficult to satisfy the computation requirements of these applications [4], [5].

With the assistance of cloud servers which possess powerful computing capability, Mobile Cloud Computing (MCC) is wildly regarded as an effective solution to tackling these issues [6]. In order to access sufficient computation resources and save energy consumption, resource-intensive applications can be offloaded to remote cloud server from the UE. Yu et al. [7] proposed a three-layer architecture of cloud resource allocation to improve the efficiency, continuity and reliability of cloud services in vehicle networks. Taking heterogeneous vehicles into consideration, Lin et al. [8] developed the optimal scheme of MCC resource allocation, which is devoted to minimizing the long-term expected consumption costs of power and time. Nabi et al. [9] presented a fully polynomial time approximation scheme for task scheduling, and then conducted a quantitative analysis of the computational power of vehicular clouds. Despite the advantages brought by MCC, the high transmission delay caused by long distance between the UE and cloud servers may lead to unacceptable latency for time-sensitive applications. Besides, the explosive growth of workload data will consume a large amount of bandwidth in backhaul networks [10].

To fully address these problems, the paradigm of Mobile Edge Computing (MEC) has been proposed, where computation services are moved to the proximity of users [11], [12]. By integrating MEC into vehicular networks, Vehicular Edge Computing (VEC) places edge servers in Road-Side Units (RSUs), by which the transmission delay can be reduced to a large extent and high bandwidth connections can be ensured during the data offloading process [13]. There exist a plethora of works that concentrate on computation offloading in VEC environment. Liu et al. [14] formulated a multi-user computation offloading game problem, proved the existence of Nash equilibrium (NE) and proposed a distributed algorithm to compute the equilibrium. Zhao et al. [15] put forward an edge caching and computation management framwork in VEC and developed an online and distributed approach to minimize the service response delay. Zhang et al. [16] presented a predictive combination-mode computation offloading scheme that greatly reduces the cost of computation offloading.

However, some critical challenges have been largely overlooked in vehicular edge computing. First, existing studies usually treat the offloading application as an “atomic” task. While with dynamic partitioning schemes [17], a computing application can be divided into several interdependent sub-tasks in the form of a directed acyclic graph (DAG). A sub-task cannot be processed until all its predecessors are mission complete. A crucial

• X. You, and Y. Xu are with School of Information Science and Technology, Fudan University, Shanghai 200237, China.
Emails: {xyyou18, ydxu}@fudan.edu.cn
L. Dai is with the Intelligent Transportation System (ITS) Research Center, Guangdong Litong Corp. Email: knuth.dai@gmail.com

question is how to make offloading decisions for these fine-grained sub-tasks without breaking the precedence constraints among them. Second, the offloading policy is usually multi-dimensional that involves both continuous and discrete, and even hierarchical decision variables. For instance, a vehicular user may choose whether a task is executed locally or at the edge, and further decide the CPU clock frequency subsequently in the former. The complex offloading problems falls in the scope of mixed integer linear programming (MINLP) that are difficult to obtain the optimality [18]. The recent trend is to employ reinforcement learning (RL) to address the complex offloading problem through trial and error [20], while standard algorithms are not suitable for handling hybrid (both continuous and discrete) and hierarchical action profiles.

In this paper, we investigate a joint computation offloading and resource allocation problem for an in-vehicle user equipment considering both task dependency and network dynamics. The local processing may consume a lot of energy and due to the limited computing capability may take a relatively longer time. Offloading task to edge servers accelerates the task processing, while experiencing the uncertain transmission delay, paying for the computing service, and risking the extra delay and charge incurred by the switching of RSUs. Hence, the total delay encountered by a task is the sum of the processing time (either locally or remotely), the transmission time and the service migration time between adjacent RSUs. The cost of processing a task is the weighted bundle of the execution delay, the edge service charge and the energy consumption, and the objective of the in-vehicle user equipment is to minimize the aggregate cost of executing all the tasks. The uncertainty of vehicular environment, i.e. the random sub-task DAG, the random resource demand of tasks and the variable vehicle speed, make the optimal decision even more complicated. The multi-dimensional policy will answer the following questions explicitly: (i) which sub-task will be selected for execution; (ii) should the selected sub-task be executed locally or offloaded to an edge server; (iii) how much CPU clock frequency should be configured for each sub-task in the local computing; (iv) how much transmission power should be allocated for each sub-task in the offloading?

In light of the inherent complexity of the formulated problem, we harness deep reinforcement learning to design a novel end-to-end solution, namely Deep Hierarchical Vehicular Offloading (DHVO). As the first innovation of DHVO, the graph neural network is introduced to extract the interdependency among the task DAGs where each sub-task is associated with the data size and the required computing resource. We implement the attention mechanism to assign different importance for the features of each node. The computation of the attention coefficient are conducted in parallel across node-neighbor pairs, despite of different degrees of each node. The impact of each node on its neighbors can be learned automatically without costly matrix operations and the access to the entire graph structure.

We further propose a novel parameterized DRL algorithm to deal with the hierarchical action space containing two discrete actions and two continuous actions. The hybrid and hierarchical action space is difficult to tackle in classical DRL. The discretization of continuous actions is restricted by the interval granularity: a small interval results in an overlarge action space that might make the training hard to converge, while a large interval will cause the performance degradation. Different from existing approaches, our design operates on the hierarchical action space directly without approximation or relaxation. We decompose the action-state value function of the hybrid action into two parts: the state value function related with discrete actions and the advantage function depending on continuous actions. The values of these two terms are estimated with a single neural network and are added correspondingly together, avoiding the over-parameterization of the action-state value function. We apply the proposed DRL algorithm to address the hybrid and hierarchical decision making in which the discrete actions are the sub-task selection and the binary offloading decision, and the continuous actions are the assignment of the local CPU frequency and the uplink transmission power.

Through the interaction with the VEC environment, the trained agent learns to dynamically adjust the computation offloading and resource allocation policy. DHVO is able to make a balance between local execution and offloading by taking both the task property and the environment information into consideration. Furthermore, DHVO tries to avoid the occurrence of task migration with speed prediction and therefore reduces migration cost greatly. To evaluate the performance of DHVO, we collect real-world car speed data and conduct several experiments by changing various environment parameters. The relationships between TESC and key parameters, including the coverage of each RSU, channel bandwidth, required computation resource of each task, are illustrated through numerical results. Compared with the counterparts, DHVO can significantly reduce the TESC in a variety of environment parameters.

The main contributions of this work are summarized as follows:

- We formulate a new joint computation offloading and resource allocation optimization problem in vehicular edge computing;
- A graph neural network with the attention mechanism is introduced to extract the interdependency among the task DAGs;
- A novel deep reinforcement learning algorithm is proposed to deal with the hybrid and hierarchical action space;
- A real-world car speed dataset based simulation is conducted to validate the efficacy of the proposed algorithm.

The remainder of this paper is organized as follows: A review of related works is presented in Section 2. Section 3 describes the system model and formulates the optimization problem. The DRL-based offloading algorithm is proposed in Section 4. Evaluation results and related analysis are elaborated in Section 5. Section 6 concludes this work and presents future directions.

2 Related Work

In this section, we briefly review the recent studies on computation offloading in VEC environment.

TABLE 1
Major Notations

Notation	Description
N	Number of tasks
v	Car speed
i	Task index $i \in N$
G, V, E	Graph, vertex set and edge set of DAG
DI_i, DO_i	Input / Output datasize of task i
C_i	Required computation resources of task i
k_i	Offloading decision of task i
f_i	Assigned CPU frequency of task i
f^l_{max}	Peak CPU frequency of the UE
f^e	CPU frequency of the edge server
t^l_i, t^o_i	Execution time of task i in local / edge computing
e^l_i, e^o_i	Energy consumption of task i in local / edge computing
t^u_i, t^c_i, t^c_i	Transmission / Execution / Receiving time of task i
t^p	Constant propagation delay
t^p_i	Task migration delay of task i
e^u_i	Energy consumption of task i in transmission phase
κ	Computing efficiency parameter
R^u_i, R^d_i	Uplink / Downlink rate of task i
W	Channel bandwidth
p_i	Transmission power of task i
p_d	Transmission power of the RSU
p_{max}	Peak transmission power of the UE
σ^2	Power of white noise
h_i, g_i	Channel gain of task i
G_A	Antenna gain
F_C	Carrier frequency
d_i	Distance between the UE and RSU
PL	Path loss exponent
c^o_i, c^m_i	Charge for edge computing service / migration of task i
u_r, u_m	Unit price of computation resource / task migration
T^s_i, T^e_i	Start / End time of task i
t_i, e_i, c_i	Total time / energy / charge of task i
U	System cost
$\beta_1, \beta_2, \beta_3$	Weighting parameters

Zhao et al. presented a collaborative computation offloading approach based on cloud assisted MEC by jointly optimizing computation offloading decision and computation resource allocation [1]. A distributed collaborative computation offloading and resource allocation optimization (CCORAO) scheme is proposed to improve the system utility and reduce computation time. Based on the consensus alternating direction method of multipliers (ADMM) [22], Zhou et al. [2] proposed a low-complexity distributed solution to energy-efficient workload offloading problem and conducted a real-world topology based simulation to validate the proposed solution In order to enhance the utilities of both the vehicles and the edge servers, Zhang et al. [23] adopted a Stackelberg game theoretic approach and an iterative distributed algorithm to design an optimal multi-level offloading scheme.

Most of the existing computation offloading and resource allocation problem in VEC environment are formulated as mixed-integer non-linear programming (MINLP) problems [18]. Since these problems are usually non-convex and NP-hard, the solving time with general solvers is too long that it is hard to meet the stringent delay requirements of time-sensitive applications in VEC. In order to efficiently solve these MINLP problems, deep reinforcement learning (DRL) based algorithms are getting more and more popular in recent years.

Luo et al. [24] formulated a unified framework with communication, computation, caching, and collaborative computing where vehicles can complete data communication via both vehicle-to-infrastructure (V2I) links and vehicle-to-vehicle (V2V) links. A data scheduling strategy based on deep Q-network (DQN) algorithm [25] is derived to reduce data processing cost. Zhang et al. [26] utilized DQN to design an optimal data transmission scheduling scheme in cognitive vehicular networks, which not only minimizes transmission costs but also fully utilizes various communication modes and resources. In [27], they proposed a DQN-based computation offloading scheme by jointly considering selection of target edge server and determination of data transmission mode. Simulation results on real traffic data shows the proposed offloading algorithm can significantly improve system utilities and offloading reliability. He et al. [28] put forward an integrated framework that can enable dynamic orchestration of networking, caching, and computing resources. Based on the combination of Double DQN [29] and Dueling DQN [30], a novel resource allocation strategy is proposed which performs well under different system parameters.

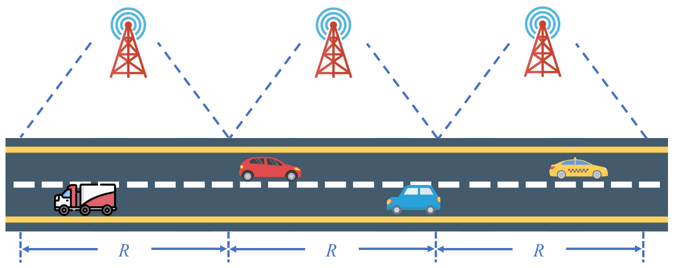


Fig. 1. An illustration of the vehicular network.

Nevertheless, the above works mainly focus on the process of a single application but have not considered the scenario involved with task dependency. Besides, the formulated problems in these papers are mainly single objective optimization which takes either the execution time or the energy consumption into account. In this paper, we propose a joint computation offloading and resource allocation scheme in the presence of task DAG and incorporates execution time, energy consumption and charge of edge computing service into the optimization goal.

3 System Model And Problem Formulation

This section presents the overall system model in VEC and formulates a joint computation offloading and resource allocation optimization problem. For the sake of convenience, the main notations used are summarized in Table 1.

3.1 System Model

As shown in Fig. 1, the vehicular network contains a series of RSUs and one vehicle running on the highway. The RSUs are evenly spaced along the highway and have the same coverage R, and the car runs with dynamic speed v. Edge servers with abundant computation resources are deployed in these RSUs.

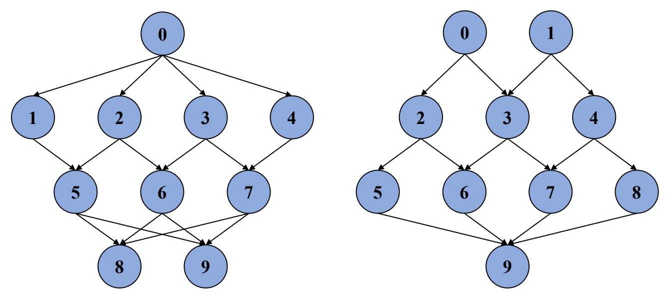


Fig. 2. Directed acyclic graphs.

A computation-intensive application in the in-vehicle user equipment (UE) can be partitioned into N tasks with the dynamic partitioning scheme [17]. As shown in Fig. 2, the inter-dependency between tasks is captured by a directed acyclic graph (DAG) $G = (V, E)$. Each node $i \in V$ represents a task and a directed edge $e(i, j) \in E$ indicates the precedence constraint between task i and j, which means the execution of task j cannot start until its precedent task i completes.

Each task i can be described by a three-tuple $\varphi_i = (DI_i, DO_i, C_i)$, where DI_i and DO_i represent the input and output data size of task i respectively, C_i is the required CPU cycles to complete task i. We denote $k_i \in \{0, 1\}$ as the offloading decision of task i, and specifically, $k_i = 0$ means that the UE decides to execute task i locally while $k_i = 1$ implies that the UE chooses to offload task i to the edge server. The wireless communication between the UE and OSUs is based on IEEE 802.11p VANET [31]. It should be noted that the UE can only access to RSU m when the vehicle runs within its coverage.

3.2 Local Computing

With the technique of dynamic voltage and frequency scaling (DVFS) [32], the UE can assign different computational capability for different tasks. We denote the CPU frequency for computing task i as f_i with $f_i \leq f^l_{max}$, where f^l_{max} is the peak CPU frequency of the UE. Then the execution time of task i for local computation is given by

$$t^l_i = \frac{C_i}{f_i}, \tag{3}$$

and the energy consumption for task i is given by

$$e^l_i = \kappa C_i f_i^2, \tag{4}$$

where κ is the effective switched capacity parameter depending on the chip architecture [3].

3.3 Edge Computing

For the edge computing, the UE will offload task i to the edge server, and then the edge server will execute the computation task and return the results to the UE. Therefore, the offloading process of task i includes three phases in sequences: the transmitting phase, the edge execution phase and the receiving phase.

1) Transmitting phase: According to Shannon's Theorem, the uplink rate for offloading task i is expressed as

$$R^u_i = W \log_2(1 + \frac{p_i h_i}{\sigma^2}), \tag{5}$$

where W is the channel bandwidth, p_i is the transmission power when offloading task i to the RSU, h_i is the channel gain due to the path loss and the shadowing attenuation, and σ^2 is the power of white Gaussian noise. We assume $p_i \leq p_{max}$, where p_{max} is the peak transmission power of the UE. The average channel gain h_i follows the free-space path loss model [5]

$$h_i = G_A(\frac{3 \cdot 10^8}{4\pi F_c d_i})^{PL}, \tag{6}$$

where G_A denotes the antenna gain, F_c denotes the carrier frequency, d_i denotes the distance between the UE and the RSU, and PL denotes the path loss exponent.

Accordingly, the transmission delay and the energy consumption during the transmission phase of task i can be expressed as

$$t^u_i = \frac{DI_i}{R^u_i}, \tag{7}$$

and

$$e^u_i = p_i t^u_i. \tag{8}$$

2) Edge execution phase: We assume the edge server possesses abundant computation resources, and its CPU frequency f^e is fixed and does not change during the execution process. In practical cases, it is assumed that the edge server has superior computation capability over the UE, and therefore $f^e > f^l_{max}$. Then the execution time of task i in the edge server can be computed as

$$t^e_i = \frac{CI_i}{f^e}. \tag{9}$$

3) Receiving phase: As for the downlink transmission, we denote the transmission power of RSU as p_d, which stays fixed across all RSUs. Likewise, the downlink rate for returning the results of task i to the vehicle is given by

$$R^d_i = W \log_2(1 + \frac{p_d h_i}{\sigma^2}). \tag{10}$$

As a result, the downlink transmission time during the receiving phase can be computed by

$$t^d_i = \frac{DO_i}{R^d_i}. \tag{11}$$

In order to avoid the task migration between adjacent RSUs, the execution of a task should be completed before the vehicle shifts from the current connected RSU to an adjacent one. We can express this constraint by

$$d_i + \int_{T^s_i}^{T^e_i} v\, dv < R, \tag{12}$$

where d_i denotes the distance between the UE and the starting point of the connected RSU when task i is offloaded to the edge server, T^s_i and T^e_i are the start time and the end time of task i.

If the task migration occurs, it will lead to a large propagation delay between neighbouring RSUs, which can be expressed by

$$t_i^p = t_p \cdot \mathbb{I}(d_i + \int_{T_i^s}^{T_i^e} v\, dv > R), \quad (13)$$

where t_p is the constant propagation delay and $\mathbb{I}(\cdot)$ is the indicator function.

By adding the results of all these three phases together, we can express the total execution time of task i during the offloading process as

$$t_i^o = t_i^u + t_i^e + t_i^d + t_i^p. \quad (14)$$

We assume that all RSUs have enough energy to execute offloaded tasks and do not take into account the energy consumed during edge execution phase and downlink transmission phase. Accordingly, the energy consumption of the vehicle in edge computing results only from the uplink transmission, which can be expressed by

$$e_i^o = e_i^u. \quad (15)$$

When offloading a task to the edge server, the UE pays for using the computation resources, and a higher resource demand leads to a higher payment. We consider a linear pricing scheme and the cost c_i^o of executing task i in the edge server can be computed as

$$c_i^o = C_i u_r, \quad (16)$$

where u_r is the unit price charged for unit computation resource by the edge server. On the contrary, the UE does not need to pay for the service when executing the task locally with its own computation resource. Besides, a relatively high cost will be charged for the task migration service, which can be computed by

$$c_i^m = C_i u_m \cdot \mathbb{I}(d_i + \int_{T_i^s}^{T_i^e} v\, dv > R), \quad (17)$$

where u_m is the per-unit price of task migration.

3.4 Task Dependency Model

As shown in Fig 2, the precedence constraints among tasks regulate the execution order of tasks. Based on the offloading decision of the UE, the relation between the start time T_i^s and the end time T_i^e of task i can be derived:

$$T_i^e = T_i^s + (1-k_i)t_i^l + k_i t_i^o. \quad (18)$$

Task i can be executed when all its immediate predecessors have completed. This constraint can be expressed by

$$T_i^s \geq \max_{j \in pred(i)} T_j^e, \quad (19)$$

where $pred(i)$ denotes the set of immediate predecessors of task i.

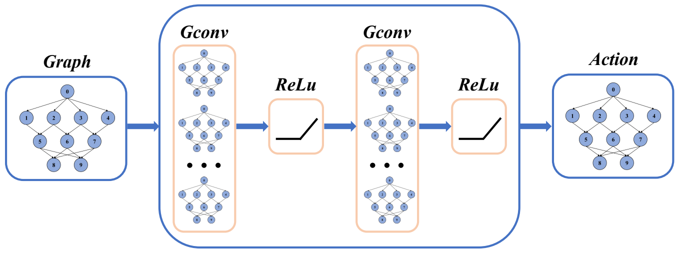


Fig. 3. Graph Attention Network.

3.5 Problem Formulation

The main objective of this paper is to minimize the time-energy-service cost (TESC), which is defined as the weighted sum of the execution time and the energy consumption of the entire application and the charge for edge computing service. Depending on the offloading decision of the UE, the execution time and the energy consumption of task i are denoted as

$$t_i = (1-k_i)t_i^l + k_i t_i^o, \quad (20)$$

and

$$e_i = (1-k_i)e_i^l + k_i e_i^o. \quad (21)$$

The charge for edge computing service is only related to offloaded tasks and is denoted as

$$c_i = k_i(c_i^o + c_i^m). \quad (22)$$

Therefore, the TESC of all the tasks can be expressed as

$$U = \sum_{i=1}^{N} \beta_1 t_i + \beta_2 e_i + \beta_3 c_i, \quad (23)$$

where $0 \leq \beta_1, \beta_2, \beta_3 \leq 1$ denote the weighting parameters, and without loss of generality $\beta_1 + \beta_2 + \beta_3 = 1$.

We aim to minimize the TESC of all the tasks by optimizing the task offloading policy $\mathcal{K} \triangleq \{k_i\}$, local CPU frequency control policy $\mathcal{F} \triangleq \{f_i\}$ and transmission power allocation policy $\mathcal{P} \triangleq \{p_i\}$. The optimization problem can be formulated as a constrained minimization problem:

$$\begin{aligned} \min_{\mathcal{Y},\mathcal{K},\mathcal{P},\mathcal{F}} \quad & U \\ \text{s.t.} \quad & C1: y_i \in \{1,...,N\}, \\ & C2: k_i \in \{0,1\}, \\ & C3: 0 \leq f_i \leq f_{max}^l, \\ & C4: 0 \leq p_i \leq p_{max}, \\ & C5: \max_{j \in pred(i)} T_j^e \leq T_i^s. \end{aligned} \quad (24)$$

Constraint C1 indicates the serial number of each task. Constraint C2 guarantees that each task is executed locally or offloaded to the edge server. Constraint C3 and C4 specify the computation resource limit and transmission power limit respectively. Constraint C5 ensures that task i cannot start execution until all its immediate predecessors have been executed. It can be seen that problem (24) is a mixed-integer programming which is in general non-convex and NP-hard. It involves the integer variable $\mathcal{Y}$ and $\mathcal{K}$ and continuous variable $\mathcal{P}$ and $\mathcal{F}$.

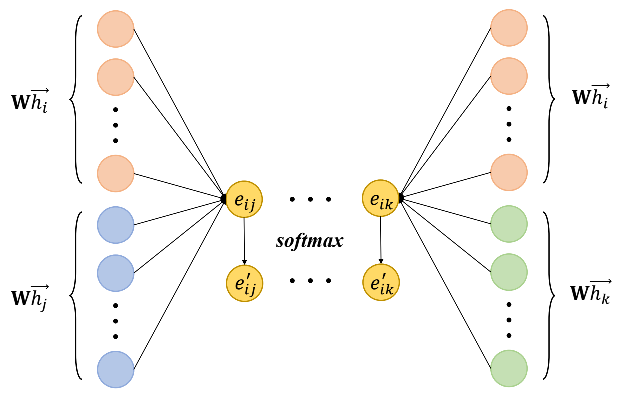


Fig. 4. Attention mechanism.

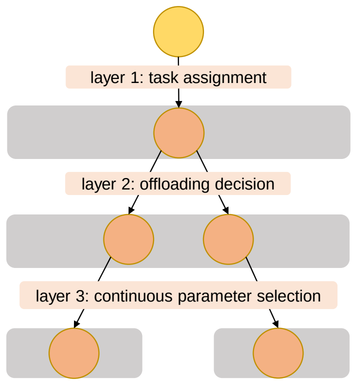

layer 1: task assignment
layer 2: offloading decision
layer 3: continuous parameter selection

Fig. 5. Hierarchical action space.

4 Design

In this section, we propose a Deep Hierarchical Vehicular Offloading (DHVO) scheme to solve the optimization problem. A graph neural network with attention mechanism is introduced to extract high-level features from the DAG. To deal with the hierarchical action space, we put forward a novel DRL algorithm. The selection of both discrete and continuous actions can be made with a single parametrized network.

4.1 Graph Attention Network

Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have been successfully applied to tackle machine learning tasks, mainly attributed to their great power of extracting high-level features from Euclidean data (e.g. images, text and videos). However, the complex relationships and interdependency between objects in graph data, such as the DAG in Fig. 2, have imposed significant challenges on existing machine learning algorithms [33].

With the introduction of attention mechanism, Graph Attention Networks (GAT) has shown its computation efficiency and superior performance in dealing with graph data [34]. Compared with spectral-based graph neural networks (GNN) such as Graph Convolution Network (GCN) [35], GAT can operate on directed graph without depending on upfront access to the entire graph structure. Accordingly, we utilize GAT to explore the hidden features of each task and the interdependency among them.

As shown in Fig. 3, our proposed GAT consists of two hidden layers with ReLU as the activation function. The input to each layer is a set of node features, $\mathbf{h} = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}, \vec{h}_i \in \mathbb{R}^F$, where F is the number of features in each node. Specially, the features of each node in the input layer are represented by $\vec{h}_i = \{DI_i, DO_i, C_i, E_i\}$, where the first three features corresponds to the three-tuple φ_i and E_i indicates whether task i has been executed.

In each hidden layer, the input features are transformed into high-level features through a shared linear transformation, which is parametrized by a weight matrix $\mathbf{W} \in \mathbb{R}^{F \times F'}$, where F' denotes the number of new generated features. Besides, we perform a shared attention mechanism $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$ to assign different importance for each node pair. The attention coefficients e_{ij}, which indicates the importance of node j's features to node i, can be computed as

$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j). \tag{25}$$

As shown in Fig. 4, the shared attention mechanism a is represented by a single-layer fully-connected network. At the output layer, we calculate the normalized attention coefficients e'_{ij} by using the softmax function across the neighbors of i (including i itself):

$$e'_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i \cup i} \exp(e_{ik})}, \tag{26}$$

where $\mathcal{N}_i$ denotes the set of neighbors of node i. For better feature extraction result, we employ the multi-head attention mechanism by independently executing attention mechanism $a(\cdot, \cdot)$ several times and then concatenating their results. The output node features $\vec{h}'_i$ after the multi-head attention can be computed by

$$\vec{h}'_i = \Big\|_{k=1}^{K} ReLU\Big(\sum_{j \in \mathcal{N}_i \cup i} ({e'_{ij}}^{k} \mathbf{W}^k \vec{h}_j\Big), \tag{27}$$

where $||$ is the concatenation operation, K is the number of attention heads, and ${e'_{ij}}^{k}$ is the normalized attention coefficients of the k-th attention head and $\mathbf{W}^k$ is the corresponding weight matrix. With the operations stated above, the dimension of node features is scaled from F to KF'.

4.2 Hierarchical Action Space

As described in Section 3.5, the action space of the UE exhibits a hierarchical structure. As shown in Fig. 5, the hierarchical action space can be divided into three parts:

(1) layer 1: task assignment. The UE needs to decide which task to be assigned without breaking the constraints of task dependency (Section 3.4).
(2) layer 2: offloading decision. The UE needs to decide the assigned task to be executed locally or offloaded to the edge server.
(3) layer 3: continuous parameter selection. If the UE decides to execute the task locally, it should consider assigning how much CPU frequency to process the task. Likewise, if the UE offloads the task to the edge server, the appropriate transmission power should be allocated.

Most of existing DRL algorithms require the action space to be either discrete (e.g. DQN [25] and A3C [36]) or continuous (e.g. DPG [37] and DDPG [38]). There are two straightforward ideas to apply these traditional DRL algorithms on discrete-continuous hybrid action space. The first is to approximate the continuous part by a finite discrete set. However, this method narrows the continuous action space and the fine-grained approximation requires complicated design. The second is to relax the discrete part to a continuous set and apply DDPG algorithm subsequently. Compared with the original action space, this approach significantly increases its complexity.

Inspired by normalized advantage functions (NAF) [39], we design a novel DRL algorithm, named parameterized NAF (PNAF), which directly works on hierarchical action space containing both discrete and continuous actions directly without approximation or relaxation.

For a simple illustration, the hierarchical action space in our paper is expressed as

$$A = \{a_d, a_c\} = \{(y_i, k_i = 0, f_i) \cup (y_i, k_i = 1, p_i)\}, \quad (28)$$

where a_d represents the discrete action set and a_c represents the continuous action set. We denote the action selected at time t by $a_t = (y_t, k_t, f_t, p_t)$ and the corresponding action-value function by $Q(s_t, a_t)$, where $s_t \in \mathcal{S}$, $y_t \in \mathcal{Y}$, $k_t \in \mathcal{K}$, $f_t \in \mathcal{F}$ and $p_t \in \mathcal{P}$. The Bellman equation in the hierarchical action space A is given by

$$Q(s_t, a_t) = \mathbb{E}[r_t + \gamma \max_{y_{t+1}} \max_{k_{t+1}} \sup_{f_{t+1}, p_{t+1}} Q(s_{t+1}, a_{t+1})], \quad (29)$$

where r_t is the immediate reward and γ is the discount factor. However, in hierarchical action space, the action-value function $Q(s,a)$ suffers from the problem of over-parameterization, which is expressed by

$$Q(s,a) = \begin{cases} Q(s,y,k,f) & \text{if } k = 0 \\ Q(s,y,k,p) & \text{if } k = 1 \end{cases} \quad (30)$$

In other words, the action-value function $Q(s,a)$ is influenced by irrelevant continuous parameters.

To overcome the over-parameterization problem, we introduce a novel neural network shown in Fig. 6, which separately outputs a state value function term $V(s, a_d)$ and an advantage term $A(s, a_d, a_c)$. The former term $V(s, a_d)$ represents the expected cumulative reward when a discrete action a_d is decided, and the latter term $A(s, a_d, a_c)$ represents the difference between the expected cumulative reward when a deterministic continuous a_c is taken and $V(s, a_d)$. Therefore, the final Q-function $Q(s, a_d, a_c)$ can be computed as

$$Q(s, a_d, a_c) = Q(s, a_d) + A(s, a_d, a_c). \quad (31)$$

Accordingly, the Bellman equation in Equation (29) becomes

$$\begin{aligned} Q(s_t, a_{d_t}, a_{c_t}) = \mathbb{E}\{r_t + \gamma \max_{a_{d_{t+1}}} \sup_{a_{c_{t+1}}} [V(s_{t+1}, a_{d_{t+1}}) + \\ A(s_{t+1}, a_{d_{t+1}}, a_{c_{t+1}})]\}. \end{aligned} \quad (32)$$

Since the state value function $V(s_{t+1}, a_{d_{t+1}}))$ is irrelevant with the continuous action $a_{c_{t+1}}$, it can be taken out of

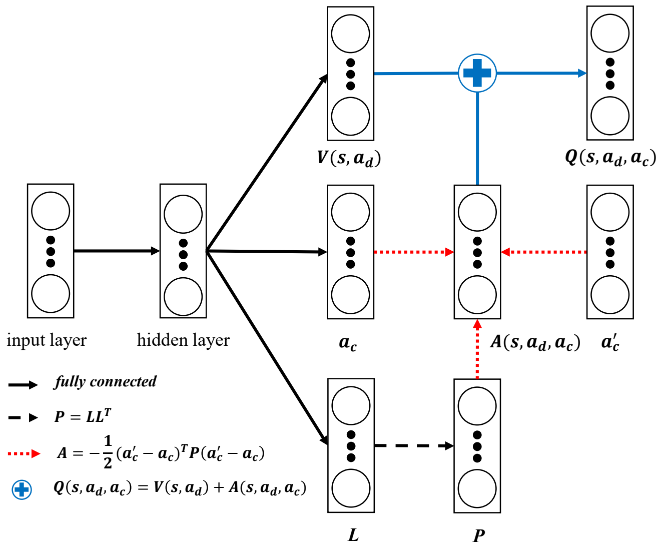


Fig. 6. The architecture of the parameterized network.

the operation of taking supremum over continuous action space, and therefore Equation (32) can be rewritten as

$$\begin{aligned} Q(s_t, a_{d_t}, a_{c_t}) = \mathbb{E}\{r_t + \gamma \max_{a_{d_{t+1}}} [V(s_{t+1}, a_{d_{t+1}}) + \\ \sup_{a_{c_{t+1}}} A(s_{t+1}, a_{d_{t+1}}, a_{c_{t+1}})]\}. \end{aligned} \quad (33)$$

To guarantee the optimality of the continuous action that maximizes the advantage function, we restrict the advantage function $A(s, a_d, a_c)$ as a quadratic function of the estimated continuous action $a_c(s; \theta)$:

$$A(s, a_d, a_c; \theta) = -\frac{1}{2}(a_c' - a_c(s;\theta))^T P(s;\theta)(a_c' - a_c(s;\theta)). \quad (34)$$

The positive-definite square matrix $P(s;\theta)$ can be computed by Cholesky decomposition [43]:

$$P(s;\theta) = L(s;\theta) L(s;\theta)^T, \quad (35)$$

where $L(s;\theta)$ is a lower-triangular matrix with positive diagonal terms. The state-dependent entries of $L(s;\theta)$ come from the output layer of the neural network.

In order to encourage the agent to explore the discrete action space sufficiently and discover a better policy, the selection of discrete actions is based on ϵ-greedy policy. The agent will choose a random action from its action space with probability ϵ or choose the action with the highest V-value with probability $1 - \epsilon$. The assignment of a_d is given by

$$a_d = \begin{cases} \text{a random action} & \text{with probability } \epsilon \\ \arg\max_{a_d} V(s, a_d; \theta) & \text{with probability } 1 - \epsilon \end{cases} \quad (36)$$

As for the continuous action, the Ornstein-Uhlenbech (OU) process is introduced to generate a temporally correlated noise sequence. The appendix specifies how the reinforcement learning with discrete-continuous hybrid action space operates.

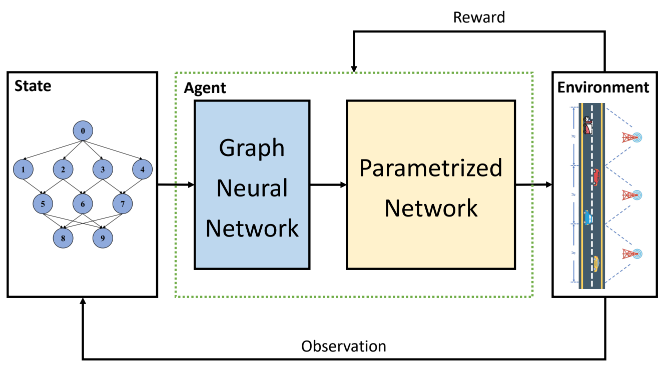


Fig. 7. The framework of DHVO.

4.3 Deep Reinforcement Learning

By combining the technique of Deep Learning (DL) [19] and Reinforcement Learning (RL) [20], deep reinforcement learning (DRL) aims at constructing an agent that can acquire knowledge by exploring the interaction with the environment without the need of external supervision in an end-to-end manner [21]. DRL can be modeled as Markov Decision Process (MDP) which is represented by a four-tuple $M = \{S, A, P, R\}$, where S is the state space, A is the action space, P is the transition probability matrix and R is the reward function. At each time step t, the agent observes the current environment state s_t and chooses an action a_t following a stochastic policy $\pi : S \to A$, which maps state space S to a probability distribution over action space A. Then the environment state changes to s_{t+1} with probability $p(s_{t+1}|s_t, a_t)$, and the agent receives an immediate reward $r_t(s_t, a_t, s_{t+1})$, which indicates the effect of action a_t on state s_t. The goal of the agent is to maximize the cumulative discounted reward $R_t = \sum_{t=0}^{T} \gamma^t r_t$, where γ is the discounted factor and T is the end episode. We define the state-value function under policy π as $V^\pi(s) = \mathbb{E}[\sum_{t=0}^{T} \gamma^t r_t|\pi]$, which denotes the expected total discounted reward at state s. Through episodic interactions with the environment, the agent tries to find the optimal policy π^* that achieves the optimal action value function V^*.

The vehicular offloading problem can be modelled as an MDP and the definition of each element in reinforcement learning are introduced as follows:

State space. The state space is composed of two parts: the task state and the environment state. The task state is represented by the DAG of tasks and its node feature vector $\mathbf{h}_t = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}$ with $\vec{h}_i = \{DI_i, DO_i, C_i, E_i\}$. The environment state includes the distance between the UE and the starting point of the connected RSU d_t, the number of remaining task n_t, and the recorded speed of last five seconds $\vec{v}_t$.

Action space. As described in Section 4.2, the action space has a hierarchical structure and is composed of task assignment y_t, offloading decision k_t, local CPU frequency f_t and transmission power p_t.

Reward. Our objective is to minimize the TESC of the whole application, we set the reward of each action as the immediate TESC achieved by the executed task u_t.

Algorithm 1 Deep Hierarchical Vehicular Offloading (DHVO)

```
Input: DAG of tasks and node feature vector $\mathbf{h}_t = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}$ with $\vec{h}_i = \{DI_i, DO_i, C_i, E_i\}$, the distance between the UE and the starting point of the connected RSU $d_t$, the number of remaining task $n_t$, the recorded speed of last five seconds $\vec{v}_t$
Output: task assignment $y_t$, offloading decision $k_t$, local CPU frequency $f_t$ and transmission power $p_t$

// initialization
Randomly initialize graph neural network $Q_g(s, a; \theta_g)$
Randomly initialize parametrized network $Q_p(s, a; \theta_p)$
Initialize target network $Q'_p(s, a; \theta'_p)$ with weight $\theta'_p \leftarrow \theta_p$
Initialize replay buffer $D \leftarrow \varnothing$

// training process
for episode = 1, $M$ do
  for t = 1, $T$ do
    Observe current state $s_t$
    Select hierarchical action $a_t$
    Execute $(y_t, k_t = 0, f_t)$ or $(y_t, k_t = 1, p_t)$
    Observe next state $s_{t+1}$ and receive reward $r_t$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $D$
    Sample a random batch $(s_i, a_i, r_i, s_{i+1})$ from $D$
    Set target value $z_i = r_i + \gamma \max_{a_d} V'(s_{i+1}, a_d; \theta_p)$
    Set loss function $L_t^g = (z_i - Q(s_i, a_i; \theta_g))^2$ and $L_t^p = (z_i - Q(s_i, a_i; \theta_p))^2$
    Update network parameters: $\theta_g \leftarrow \theta_g + \alpha_g \nabla_{\theta_g} L_t^g$, $\theta_p \leftarrow \theta_p + \alpha_p \nabla_{\theta_g} L_t^p$ and $\theta'_p \leftarrow \tau\theta_p + (1-\tau)\theta'_p$
  end for
end for
```

4.4 Deep Hierarchical Vehicular Offloading

By introducing the technique of GAT and PNAF into DRL, we propose an adaptive vehicular offloading algorithm called Deep Hierarchical Vehicular Offloading (DHVO). The framework of DHVO is shown in Fig. 7. The decision-making of a UE consists of a graph neural network and a parametrized network, which establishes the mapping between a state and a hierarchical action.

The pseudo-code of DHVO is shown in Algorithm 1. We first initialize the graph neural network $Q_g(s, a; \theta_g)$, the parametrized network $Q_p(s, a; \theta_p)$ and its target network $Q'_p(s, a; \theta'_p)$ with $\theta'_p = \theta_p$. The replay buffer D is set to $\varnothing$ initially.

At each decision epoch t when the UE finishes the last task $TASK_{t-1}$ and starts the execution of a new task $TASK_t$, the UE will observe the task state $\mathbf{h}_t = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}$ with $\vec{h}_i = \{DI_i, DO_i, C_i, E_i\}$ and the environment state $s_t^e = \{d_t, n_t, \vec{v}_t\}$. Then, these features are sent to GNN for the extraction of high-level features, and finally the Q-value of each task can be estimated. The task with the highest Q-value is set to $TASK_t$, and thus the action of task assignment y_t is determined. Afterwords, the features of $TASK_t$ and the environment state are passed to the parametrized network Q_p as the input, which then outputs the hierarchical action $a_t \in \{(y_t, k_t = 0, f_t) \cup (y_t, k_t = 1, p_t)\}$ and the corresponding Q-value $Q_p(s_t, a_t; \theta_p)$. After

TABLE 2
Simulation Parameters

Description	Parameter	Value
Coverage of each RSU	R	200 m
Length of time-slot	Δt	1 s
Number of tasks	N	10
Input datasize of each task	DI	[2.5, 3.5] MByte
Output datasize of each task	DO	[2.5, 3.5] MByte
Required computation resources	C	[800, 1200] Mcycles
Bandwidth	W	2 MHz
Peak local computing capability	f^l_{max}	10^8 cycles/s
Edge server computing capability	f^e	10^9 cycles/s
Peak transmission power	p^l_{max}	200 mW
Computing efficiency parameter	κ	10^{-25}
Atenna gain	G_A	4.11
Distance from the RSU	d	100 m
Constant propagation delay	t_p	5 s
Carrier frequency	F_c	915 MHz
Path loss exponent	PL	3
Noise power	δ^2	10^{-12} W
Computation resource price	u_r	0.1 \$/Mcycles
Task migration price	u_m	2 \$/Mcycles
System cost weighting parameters	$\beta_1, \beta_2, \beta_3$	0.33, 0.33, 0.33

TABLE 3
Parameters of neural networks

Parameter	Value
Attention heads of GAT	(8, 1)
Number of features of each attention head	(8, 1)
Network hidden layers of parametrized network	(128,)
Reward discounted factor	0.99
Learning rate of graph neural network	0.001
Learning rate of parametrized network	0.01
Soft-update coefficient of the target network	0.1
Activation function	ReLU
Network optimizer	Adam
Max training episodes	2000

the selection of hierarchical action a_t, task $TASK_t$ will be executed locally with CPU frequency f_t if $k_t = 0$, or it will be offloaded to the edge server with transmission power p_t if $k_t = 1$. Afterwards, the environment state turns into s_{t+1} and the UE receives the immediate reward r_t. The UE will record the environment transition (s_t, a_t, s_{t+1}, r_t) into the replay buffer D.

At the beginning of parameter update, a training batch (s_i, a_i, s_{i+1}, r_i) is sampled randomly from D. As for the graph neural network Q_g and the parametrized network Q_p, the target value z_i is set to the sum of the immediate reward r_i and the highest V-value of target network $Q_{p'}$:

$$z_i = r_i + \gamma \max_{a_d} V'(s_{i+1}, a_d; \theta'_p), \tag{37}$$

where γ is the discount factor. In order to adjust the estimated Q-value toward the target value, we update the parameters of Q_g and Q_p by minimizing the mean squared error between the target value z_i and its current output. The loss function of the graph neural network L^g_t and the parametrized network L^p_t are separately given by

$$L^g_t = (z_i - Q(s_i, a_i; \theta_g))^2, \tag{38}$$

$$L^p_t = (z_i - Q(s_i, a_i; \theta_p))^2. \tag{39}$$

Then we perform gradient descent to update the parameters of three neural networks as below:

$$\theta_g \leftarrow \theta_g + \alpha_g \nabla_{\theta_g} L^g_t, \tag{40}$$

$$\theta_p \leftarrow \theta_p + \alpha_p \nabla_{\theta_g} L^p_t, \tag{41}$$

$$\theta'_p \leftarrow \tau\theta_p + (1-\tau)\theta'_p, \tag{42}$$

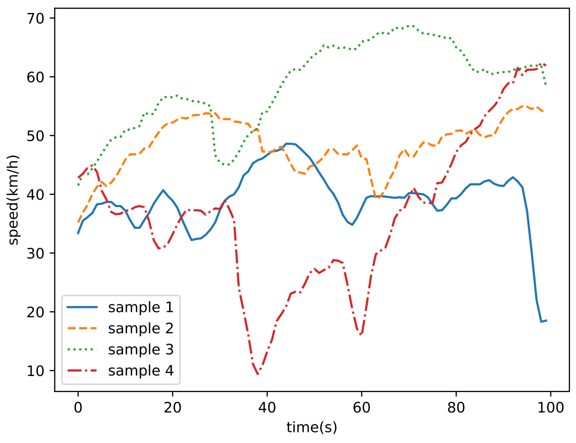


Fig. 8. Four samples of speed trajectories.

where α_g and α_p are the learning rates of the graph neural network Q_g and the parametrized network Q_p, and τ is the soft-update coefficient of the target network. In this way, the network parameters are updated with episodic training until convergence.

5 Evaluation

5.1 Simulation Setup

Simulation environment. We consider one vehicle running on the highway along which several APs are evenly deployed with coverage $R = 200$ m. The speed of the vehicle is collected from a real-world dataset provided by CPIPC (China Postgraduate Innovation & Practice Competition) [40]. The speed data collection lasts for seven days and is measured in seconds. We tailor this dataset to four trajectories, each of which contains 100s speed data. As shown in Fig. 8, the speed of the vehicle varies with time dynamically and therefore is hard to predict. The input datasize DI_i, output datasize DO_i and required computation resources C_i follows uniform distribution in the range of [2.5, 3.5]

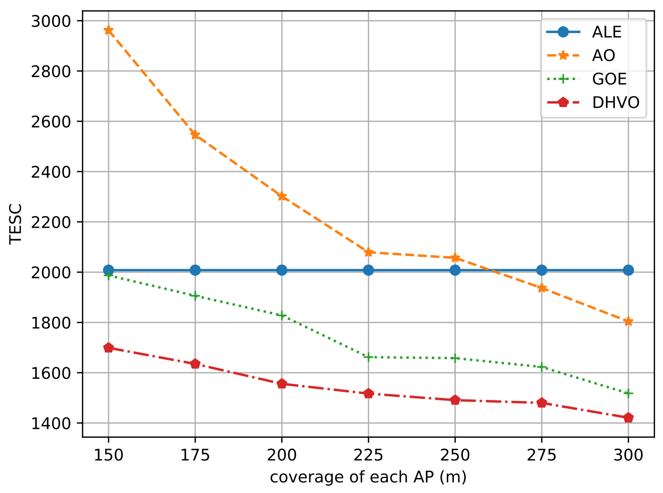


Fig. 9. Total cost with the coverage of each RSU.

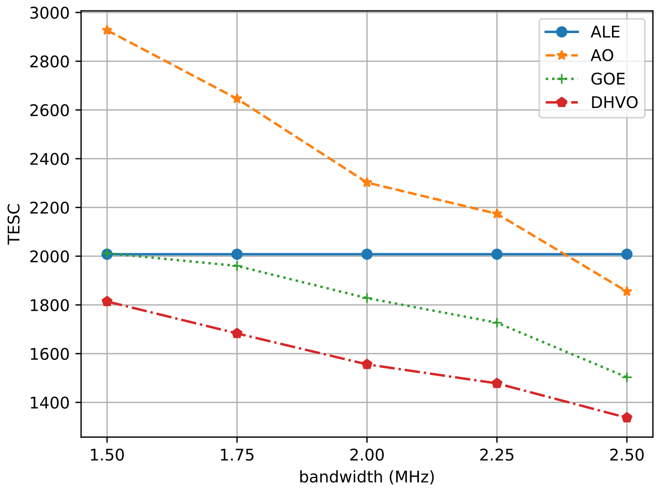


Fig. 10. Total cost with bandwidth.

MBytes, [2.5, 3.5] MBytes and [800, 1200] Mcycles, respectively. The complete simulation parameters are shown in Table 2 and are set as default unless otherwise specified.

Neural network parameters. We apply a two-layer GAT model, where the fist layer consists of $K = 8$ attention heads computing $F' = 8$ features (for a total of 64 features), and the second layer uses a single attention head which computes one feature (Q-value). We eliminate dropout and L_2 regularization for stable training effect. The parametrized network is fully connected with one hidden layer containing 128 neurons. We set the learning rate of graph neural network and parametrized network to 0.001 and 0.01 separately, and set the soft-update coefficient of the target network to 0.1. The reward discounted factor γ is set to 0.99. We use Rectified Linear Unit (ReLU) as the activation function and Adam as the optimization algorithm. The detailed parameters setting of the neural network is listed in Table 3.

Simulation platform. We implemented DHVO algorithm based on Pytorch library with Python 3.5. The simulation platform is a Ubuntu Server 16.04 with 32GB RAM and an Intel (R) Core (TM) i7-6800K CPU, which has 6 cores and 12 threads.

Compared algorithms. We compare our proposed algorithm DHVO with three representative benchmarks:

- All locally executed (ALE): All of the tasks are executed locally with peak local computing capability;
- All offloaded (AO): All of the tasks are offloaded to the edge server with peak transmission power;
- Greedy on edge (GOE): The UE will offload the task to the edge server with peak transmission power if it predicts that the task can be returned within the coverage of the current RSU based on a constant speed (30 km/h). Otherwise, the UE will execute the task locally with peak local computing capability.

5.2 Performance Evaluation

In this part, we evaluate the performance of four algorithms under different environment parameters and explore the relationship between the system cost and these parameters. The environment parameters are set default as listed in Table 2 unless otherwise specified. For a fair comparison, we conduct 50 individual test and take the average value as the final result.

We varies the coverage of each RSU to find its relationship with the TESC of each algorithm. As shown in Fig 9, the system cost of each algorithm except for ALE decreases with the rising of RSU's coverage. Since ALE executes all the tasks locally and does not involve the task offloading process, its system cost remains constant regardless of the change of RSU's coverage. As for the other three algorithms, when the coverage of each RSU increases, the frequency of the task migration will decrease, which causes the cost reduction of task migration service. Compared with AO and GOE, the slope of DHVO is much smaller and therefore is less sensitive to the change of RSU's coverage.

Fig. 10 indicates the system cost versus the bandwidth. Comparing ALE and AO, we can see that when the bandwidth is less than 2.25 MHz, the performance of ALE is much better than AO. Since the uplink/downlink rate and the bandwidth are in direct proportion relationship (Equation (5)(10)), the transmission time during the transmitting and receiving phase with low bandwidth is much longer than the local execution time. In addition, too long time spent in edge computing will increase the risk of the task migration. However, when the bandwidth keeps rising, edge computing shows its advantage of lower task completion time and therefore is more recommended than local execution. It can be clearly seen that DHVO is more robust to the change of the bandwidth than AO and GOE, which is attributed to its ability to forecast the occurrence of task migration.

Fig. 11 and Fig. 12 illustrate the relationship of the required computation resource and the input/output datasize

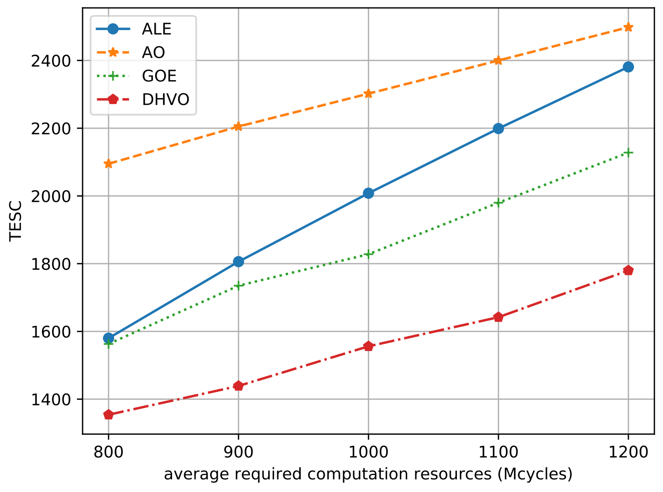


Fig. 11. Total cost with required computation resources.

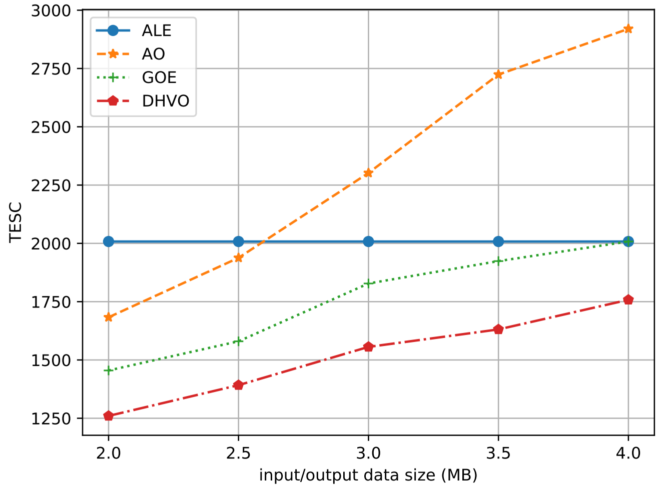


Fig. 12. Total cost with input / output data size.

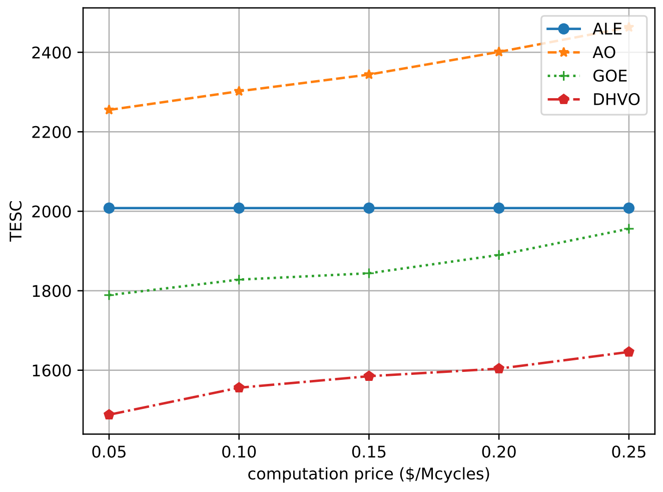


Fig. 13. Total cost with computation resource price.

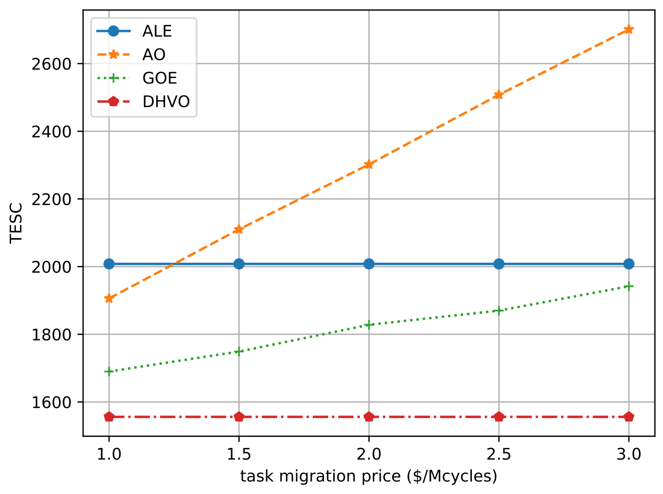


Fig. 14. Total cost with task migration price.

with the system cost. We can find that the required computation resource of each task has more impact on local computing than edge computing, while the input/output datasize only affects edge computing. This is because the execution time and energy consumption during local execution are in direct proportion to the required computation resource (Equation (3)(4)), while the delay and energy during edge computing depends more on the data size of each task. Although edge servers can provide powerful computing capability, the risk of task migration will cause extra service cost and propagation delay. When the datasize is less than 2.5 MB, short transmission time during the transmitting and receiving phase can be achieved. Moreover, compared with GOE which prefers edge computing, DHVO can adjust its offloading policy based on tasks' features and select proper offloading decision for each task.

Fig. 13 and Fig. 14 illustrate the relationship between the system cost and the computation resource price and the task migration price respectively. Since all the tasks of ALE are executed locally and thus ALE needs not to pay for the edge computing service, the total cost of ALE remains constant in both cases. Comparing these two pictures, we can see that the task migration price has more impact on the performance of AO and GOE than the computation resource prices. The main reason is that the value of the former price is much higher than the latter price, and the occurrence of task migration will lead to extra payment. Although edge computing can achieve lower execution time and energy consumption, the high cost brought by frequent task migration in AO and GOE makes edge computing a poor choice. On account of much lower risk of task migration, DHVO is not affected by the change of the task migration price and performs quite better than AO and GOE.

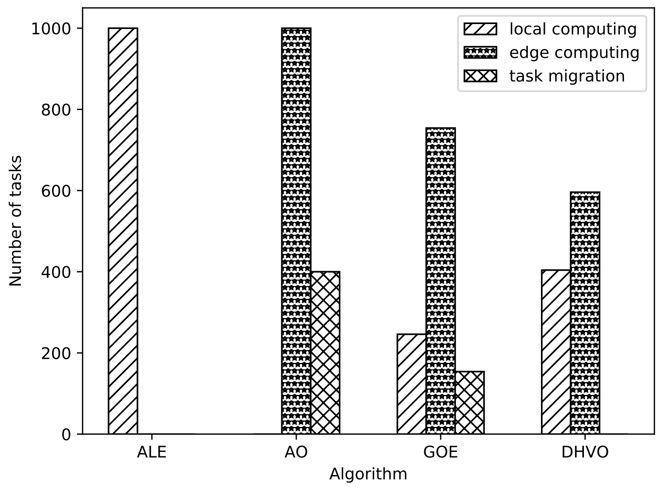


Fig. 15. Action of each algorithm.

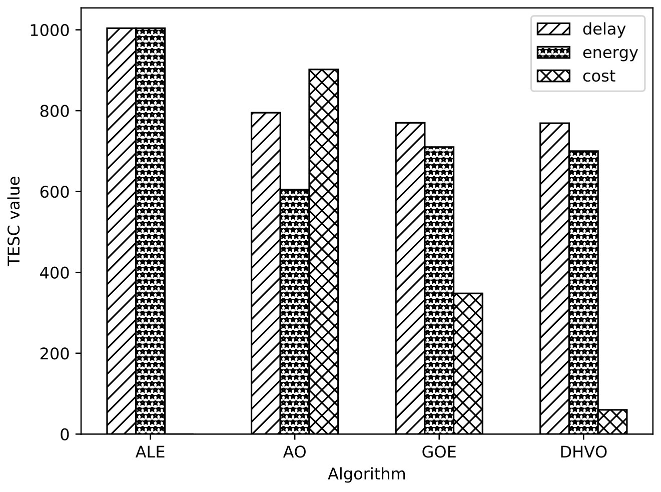


Fig. 16. The cost value of each algorithm.

5.3 Performance Analysis

In this part, we make a detailed analysis on the reasons why the performance of DHVO is much better than the compared algorithms from the perspective of the offloading policy and the TESC distribution of each algorithm.

Fig. 15 shows the number of each action taken by each algorithm when dealing with 100 application, each of which contains 10 tasks. We can find that DHVO makes a balance between local execution and edge computing, while GOE is greedy on offloading tasks to the edge server. However, since the speed of the vehicle varies dynamically, the constant speed used by GOE is wildly inaccurate compared with the real speed value, and accordingly GOE suffers a high penalty caused by task migration between consecutive RSUs. On the contrary, DHVO utilizes the speed data of last five seconds to make an accurate estimation on future speed, and thus avoids the occurrence of task migration. More precisely, if DHVO finds that the current task cannot be returned within the coverage of the current RSU when the vehicle runs at the predicted speed, it will decide to execute the task locally, and otherwise it will offload the task to the edge server to meet the low execution delay brought by edge computing.

Fig. 16 depicts the cost distribution of each algorithm. We can see that the energy consumption of ALE is the highest among four algorithms, which explains the reasons why task offloading is necessary for energy conservation. As for AO, though edge computing brings lower task completion delay and energy consumption, it suffers from high cost of edge computation resource and task migration. In spite of lower probability of the task migration, GOE also suffers from high cost of task migration as a result of wrong speed estimation. As for DHVO, it learns to select proper offloading decision for each task by considering both the individual features of each task and the dynamic environment, and learns to select proper local computing capability and transmission power to make a balance among delay, energy and service cost.

6 Conclusion

In this paper, we study the joint computation offloading and resource allocation problem in VEC environment. The problem is formulated as a MINLP optimization by jointly optimizing the task assignment, task offloading decision, local computation resource allocation and transmission power control. The system cost is defined as time-energy-service cost (TESC), which is the weighted sum of execution time, energy consumption and charge of service. Based on DRL, we propose deep hierarchical vehicular offloading (DHVO) scheme to solve this problem in an end-to-end manner. With the introduction of graph neural networks, the hidden features of each sub-task and the interdependency between them. Moreover, we design a novel neural network architecture to deal with the hierarchical action space which contains both discrete and continuous actions. We conduct simulations based on a real-world car speed dataset, and the evaluation results show that our proposed algorithm can significantly reduce the system cost under various environment parameters.

For the future work, we are going to consider the scenario where multiply UEs compete for communication resources in radio spectrum and computation resources in edge servers. In this case, each UE would make a more careful offloading decision in the presence of other competitors.

References

[1] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation Offloading and Resource Allocation For Cloud Assisted Mobile Edge Computing in Vehicular Networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 7944–7956, Aug. 2019.

[2] Z. Zhou, J. Feng, Z. Chang, and X. Shen, "Energy-efficient edge computing service provisioning for vehicular networks: A consensus ADMM approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 5087–5099, 2019.

[3] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, Apr. 2016, pp. 1–9.

[4] J. Yan, S. Bi, Y.-J. A. Zhang, and M. Tao, "Optimal Task Offloading and Resource Allocation in Mobile-Edge Computing with Inter-user Task Dependency," *IEEE Transactions on Wireless Communications*, pp. 1–1, 2019.

[5] L. Huang, S. Bi, and Y. J. Zhang, "Deep Reinforcement Learning for Online Computation Offloading in Wireless Powered Mobile-Edge Computing Networks," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2019.

[6] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[7] R. Yu, Y. Zhang, S. Gjessing, W. Xia, and K. Yang, "Toward cloud-based vehicular networks with efficient resource management," *IEEE Network*, vol. 27, no. 5, pp. 48–55, Sep. 2013.

[8] C.-C. Lin, D.-J. Deng, and C.-C. Yao, "Resource Allocation in Vehicular Cloud Computing Systems With Heterogeneous Vehicles and Roadside Units," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3692–3700, Oct. 2018.

[9] M. Nabi, R. Benkoczi, S. Abdelhamid, and H. S. Hassanein, "Resource assignment in vehicular clouds," *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–6.

[10] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: A survey," *arXiv preprint arXiv:1908.06849*, 2019.

[11] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4268–4282, 2016.

[12] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint computation offloading and user association in multi-task mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 12, pp. 12313–12325, 2018.

[13] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[14] Y. Liu, S. Wang, J. Huang, and F. Yang, "A computation offloading algorithm based on game theory for vehicular edge networks," *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[15] J. Zhao, X. Sun, Q. Li and X. Ma, "Edge Caching and Computation Management for Real-Time Internet of Vehicles: An Online and Distributed Approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 4, pp. 2183-2197, April 2021.

[16] 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 Vehicular Technology Magazine*, vol. 12, no. 2, pp. 36–44, Jun. 2017.

[17] L. Yang, J. Cao, H. Cheng, and Y. Ji, "Multi-User Computation Partitioning for Latency Sensitive Mobile Cloud Applications," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2253–2266, 2015.

[18] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Transactions on Communications*, vol. 65, no. 8, pp. 3571–3584, 2017.

[19] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[20] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," MIT press, 2018.

[21] V. Mnih et al., "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[22] S. Boyd, N. Parikh, and E. Chu, "Distributed optimization and statistical learning via the alternating direction method of multipliers," Now Publishers Inc, 2011.

[23] K. Zhang, Y. Mao, S. Leng, S. Maharjan, and Y. Zhang, "Optimal delay constrained offloading for vehicular edge computing networks," *2017 IEEE International Conference on Communications (ICC)*, 2017, pp. 1–6.

[24] Q. Luo, C. Li, T. H. Luan, and W. Shi, "Collaborative Data Scheduling for Vehicular Edge Computing via Deep Reinforcement Learning," *IEEE Internet of Things Journal*, 2020.

[25] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[26] K. Zhang, S. Leng, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Artificial intelligence inspired transmission scheduling in cognitive vehicular communications and networks," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1987–1997, 2018.

[27] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Deep learning empowered task offloading for mobile edge computing in urban informatics," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7635–7647, 2019.

[28] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, pp. 44–55, 2017.

[29] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *Thirtieth AAAI conference on artificial intelligence*, 2016, pp. 2094–2100.

[30] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," *International conference on machine learning*, 2016, pp. 1995–2003.

[31] C. Campolo and A. Molinaro, "Improving V2R connectivity to provide ITS applications in IEEE 802.11 p/WAVE VANETs," *2011 18th International Conference on Telecommunications*, 2011, pp. 476–481.

[32] X. Lin, Y. Wang, Q. Xie, and M. Pedram, "Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 175–186, 2014.

[33] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, “A comprehensive survey on graph neural networks,” *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[34] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” *Proc. of ICLR*, 2017.

[35] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *Proc. of ICLR*, 2017.

[36] V. Mnih et al., “Asynchronous methods for deep reinforcement learning,” *International conference on machine learning*, 2016, pp. 1928–1937.

[37] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” *International conference on machine learning*, 2014.

[38] T. P. Lillicrap et al., “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.

[39] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, “Continuous deep q-learning with model-based acceleration,” *International Conference on Machine Learning*, 2016, pp. 2829–2838.

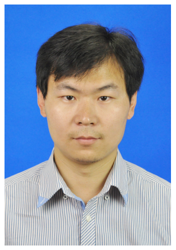
[40] China Postgraduate Innovation & Practice Competition (CPIPC) https://cpipc.chinadegrees.cn/

[41] J. Xiong et al., “Parametrized Deep Q-Networks Learning: Reinforcement Learning with Discrete-Continuous Hybrid Action Space,” *arXiv preprint arXiv:1810.06394*, 2018.

[42] C. J. Bester, S. D. James, and G. D. Konidaris, “Multi-pass q-networks for deep reinforcement learning with parameterised action spaces,” *arXiv preprint arXiv:1905.04388*, 2019.

[43] A. Krishnamoorthy and D. Menon, “Matrix inversion using Cholesky decomposition,” *2013 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, 2013, pp. 70-72.

Xinyu You recevied the B.Eng. degree from the School of Electronic Information, Wuhan University, Wuhan, China, in 2018. He is currently pursuing the M.S. degree at Fudan University, Shanghai, China. His research interests include deep reinforcement learning, routing, and edge computing.



Yuedong Xu received B.S. degree from Anhui University in 2001, M.S. degree from Huazhong University of Science and Technology in 2004, and Ph.D. degree from The Chinese University of Hong Kong, Hong Kong, China, in 2009. He is a tenured Associate Professor with the School of Information Science and Technology, Fudan University, Shanghai, China. From late 2009 to 2012, he was a Postdoc with INRIA Sophia Antipolis and Universite d’Avignon, France. He has published more than 30 papers in reputable vents including ACM Mobisys, ACM Mobihoc, ACM CoNEXT, IEEE Infocom, IEEE/ACM ToN, IEEE TMC, etc. His research interests include performance evaluation, machine learning, data analytics of communication networks, and mobile computing.

PLACE PHOTO HERE

Liangui Dai Liangui Dai received Ph.D. degree in Control Theory and Application from Northeastern University, ShenYang, China in 1997. He is a senior researcher with the intelligent transportation system (ITS) research institute of Guangdong Litong Corp. His research interests include data analytics of ITS and information system development for ITS.

Appendix A
Parameterized Normalized Advantage Functions (PNAF)

The parameterized action space consists of the discrete action set $\mathcal{A}_d = [k] = \{k_1, k_2, \dots, k_K\}$ and its corresponding continuous action set $\mathcal{X}_k \subseteq \mathbb{R}^{m_k}$, where K is the number of discrete actions and m_k is the dimension of the continuous action associated with discrete action k. Therefore, the parameterized action space $\mathcal{A}$ can be denoted as

$$\mathcal{A} = \{(k, x_k) | x_k \in \mathcal{X}_k \text{ for all } k \in [k]\}. \tag{1}$$

The Bellman equation under parameterized action space can be expressed as

$$Q(s_t, k_t, x_{k_t}) = \mathbb{E}[r_t + \gamma \max_{k_{t+1}} \sup_{x_{k_{t+1}}} Q(s_{t+1}, k_{t+1}, x_{k_{t+1}})]. \tag{2}$$

The neural network is usually a nonconvex function, and therefore taking supremum over the continuous space $\mathcal{X}_k$ is computationally intractable. We decompose the action-state value function Q into the state value function V and the advantage function A, and the Bellman equation in Equation (1) becomes

$$\begin{aligned} Q(s_t, k_t, x_{k_t}) = \mathbb{E}\{r_t + \gamma \max_{k_{t+1}} \sup_{x_{k_{t+1}}} [V(s_{t+1}, k_{t+1}) + \\ A(s_{t+1}, k_{t+1}, x_{k_{t+1}})]\}. \end{aligned} \tag{3}$$

Since the state value function $V(s_{t+1}, k_{t+1})$ is irrelevant with the continuous action $x_{k_{t+1}}$, it can be taken out of the operation of taking supremum over continuous space, and therefore Equation (43) can be rewritten as

$$\begin{aligned} Q(s_t, k_t, x_{k_t}) = \mathbb{E}\{r_t + \gamma \max_{k_{t+1}} [V(s_{t+1}, k_{t+1}) + \\ \sup_{x_{k_{t+1}}} A(s_{t+1}, k_{t+1}, x_{k_{t+1}})]\}. \end{aligned} \tag{4}$$

With the carefully designed neural network in Fig. 1, the state value function $V(s_{t+1}, k_{t+1})$ and the advantage function $A(s_{t+1}, k_{t+1}, x_{k_{t+1}})$ can be estimated simultaneously. The pseudo-code of Parameterized Normalized Advantage Functions (PNAF) is shown in Algorithm 1.

We first initialize the normalized Q network $Q(s, k, x_k; \theta)$ randomly and initialize the target network $Q'(s, k, x_k; \theta')$ with the same network parameters $\theta' = \theta$. The replay buffer D is initialized to store the environment transitions.

At each decision epoch t, the agent observes the current state s_t and chooses the discrete action k_t and the continuous action x_{k_t} according to the output of neural network. In order to encourage the agent to discover the action space sufficiently, the discrete and continuous actions will be decided according to the ϵ-greedy policy and OU process separately. The agent will choose a random discrete action from its action space $[K]$ with probability ϵ, and otherwise will select the action with the highest V-value, that is

$$k_t = \begin{cases} \text{a random action} & \text{with probability } \epsilon \\ \arg\max_k V(s, k; \theta) & \text{with probability } 1 - \epsilon \end{cases} \tag{5}$$

Then the continuous action x_{k_t} associated with the discrete action k_t will be selected with the injection of temporally correlated noise generated by OU process. Once the parameterized action $a_t = (k_t, x_{k_t})$ is executed, the environment state turns to s_{t+1}, and at the same time the agent will receive the immediate reward r_t. After that, the agent records the environment transition (s_t, a_t, s_{t+1}, r_t) into the experience replay buffer D.

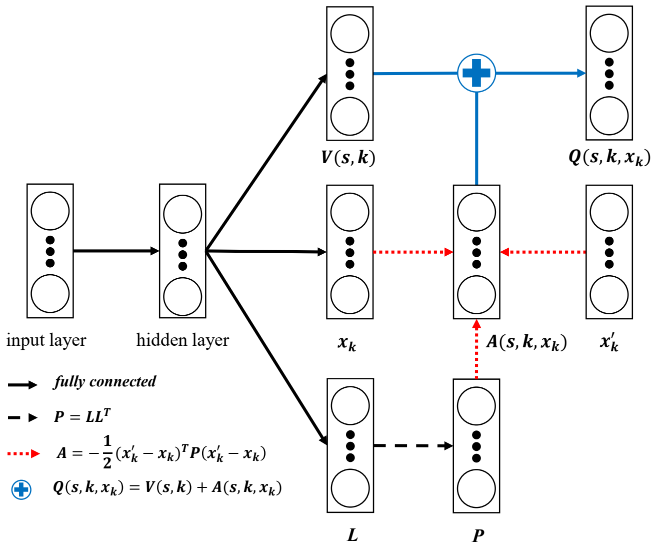


Fig. 1. The architecture of the parameterized network.

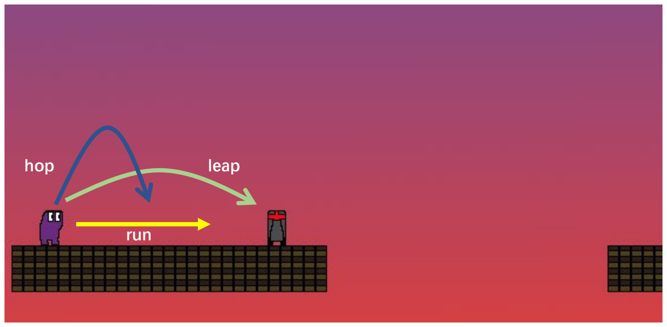


Fig. 2. The platform domain.

At the beginning of parameter update, a training batch (s_i, a_i, s_{i+1}, r_i) is sampled randomly from D. The target value y_i is set to the sum of the immediate reward r_i and the highest V-value of target network Q', which can be expressed as

$$y_i = r_i + \gamma \max_{k \in [K]} V'(s_{i+1}, k; \theta'), \tag{6}$$

where γ is the discount factor. We update θ, the parameters of the Q network, by minimizing the mean squared error between the target Q-value y_i and its current output. The loss function of Q network L_t is expressed by

$$L_t = (y_i - Q(s_i, k_i, x_{k_i}; \theta))^2. \tag{7}$$

Then we update the parameters of the target network θ' in a soft-update manner:

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta', \tag{8}$$

where τ is the soft-update coefficient.

Algorithm 1 Parameterized Normalized Advantage Functions

```
// initialization
Randomly initialize normalized Q network $Q(s, k, x_k; \theta)$
Initialize target network $Q'(s, k, x_k; \theta')$ with $\theta' \leftarrow \theta$
Initialize replay buffer $D \leftarrow \varnothing$

// training process
for episode = 1, $M$ do
  Initialize $\epsilon$-greedy policy for discrete action exploration
  Initialize OU process for continuous action exploration
  Receive initial observation state $s_1$
  for t = 1, $T$ do
    Select discrete action $k_t$ and continuous action $x_{k_t}$
    Execute parameterized action $a_t = (k_t, x_{k_t})$
    Calculate and collect reward $r_t$
    Observe next state $s_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $D$
    Sample a random batch $(s_i, a_i, r_i, s_{i+1})$ from $D$
    Set target value $y_i = r_i + \gamma \max_{k \in [K]} V'(s_{i+1}, k; \theta')$
    $\theta \leftarrow \mathit{GradientDescent}\ ((y_i - Q(s_i, k_i, x_{k_i}; \theta))^2\ )$
    Update target network $Q'$: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$
  end for
end for
```

To validate the efficiency of our proposed algorithm, we conduct experiments on Platform domain [2]. As shown in Fig. 2, the discrete action space consists of {run, hop, leap}, and each of them is associated with a continuous action to control the position of the agent. The goal of the agent is to reach the destination with proper policy, and if the agent touches the enemy or falls into a gap, it will lose the game. The agent can observe 9-dimensional environment state including the position and velocity of itself and the enemy. The immediate reward is set to the moving distance before and after the execution of hybrid action.

The baseline algorithms consist of P-DQN [1] and MP-DQN [2], which are the state-of-the-art algorithms in the domain of parameterized DRL. By combining the technique of DQN and DDPG, P-DQN constructs a Q-network to estimate the Q-value of each discrete action and an actor network to select the optimal continuous action. Since P-DQN suffers from the over-parameterization problem, MP-DQN introduces a multi-pass method to address it and achieves better performance than P-DQN.

We conduct five individual experiments with different random seeds and take the mean value as the final result. The training result is shown in Fig. 3. We can find that the total reward of each algorithm rises along with the increase of training episodes. Among these three algorithms, PNAF can achieve the fastest convergence speed and the highest cumulative reward. In order to evaluate the performance of each algorithm without ϵ-greedy policy and the OU process, we conduct all-exploitation test every ten episodes during the training process. More specifically, the discrete action with the highest Q-value or V-value and the continuous action without OU noise are selected. As shown in Fig. 4, we can observe that PNAF can find and stabilize at the optimal policy at 800 episodes, while MP-DQN takes 3000 episodes

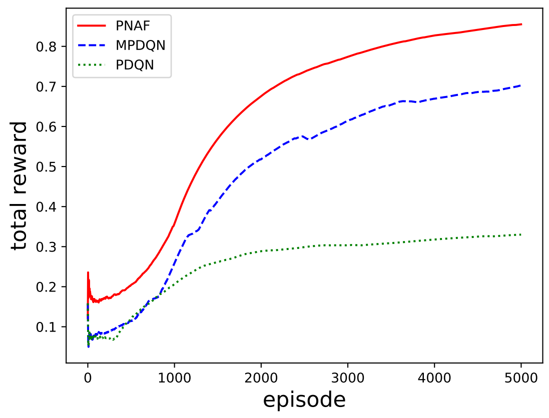


Fig. 3. Training result in Platform domain.

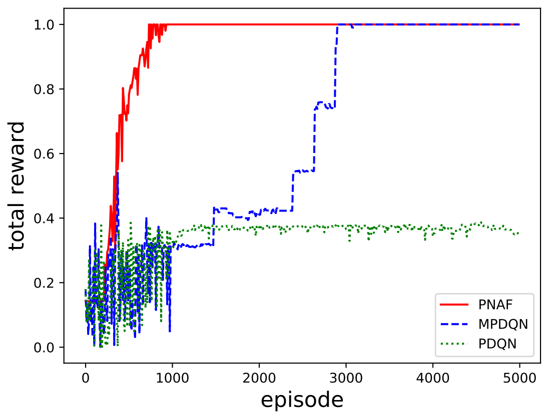


Fig. 4. Test result in Platform domain.

and P-DQN lags far to converge behind PNAF and MP-DQN.

From the neural network in Fig. 1 and Equation (4), we can find that when computing the action-state value function $Q(s_t, a_t)$ of the hybrid action $a_t = (k_t, x_{k_t})$, the advantage function $A(s_t, k_t, x_{k_t})$ of the continuous action x_{k_t} is added correspondingly to the state value function $V(s_t, k_t)$ of the associated discrete action k_t, thus ensuring the correlation of the discrete and continuous actions. Compared with P-DQN and MP-DQN, PNAF has three main advantages: 1) with a single neural network, the state value function of discrete actions and the advantage function of continuous actions can be estimated; 2) the value of discrete actions depends only on its corresponding continuous actions and is not affected by unassociated continuous actions; 3) the upgrade process of continuous actions is only related to its corresponding discrete action without the interference of other discrete actions.

References

[1] J. Xiong et al., "Parametrized Deep Q-Networks Learning: Reinforcement Learning with Discrete-Continuous Hybrid Action Space," *arXiv preprint arXiv:1810.06394*, 2018.
[2] C. J. Bester, S. D. James, and G. D. Konidaris, "Multi-pass q-networks for deep reinforcement learning with parameterised action spaces," *arXiv preprint arXiv:1905.04388*, 2019.