CCF Transactions on Pervasive Computing and Interaction
https://doi.org/10.1007/s42486-024-00156-x

REGULAR PAPER

A bandwidth-fair migration-enabled task offloading for vehicular edge computing: a deep reinforcement learning approach

Chaogang Tang[1] · Zhao Li[1] · Shuo Xiao[1] · Huaming Wu[2] · Wei Chen[1]

Received: 14 December 2023 / Accepted: 10 April 2024

Abstract
Vehicular edge computing (VEC), which extends the computing, storage, and networking resources from the cloud center to the logical network edge through the deployment of edge servers at the road-side unit (RSU), has aroused extensive attention in recent years, by virtue of the advantages in meeting the stringent latency requirements of vehicular applications. VEC enables the tasks and data to be processed and analyzed in close proximity to data sources (i.e., vehicles). VEC reduces the response latency for vehicular tasks, but also mitigates the burdens over the backhaul networks. However, how to achieve cost-effective task offloading in VEC remains a challenging problem, owing to the fact that the computing capabilities of the edge server are not sufficient enough compared to the cloud center and the uneven distribution of computing resources among RSUs. In this paper, we consider an urban VEC scenario and model the VEC system in terms of delay and cost. The goal of this paper is to minimize the weighted total latency and vehicle cost by balancing the bandwidth and migrating tasks while satisfying multiple constraint conditions. Specifically, we model the task offloading problem as a weighted bipartite graph matching problem and propose a Kuhn-Munkres (KM) based Task Matching Offloading scheme (KTMO) to determine the optimal offloading strategy. Furthermore, considering the dynamic time-varying features of the VEC environment, we model the task migration problem as a Markov Decision Process (MDP) and propose a Deep Reinforcement Learning (DRL) based online learning method to explore optimal migration decisions. The experimental results demonstrate that our strategy has better performance compared to other methods.

Keywords Vehicular edge computing · Bandwidth fairness · Task offloading · Task migration · Deep reinforcement learning

1 Introduction

With the rapid development of the fifth generation (5 G) mobile networks, the Internet of Things (IoT) and artificial intelligence (AI), intelligent transport systems (ITS) have attracted considerable attention from both industry and academia in recent years. As an important branch of IoT, the Internet of Vehicles (IoV) system is of particular significance in the development of ITS and smart cities (Tang et al. 2022; Liu et al. 2017). The IoV is an interactive network that consists of intelligent vehicles, stationary infrastructures, travelling roads and other entities, with the abilities to communicate, store and process vehicular requests. Deploying an array of sensing devices and intelligent modules on the vehicles enables the dynamic collection of data about vehicles and surrounding environments, which can guarantee driving safety to a great degree. Generally, the rapid development of the IoV has facilitated the widespread use of intelligent vehicle services for autonomous driving, entertainment applications, smart navigation, and intelligent parking (Tang et al. 2018; Zeng et al. 2021). Such applications usually generate a mass of data that need to be handled immediately to satisfy the ultra-low response latency requirement in IoV.

Although vehicles have certain storage and computing capabilities, various vehicular applications are making these vehicles increasingly inadequate to support the latency-sensitive and computation-intensive tasks (Liu et al. 2019). To address the shortage of resources for IoV,

✉ Shuo Xiao
sxiao@cumt.edu.cn

[1] School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, Jiangsu, China

[2] Center for Applied Mathematics, Tianjin University, Tianjin 300072, China

Published online: 26 May 2024

extensive research has been conducted to incorporate cloud resource (Wang et al. 2020). However, explosively growing number of tasks offloaded to the cloud consumes large backhaul networking resources, thus incurring long response latency for the vehicular tasks. In this case, the response latency may hardly meet the stated Quality of Service (QoS) or the Quality of Experience (QoE) from the viewpoint of resource requestors.

Accordingly, vehicular edge computing (VEC) is proposed in response to the above challenges (Tang et al. 2022). VEC pushes various computing, storage, and networking resources to the network edge through deploying basic computing and communication facilities at the road-side unit (RSU). By doing so, these resources can be provisioned in close proximity to data sources (e.g., smart vehicles), thus significantly reducing various networking latency (Han et al. 2019; Qiao et al. 2018). However, several issues around VEC still need to be addressed. For instance, the computing resources are not evenly distributed among RUSs and the bandwidth resources of RSUs are also limited. Inappropriate task offloading will cause network congestion and long transmission latency. Therefore, designing appropriate task offloading strategy plays a significant role in reducing response delay and improving the efficiency of VEC systems.

At the same time, various network operators are beginning to realize the potential opportunities and benefits of RSUs. The deployment of RSUs by multiple operators on the road can achieve wider coverage, provide real-time traffic information and services, and facilitate data exchange and collaboration among operators. However, there are various costs associated with the deployment of RSUs, including purchase, installation and maintenance, as well as energy consumption during operation. Operators may determine the charges for RSU resources according to their own strategies, business models and market competition. On the other hand, the computing resources are also provisioned in a pay-as-you-go fashion (Tang and Wu 2021). Therefore, the cost of requesting resources is a matter of consideration for users, in addition to a better quality of service. It easily occurs to people that task migration among RUSs may be a viable option to minimize the overall costs. However, it remains a major challenge for traditional methods to find an appropriate migration strategy in such a dynamic VEC environment. In recent years, Deep Reinforcement Learning (DRL) has become widely used in edge computing due to its powerful perception and decision-making abilities. DRL introduces neural networks with perceptual capabilities that explore the best actions to take by constantly interacting with the environment, with the aim of maximizing future cumulative rewards. Thus, DRL as a machine learning method is well suited for finding optimal migration strategies in dynamic and complex VEC environments.

To overcome the aforementioned issues such as centralized offloading and the uneven distribution of computing resources, in this paper, we study and analyse the task scheduling and migration problems in VEC, respectively. We try to minimize the weighted sum of the processing latency of tasks and vehicle cost. The main contributions of this paper are listed as follow:

- Considering the impact of uneven offloading of tasks, we model the task scheduling problem as a weighted bipartite graph best matching problem. A matching offloading scheme based on the Kuhn-Munkres (KM) (Munkres 1957) algorithm is proposed to keep the bandwidth allocation and interference relatively balanced for all tasks.
- As the unit price of resources varies among edge servers and has time-varying characteristics, we design an efficient migration strategy to reduce the overall cost of the vehicle. Specifically, we model task migration as a Markov Decision Process (MDP), defining the system state space, action space and reward function. An adaptive migration algorithm is proposed based on Actor-Critic(AC).
- Extensive simulation is carried out to evaluate the proposed offloading scheme compared to other methods. The results show that our proposed method has better performance in both transmission delay and vehicle cost.

The rest of the paper is organized as follows. In Sect. 2, we review some related works on this topic. We present the system model in Sect. 3. In Sect. 4, we formulate the optimization problem and design the corresponding strategies to solve it. Simulation results are presented and analyzed in Sect. 5. In Sect. 6, we summarize the work of this paper.

2 Relate work

In VEC, as the edge servers are located close to the vehicles, it is feasible for the vehicles to utilize edge servers to offload computing tasks, which not only reduces the computational loads on vehicles, but also enables more real-time responses to vehicular offloading requests. In addition, edge servers are able to provide better QoS for vehicular applications compared to remote clouds (Raza et al. 2019). Many researchers have contributed to overcome the challenges of VEC (Anawar et al. 2018; Hou et al. 2016), especially in offloading decisions and resource allocation, while taking into account a number of factors, such as task characteristics, network traffic, edge node resources, energy consumption, and QoS. Related research is focused on developing intelligent offloading decision-making algorithms to optimize the selection and collaboration of edge nodes with the goal of improving system efficiency and QoS (Kim et al. 2018; Liu et al. 2021).

The elaborate resource allocation aims to provision efficient and elastic services, given limited resources and various task constraints. By optimizing offloading decisions and resource allocation, better overall performance, energy efficiency, and user experience can be achieved while maximizing the benefits and potential of edge computing.

The authors in Dai et al. (2019) try to maximize system utility via a joint load balancing and offloading approach. The system utility is expressed as the task computation delay in VEC networks. They proposed an efficient algorithm with low time complexity to jointly optimize selection decision, offloading ratio, and computation resource among edge servers. In Zhang et al. (2020), the authors proposed an approximate computational offloading scheme to achieve load balancing of computing resources among edge servers. The method aims to reduce processing delay by appropriately allocating the computational resources among the edge servers. The authors in Guo et al. (2019) proposed a VEC network that incorporates Fiber-Wireless enhanced capabilities. They studied the cooperative task scheduling problem, in the hope to reduce the latency in processing tasks. Two task offloading approaches were proposed and their superior performance was verified through simulation experiments. The authors in Yuan and Zhou (2021) studied a cloud-edge collaborative computing problem that jointly considered the features of edge servers (e.g., the mount of resources, the load balance of edge nodes) and the cloud center (e.g., the task queue stability). A fine-grained collaborative task scheduling strategy has been put forward, with the goal of profit maximization from the angles of cloud and edge computing systems.

However, the research works mentioned above do not fully consider the dynamic nature of the VEC networks. On the other hand, some researchers have shifted their attention to the applicability of machine learning (ML) in VEC networks, since ML technologies have presented broad applicability in dynamic environments, by virtue of powerful capabilities in dealing with variable data distributions and patterns (Sun et al. 2019; Qi et al. 2019).

The authors in Qi et al. (2019) proposed a knowledge-driven service offloading scheme for the IoV that combines DRL to explore optimal task offloading strategy. This knowledge-driven mechanism supports pre-training and online learning to effectively adapt to any changes in the environment. To avoid service-disruption-incurred failures when vehicles leave the server coverage, service migration has been proposed as an efficient countermeasure in Taleb et al. (2019). Depending on the movement trajectory of the vehicle, the relevant computing services can be migrated to another edge server that may be associated with the vehicle in the future. Moon et al. (2021) defined a task migration problem that aims to balance the computational load of edge servers through task migration. The authors introduced a DRL algorithm to explore the optimal migration strategy while reducing the migration cost according to the observed environmental state. Huang et al. (2020) proposed a Deep Reinforcement Learning-based Online Offloading (DROO) framework to optimally adapt to time-varying wireless channel and wireless devices with different transmit power. Additionally, a novel order-preserving action generation method is developed to efficiently generate binary offloading actions and significantly reduce the execution time.

Liu et al. (2022) proposed a Deep Learning-based Edge Cache Optimization method (DLECO) to reduce the cost of the cache planning process. The method used the Long Short-Term Memory (LSTM) unit as the core component of the deep learning model and combined pointer networks with attention mechanisms as a novel deep neural architecture. The experimental results confirm the superior performance of their proposed approach. Ning et al. (2019) constructed an intelligent offloading system for VEC by DRL. The communication and computation states were modeled as finite Markov chains. In addition, with the overall objective of maximizing the quality of the user experience, a two-sided matching scheme and a DRL approach were developed to schedule task requests and allocate network resources, respectively.

In VEC, when multiple vehicles choose to offload tasks to a particular RSU at the same time, it can cause network congestion and increase transmission delay. The aforementioned works have rarely taken into account this situation. Compared to the existing works, we pay more attention to the impact of balancing bandwidth allocation and transmission interference. More importantly, we simultaneously consider task migration in the context of uneven distribution of computing resources to reduce the computing time and cost.

3 System model

Figure 1 shows the considered application architecture which consists of four streets and related intersections. Base station (BS) and RSUs are deployed in this area with edge servers to provide computing resources for passing

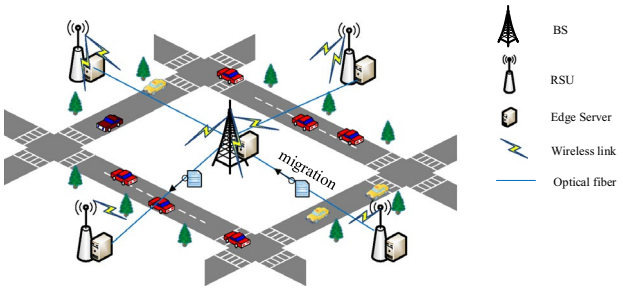


Fig. 1 System model

vehicles. They are connected by optical fiber, allowing task migration and ensuring transmission quality.

In this paper, the VEC system consists of one BS, M RSUs and N vehicles. The sets of vehicles and RSUs are indexed by $\mathcal{N} = \{1, ..., N\}$ and $\mathcal{M} = \{1, ..., M\}$, respectively. To provide a clear definition, we add the BS to the set of RSUs with index 0. The BS covers all vehicles and RSUs in the current area and each vehicle is in the communication range of at least one RSU. It is assumed that there exists a global controller on the BS that not only has full knowledge of the system but also makes decisions for all the vehicles in a centralized manner. At the start of each time slot, the vehicles and RSUs send their own task and resource information to the BS, which is handed over to the global controller for decision-making. The vehicles on the roads are spaced following a random distribution, and the velocity of vehicle i is vel_i. Within the communication range of RSUs, the vehicles can offload their tasks to them. In addition, time is divided into a set of discrete time slots, indexed by $\{0, 1, ..., T-1\}$, where T is the number of time slots. At each time slot, vehicle n will generate a computation-intensive task $Task_n$. $Task_n$ can be described as $Task_n = \{D_n, W_n, T_n^{max}\}$, where D_n denotes the size of the computing task, $W_n = \tau D_n$ represents the computing resources required to process the task where τ denotes the required CPU cycles per data bit, and T_n^{max} indicates the latency requirement of the task.

3.1 Communication model

The communication model depicts the transmission time for a computing task offloaded from vehicle n to RSU m via a wireless channel. The data transmission rate between vehicle n and RSU m is given as

$$R_{n,m} = \frac{B_m}{\sum_{i\in\mathcal{N}} a_{i,m}} \log\left(1 + \frac{P_n H_{n,m}}{\sigma + \sum_{i\in\mathcal{N}/\{n\}}^{N} a_{i,m} P_i H_{i,m}}\right), \tag{1}$$

where B_m represents the total bandwidth of the RSU m and $a_{i,m}$ is a binary variable indicating whether the vehicle i offloads its task to RSU m. If $Task_i$ is offloaded to RSU m, $a_{i,m} = 1$, otherwise $a_{i,m} = 0$. $H_{n,m} = 4.11(\frac{3*10^8}{4\pi f_c d_{n,m}})^{d_e}$ represents the channel gain between vehicle n and RSU m, where $d_{n,m}$ denotes the distance, f_c denotes the carrier frequency and d_e denotes the path loss exponent (Huang et al. 2020). P_n is the transmission power of vehicle n and σ is the white gaussian noise power. Each vehicle can only offload its computing task to one RSU, so the following constraint should be satisfied

$$\sum_{m=1}^{M} a_{n,m} = 1, \ \forall n \in \mathcal{N}. \tag{2}$$

The transmission delay for offloading the computing task of vehicle n is given as follows:

$$T_n^{trans} = \frac{D_n}{\sum_{m=1}^{M} a_{n,m} R_{n,m}}. \tag{3}$$

3.2 Migration model

Migration time refers to the duration required for transferring a computing task from one edge server to another. Usually, the migration time depends upon the task size, the transmission rate among RSUs, and the number of migration hops between the source RSU and destination RSU. We assume that the optical fiber is used for the wired communication among servers to guarantee a higher transmission rate. The migration time of the $Task_n$ is expressed as follows:

$$T_n^{mig} = \frac{D_n}{R_o} \sum_{m=1}^{M} a_{n,m} \left(\sum_{m'=1}^{M} b_{n,m'} h_{m,m'}\right), \tag{4}$$

where R_o represents the transmission rate of optical fiber. $b_{n,m'} \in \{0, 1\}$ indicates whether the $Task_n$ is eventually processed by RSU m' and $h_{m,m'}$ is a non-negative integer indicating the least number of hops between RSU m and RSU m'. If $m = m'$, the task does not migrate and thus $h_{m,m'} = 0$. Each task can only be processed by one server, and thus the following constraint should be satisfied

$$\sum_{m'=1}^{M} b_{n,m'} = 1, \ \forall n \in \mathcal{N}. \tag{5}$$

3.3 Computation model

The computation model describes the time required to accomplish a computing task at the edge server. Let $f_{n,m'}$ be the computing resources allocated to the $Task_n$ by the edge server in RSU m', so the computing delay for the task $Task_n$ is expressed as follows:

$$T_n^{com} = \frac{W_n}{\sum_{m'=1}^{M} b_{n,m'} f_{n,m'}}. \tag{6}$$

The total latency for $Task_n$ is given as

$$T_n^{total} = T_n^{trans} + T_n^{mig} + T_n^{com}. \tag{7}$$

The processing result of the task is transmitted to the RSU closest to the vehicle via optical fiber and then returned to the vehicle over the wireless channel. Since the computing

result is smaller than that of the offloading task, we ignore the time spent on the transmission of computing result in this paper.

3.4 Cost model of requesting vehicle

We study the energy consumption and cost of the vehicles when they use the RSU services. Certain energy on the link communication is consumed when vehicle offloads a task to the RSU. Additionally, the services at the edge server including storage and computing are all provisioned in a pay-as-you-go model. Accordingly, we assume that the network operators will charge the served vehicles. Note that from the viewpoint of vehicles, the cost mainly includes the migration cost and computing cost.

3.4.1 Communication energy consumption

Transferring tasks from the vehicle to the RSU will consume energy, which is directly related to the vehicle's transmission power and transmission time. The energy consumption generated by vehicle n in link communication is

$$E_n^{trans} = P_n \sum_{m=1}^{M} a_{n,m} T_{n,m}^{trans}, \tag{8}$$

where P_n is the transmission power of vehicle n.

3.4.2 Migration cost

Costs are generated during task migration including computing replication from source server to destination server and resources releasing at the currently hosted edge server (Yuan et al. 2020). In addition, the bandwidth resources of the optical fiber are also occupied during task migration. Therefore, taking these factors into account, the migration cost is jointly determined by the data size and the number of migration hops for $Task_n$ migrated from RSU m to m'. The migration cost is given as follows, where P^{mig} represents the unit price for migrating one bit of data.

$$Cost_n^{mig} = D_n \sum_{m=1}^{M} a_{n,m} \Big(\sum_{m'=1}^{M} b_{n,m'} h_{m,m'} \Big) P^{mig}. \tag{9}$$

3.4.3 Computing cost

After task migration, each task eventually arrives at an edge server and are executed there. Certain amount of energy is consumed in each CPU cycle during task execution. To motivate RSUs to contribute their computing resources, the operator usually sets appropriate prices for computing resources to guarantee the profits of RSUs. We define the initial unit price for the computing resources of the edge server m' as $P_{m',init}^{com}$. Besides, we assume that the price of computing resources denoted $P_{m'}^{com}$ is variable. For instance, $P_{m'}^{com}$ usually increases, when the amount of the available resources decrease. As a result, we can define $P_{m'}^{com}$ as

$$P_{m'}^{com} = P_{m',init}^{com} \beta^{(1-\frac{f_{m'}}{F_{m'}})}, \tag{10}$$

where $\beta(>1)$ represents the growth factor for unit price of the computing resources. $F_{m'}$ and $f_{m'}$ denote total resources and remaining resources of edge server m', respectively. The server operator will give an appropriate discount on the cost based on the time that the resource is rented. The longer the resource is rented, the greater the discount. We define a discount function for the rental time which is expressed as follows:

$$g(x) = \frac{\mu}{x + \mu}, \tag{11}$$

where $\mu(>0)$ is a hyperparameter used to control the discount intensity. The computing cost required for the $Task_n$ is given as follows:

$$\begin{aligned} Cost_n^{com} &= \sum_{m'=1}^{M} b_{n,m'} P_{m'}^{com} f_{n,m'} g\Big(\frac{W_n}{f_{n,m'}}\Big) \\ &= \sum_{m'=1}^{M} b_{n,m'} \frac{P_{m'}^{com} \mu f_{n,m'}^2}{W_n + \mu f_{n,m'}}. \end{aligned} \tag{12}$$

Therefore, the total cost for the $Task_n$ can be expressed as the weighted sum of energy consumption, migration cost and computating cost,

$$Cost_n^{total} = \varphi_1 E_n^{trans} + \varphi_2 Cost_n^{mig} + \varphi_3 Cost_n^{com}, \tag{13}$$

where $\varphi_i, i \in \{1,2,3\}$, is a weight coefficient used to balance the corresponding cost among the above three metrics. For example, more attention is focused on energy consumption when φ_1 is larger than the other two weights. When the size of the task data is large, more attention should be paid to the migration cost, so φ_2 can be set to a larger value than others. Conversely, when the vehicle is concerned with the price of computing resources, φ_3 will be set to a larger value.

4 Problem formulation

In this section, we formulate the optimization problem and design the offloading and migration strategies.

4.1 Optimization objective

As assumed earlier, the global controller can acquire necessary information about the VEC system and network. On one hand, it detects the mobility information of an arbitrary vehicle $n(\in \mathcal{N})$ and the achievable RSUs that vehicle n can communicate with. On the other hand, the controller develops appropriate policies to decide whether and where to offload and migrate the computing tasks among RSUs. In this study, we aim to minimize the total delay and vehicle cost while satisfying multiple constraints. The two objectives are defined as a weighted sum with the weight factor $\omega \in (0, 1)$. Consequently, the objective function is defined as

$$U(a,b,f) = \sum_{n=1}^{N} (\omega T_n^{total} + (1-\omega) Cost_n^{total}), \tag{14}$$

where $a = \{a_{n,m} | n \in \mathcal{N}, m \in \mathcal{M}\}, b = \{b_{n,m'} | n \in \mathcal{N}, m' \in \mathcal{M}\}$ and $f = \{f_{n,m'} | n \in \mathcal{N}, m' \in \mathcal{M}\}$. The problem P1 is formulated as

$$\begin{aligned}
&P1: \min_{a,b,f} U \\
&\text{s.t.} \\
&C1: \sum_{m=1}^{M} a_{n,m} = 1, \quad \forall n \in \mathcal{N}, \\
&C2: \sum_{m'=1}^{M} b_{n,m'} = 1, \quad \forall n \in \mathcal{N}, \\
&C3: f_{n,m'} \geq 0, \quad \forall n \in \mathcal{N}, \forall m' \in \mathcal{M}, \\
&C4: \sum_{n=1}^{N} b_{n,m'} f_{n,m'} \leq f_{m'}, \quad \forall m' \in \mathcal{M}, \\
&C5: h_{m,m'} \geq 0, \quad \forall m, \forall m' \in \mathcal{M}, \\
&C6: a_{n,m} \in \{0,1\} \quad \forall n \in \mathcal{N}, \forall m \in \mathcal{M}, \\
&C7: b_{n,m'} \in \{0,1\} \quad \forall n \in \mathcal{N}, \forall m' \in \mathcal{M}.
\end{aligned} \tag{15}$$

Specifically, C1 indicates that a task can only be offloaded to one RSU. C2 represents that a task only be eventually processed at one RSU (edge server). C3 indicates that the resources allocated to the $Task_n$ must be non-negative. C4 shows that the sum of computing resources allocated to the tasks by the server cannot exceed the remaining resources. C5 ensures that $h_{m,m'}$ is a non-negative integer. C6 and C7 mean that $a_{n,m}$ and $b_{n,m'}$ are both binary variables.

Problem P1 is a mixed integer programming problem, which is hard to solve. However, once a and b are given, the problem is reduced to a multi-objective optimization problem for resource allocation on each server. It is assumed that there are $K = \sum_{n=1}^{N} b_{n,m}$ tasks to be executed by server m. From the viewpoint of RSU m, the optimization problem P1 can decomposed into the optimization of minimizing the total delay and vehicle cost for all the tasks offloaded to RSU m. In particular, denote by $G_m(f_{1,m}, ..., f_{K,m})$ the total delay and vehicle cost for all the tasks processed by RSU m, given a and b, and G_m can be expanded as

$$\begin{aligned}
G_m(f_{1,m}, ..., f_{K,m}) &= \sum_{k=1}^{K} (\omega T_k^{com} + (1-\omega) Cost_k^{com}) \\
&= \sum_{k=1}^{K} (\frac{\omega W_k}{f_{k,m}} + (1-\omega) \frac{P_m^{com} \mu f_{k,m}^2}{W_k + \mu f_{k,m}}).
\end{aligned} \tag{16}$$

Up to now, the original optimization problem P1 can be transformed into the following subproblems P2, which aims to separately minimize the computing delay and computing cost for all the tasks processed by RSU m ($\forall m \in \mathcal{M}$) by designing an optimal resource allocation scheme $f = \{f_{n,m} | n \in \mathcal{N}\}$,

$$\begin{aligned}
&P2: \min_{f} G_m \\
&\text{s.t.} \quad C3, C4.
\end{aligned} \tag{17}$$

The optimization function G_m ($\forall m \in \mathcal{M}$) is convex, which can be proved as follows. We calculate the hessian matrix for G_m with regards (w.r.t.) to f as follows:

$$H(G_m) = \begin{bmatrix} \frac{\partial^2 G_m}{\partial f_{1,m}^2} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{\partial^2 G_m}{\partial f_{K,m}^2} \end{bmatrix}, \tag{18}$$

where $H(G_m)$ is a diagonal matrix and the second-order mixed partial derivative on the diagonal are expressed as follows, $\forall k \in \{1, ..., K\}$

$$\frac{\partial^2 G_m}{\partial f_{k,m}^2} = \frac{2\omega W_k}{f_{k,m}^3} + \frac{2(1-\omega) P_m^{com} \mu W_k^2}{(W_k + \mu f_{k,m})^3} > 0. \tag{19}$$

Therefore, $H(G_m)$ is a positive definite matrix and $G_m(f)$ is a convex problem. Therefore, the problem P2 can be solved by existing optimization technologies such as interior point method. And the resource allocation optimization for all servers is also a convex problem that can be expressed as $\sum_{m=1}^{M} G_m$.

To sum up, problem P1 can be decomposed into three sub-problems, namely, offloading decision, migration policy and resource allocation. The resource allocation problem can be solved by existing approaches. In the next, we can pay our attention to the offloading decision and migration strategy problems. The overall algorithmic framework is shown in the Fig. 2.

4.2 Offloading task scheduling

When multiple tasks are offloaded to a particular RSU at the same time, it can cause network congestion and increase transmission delay. High transmission delay is also partly attributed to the bandwidth sharing as shown in Eq. (1). To avoid these effects, a KM-based Task Matching Offloading scheme (KTMO) is proposed to achieve bandwidth-fair task offloading among RSUs in this paper.

The KM algorithm, well known as the bipartite graph best matching algorithm, is utilized to solve the best matching problem in weighted bipartite graphs. It is an extension of the Hungarian algorithm and aims at maximizing the weight sum of matching edges in the weighted bipartite graph by continually searching for augmented paths. The optimal time complexity of KM algorithm is $O(n^3)$.

Bipartite graph is a special model in graph theory. Let $G = (V, E)$ be an undirected graph. A graph G is considered as a bipartite graph if the vertex set V can be divided into two disjoint subsets (A, B), where each edge (i, j) in the graph is connected to two vertices i and j belonging to different vertex sets ($i \in A, j \in B$). In short, the vertex set V can be divided into two disjoint subsets, and the two vertices attached to each edge in the graph belong to the two disjoint subsets.

Our task offloading problem can be mapped into the bipartite graph as follows. We can respectively view the sets of vehicular tasks and RSUs as two different vertex sets in the graph, similar to the sets (A, B) above. Notably, the two sets are also disjoint with each other. Let the task vertex set be $X = \{X_1, X_2, ..., X_N\}$, and the RSU vertex set be $Y = \{Y_0, Y_1, Y_2, ..., Y_M\}$. A vehicle may offload its task to any of the candidate RSUs that wirelessly cover it. We define a weight function to evaluate the weight of the edge (X_i, Y_i). The weight value of (X_i, Y_i) can indicate the efficiency of task offloading from the vehicle to the RSU. Usually, the larger the weight value, the higher the efficiency of task offloading. Thus, the task offloading decision can be abstracted as a weighted bipartite graph matching problem, with the goal of maximizing the total weight of the edges between the tasks and the RSUs.

Algorithm 1 The description of KTMO Algorithm

Input: $\mathcal{N}, \mathcal{M}$.
Output: Matching results, *mat-res*.

Gather all task information through interactions and form a task queue TQ in descending order of urgency.
Initialize the matching task queue *mtq*.
while TQ is not empty **do**
  Clear *mtq*
  for $i \in [1, M+1]$ **do**
    if TQ is not empty **then**
      Pop a task and add it to *mtq*.
    else Add a virtual task to *mtq*.
    end if
  end for
  for each task $X_i \in mtq, i \in [1, M+1]$ **do**
    for each RSU $Y_j, j \in [1, M+1]$ **do**
      if task X_i is outside the communication range of the RSU Y_j or task X_i is virtual **then**
        $BG[X_i][Y_j] = 0$
      else Update $BG[X_i][Y_j]$ through Eq. (20)
      end if
    end for
  end for
  Use KM algorithm to get one round of matching results *km-res*
  for task, RSU in *km-res* **do**
    if The weight of the edge (task, RSU) is 0 **then**
      Push the task to TQ.
    else Add the tuple (taks, RSU) to *mat-res*
    end if
  end for
end while
return *mat-res*;

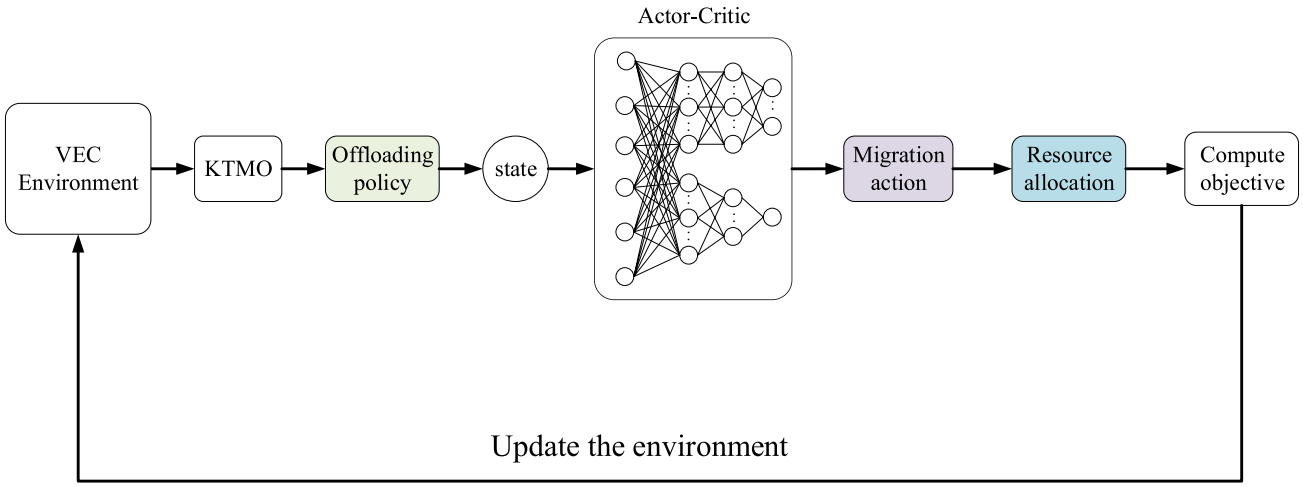


Fig. 2 Overall algorithmic framework

Generally, the number of tasks is much larger than that of RSUs. However, the KM algorithm requires that the number of task vertices is the same as the number of RSU vertices. Considering the effect of time complexity, we take the side with the fewer elements as the benchmark. It should be noted that, for the sake of clarity, we include BS in the set of RSUs so that the total number is $M+1$. Specifically, we firstly prioritize the vehicular tasks according to their urgency to form a task queue. In each round of matching, $M+1$ tasks with the highest priority are populated from the task queue as a set of tasks to be matched. If the length of the task queue is less than $M+1$, virtual tasks will be added. For this batch of tasks, we calculate the weight of each edge that links a task and an RSU according to Eq.(20). Note that if the vehicle generating the task is not covered by an RSU, or the task is virtual, the corresponding weight edge is set to zero. From this, a weighted bipartite graph is obtained and the KM algorithm is applied to return a set of matching results. To ensure accurate matching results, it is crucial to take into account tasks and RSUs with zero edge weights. A virtual task can be safely ignored as it will not affect the offload decision. However, if the vehicle generating the task is not wirelessly covered by the RSU it matches, then a false match has certainly been generated. Tasks that are incorrectly matched will be pushed to the top of the queue for the next round of matching. Finally, the offloading decision is updated based on the matching results, ending the round of matching. The KTMO algorithm is terminated when the task queue is empty, indicating that all tasks have successfully found the offloading target. The process and detail of the KTMO algorithm is depicted in Fig. 3 and Algorithm 1, respectively.

Multiple factors may influence the task offloading decision and further significantly affect system efficiency. It is efficient and important to consider the sigmoidal behavior feature in the users' utility function for resource allocation (Wang et al. 2017). Following the work (Ning et al. 2019) in which the authors adopt a sigmoid-like function to model the utility of vehicles, we in this paper also adopt this kind of function to describe the weights between vehicles and RSUs. For example, the weight function in this paper between vehicle n and RSU m is defined as follows,

$$Weight_{n,m} = \frac{w_0}{(w_1 d_{n,m} + e^{w_2 \rho_n h_{n,m}})}. \tag{20}$$

where $d_{n,m}$ denotes the distance between vehicle n and RSU m, $\rho_n = \frac{D_n}{T_n^{max}}$ indicates the urgency of the task, and $h_{n,m}$ is the channel gain. w_0 is a weight factor that can be set to a larger value. w_1 and w_2 control the magnitude of the parameter $d_{n,m}$ and ρ_n, respectively. In the task scheduling phase, our purpose is to fairly allocate tasks to each RSU in such a way as to balance bandwidth allocation and transmission interference, and ultimately reduce transmission latency.

Complexity Analysis. In each round, taking $M+1$ tasks from the task queue in $O(M)$ time (lines 5-10). Calculate the weights between this group of tasks and RSUs with time complexity $O(M^2)$ (lines 11-18). Then, the KM algorithm with time complexity $O(M^3)$ is used to get one round of matching results (line 19). Finally, checking the results and update offloading decisions in $O(M)$ time (lines 20-25). Therefore, the time complexity of each round is $O(M^3)$. Since the number of tasks is larger than that of RSUs, multiple rounds of matching are required to obtain offloading decisions, at least $\lceil N/(M+1) \rceil$ rounds. The best time complexity of the KTMO algorithm is $O(NM^2)$, and the worst is $O(NM^3)$.

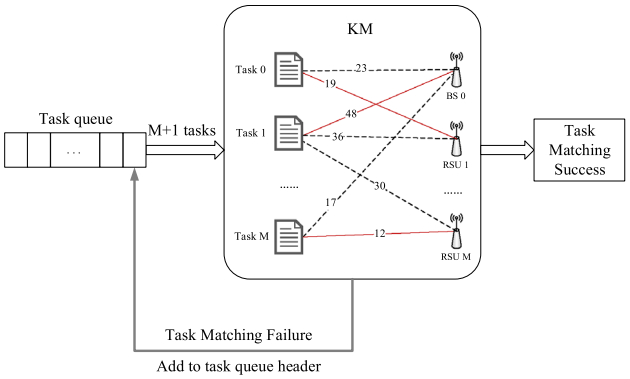


Fig. 3 The process of the KTMO algorithm

4.3 Migration decision

Due to the mobility of the vehicles, the stochasticity of the tasks and the time-varying characteristics of the resource prices, it is very challenging to solve task offloading and migration problems using traditional approaches. Therefore, we introduce a DRL approach to solve it in this paper.

DRL is a technique born from the combination of Deep Neural Networks(DNN) and Reinforcement Learning(RL). DNN consist of an input layer, multiple hidden layers, and an output layer, which provide them with strong perceptual capabilities. After continuous training on a specific dataset, it is possible to identify the internal connections between the data and output the corresponding predictions. DNN play a crucial role in various fields of AI owing to their exceptional perceptual abilities. Reinforcement learning has a strong decision-making capability where the agent interacts with the environment by performing actions and continuously learns the optimal action with the goal of maximizing future cumulative rewards. DRL combines the perceptual ability of DNN with the decision-making capability of RL by inputting the state of the environment to DNN, directly outputting the corresponding actions, and training the network parameters based on the feedback rewards.

Specifically, denote by s_t the VEC environment in time step t, and a_t denotes a migration action performed by the agent in time step t. Then, the current state s_t may transfer to any achievable following state s_{t+1}. The agent then receives a feedback reward r_t. In the long run, the agent takes a policy π to maximize future rewards $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$, where γ denotes the discount factor for future rewards. The policy π generates one action based on the observations on the environment.

In this paper, we propose a migration strategy based on Actor-Critic to explore and learn optimal actions. Specifically, we consider the global controller as an agent for the AC model. The agent constantly interacts with the VEC environment via a series of observations, actions, and rewards and makes migration decisions. The target is to choose appropriate actions to reduce vehicular cost in the system. The AC algorithm is shown in Algorithm 2. The elements in the DRL model which corresponds to our task migration problem are respectively defined as follows

1. State: The state information, as an input to the neural network in the DRL, needs to reflect the overall state of the system environment. The system state consists of the vehicle's mobility, task information, initial policy for task offloading, remaining resources of the edge server, and the unit price of resources. Therefore, state s_t is defined as

$$s_t = \{V_1, ..., V_N, R_1, ..., R_M\}, \tag{21}$$

where $V_n = \{loc_n, vel_n, p_n, dir_n, u_n, Task_n\}$. These parameters represent the vehicle's location, velocity, transmission power, direction, offloading target and the task information generated by the vehicle, which include the data size, cpu cycles and constraint latency, respectively. $R_m = \{f_m, P_m^{com}\}$, where f_m denotes the available computing resources and P_m^{com} denotes the unit price of the resources in the current time slot.
2. Action: For each step, the controller chooses an action according to the current state. In the environment, the controller decides to migrate tasks from source server to optimize the resource cost in the system. Migration action is defined as a vector with each element indicating the migration destination for the corresponding task. Particularly, action a_t is described as follows

$$a_t = \{e_1, ..., e_N\}, \tag{22}$$

where $e_n \in \mathcal{M}, \forall n \in \mathcal{N}$ denotes the migration target for the $Task_n$. If $e_n = u_n$, the task does not migrate, otherwise the task is migrated to e_n.
3. Reward: At the end of each step, the controller receives feedback from the environment as a reward. The objective of our optimization problem is to minimize the weighted total delay and vehicle cost, while the objective of DRL is to maximize the long-term reward of the system. Therefore, we formulate the reward function as the negative of the objective function

$$r_t = -U. \tag{23}$$

Usually, AC adopts the policy gradient which comprises two networks, i.e., the actor and critic networks, respectively.

The actor network generates a policy action to interact with the environment based on the system state, and the critic network fits a value function $V(s_t)$ to evaluate the action generated by the actor. It is assumed that the parameters of the actor network and critic network are θ and $\theta^{'}$.

To evaluate the action in the current state, we take the temporal difference (TD) error, which is defined as the difference between the estimated value and the real value, with the following expression:

$$\delta_t = r_t + \gamma V(s_{t+1};\theta^{'}) - V(s_t;\theta^{'}). \tag{24}$$

The TD error represents the distance between the real value and the estimated value. The smaller the TD error, the better the critic network is fitted. Therefore, to reduce the TD error, the critic network will be updated, so as to minimize the TD error with a loss function expressed as follows:

$$loss_{\theta'} = (r_t + \gamma V(s_{t+1};\theta^{'}) - V(s_t;\theta^{'}))^2. \tag{25}$$

Then, the parameter θ of actor network is updated by policy gradient, which is given as follows:

$$Loss_\theta = \frac{1}{N}\sum log\pi(a_t|s_t;\theta)\delta_t. \tag{26}$$

Here, N denotes the size of the action space. The gradient of θ can be expressed as

$$\theta_{t+1} = \theta_t + \alpha\nabla_\theta log\pi(a_t|s_t;\theta)\delta_t. \tag{27}$$

The Actor and Critic networks both consist of four layers. The layers of the Actor network are presented as follows:

1) First Layer: The layer takes the environment state as input, with the number of neurons consistent with the state space, which is then transformed by a Sigmoid function.
2) Second Layer: This is a hidden layer with 128 neurons, which is then transformed by the Relu function.
3) Third Layer: It's exactly the same as the previous layer.
4) Last Layer: This is the output layer. The layer is first reshaped into a matrix with the same number of rows and columns as the tasks and edge servers, respectively. Each row of the matrix is then normalized using Softmax function to derive the probability of each task migrating to each server.

The Critic network also has four layers, and unlike the Actor, the last layer has a neuron which directly outputs the fitted state value. The training process of DRL does not require a dataset, the agent acquires the state space from the environment at each time slot, performs the migration actions, and then updates the network gradient based on the reward feedback for online decision updating.

Algorithm 2 The description of the Actor-Critic

Input: System state s_t.
Output: Migration strategy.
Initialize actor and critic networks with random weights θ and $\theta^{'}$.
Initialize initial observation s_0
for each time slot $t = 1, 2, ...T-1$ **do**
  actor generates an action a_t
  Agent performs a_t, observes the next observation s_{t+1} and gets reward r_t
  Compute the TD error:
  $\delta_t = r_t + \gamma V(s_{t+1};\theta^{'}) - V(s_t;\theta^{'})$
  Compute the loss of the critic network and perform gradient decent:
  $\theta^{'}_{t+1} = \theta^{'}_t - \alpha\nabla_{\theta'} V(s_t;\theta^{'})\delta_t$
  Update the gradient of the actor network:
  $\theta_{t+1} = \theta_t - \alpha\nabla_\theta log\pi(a_t|s_t;\theta)\delta_t$
end for

5 Performance evaluation

In this section, we evaluate the performance of the proposed KTMO scheme through extensive simulations. First, the simulation scenario and basic parameters of the VEC environment are introduced. Then, the task offloading policy based on the KTMO algorithm is evaluated and compared with three other algorithms:

1. AC: The offloading decision is given directly by the AC with the reward function defined as the total transmission delay. Note that since each vehicle is not covered by all RSUs, so incorrect offloading actions may occur. To suppress incorrect actions, it is necessary to impose a penalty term on the reward function, which is related to the number of wrong actions.
2. DVIM (Ning et al. 2019): DVIM is a dynamic algorithm that matches vehicle-to-infrastructure communications. Each infrastructure has an accepted list and a forbidden list, and the overall system utility is optimized by updating these lists dynamically through Eq. (20).
3. Greedy approach: The greedy method prefers to select the RSU with the maximum weight.

5.1 Simulation setup

We abstract the set of edge servers linked by optical fiber as a connected graph and preserve the minimum number of hops between any two edge nodes using a shortest path algorithm. At the beginning, the environment is established

by initializing both vehicles and edge servers. Then, at each time slot, the KTMO algorithm is implemented to determine the offloading policy for vehicular tasks, which is then fed into the AC network along with the vehicle information and edge server information to obtain the migration strategy. Next, the optimal resource allocation scheme is implemented for the tasks received on each edge server. Finally, the state information of the vehicles and edge servers is updated at the end of each time slot.

We consider four traffic roads that intersect each other vertically with vehicles following a random distribution. It is assumed that the number of vehicles remains constant at each time slot. To this end, we assume one vehicle enters the considered zone if a vehicle within the zone leaves. BS and Four RSUs are uniformly distributed in this area and connected to each other through optical fiber to form a connected graph. Each vehicle has only one computing task to be completed in each time slot, and the size of its input data is randomly generated between the interval [100, 200]KB. In addition, the CPU cycles required for accomplishing the task range from 100 MHz to 200 MHz, and the maximum allowable processing delay is 0.5–1.0s.

The coverage area for BS is assumed to be 500 m, while each RSU is 300 m, with a bandwidth of 100MHz, and the CPU cycle frequency of edge server being [40, 20, 30, 30, 20]GHz, respectively. The initial price for computing resource of each server can be expressed [0.003, 0.001, 0.002, 0.002, 0.001]/MHz and the price of data migration is a constant 0.0002/KB. It should be noted that in the simulation, the migration cost is set relatively lower, as we pay more attention to the cost of computing resources. The discount factor $\gamma = 0.9$ in the AC model. For your easy reference, some key parameters to be used in the simulation are listed in Table 1.

Table 1 Simulation Parameters

Notation	Description	Value
M	The number of RSUs	4
N	The number of vehicles	20–40
D_n	The data size of a task	100–200KB
W_n	CPU cycles required for a task	100–200MHz
T_n^{max}	Maximum delay constraint	0.5–1.0s
vel_n	The velocity of a vehicle	10–20m/s
R	Communication range of RSUs	300 m
B	Bandwidth	100MHz
R_o	Transmission rate of optical fiber	$1 * 1e9$bits/s
F	Computing resources	[40, 20, 30, 30, 20]GHz
P_m^{com}	Initial unit price for each server	[0.003, 0.001, 0.002, 0.002, 0.001] /MHz
P^{mig}	Unit price for task migration	0.0002/KB
P	Transmission power of the vehicle	0.5w
f_c	Carrier frequency	615MHz
d_e	Path loss exponent	2.0
σ	Communications noise	10^{-11}

5.2 Simulation results

In this study, the ω serves as a strategy parameter. The objective function balances the ω between total delay and vehicle cost. To determine the optimal ω for the objective function, performance is measured by varying the weight factor ω as presented in Fig. 4. It is apparent that the impact on delay and cost is similar when ω is approximately 0.57. To take account of the importance of time, we set the weight factor ω to 0.6 in the next few experiments.

The simulation result shown in Fig. 5 reveals how the fairness of the offloading decision changes with the number of tasks under different offloading schemes. Based on bandwidth allocation method, the fairer the task offloading, the fairer the bandwidth allocation will be. Here, we measure the degree of fairness based on the standard deviation of the number of tasks offloaded to each RSU, which

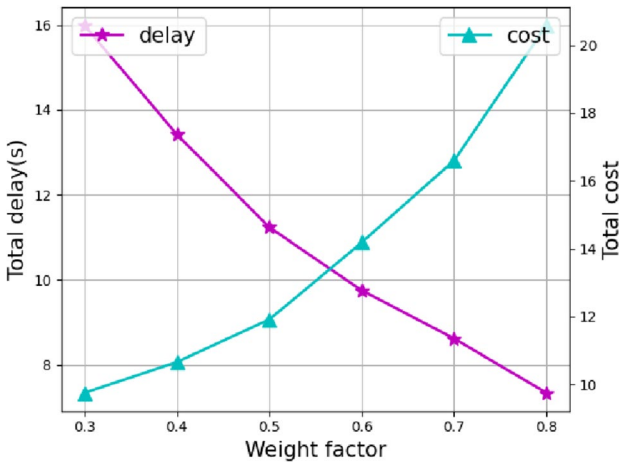


Fig. 4 Effect of weight factor ω on total delay and vehicle cost

is defined as $DF = \sqrt{\frac{\sum_{m=1}^{M}(\lambda_m - \bar{\lambda})^2}{M}}$, where $\lambda_m = \sum_{i=1}^{N} a_{i,m}$ indicates the number of tasks received by the RSU m and $\bar{\lambda} = \frac{N}{M}$ represents the mean value. The smaller the DF, the higher the fairness in bandwidth allocation. It is evident that our proposed KTMO matching scheme has a lower DF. The reason is that, an RSU is matched to only one task during each round of the KTMO algorithm, which can obtain an offloading strategy in a fairer way compared to other algorithms. This means that there is little difference in the number of tasks accepted by each RSU so does the

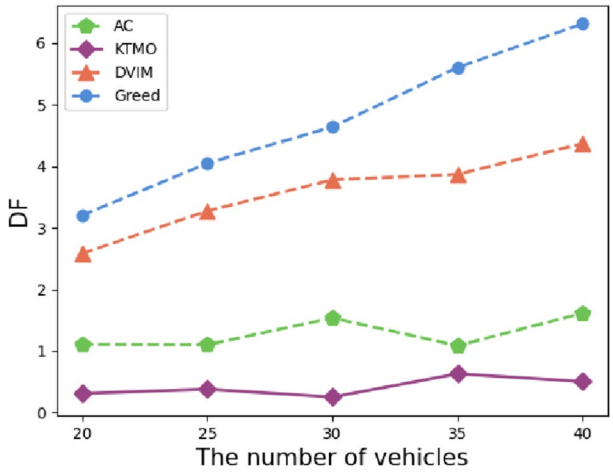


Fig. 5 Comparison of fairness in bandwidth with different numbers of vehicles

bandwidth, as the bandwidth is evenly distributed to each task. The greedy algorithms are more likely to centralize offloading, leading to a gradual increase in *DF* with the number of tasks. The AC algorithm adapts automatically as environmental parameters change to improve the strategy. Furthermore, it maintains a relatively lower *DF* value, which substantiates the benefits of fair offloading.

Figure 6 shows a comparison of the average transmission delay that is determined by offloading decision for different number of tasks. It can be seen that the KTMO offloading scheme shows the minimum total transmission delay compared to the other three algorithms. In our model, we evenly distribute the communication bandwidth of an RSU to the vehicles. When numerous tasks are offloaded to one specific RSU, the average bandwidth allocated to the vehicle decreases and the interference increases, which will lead to a significant increase in the transmission delay for this group of tasks. As depicted in Fig. 5, the KTMO algorithm can achieve a better fairness in task offloading and reduce the negative effects mentioned above. The other algorithms, particularly the greedy algorithm, could lead to a concentration of certain tasks on a specific RSU, resulting in higher transmission latency. However, we note that the AC has a less transmission time than KTMO for a vehicle number of twenty-five. It is true that balanced offloading is advantageous most of the time, but the dynamics of the environment and the stochastic nature of the task bring other possibilities.

Figure 7 shows the effect of weight factor ω on task completion rate, and we can see that the task completion rate rises as the weight factor increase. Firstly, it is worth noting that the values of transmission time and migration time do not fluctuate with changes in the weight factor ω, as the former depends on the offloading decision and the latter is determined by the migration policy. We put restrictions on the number of hops connecting any two edge servers via optical fiber which has a very fast transmission rate. Therefore, the migration time have less impact on the total processing delay of tasks. Given the offloading and migration decisions, the problem P2 is directly related to the weight factor. Normally, the weight factor in the simulation is intended for a trade-off between computing time and computing cost. When the weight factor increases, the objective function pays more attention to time and thus the edge server allocates more computing resources to the task to reduce the computing time and increase the task completion rate. Moreover, the KTMO scheme has better performance in the total delay reduction, compared to other approaches. When a task spends more time in transmission, its urgency increases, taking a greater risk of failure within limited resource constraints. The transmission time under the KTMO scheme is shorter than that of other schemes, and hence, KTMO is able to achieve higher task completion rates.

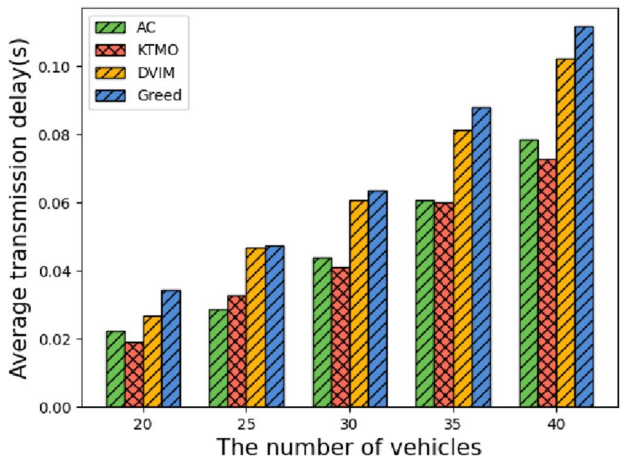


Fig. 6 The average transmission delay comparison under different numbers of vehicles

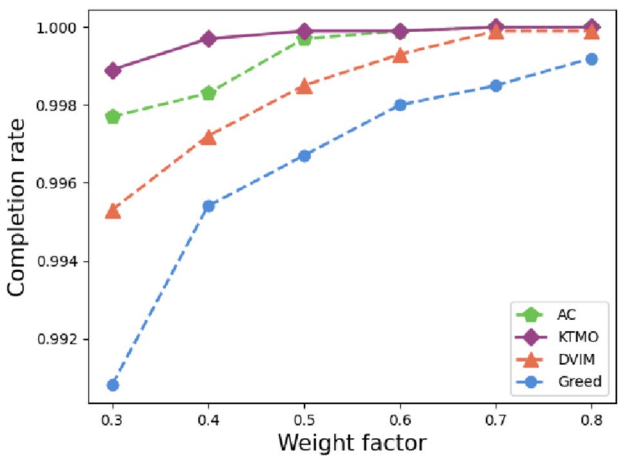


Fig. 7 Task completion rate comparison with different weights

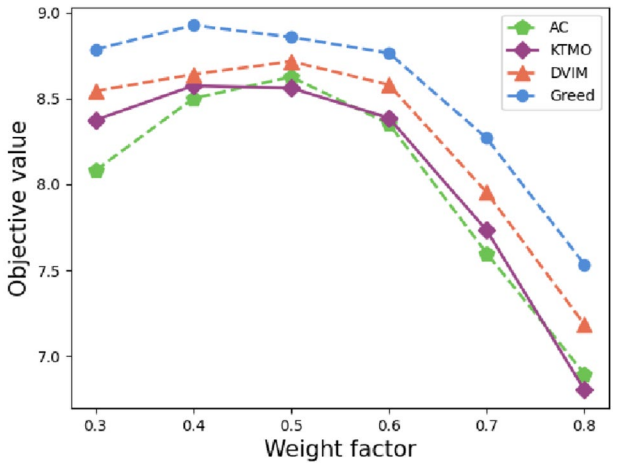


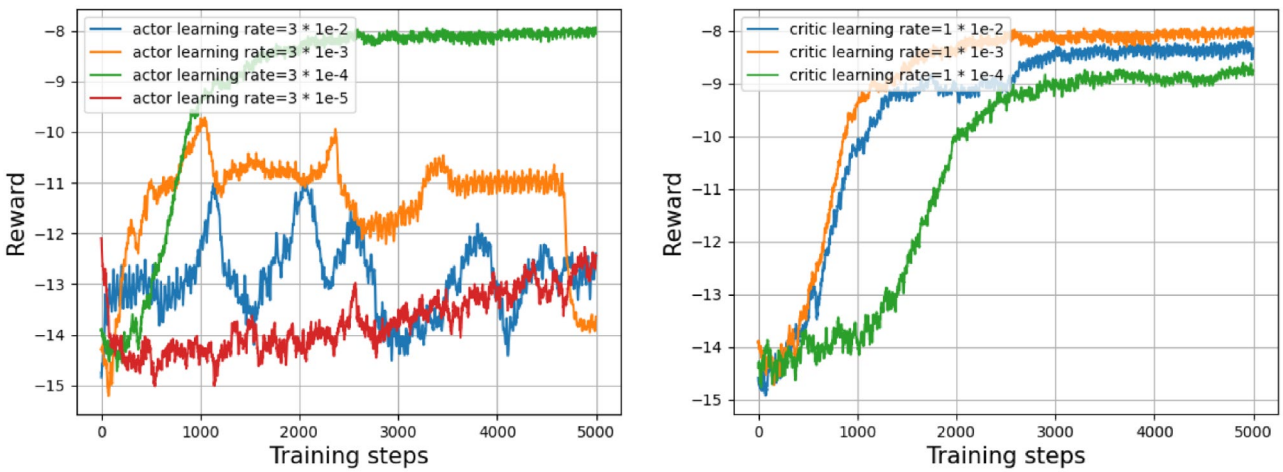
Fig. 8 Comparison of the objective value with the weight factor under four different offloading schemes

Figure 8 demonstrates the clear relationship between the value of objective function and weight factor. When the weight factor increases, the objective function focuses more on time and the server allocates more computing resources to the task to reduce the computing time. If the decrease in time is less than the increase in cost, the value of the objective function will increase. Conversely, a downward trend can be observed. As you can see from Fig.4, both indicators are of the same magnitude and have similar vertical scales. Before the crossover point, the delay is greater than the cost, so the objective function rises as ω increases. And after the intersection point, the delay is less than the cost, so the objective function decreases.

In Fig. 9, We study the effect of the learning rate on the convergence performance of the AC model. In Fig. 9(a), we set the learning rate of the critic network to 0.001 and compare the performance of the AC model for the actor network at different learning rates. Similarly, in Fig. 9(b), we fix the learning rate of the actor network and observe the performance of the AC model by varying the learning rate of the critic network. It is observed that a larger learning rate makes the training curve oscillate constantly, while a smaller learning rate produces stable curve but converges too slowly. Considering the trade-off between stability and training speed, we set the learning rate to 3*1e-4 for the actor network and $1e-3$ for the critic network in the experiments.

In the resource allocation problem P2, computing time decreases as computing resources increase, while computing cost increases. When the unit price of computing resources is lower, more attention is paid to latency optimization and more computing resources are allocated to reduce computing latency. Therefore, transferring tasks from a server with a higher unit cost to a lower one can effectively reduce computing latency as shown in Fig. 10. Despite increased resource requirements, computing cost has shown a downward trend, as shown in Fig. 11. Our proposed KTMO scheme, which is slightly better in terms of transmission time, has only slightly less computation delay than AC, which is more intelligent in making decisions for each task.

Figure 12 shows the changes in average task completion latency for different number of tasks. As the number of tasks grows, the average task execution latency gradually increases. An increase in the number of vehicles reduces the bandwidth allocated to each task, resulting in a higher transmission delay. On the other hand, as the number of



(a) Actor (b) Critic

Fig. 9 Training processes comparison of with different learning rates

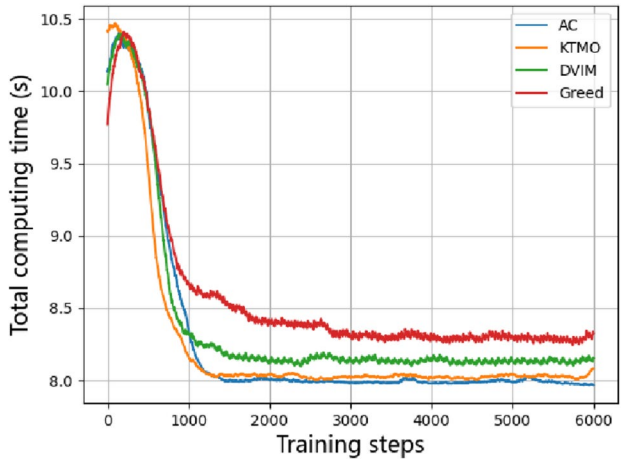


Fig. 10 Comparison of the computing time along the training steps

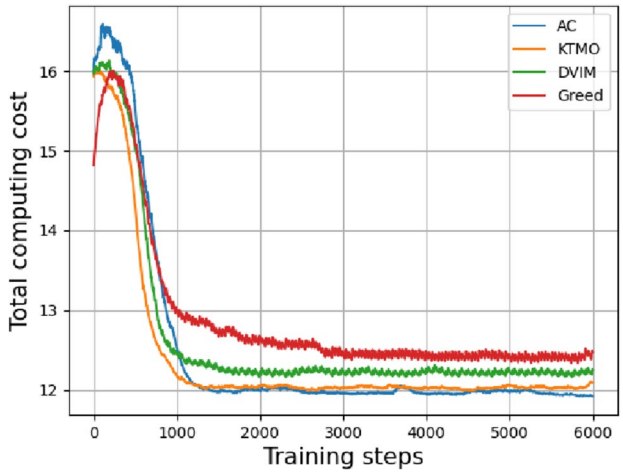


Fig. 11 Comparison of the computing cost along the training steps

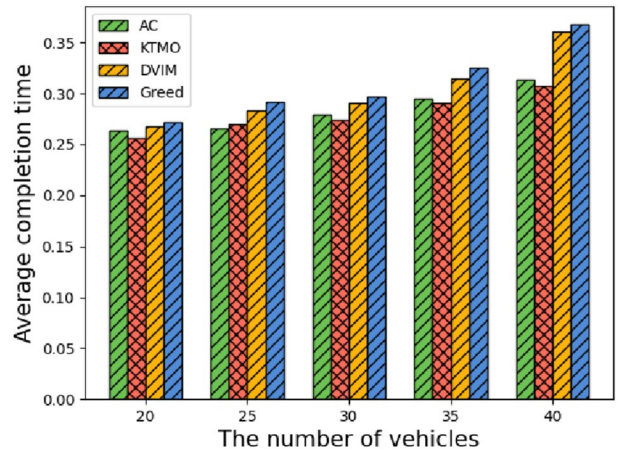


Fig. 12 Comparison of average completion delay with different numbers of vehicles

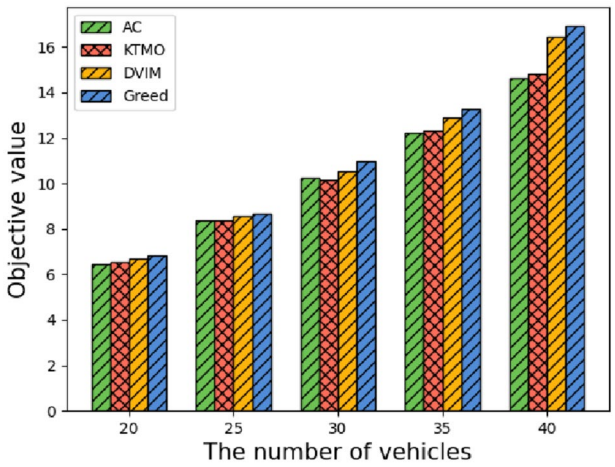


Fig. 13 Comparison of objective value with different numbers of vehicles

task increases, so does the demand for finite resources. Resources should be allocated more rationally to meet the demands of a greater number of tasks. Our proposed KTMO scheme achieves a lower execution delay by taking advantage of the transmission phase.

Figure 13 illustrates how the objective function value changes with different number of tasks. It is obvious that as the number of tasks increases, so does the value of the objective function. As mentioned earlier, with a growing number of tasks, the average bandwidth allocated to the vehicle diminishes and the interference increases, resulting in an increase in transmission delay and computing cost. Additionally, the more tasks, the more computing resources required, and at each time slot, the unit price of the resources increases dramatically, which also has a direct impact on the overall cost of the tasks. Generally, when the computing resources of the edge servers with lower prices are running out, tasks are forced to migrate to edge servers with relatively higher prices. This case can also increase the total cost. For these reasons, the value of the objective function increases as the number of tasks grows.

6 Conclusion

In this paper, we have mainly studied the offloading strategy and migration policy for VEC. For the task scheduling problem, we have designed a fair offloading scheme for tasks named KTMO based on weighted bipartite graph matching to balance the bandwidth resources allocated to each

task. Furthermore, considering the time-varying features of the computing resources, we modeled the task migration process as a MDP that determines the system state, action, and reward function by continuously exploring the environment to find the optimal strategy and reduce vehicle cost. Finally, we verified the performance of our proposed method through simulation experiments. The simulation results have demonstrated the advantages compared to other approaches. For the future works, we will focus on the fair caching of service in VEC.

Acknowledgements This work is supported by the National Natural Science Foundation of China under Grant Number 62071327, 62271486 and 62071470.

Data availability The data supporting the findings of this work can be obtained from the corresponding author upon reasonable request.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Tang, C., Chen, W., Zhu, C., et al.: When cache meets vehicular edge computing: architecture, key issues, and challenges. IEEE Wirel. Commun. **29**(4), 56–62 (2022). https://doi.org/10.1109/MWC.202.2100159

Liu, B., Jia, D., Wang, J., et al.: Cloud-assisted safety message dissemination in VANET–cellular heterogeneous wireless network. IEEE Syst. J. **11**(1), 128–139 (2017). https://doi.org/10.1109/JSYST.2015.2451156

Tang, C., Wei, X., Zhu, C., et al.: Towards smart parking based on fog computing. IEEE Access **6**, 70172–70185 (2018). https://doi.org/10.1109/ACCESS.2018.2880972

Zeng, F., Chen, Q., Meng, L., et al.: Volunteer assisted collaborative offloading and resource allocation in vehicular edge computing. IEEE Trans. Intell. Transp. Syst. **22**(6), 3247–3257 (2021). https://doi.org/10.1109/TITS.2020.2980422

Liu, S., Liu, L., Tang, J., et al.: Edge computing for autonomous driving: opportunities and challenges. Proc. IEEE **107**(8), 1697–1716 (2019). https://doi.org/10.1109/JPROC.2019.2915983

Wang, B., Wang, C., Huang, W., et al.: A survey and taxonomy on task offloading for edge-cloud computing. IEEE Access **8**, 186080–186101 (2020). https://doi.org/10.1109/ACCESS.2020.3029649

Tang, C., Zhu, C., Wu, H., et al.: Toward response time minimization considering energy consumption in caching-assisted vehicular edge computing. IEEE Internet Things J. **9**(7), 5051–5064 (2022). https://doi.org/10.1109/JIOT.2021.3108902

Han, S., Li, Y., Meng, W., et al.: Indoor localization with a single Wi-Fi access point based on OFDM-MIMO. IEEE Syst. J. **13**(1), 964–972 (2019). https://doi.org/10.1109/JSYST.2018.2823358

Qiao, G., Leng, S., Zhang, K., et al.: Collaborative task offloading in vehicular edge multi-access networks. IEEE Commun. Mag. **56**(8), 48–54 (2018). https://doi.org/10.1109/MCOM.2018.1701130

Tang, C., Wu, H.: Optimal computational resource pricing in vehicular edge computing: a Stackelberg game approach. J. Syst. Architect. **121**, 102331 (2021). https://doi.org/10.1016/j.sysarc.2021.102331

Munkres, J.: Algorithms for the assignment and transportation problems. J. Soc. Ind. Appl. Math. **5**(1), 32–38 (1957). https://doi.org/10.1137/0105003

Raza, S., Wang, S., Ahmed, M., et al.: A survey on vehicular edge computing: architecture, applications, technical issues, and future directions. Wirel. Commun. Mob. Comput. **2019**, 1–19 (2019). https://doi.org/10.1155/2019/3159762

Anawar, M.R., Wang, S., Azam Zia, M., et al.: Fog computing: an overview of big IoT data analytics. Wirel. Commun. Mob. Comput. **2018**, 1–22 (2018). https://doi.org/10.1155/2018/7157192

Hou, X., Li, Y., Chen, M., et al.: Vehicular fog computing: a viewpoint of vehicles as the infrastructures. IEEE Trans. Veh. Technol. **65**(6), 3860–3873 (2016). https://doi.org/10.1109/TVT.2016.2532863

Kim, Y., Kwak, J., Chong, S.: Dual-side optimization for cost-delay tradeoff in mobile edge computing. IEEE Trans. Veh. Technol. **67**(2), 1765–1781 (2018). https://doi.org/10.1109/TVT.2017.2762423

Liu, H., Zhao, H., Geng, L., et al.: A Distributed Dependency-Aware Offloading Scheme for Vehicular Edge Computing Based on Policy Gradient. In: 2021 8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2021 7th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), pp 176–181, https://doi.org/10.1109/CSCloud-EdgeCom52276.2021.00040 (2021)

Dai, Y., Xu, D., Maharjan, S., et al.: Joint load balancing and offloading in vehicular edge computing and networks. IEEE Internet Things J. **6**(3), 4377–4387 (2019). https://doi.org/10.1109/JIOT.2018.2876298

Zhang, J., Guo, H., Liu, J., et al.: Task offloading in vehicular edge computing networks: a load-balancing solution. IEEE Trans. Veh. Technol. **69**(2), 2092–2104 (2020). https://doi.org/10.1109/TVT.2019.2959410

Guo, H., Zhang, J., Liu, J.: FiWi-enhanced vehicular edge computing networks: collaborative task offloading. IEEE Veh. Technol. Mag. **14**(1), 45–53 (2019). https://doi.org/10.1109/MVT.2018.2879537

Yuan, H., Zhou, M.: Profit-maximized collaborative computation offloading and resource allocation in distributed cloud and edge computing systems. IEEE Trans. Autom. Sci. Eng. **18**(3), 1277–1287 (2021). https://doi.org/10.1109/TASE.2020.3000946

Sun, Y., Guo, X., Song, J., et al.: Adaptive learning-based task offloading for vehicular edge computing systems. IEEE Trans. Veh. Technol. **68**(4), 3061–3074 (2019). https://doi.org/10.1109/TVT.2019.2895593

Qi, Q., Wang, J., Ma, Z., et al.: Knowledge-driven service offloading decision for vehicular edge computing: a deep reinforcement learning approach. IEEE Trans. Veh. Technol. **68**(5), 4192–4203 (2019). https://doi.org/10.1109/TVT.2019.2894437

Taleb, T., Ksentini, A., Frangoudis, P.A.: Follow-me cloud: when cloud services follow mobile users. IEEE Trans. Cloud Comput. **7**(2), 369–382 (2019). https://doi.org/10.1109/TCC.2016.2525987

Moon, S., Park, J., Lim, Y.: Task migration based on reinforcement learning in vehicular edge computing. Wirel. Commun. Mob. Comput. **2021**, 1–10 (2021). https://doi.org/10.1155/2021/9929318

Huang, L., Bi, S., Zhang, Y.J.A.: Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. IEEE Trans. Mob. Comput. **19**(11), 2581–2593 (2020). https://doi.org/10.1109/TMC.2019.2928811

Liu, B., Jiang, X., He, X., et al.: A deep learning-based edge caching optimization method for cost-driven planning process over IIoT. J. Parallel Distrib. Comput. **168**, 80–89 (2022). https://doi.org/10.1016/j.jpdc.2022.06.007

Ning, Z., Dong, P., Wang, X., et al.: Deep reinforcement learning for vehicular edge computing an intelligent offloading system. ACM Trans. Intell. Syst. Technol. **10**(6), 1–24 (2019). https://doi.org/10.1145/3317572

Yuan, Q., Li, J., Zhou, H., et al.: A joint service migration and mobility optimization approach for vehicular edge computing. IEEE Trans.

Veh. Technol. **69**(8), 9041–9052 (2020). https://doi.org/10.1109/TVT.2020.2999617

Wang, Z., Ng, D.W.K., Wong, V.W.S., et al.: Robust beamforming design in C-RAN with sigmoidal utility and capacity-limited backhaul. IEEE Trans. Wirel. Commun. **16**(9), 5583–5598 (2017). https://doi.org/10.1109/TWC.2017.2712645