Peer-to-Peer Networking and Applications
https://doi.org/10.1007/s12083-021-01252-w

Joint optimization of task caching and computation offloading in vehicular edge computing

Chaogang Tang[1] · Huaming Wu[2]

Received: 19 July 2021 / Accepted: 30 September 2021

Abstract

The recent surge in the number of connected vehicles and vehicular applications really benefits citizens. Various vehicular applications are developed to cater for the increasingly sophisticated demands of drivers. Against this background, vehicular edge computing (VEC) is put forward as a promising solution to meet the strict latency requirement of these vehicular applications, by undertaking the computation offloaded from the nearby vehicles. Furthermore, task-oriented caching strategies are also applied to VEC for performance improvement. However, challenges faced by caching-enabled VEC still need to be addressed. For example, many factors can restrict the application of task caching in VEC, which usually include limited caching capability, extra energy consumption incurred by task caching, caching results delivery and so on. To overcome these issues, we propose a general caching-enabled VEC scheme and aim to jointly optimize the task caching and computation offloading in the VEC system. Moreover, we consider not only the response latency reduction benefitting from task caching, but also the energy consumption incurred by task caching. In particular, we strive to minimize the weighted sum of the service time and energy consumption for all the offloading requests in VEC. Due to the exponential time taken to obtain the optimal value, we in this paper propose a genetic algorithm-based task caching and computation offloading strategy. Extensive simulation has been carried out to investigate its efficiency compared to the benchmark algorithms. The simulation results reveal that the proposed strategy outperforms other approaches including the greedy approach and the random approach.

Keywords Vehicular edge computing · Task caching · Optimization · Computation offloading · Genetic algorithm

1 Introduction

The rapid development of intelligent transportation systems has brought considerable benefits for citizens, e.g., the recent surge in the number of connected vehicles and vehicular applications. Various vehicular applications are developed to cater for the increasingly sophisticated demands of drivers, in addition to the basic demands for driving safety [1]. As such, smart vehicles are taking on more and more responsibility. For instance, with integrated communication and computing modules, not only are they responsible for communicating with each other in case of car accidents, but also they perform vehicular applications and tasks to satisfy non-functional requirements of drivers. By non-functional requirements, we mean those requirements imposed from the perspective of social communication and infotainment service provisioning. The ultra-low response latency becomes one of the most urgent needs for these vehicular applications such as virtual reality games and in-car cloud games. However, cloud computing paradigm falls short of such a goal, since computation offloading to the cloud center via the core networks incurs unpredictable transmission delay.

In this context, vehicular edge computing (VEC) is put forward as a promising solution to meet the strict latency requirement of these applications [2–4]. Specifically, it extends the cloud-like characteristics to the logical edge of the networks such as road side units (RSU), and thus provides the computing resources in close proximity to the vehicles. Task offloading using the fronthaul links instead of the backhaul links can drastically reduce the response latency.

✉ Huaming Wu
whming@tju.edu.cn

Chaogang Tang
cgtang@cumt.edu.cn

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

2 The Center for Applied Mathematics, Tianjin University, Tianjin 300072, China

Published online: 19 October 2021

The response latency can be further reduced by applying task-oriented caching strategies in VEC systems [5]. Note that task-oriented caching (TOC) is not a fancy term for information-centric caching (ICC) [6]. In this paper, TOC refers to the task execution result caching for the future reuse in VEC systems. Tasks can be repeatedly offloaded by vehicles with similar behaviors. For instance, drivers with similar backgrounds and characters display similar driving patterns to a certain extent. With the aid of TOC, the execution result can be reused for further reducing the response latency, when vehicular applications are offloaded and executed at the edge. The difference between TOC and ICC lies in that TOC can be implemented at different granularities. The tasks offloaded to the edge usually consist of processing codes and user parameters. A fine-granularity task caching usually reserves both the processing codes and user parameters, while a coarse-granularity task caching only reserves the execution result. The fine-granularity task caching considers the different preferences with regards to (w.r.t.) the input parameters. In spite of such an advantage, it incurs more storage overheads and energy consumptions in comparison to the coarse-granularity task caching [7]. In this paper, TOC only focuses on the coarse-granularity task caching in VEC.

Despite enticing advantages by applying TOC to VEC systems, there are still several challenges that need to be addressed. First, the limited caching capability of the edge makes it prohibitively costly to cache all the tasks. Second, the small coverage of RSU and high mobility of vehicles may increase the data loss during task offloading or result return, which definitely degrades the quality of experience (QoE). Third, the performance of caching enabled VEC systems depends on not only which tasks should be cached, but also how the caching results are delivered, especially when cooperative caching is enabled in VEC with geographically dispersed edge servers. Fourth, TOC not only consumes the storage resources, but also incurs certain energy consumption. For example, virtual machine environments may hold for a while to facilitate task performing in a fine-granularity task caching scenario. Therefore, there should be a trade-off between response latency and energy consumption. Last but not least, most of existing works have not considered the caching result delivering in VEC. To simplify the model, they just assume that the time taken to retrieve the caching results in VEC can be negligible. Such an assumption, however, does not always hold in vehicular applications characterized by strong interactions and a large size of execution results.

To overcome these issues, in this paper we propose to jointly optimize the task caching and computation offloading in the VEC system. To facilitate computation calculation, we assume that RSUs can perform the tasks in a cooperative way. Furthermore, we consider not only the response latency reduction brought by task caching, but also the energy consumption incurred by task caching. Specifically, we strive to minimize the weighted sum of the service time and energy consumption for all the offloading requests. The major contributions are given as below:

- A general model is proposed in this paper with the aim to jointly optimize the task caching and computation offloading in VEC system, which takes into account not only the benefits of task caching such as response latency reduction but also the incurred energy consumption when computation is offloaded and undertaken at RSUs.
- We mathematically formulate the joint optimization of task caching and computation offloading in this paper. Owning to the exponential time taken to obtain the optimal solution, we propose a genetic algorithm based strategy to obtain the proximate optimum solution.
- A series of experiments have been carried out to evaluate our approach in comparison to the benchmark algorithms. The simulation results have revealed that our approach outperforms other approaches in terms of efficiency and effectiveness.

The rest of the paper is organized as follows. Some related works are reviewed in Sect. 2. The system model comes in Sect. 3 which introduces the three different cases of task caching and then formulates our optimization problem. A genetic algorithm based strategy is put forward in Sect. 4 for jointly optimizing the task caching and computation offloading in VEC. The simulation results are reported in Sect. 5, followed by the conclusion in Sect. 6.

2 Related works

The explosive growth in the number of vehicular applications has posed great pressure on the limited computing capabilities of vehicle-loaded computers, which stimulates the rapid development of VEC. As a new computing paradigm, VEC can undertake all or part of computation offloaded from vehicles, in hope to satisfy multiple purposes from the drivers and vehicles. Such purposes include response latency reduction, energy consumption saving and QoE improvement. In this section, we will review some related works revolved around VEC for the purpose of performance optimization.

2.1 Optimization objectives revolved around VEC

Unlike cloud computing where there are sufficient computing and storage resources, VEC pushes the computing resources to the edge of networks such as RSU at the expense of limited computing capabilities owing to the sporadic computing resources. As a result, it is necessary to exploit all the dispersive computing resources of vehicles in the vicinity, which however is still challenging and an important research direction. Authors in [8] propose the notion of Virtual Edge that is a collaborative VEC framework where vehicles with idle computing resources can serve as the virtual edge to assist computation. Furthermore, an algorithm for virtual edge formation is put forward, which pays attention to not only the idle computing resources but also the state of virtual edge.

One major benefit of applying VEC to computation offloading is to cater for the increasingly sophisticated demands of vehicular applications. Note that the wireless coverage of RSUs is usually limited and the vehicles are characterized by high mobility, thus making it pretty hard to maintain high communication quality all the time. Authors in [9] put forward an energy aware task offloading strategy for VEC, which balances the response latency and energy consumption when performing the computational tasks.

With the advent of the sixth generation (6G) vehicle-to-everything (V2X) applications, it is necessary to construct three-dimensional (3D) and ubiquitous networking coverage for the time critical task offloading. An intelligent VEC system assisted by unmanned aerial vehicle is proposed in [10], so as to meet 6G-V2X requirements including 3D and adaptive service coverage.

Considering the strict delay requirement for end-user applications, authors in [11] propose a routing scheme based on collaborative learning in VEC, which aims to proactively find routes using a reinforcement learning algorithm. They have proven that their strategy can achieve better performance in comparison to the existing works.

Although the response delay of vehicular applications can be reduced with the help of edge computing, it is very difficult to ensure the communication quality owing to the building obstruction or lack of infrastructure. In view of this, authors [12] resort to UAVs for addressing such concerns. In particular, they propose a computation offloading optimization framework where both SDN technology and UAV are introduced for optimizing the cost of vehicular tasks.

When considering the self-driving vehicles, the passenger profiles including sophisticated infotainment applications are supposed to be constructed. Such information should be processed in real time. Therefore, a streamlined edge computing infrastructure is needed where computationally intensive workloads are offloaded to a nearby VEC infrastructure. To realize the purpose, authors in [13] propose a two-stage machine learning-based vehicular edge orchestrator. Such an orchestrator considers both task completion success and the service time at the same time. Extensive simulation is carried out to evaluate the performance of their strategy.

On another hand, the security issues are still challenging VEC. One of the main reasons is that the incentive mechanism is insufficient in the vehicular ad-hoc networks which is an untrusted and opaque environment. To cope with such issues, a consortium blockchain is proposed which aims to realize secure resource sharing in VEC [14]. Specifically, a contract-based incentive mechanism is leveraged to encourage vehicles to contribute idle computation resources.

The intelligent VEC (IVEC) infrastructure has attracted extensive attention recently, which benefits from the rapid development of AI algorithms recently. Despite the benefits, IVEC is vulnerable to fake computation feedback, unfair or biased resource allocation. One of the main causes is the centralized governance that is transparent to the user. Therefore, authors in [15] put forward a blockchain-based decentralized architecture to improve the resource management in terms of transparency in IVEC. They also try to solve the load balancing issue and further design a secure IVEC federation model for workloads balancing.

2.2 Caching aided performance improvement in VEC

We also notice that interest is aroused about application of caching strategies to the task offloading in VEC system [5, 16, 17].

Authors in [18] put forward an architecture for content caching in VEC. This architecture is task oriented and at least three tasks can be identified, i.e., they can realize popularity prediction of contents, content placement and retrieval from the cache, via the artificial intelligence technologies. Furthermore, future research opportunities in areas are also discussed in depth.

Authors in [19] put forward a cooperative edge caching framework. They try to exploit the cooperations among base station, RSU and connected vehicles, with the purpose of jointly optimizing the content placement and content delivery in VEC. Specifically, they model such an optimization problem as a double time-scale Markov decision process and solve it by a nature-inspired method with a low computation complexity.

Authors in [20] propose a joint optimization for communication, caching and computing strategy, with the aim to realize the cost efficiency in VEC. Specifically, an artificial intelligence-based multi-timescale framework is designed and they combine the particle swarm optimization with deep reinforcement learning to achieve their goals.

Considering the fact that a vast number of parked vehicles may have unexploited computing and storage resources, authors in [21] propose a caching strategy for VEC where vehicles in the parking lots are made full used, to serve as the content providers to cache popular contents in a collaborative way. They aim to minimize the average response delay by an efficient content placement algorithm. In our previous work [22], we pay attention to a caching enabled task offloading in mobile edge computing, and try to optimize the weighted sum of energy consumption and response latency by an alternate optimization algorithm.

In comparison to these works, we in this paper focus on a more general VEC system where multiple RSUs work together to undertake the computation offloaded from vehicles in the vicinity and the task caching is further enabled for the performance improvement in VEC. In the meanwhile, we consider both the response latency reduction brought by task caching, but also the energy consumption incurred by task caching.

3 System model

A system model considered in this paper is denoted in Fig. 1, which consists of multiple edge servers with limited wireless coverage. The edge servers are respectively deployed at geographically dispersed RSUs. Thus, the computing resources can be provisioned at RSUs. Owing to the limited wireless coverage, each RSU can only serve the vehicles within its own coverage. RSUs are connected with each other in a wired way (e.g., fiber-optic networks). In the meanwhile, they also connect to the remote cloud center via the backhaul links. On another hand, each edge is equipped with a caching unit for caching the task execution results.

Let $\mathcal{R} = \{R_1, \cdots, R_M\}$ denote the set of RSUs (edge servers) and $\mathcal{V}_i = \{v_i^1, \cdots, v_i^{n_i}\}$ the set of vehicles within the coverage of R_i in which n_i is the number of vehicles that R_i can serve. Assume that the computations that need to be offloaded come from the set of K tasks, indexed by $\mathcal{T} = \{t_1, t_2, \cdots, t_K\}$. Each task t_k can be denoted by $t_k = (d_k, s_k, r_k)$ where d_k denotes the input size of t_k which usually consists of the processing codes and parameters, s_k the needed number of CPU cycles to accomplish t_k, and r_k the data size of execution result. Let c_i denote the caching capability of R_i, and hence the total amount of cached results at R_i should be no larger than c_i. Let $Req_{i,j}^k$ denote a computation offloading request for task t_k from the jth vehicle in the serving area of R_i. Define $\boldsymbol{\phi} \triangleq \{\boldsymbol{\phi_1}, \cdots, \boldsymbol{\phi_M}\}$ as a decision matrix, where $\boldsymbol{\phi_m} = \{\phi_m^1, \cdots, \phi_m^K\}$ and $\phi_m^k \in \{0, 1\}$. The binary variable ϕ_m^k represents whether task t_k is cached at R_m. Specifically, ϕ_m^k is equal to 1 when task t_k is cached at R_m, and 0, otherwise.

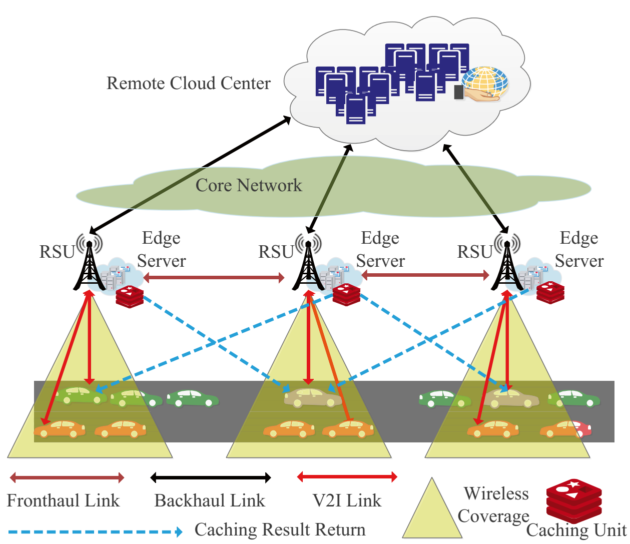


Fig. 1 An application scenario considered in this paper

The computation offloading procedure in caching enabled VEC system can be described as below. A vehicle v_i^j sends $Req_{i,j}^k$ together with the beacon information to R_i. R_i checks its own caching unit. If the task t_k is cached in the caching unit, R_i will directly return the execution result to v_i^j. If t_k is not cached at R_i, R_i will communicate with other RSUs in $\mathcal{R}$ to check whether they have cached t_k. If t_k is cached, R_i retrieves the caching result from the edge server which has the least transmission delay. If none of the edge servers in $\mathcal{R}$ cache t_k, v_i^j offloads the computation to R_i for execution. It shall be noted that in reality the bandwidth among RSUs is much higher than between RSU and vehicles. Thus, in comparison to task offloading, it takes much less time to retrieve the caching result from other RSUs even through multiple hops. In this way, computation offloading in caching assisted VEC can reduce the service delay for vehicles, which can mitigate the high demands for computing resources and further improve QoE.

3.1 Communication and computation model

When a request $Req_{i,j}^k$ arrives, there are three cases for R_i to consider, given as below.

3.1.1 Case 1: Task t_k cached at R_i

The first case is that the task t_k has been cached at R_i. Namely, $\phi_i^k = 1$. Then, there is no need for v_i^j to offload the computation and thus the offloading time, denoted by $t_{i,j}^{off,k}$, is zero. Similarly, the execution time of t_k, denoted by $t_{i,j}^{exe,k}$, is also zero. The return time, denoted by $t_{i,j}^{rtn,k}$, can be calculated as:

$$t_{i,j}^{rtn,k} = \frac{r_k}{b_{i,j}}, \tag{1}$$

where $b_{i,j}$ is the transmission rate from R_i to v_i^j and expressed as:

$$b_{i,j} = w \log_2(1 + \frac{P_i}{\sigma^2}), \tag{2}$$

where w is the bandwidth of the wireless channel, P_i is the transmission power of R_i, and σ^2 is the noise power.

Accordingly, the service time l_{case1}, which includes the offloading time, the execution time and the return time, can be calculated as:

$$l_{case1} = \phi_i^k (t_{i,j}^{off,k} + t_{i,j}^{exe,k} + t_{i,j}^{rtn,k}) = \phi_i^k \frac{r_k}{b_{i,j}}. \tag{3}$$

On the other hand, the energy consumption in the first case denoted by e_{case1} should also be considered as follows. In this paper, the energy consumption usually includes four parts, i.e., the energy consumption for offloading the computation to the edge denoted by $e_{i,j}^{off,k}$, energy consumption for task execution at the edge denoted by $e_{i,j}^{exe,k}$, energy consumption for results return denoted by $e_{i,j}^{rtn,k}$, and the energy consumption for task caching denoted by $e_{i,j}^{c,k}$. In terms of the first case where the requested computation has been cached at R_i, the energy consumption e_{case1} only consists of $e_{i,j}^{rtn,k}$ and $e_{i,j}^{c,k}$. $e_{i,j}^{rtn,k}$ can be calculated as follows.

$$e_{i,j}^{rtn,k} = t_{i,j}^{rtn,k} P_i. \tag{4}$$

The energy consumption for task caching $e_{i,j}^{c,k}$ can be given as:

$$e_{i,j}^{c,k} = \gamma_i r_k, \tag{5}$$

where γ_i is the static power consumption for one unit data/task caching which only depends on R_i. $e_{i,j}^{c,k}$ is independent of vehicles which send the offloading requests. Thus, the total energy consumption e_{case1} can be given as:

$$e_{case1} = \phi_i^k (e_{i,j}^{rtn,k} + e_{i,j}^{c,k}) = \phi_i^k (t_{i,j}^{rtn,k} P_i + \gamma_i r_k). \tag{6}$$

The weighted sum of response latency and energy consumption denoted by le_{case1} is given as:

$$le_{case1} = w_1 l_{case1} + w_2 e_{case1}, \tag{7}$$

where $w_1 (w_2) \in [0, 1]$ and $w_1 + w_2 = 1$. They are used to strike a balance between response latency and energy consumption.

3.1.2 Case 2: Task t_k cached at other RSUs

The second case is that R_i does not cache t_k, but at least one of other RSUs in $\mathcal{R}$ caches t_k. In this case, $\phi_i^k = 0$. Let $\mathcal{R}_{-i}$ denote the set of RSUs except R_i. Define ϕ_{-i}^k as:

$$\phi_{-i}^k = \prod_{j \in \{1, \cdots, m\} \bigwedge j \neq i} (1 - \phi_j^k) = 0. \tag{8}$$

The above equation can guarantee that there exists at least one RSU, say $R_m (\in \mathcal{R}_{-i})$, which caches t_k. According to the computation offloading procedure mentioned earlier, R_i will retrieve the caching result from R_m and then deliver it to v_i^j. Accordingly, the service time l_{case2} can be expressed as:

$$l_{case2} = (1 - \phi_i^k)(1 - \phi_{-i}^k)(\frac{r_k}{b_{i,j}} + \frac{r_k}{b_R}), \tag{9}$$

where $\frac{r_k}{b_R}$ denotes the time taken for R_i to retrieve the caching result from R_m and b_R is the transmission rate between R_m and R_i. Since RSUs in $\mathcal{R}$ are connected with each other in the ultra-fast fiber-optic networks, we assume that transmission rate between two RSUs are the same for simplicity. On the other hand, in addition to $e_{i,j}^{rtn,k}$ and $e_{i,j}^{c,k}$, the energy consumption in the second case also includes the energy consumption for delivering the caching result from the source RSU R_m to the destination RSU R_i, denoted by $e_{i,j}^{dlv,k}$ as follows:

$$e_{i,j}^{dlv,k} = \frac{r_k}{b_R} P_m, \tag{10}$$

where P_m is the transmission power of R_m. Accordingly, the total energy consumption in the second case can be expressed as:

$$e_{case2} = (1 - \phi_i^k)(1 - \phi_{-i}^k)(e_{i,j}^{rtn,k} + e_{i,j}^{c,k} + e_{i,j}^{dlv,k}). \tag{11}$$

Thus, the weighted sum of response latency and energy consumption can be calculated as:

$$le_{case2} = w_1 l_{case2} + w_2 e_{case2}. \tag{12}$$

3.1.3 Case 3: Task t_k not cached at $\mathcal{R}$

The last case is that none of RSUs in $\mathcal{R}$ has cached the task t_k. In this case, $\phi_i^k = 0$ and $\phi_{-i}^k = 1$. v_i^j needs to offload t_k to R_i for execution. Thus, the service time actually includes the offloading time, execution time and return time. The offloading time can be given as:

$$t_{i,j}^{off,k} = \frac{d_k}{b_{j,i}}, \tag{13}$$

where $b_{j,i}$ is the transmission rate from v_i^j to R_i and expressed as:

$$b_{j,i} = w \log_2 (1 + \frac{P_i^j}{\sigma^2}). \tag{14}$$

where w is the bandwidth of the wireless channel, P_i^j is the transmission power of v_i^j, and σ^2 is the noise power. The execution time $t_{i,j}^{exe,k}$ is given as:

$$t_{i,j}^{exe,k} = \frac{s_k}{\rho_i^j}, \tag{15}$$

where ρ_i^j is the processing capability of v_i^j. The return time is given as the same as Eq. (1). Therefore, the service time l_{case3} can be expressed as:

$$\begin{aligned} l_{case3} &= \phi_{-i}^k (1 - \phi_i^k)(t_{i,j}^{off,k} + t_{i,j}^{exe,k} + t_{i,j}^{rtn,k}) \\ &= \phi_{-i}^k (1 - \phi_i^k)(\frac{d_k}{b_{j,i}} + \frac{s_k}{\rho_i^j} + \frac{r_k}{b_{i,j}}). \end{aligned} \tag{16}$$

The energy consumption in this case actually includes $e_{i,j}^{off,k}$, $e_{i,j}^{exe,k}$, and $e_{i,j}^{rtn,k}$.

$$e_{i,j}^{off,k} = t_{i,j}^{off,k} P_i^j. \tag{17}$$

And $e_{i,j}^{exe,k}$ is given as:

$$e_{i,j}^{exe,k} = \kappa_i s_k \rho_i^2, \tag{18}$$

where κ_i is the effective capacitance coefficient of the CPU chip of R_i, and ρ_i is the processing frequency of R_i. $e_{i,j}^{rtn,k}$ is the same as that in the first case. Accordingly, the total energy consumption is given as:

$$e_{case3} = \phi_{-i}^k (1-\phi_i^k)(e_{i,j}^{off,k} + e_{i,j}^{exe,k} + e_{i,j}^{rtn,k}). \tag{19}$$

The weighted sum of response latency and energy consumption in this case is

$$le_{case3} = w_1 l_{case3} + w_2 e_{case3}. \tag{20}$$

3.2 Problem formulation

In this paper, we aim to optimize the weighted sum of service time and energy consumption in caching assisted VEC system for the offloading requests, as defined in Eq. (21). Usually, the more the number of tasks that are cached, the less the service time. However, the more the number of tasks that are cached, the more the energy consumption in the VEC system. Therefore, the minimization of both of them actually seeks a balance between response latency reduction and energy consumption saving. In particular, we take into account not only the application of TOC, but also the horizontal cooperation among RSUs in $\mathcal{R}$. The rationale behind this is that the tasks cached at other RSUs could be very beneficial to the current RSU with the task offloading request. Based on these descriptions, the optimization problem in this paper can be formulated as below:

$$\begin{aligned} O(\boldsymbol{\phi}) &= \sum_{k=1}^{K}\sum_{i=1}^{M}\sum_{j=1}^{n_i}[le_{case1} + le_{case2} + le_{case3}] \\ &= \sum_{k=1}^{K}\sum_{i=1}^{M}\sum_{j=1}^{n_i} w_1[(\frac{r_k}{b_{i,j}} + \frac{r_k}{b_R}) - \phi_i^k \frac{r_k}{b_R} \\ &+ \phi_{-i}^k(\frac{d_k}{b_{j,i}} + \frac{s_k}{\rho_i^j} - \frac{r_k}{b_R}) - \phi_i^k\phi_{-i}^k(\frac{d_k}{b_{j,i}} + \frac{s_k}{\rho_i^j} - \frac{r_k}{b_R})] \\ &+ w_2[\phi_i^k(e_{i,j}^{rtn,k} + e_{i,j}^{c,k}) + (1-\phi_i^k)(e_{i,j}^{rtn,k} + e_{i,j}^{c,k} + e_{i,j}^{dlv,k} \\ &+ \phi_{-i}^k(e_{i,j}^{off,k} + e_{i,j}^{exe,k} - e_{i,j}^{c,k} - e_{i,j}^{dlv,k}))]. \end{aligned} \tag{21}$$

$$(\mathcal{P}1) \qquad \min_{\boldsymbol{\phi}} O(\boldsymbol{\phi}), \tag{22}$$

$$s.t. \qquad \sum_{k=1}^{K} \phi_i^k r_k \le c_i, \quad \forall i \in \{1,\cdots,M\}, \tag{23}$$

$$\sum_{i=1}^{m} \phi_i^k \le 1, \quad \forall k \in \{1,\cdots,K\}, \tag{24}$$

$$w_1 + w_2 = 1, \quad w_1, w_2 \in [0,1], \tag{25}$$

$$\rho_{i,min} \le \rho_i \le \rho_{i,max}, \quad \forall i \in \{1,\cdots,M\}, \tag{26}$$

$$\rho_i^{j,min} \le \rho_i^j \le \rho_i^{j,max}, \quad \forall i \in \{1,\cdots,M\}, \quad j \in \{1,\cdots,n_i\}, \tag{27}$$

$$P_{i,min} \le P_i \le P_{i,max}, \quad \forall i \in \{1,\cdots,M\}, \tag{28}$$

$$P_i^{j,min} \le P_i^j \le P_i^{j,max}, \quad \forall i \in \{1,\cdots,M\}, \quad j \in \{1,\cdots,n_i\}, \tag{29}$$

$$\phi_i^k, \phi_{-i}^k \in \{0,1\}, \quad \forall i \in \{1,\cdots,M\}, \quad \forall k \in \{1,\cdots,K\}, \tag{30}$$

$$\phi_{-i}^k \in \{0,1\}, \quad \forall i \in \{1,\cdots,M\}, \quad \forall k \in \{1,\cdots,K\}, \tag{31}$$

where the constraint (23) guarantees that the total amount of cached tasks at each RSU in $\mathcal{R}$ should not exceed its caching capability. As mentioned earlier, owing to the ubiquitous connections among RSUs in fiber-optic networks, the transmission rate among them is assumed to be b_R. Such an assumption implies that for the RSU with the offloading request, other RSUs are equally important to it in the sense that any one of them caching the requested task can make an equal contribution to it. As a result, for each task, say t_i in $\mathcal{T}$, it is actually unnecessary to repeatedly cache it in $\mathcal{R}$. Thus, we use the constraint (24) to guarantee that each task can be cached at most once. Constraints (26)–(27) denote that the processing frequencies of vehicles and RSUs are adjusted for the sake of energy consumption saving. Similarly, the constraints (28)–(29) mean that the transmission power of vehicles and RSUs are also adjusted. The constraint (30) guarantees that the decision variable is binary in this paper.

Intuitively, problem $\mathcal{P}1$ aims to optimize both the response latency and energy consumption by placing the cached task results at RSUs, on the condition that the sets of tasks cached at different RSUs are disjoint. However, it is different from the set partitioning since the set partitioning requires that the union of these subsets should be the whole set. Whereas, in our problem, the union does not have to be it, owing to the fact that some tasks can be cached by none of these RSUs. The simplest way to solve this problem is to enumerate all the possible solutions over the searching space, which however is prohibitively costly due to the exponential amount of time.

4 Algorithm design

Let $J(\boldsymbol{\phi})$ denote the objective value for a single offloading request $Req_{i,j}^k$, which can be defined as:

$$
\begin{aligned}
J(\boldsymbol{\phi}) =& w_1[(\frac{r_k}{b_{i,j}} + \frac{r_k}{b_R}) - \phi_i^k \frac{r_k}{b_R} + \phi_{-i}^k(\frac{d_k}{b_{j,i}} + \frac{s_k}{\rho_i^j} - \frac{r_k}{b_R}) \\
& - \phi_i^k \phi_{-i}^k(\frac{d_k}{b_{j,i}} + \frac{s_k}{\rho_i^j} - \frac{r_k}{b_R})] + w_2[\phi_i^k(e_{i,j}^{rtn,k} + e_{i,j}^{c,k}) \\
& + (1-\phi_i^k)(e_{i,j}^{rtn,k} + e_{i,j}^{c,k} + e_{i,j}^{dlv,k} + \phi_{-i}^k(e_{i,j}^{off,k} \\
& + e_{i,j}^{exe,k} - e_{i,j}^{c,k} - e_{i,j}^{dlv,k}))].
\end{aligned} \tag{32}
$$

We can rewrite $J(\boldsymbol{\phi})$ as:

$$
\begin{aligned}
J(\boldsymbol{\phi}) =& [w_1(\frac{r_k}{b_{i,j}} + \frac{r_k}{b_R})) + w_2(e_{i,j}^{rtn,k} + e_{i,j}^{c,k} + e_{i,j}^{dlv,k})] \\
& + \phi_i^k[w_2 e_{i,j}^{dlv,k} - w_1 \frac{r_k}{b_R}] + \phi_{-i}^k[w_1(\frac{d_k}{b_{j,i}} + \frac{s_k}{\rho_i^j} - \frac{r_k}{b_R}) \\
& + w_2(e_{i,j}^{off,k} + e_{i,j}^{exe,k} - e_{i,j}^{c,k} - e_{i,j}^{dlv,k})] \\
& - \phi_i^k \phi_{-i}^k[w_1(\frac{d_k}{b_{j,i}} + \frac{s_k}{\rho_i^j} - \frac{r_k}{b_R}) \\
& + w_2(e_{i,j}^{off,k} + e_{i,j}^{exe,k} - e_{i,j}^{c,k} - e_{i,j}^{dlv,k})].
\end{aligned} \tag{33}
$$

From the above equation we can observe that for the offloading request $Req_{i,j}^k$, the minimization of $J(\boldsymbol{\phi})$ depends upon not only the caching decision of task t_k at R_i, but also that at other RSUs in $\mathcal{R}_{-i}$. To efficiently tackle this issue, a joint optimization of task caching and computation offloading in this paper can be decomposed into two phases. First, an initial task caching decision $\boldsymbol{\phi}$ is made. Then based on the current task caching decision, $J(\boldsymbol{\phi})$ can be calculated, and moreover the corresponding value of $O(\boldsymbol{\phi})$ can be obtained.

It is worthwhile mentioning that the value of $O(\boldsymbol{\phi})$ is unique given the caching decision $\boldsymbol{\phi}$. In the next, we try to replace the current decision with a newly resulting caching decision as long as the new value of $O(\boldsymbol{\phi})$ is better than the original one. The procedure can be repeated until a convergence condition is achieved, e.g., the value of $O(\boldsymbol{\phi})$ does not decrease any more. From the description, we can see that the iteration-based algorithms can be adopted to solve our problem in this paper. In view of this, we propose a genetic algorithm (GA) to jointly optimize the caching decision and computation in VEC. Although GA is sometimes time consuming when the searching space is huge, it is especially suited for our optimization problem in this paper. This is because the searching space is not very large when the solution is encoded, due to the constraints such as (24).

Lemma 1 *Given the task caching decision $\boldsymbol{\phi}$, the time taken for the offloading requests to seek the optimal value of $O(\boldsymbol{\phi})$ is $O(TM\sum_{i=1}^{M} n_i)$ in the worst case.*

Proof Given the task caching profile $\boldsymbol{\phi}$, the offloading request $Req_{i,j}^k$ can be processed based on the aforementioned three cases. Obviously, the best case is that R_i has cached the task t_k, which will take the time of $O(1)$ to calculate $J(\boldsymbol{\phi})$. For the other two cases, i.e., R_i has not cached the task t_k, each RSU in $\mathcal{R}_{-i}$ should be checked to see whether there is one RSU which caches t_k. As a result, the time complexity of RSU checking is $O(M)$ in the worst case. There are M RSUs with each serving $n_i (i = 1, \cdots, M)$ vehicles. Therefore, to find the optimal value of $O(\boldsymbol{\phi})$ needs the time of $O(TM\sum_{i=1}^{M} n_i)$ in the worst case. □

Lemma (1) reveals that given the task caching profile $\boldsymbol{\phi}$, the optimal value of $O(\boldsymbol{\phi})$ can be obtained almost in real time. Therefore, we utilize GA to iteratively calculate the value of $O(\boldsymbol{\phi})$ by repeatedly constructing the caching decision. GA has demonstrated powerful searching capability over the solution domain. As a representative of population-based searching algorithms, GA generally includes selection, crossover and mutation operations.

4.1 Encoding

We need to encode our problem in line with the requirements of GA at the beginning. Usually, each individual (i.e., the so-called phenotype) in the population can denote a possible solution. The corresponding genotype can be obtained as follows. We aim to jointly optimize the task caching and computation offloading in VEC. Recall that a matrix $\boldsymbol{\phi}$ is used to represent the caching decisions of $\mathcal{R}$ over the tasks $\mathcal{T}$. Each row in $\boldsymbol{\phi}$ denotes the decision profile of one RSU over all the tasks in $\mathcal{T}$. Thus, this matrix is actually the decision profile of all the RSUs over the tasks. Accordingly, this matrix $\mathcal{M}$ can be used to present the chromosome (the genotype), i.e.,

$$
\mathcal{M} = \begin{bmatrix}
\phi_{1,1} & \cdots & \cdots & \cdots & \phi_{1,M} \\
\phi_{2,1} & \cdots & \cdots & \cdots & \phi_{2,M} \\
\phi_{3,1} & \cdots & \cdots & \cdots & \phi_{3,M} \\
\phi_{4,1} & \cdots & \cdots & \cdots & \phi_{4,M} \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
\phi_{K,1} & \cdots & \cdots & \cdots & \phi_{K,M}
\end{bmatrix} \tag{34}
$$

where each element $\phi_{k,m} (\in \{0,1\}) (1 \leq k \leq K, 1 \leq m \leq M)$ denotes whether task t_k is cached at R_m. Namely, $\phi_{k,m}$ in $\mathcal{M}$ is the same meaning as ϕ_m^k defined earlier. Thus, the population can be constituted by numerous chromosomes. The whole population denotes the potential solution space. Since $\phi_{k,m}$ is a binary variable, the potential solution space consists of 2^{KM} potential solutions. Searching over such a huge solution space is impracticable in latency sensitive scenarios.

On the other hand, our constraint condition (24) represents that each task in $\mathcal{T}$ is cached at most once. This constraint, if leveraged properly, can greatly prune the searching space. To be more specific, in terms of the chromosome $\mathcal{M}$, the constraint (24) means that for each row k, also known as the gene segment, there is at most one element $\phi_{k,m}$ which is equal to 1. Hence, there are total $M+1$ cases for row k. The total searching space is actually $(M+1)^K$, which is much smaller than 2^{MK}. However, it is worth noting that $\mathcal{M}$ is actually a sparse matrix due to the fact that there is at most one element $\phi_{k,m}$ equal to 1 in row k. Thus, there are at most K nonzero elements in $\mathcal{M}$ of KM elements.

If we do the mutation operations based on $\mathcal{M}$ for producing the offsprings, there would be lots of constraint-violated offsprings, since it is only allowed to have at most one nonzero element in each gene segment. To cope with this issue, we reconstruct the chromosome by leveraging the constraint (24). To be more specific, the chromosome $\mathcal{M}^\dagger$ is expressed as:

$$\mathcal{M}^\dagger = \begin{bmatrix} \psi_{1,1} & \cdots & \cdots & \cdots & \psi_{1,U} \\ \psi_{2,1} & \cdots & \cdots & \cdots & \psi_{2,U} \\ \psi_{3,1} & \cdots & \cdots & \cdots & \psi_{3,U} \\ \psi_{4,1} & \cdots & \cdots & \cdots & \psi_{4,U} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \psi_{K,1} & \cdots & \cdots & \cdots & \psi_{K,U} \end{bmatrix} \tag{35}$$

where $\psi_{k,u} \in \{0,1\}, 1 \le k \le K, 1 \le u \le U$ and $U = \lceil \log_2^{M+1} \rceil$. The row vector $\boldsymbol{\psi}_i = (\psi_{i,1}, \cdots, \psi_{i,U})$ $(1 \le i \le K)$ represents which RSU task t_i is cached at. Specifically, we regard $\boldsymbol{\psi}_i = (\psi_{i,1}, \cdots, \psi_{i,U})$ as the binary number, and convert it to the decimal numeral as the index of the RSU in $\mathcal{R}$. When the resulting decimal numeral is zero, it means that no RSU in $\mathcal{R}$ caches t_i. It is obvious that the mutation operated upon $\mathcal{M}^\dagger$ will avoid the above issue.

4.2 Fitness function

Fitness function denotes the individual adaptability to the environments during evolution. Such a quantitative evaluation can help GA decide which individuals can be reserved and which individuals cannot. In terms of our problem in this paper, we want to minimize both the service time and energy consumption. When it comes to the individual in GA, an individual with more powerful environmental adaptability always has a lower value of $O(\boldsymbol{\phi})$. Therefore, we use Eq. (21) as the fitness function. Given the task caching profile $\boldsymbol{\phi}$, a smaller value of the fitness function always denotes a better individual w.r.t. the environmental adaptability.

4.3 Selection operation

Selection operation plays a key role in the process of individual evolvement in GA. Furthermore, it also affects the convergence rate of GA. Generally, the better individuals in terms of the fitness values have a higher probability to be reserved during the evolvement of the population. By doing this, the optimal solution can be found at a high speed. In this paper, we adopt the simple roulette-wheel selection to reserve the better individuals. Specifically, in each generation, a portion of individuals are reserved for the next generation by selection operation based on the selection probability. After the selection operation, we supplement the individuals to keep the same population size.

4.4 Crossover operation

Crossover operation is used for generating the offsprings, which usually requires the cooperation between two parents. As an important means to preserve species diversity,

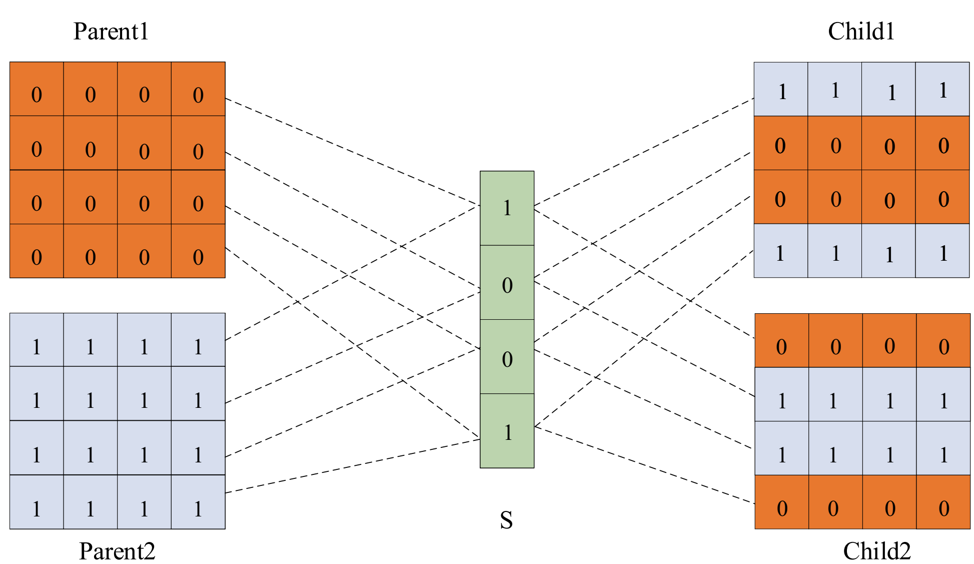


Fig. 2 Example of multiple point crossover

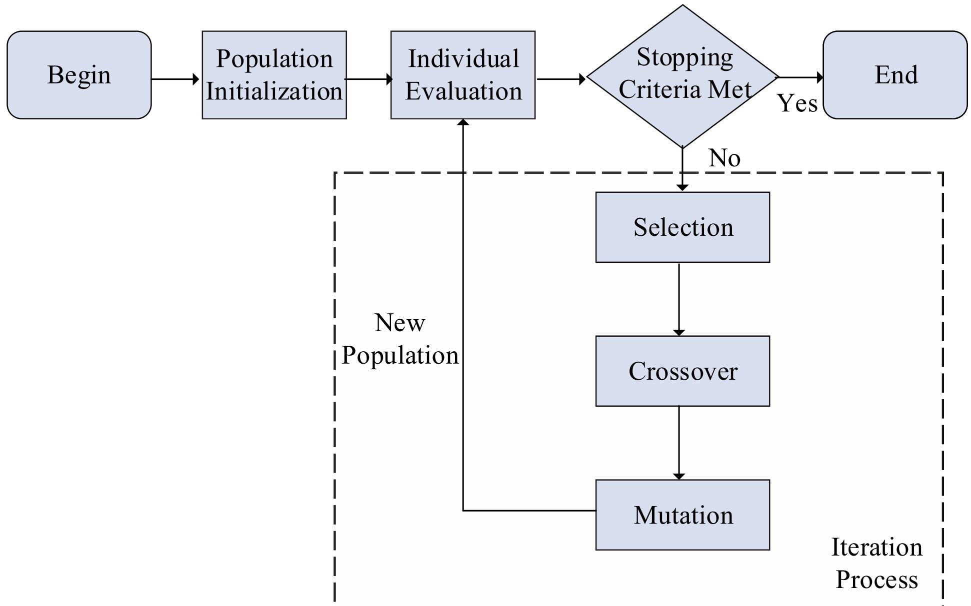


Fig. 3 The general framework for GA

crossover operation is implemented by exchanging the gene segments between two parents. In this paper, we adopt a multiple point crossover to generate the offsprings [23]. Specifically, an example of multiple point crossover operation is depicted in Fig. 2.

As depicted, the crossover operation is assisted by a binary vector S. The number of elements in S is the same as the number of rows in $\mathcal{M}^{\dagger}$. The value of elements in S denotes whether the corresponding gene segments (rows) in two parents need to be exchanged. For instance, the first element of S is one, which means that the gene segments (0,0,0,0) from Parent1 and (1,1,1,1) from Parent2 should be exchanged.

4.5 Mutation operation

The mutation operation is also an important means to preserve the spaces diversity. Specifically, the mutation operation, which only involves one individual (parent), is usually implemented after the crossover operation by altering a gene in the gene segment of the parent. As a result, the resulting offspring is sort of "close" to the parent, in terms of distance between them in the multi-dimensional solution space. Thus, the crossover and mutation operations serve different purposes, respectively, i.e., the crossover operation focuses on the improvement of the global search capability while the mutation operation pays attention to the local search capability improvement. In terms of our problem, a single point mutation is used, to prevent GA from falling into the local best solutions. Specifically, the mutation needs another binary vector S which has only one nonzero element. The index of the nonzero element in S denotes the location (i.e., gene segment) of the mutation. Then gene is also randomly determined for the mutation operation in the chosen gene segment.

4.6 GA-based algorithm design

The general framework for GA is depicted in Fig. 3, which consists of several steps such as selection, crossover, and mutation operations. Generally, the running time of GA depends upon when the stopping criteria are met. In terms of our optimization problem, the GA based joint optimization of task caching and computation offloading in VEC is shown in Algorithm 1.

In the algorithm, we use the crossover probability and mutation probability to control the crossover and mutation operations, respectively. It is worth mentioning that the crossover and mutation are operated on the same population S_1, which makes the order of the two operations irrelevant in this algorithm.

There are many constraint conditions in the problem formulation such as constraints (23)–(31), so many resulting individuals may be constraint-violated after crossover and mutation operations. It is necessary to check and remove those individuals which violate the constraints when a new population is generated. In case of a constraint violation, new individuals should be added to keep the same size of population. The running time of this algorithm depends upon the stopping criteria adopted in this paper. Usually, there are two most popular ways to serve as the stopping criteria. One is to directly set the number of iteration steps. The other is to define the optimization gap between the globally optimal fitness value explored so far and the current fitness value. If the value of optimization gap is smaller than the given threshold, we can draw a conclusion that the algorithm has converged.

Algorithm 1: GA based optimization of task caching and computation offloading (GATC)

Input: Parameters required for GA and problem $\mathcal{P}1$, respectively
Output: The optimal fitness value V
Initialize a population S for GA;
Evaluate the fitness values of individuals in S using Eq.(21);
Record the best fitness value V;
while *stopping criteria not met* **do**
  Produce new population S_1 by selection operation on S;
  for *each pair of individuals in* S_1 **do**
    Perform the crossover operations on it based on the crossover probability;
  end
  Form new population S_2 based on crossover operations on S_1;
  Remove the individuals in S_2 with constraint violations;
  for *each individual in* S_1 **do**
    Perform the mutation operation on it based on the mutation probability;
  end
  Form new population S_3 based on mutation operations on S_1;
  Remove the individuals in S_3 with constraint violations;
  Form new population S by combining S_2 with S_3;
  Supplement S with randomly generated individuals to keep the same population size;
  Evaluate the fitness values of individuals in S;
  Record and update the best fitness value V;
end
Return the best fitness value V;

5 Simulation results and analysis

5.1 Experimental settings

We have conducted a series of experiments to investigate the GA based joint optimization of task caching and computation offloading in terms of effectiveness and efficiency in this section. First, our experiments are run on a laptop with 1.8GHz Intel I5 Quad-Core CPU, 8G of RAM, Microsoft Windows 10 Operating System, Python 3.7. Second, the initial parameter settings in the experiments are set appropriately. For instance, the number of RSUs serving the moving vehicles is set to 5, and the number of vehicles served by each RSU ranges from 15 to 30. For the three components of tasks, say $t_k = (d_k, s_k, r_k)$, d_k ranges from 1 to 51, s_k ranges from 30 to 70 and r_k ranges from 30 to 40. The caching capability of each RSU c_i ranges from 100 to 200. On the other hand, for the GA involved parameters, the population size and the number of iterations are set to 100 and 50, respectively.

It shall be noted that the proposed algorithm GATC can be affected by many factors. For instance, parameters, such as the crossover probability, mutation probability, population size, and the number of iterations, can greatly influence the performance of GATC w.r.t. running time, convergence rate and obtained solution. Parameters, including the number of RSUs, the number of offloading requests and so on, can also affect the performance of GATC in comparison to other approaches. To investigate the efficiency and effectiveness of GATC, two approaches are introduced as the benchmark algorithms.

Random approach. One simple way for task caching is to select the RSU in a random way. To be more specific, tasks from $\mathcal{T}$ can be cached arbitrarily at RSUs without constraint violations. There are at least two constraints which are imposed upon the task caching. For example, one is that the total caching results at one RSU should not exceed its own caching capability, which corresponds to the constraint (23). The other is that each task can only be cached at most once, which corresponds to the constraint (24).

Greedy approach. There are many rules to guide the solution searching when the greedy approaches are adopted. For instance, different RSUs may have different caching capabilities. Thus an intuitive rule is that RSUs with more powerful caching capabilities should cache more tasks. Therefore, these RSUs can choose either the tasks with larger result size or the tasks with smaller result size first. Different caching rules may result in different performance. However, this greedy rule does not consider the dynamic distribution of these offloading requests. Specifically, although the task is cached, most of the offloading requests for it come from other RUSs rather than the one where it is cached. Accordingly, the bandwidth resources among these RSUs can be seriously consumed to transmit the caching results. To avoid this situation, we in this paper adopt another greedy rule, i.e., each RUS decides the tasks to be cached based on its own offloading requests. They tend to cache those most frequently requested tasks in their own serving area. One inevitable case is that tasks can be repeatedly cached at different RSUs. To cope with this issue, state information about cached tasks should be disseminated among these RSUs. Despite extra communication resources consumed by RSUs, it is still more efficient than the former one.

5.2 Impact of parameters

In this section, we have conducted a series of experiments to evaluate the influence of the GA-involved parameters upon our strategy. These parameters include the crossover probability, mutation probability and the population size.

First, the effects of the crossover probability on GATC are investigated in terms of the obtained optimal values and the response latency. The results are shown in Figs. 4 and 5, respectively. We denote the crossover probability by cp in the experiments. Five values for cp are evaluated. Other parameters are set to default values empirically. For example, the population size is set to 50 and the mutation probability is set to 0.02. The number of iteration steps is 50 in Fig. 4 and 20 in Fig. 5. From Fig. 4, we can observe that the performance of GATC is affected by the crossover probability to some extent. In terms of the obtained fitness values, the best crossover probability is 0.3 and the worst is 0.2. When the crossover probability is set appropriately (i.e., $cp = 0.3$), the fitness values are averagely reduced by 5%, 36%, 2% and 18%, compared to the cases $cp = 0.1, 0.2, 0.4, 0.5$ respectively. These different fitness values denote that GATC are

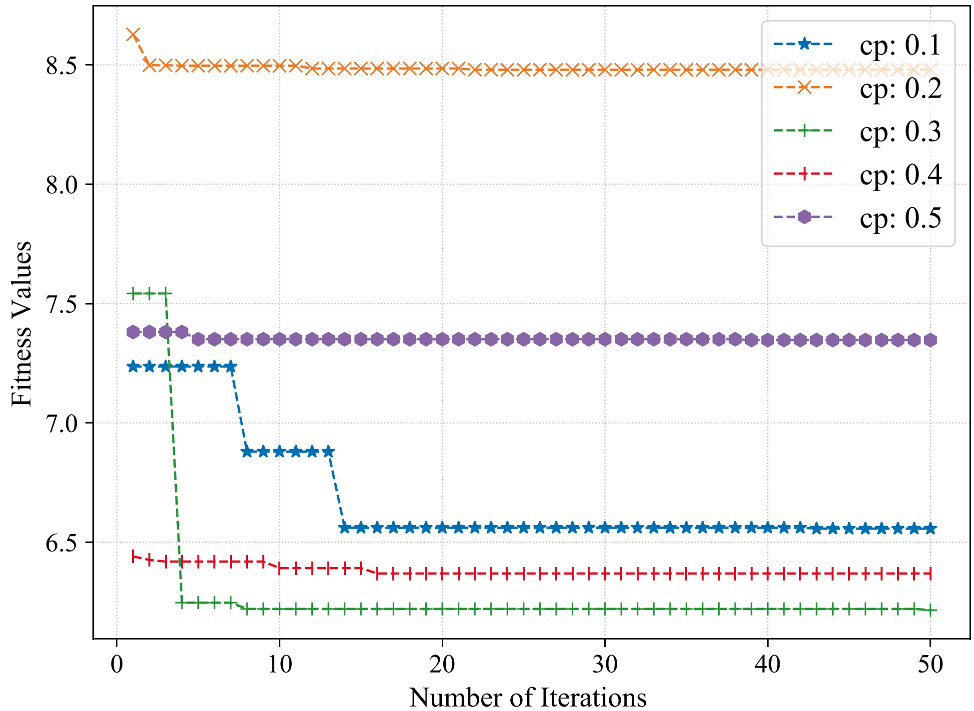


Fig. 4 Fitness values with different crossover probabilities

stuck in different locally optimal solutions respectively. We also notice that no matter which case is considered, the obtained fitness values do not change any more when the number of iterations comes to 20. The response times are shown in Fig. 5 when the different crossover probability is considered. The number of iterations is up to 20. This is because the fitness values as shown in Fig. 4 do not change since then. The time taken to obtain the approximately optimal fitness values is acceptable. On one hand, GATC takes time to search for the best solution over the huge searching space, and on the other hand, GATC takes time to check the validity of these individuals in the population. When the individuals are invalid, it still needs to supplement new individuals to the population.

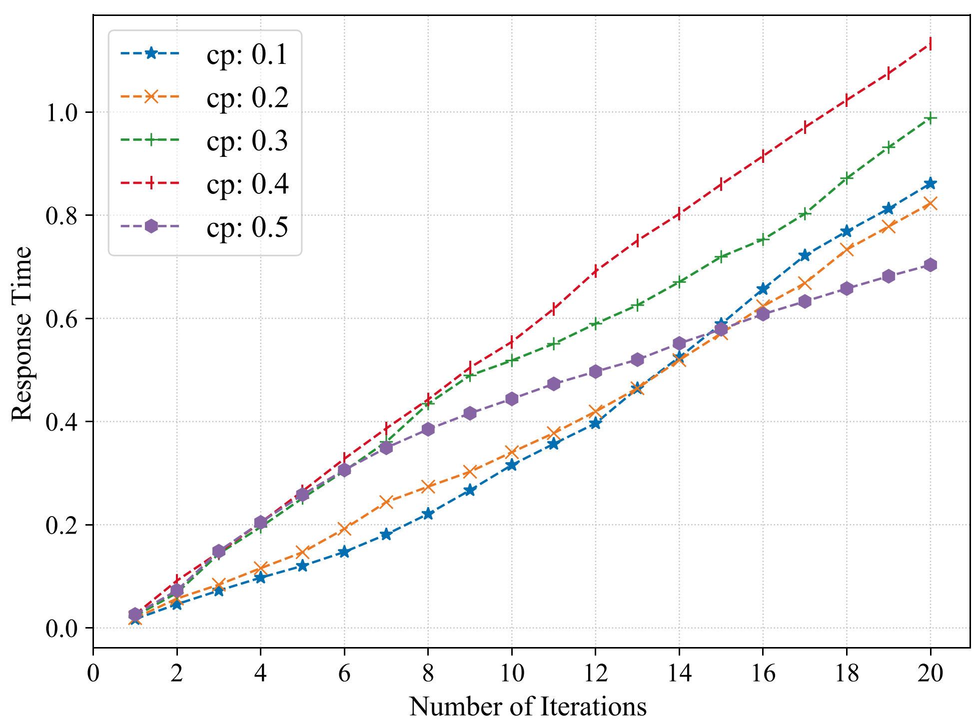


Fig. 5 Response times with different crossover probabilities

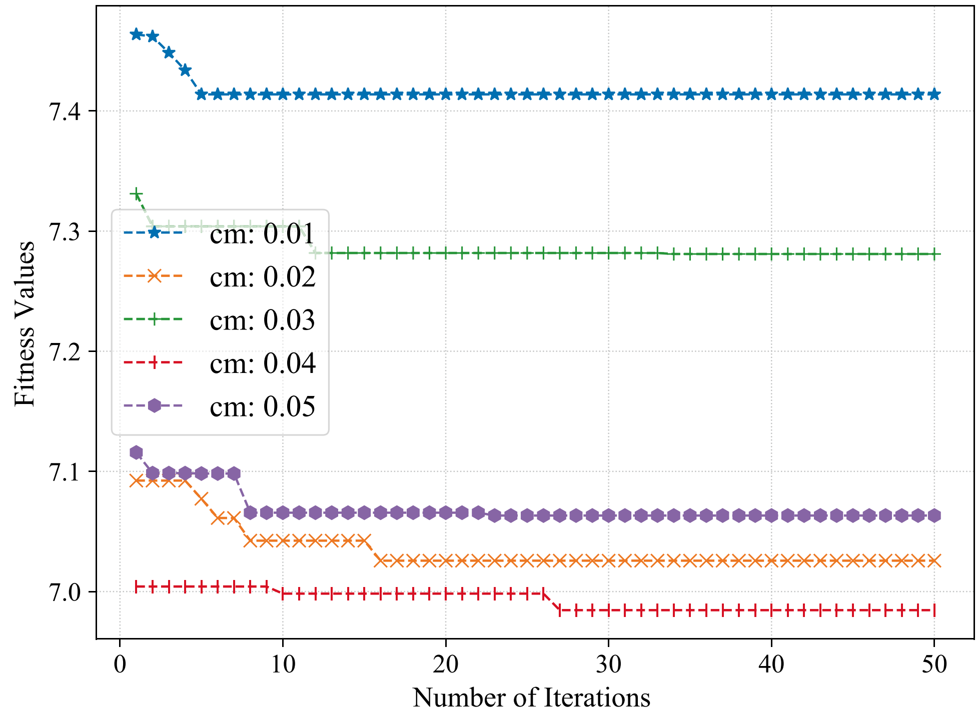


Fig. 6 The fitness values with different mutation probabilities

Second, the effects of the mutation probability on GATC are investigated in terms of the obtained optimal values and the response latency. The results are shown in Figs. 6 and 7, respectively. We denote the mutation probability by *cm* in the experiments. Similar to the evaluation of crossover probability in the first set of experiments, five values for *cm* are investigated. Other parameters are also set to default values empirically. Note that based on the results shown in the first set of experiments, the crossover probability is set to 0.3. The population size is set to 50. The number of iteration steps is 50 in Fig. 6 and 30 in Fig. 7. Figure 6 shows that the performance of GATC is also affected by the mutation probability to some extent. In terms of the obtained fitness values, the best mutation probability is 0.04 and the worst is

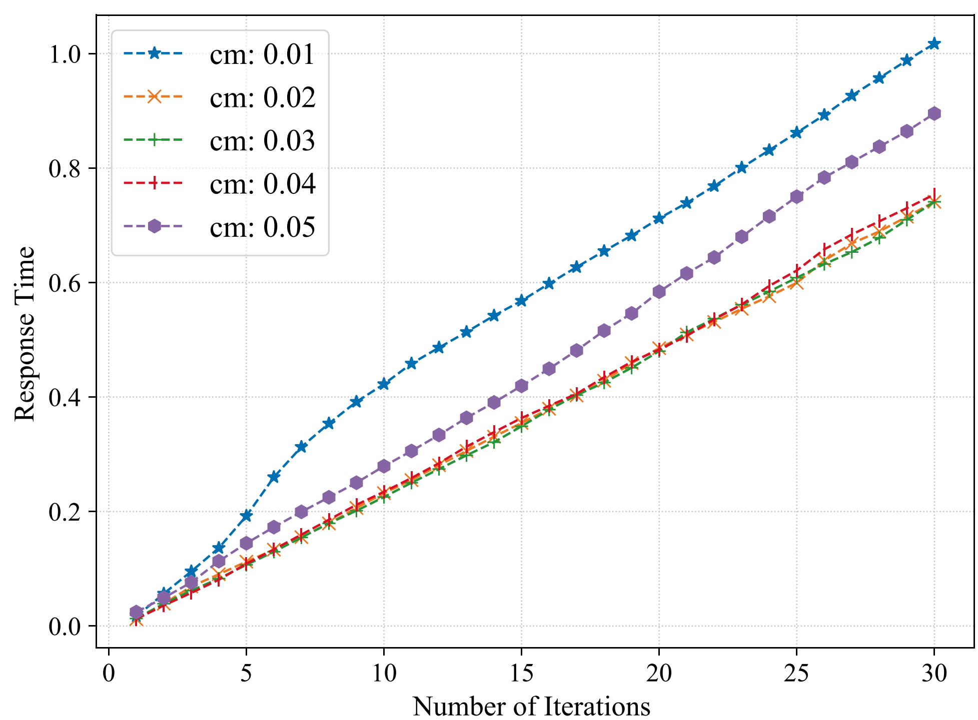


Fig. 7 The response times with different mutation probability

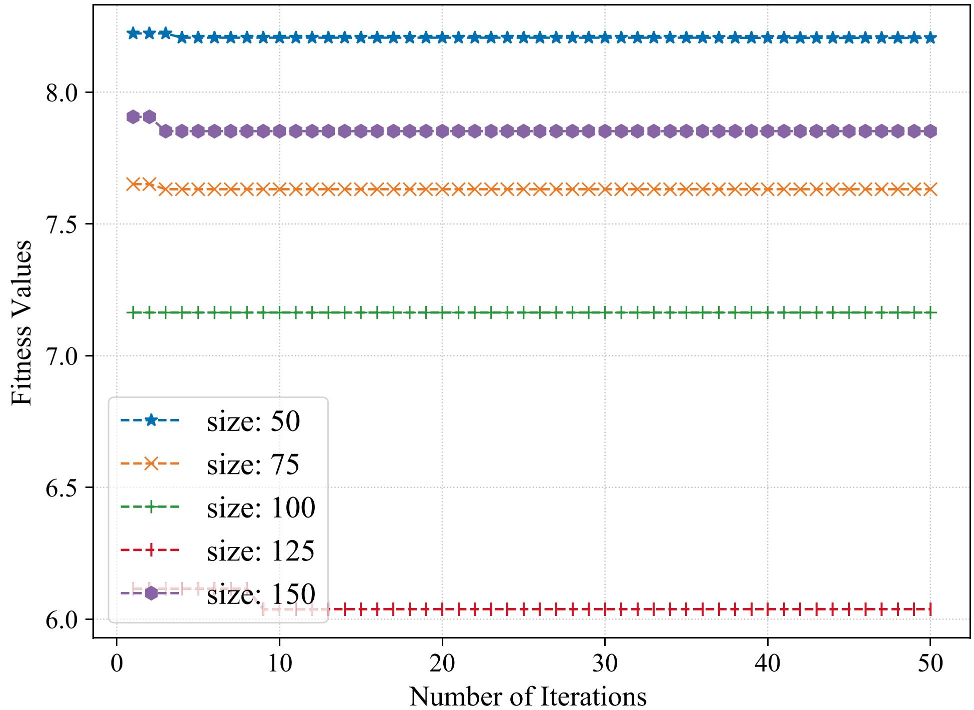


Fig. 8 The fitness values with different population sizes

0.01. That means the local searching capability of GATC can achieve the best with the mutation probability equal to 0.04. To be more specific, when the mutation probability is 0.04, the fitness values are averagely reduced by 6%, 0.5%, 4% and 1%, compared to the cases $cm = 0.01, 0.02, 0.03, 0.05$ respectively. These different fitness values also denote that GATC can be stuck in different locally optimal solutions respectively. Furthermore, GATC with different mutation probability has different convergence rates. In the meanwhile, we also notice that no matter which case is considered, the obtained fitness values do not change any more when the number of iterations comes to 30. The response times are shown in Fig. 7 when the different mutation probability is considered. The number of iterations is up to 30, for the reason that the fitness values as shown in Fig. 6 do not decrease any more. The analysis about time consumption is similar to the first set of experiments which also includes two parts – one for solution searching in huge solution space and the other for the validity checking for the individuals in the population. We do not detail them further.

Third, the effect of the population size on GATC is investigated in terms of the obtained optimal values. Generally, the larger the population size, the better the obtained fitness values. This is because a larger population includes more individuals which can explore more potential solutions at the same time. However, a larger population size also incurs more time consumption. Therefore, there should be a tradeoff between the time consumption and the optimal value. Specifically, the experimental results are shown in Fig. 8. We denote the population size by *size* in the figure. The population size varies from 50 to 150 with a step of 25. Other parameters are set empirically, e.g., the crossover probability is set to 0.3, and the mutation probability is 0.04. The number of iteration steps is 50.

From this figure, we can draw several conclusions based on the observations. First, the population size can affect GATC w.r.t. the fitness values. For example, the optimal values are different when the population size is different. In this experiment, the performance can reach the best when the population size is 125, which however contradicts the cognition that a larger population size always brings about a better fitness value. One possible and acceptable explanation is that GATC has been stuck in locally optimal solutions very soon. For example, as far as the population size equal to 150 is concerned, GATC converges to the local optimum value at the fastest speed. After that, the fitness value does not decrease any more. Second, generally speaking, the convergence rate is high no matter which population size is investigated. Last but not least, as discussed earlier, a larger population size does not always yield a better solution.

5.3 Approach comparison

In this section, we compare our approach with the benchmark algorithms. Since we have evaluated the GA involved parameters in the previous section, GATC will be run with parameters denoting the best performance. Specifically, we compare our approach with others from multiple perspectives. The first set of experiments is conducted to compare them when the number of RSUs changes. When the

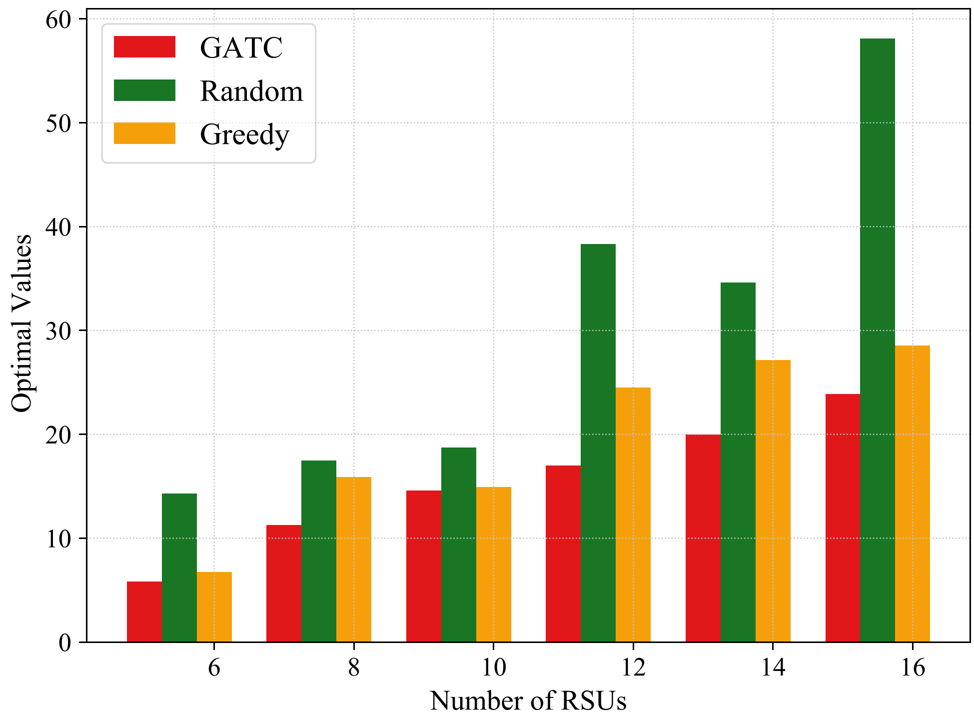


Fig. 9 Performance comparison with different numbers of RSUs

number of RSUs in the VEC system is large, the number of vehicular applications served by RSUs is also very large. In this context, the number of tasks that can be cached also increases. Accordingly, the probability that the tasks requested by vehicles are already cached in the VEC systems increases. However, the fact that a large number of tasks cached in the VEC system not only incurs energy consumptions as denoted in Eq. (5), but also consumes more bandwidth resources for state information dissemination and sharing among RSUs.

The simulation result is shown in Fig. 9. It is obvious that GATC can achieve the best performance compared to both the random approach and the greedy approach. The random approach is always the worst among the three approaches.

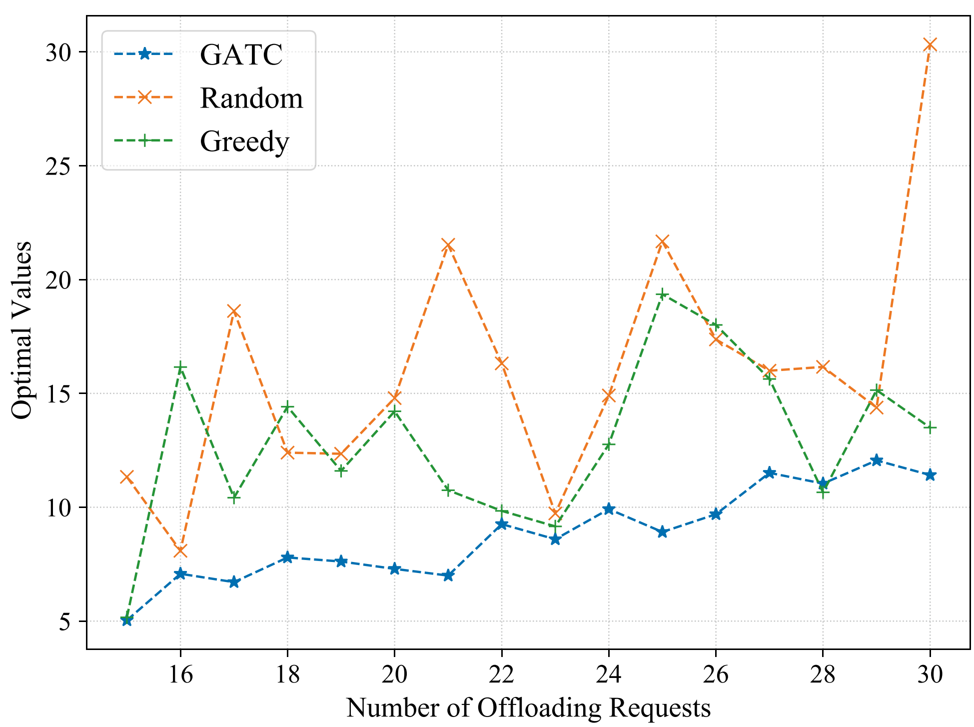


Fig. 10 Performance comparison with different numbers of offloading requests

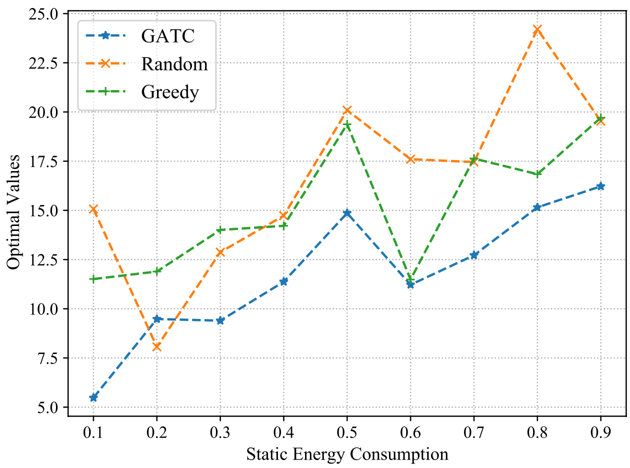


Fig. 11 Performance comparison with different static energy consumption

With the increasing number of RSUs, the number of offloading requests also increases. Accordingly, the optimal value defined by Eq. (21) becomes larger and larger. The optimal value of the random approach varies sharply compared to the other two approaches.

In addition, the number of vehicles served by each RSU can also affect the performance of our strategy. An assumption has been made in the experiment that the number of offloading requests is the same as the number of vehicles in the serving area of each RSU. The second set of experiments has been carried out to evaluate the influence of the number of offloading requests upon the strategy. In the experiment, the number of RSUs is 5, and the number of offloading requests for each RSU varies from 15 to 30. The simulation result is shown in Fig. 10. As expected, GATC is the best while the random approach is the worst among the three approaches. However, when the number of requests is 16, the random approach is better than the greedy approach. One possible explanation is that the placement of task results at RSUs in a random way caters for the randomly generated offloading requests much more than the greedy approach. This figure also reveals that the optimal values increase for the three approaches as the number of offloading requests increases.

Last, the three approaches are compared with each other from the viewpoint of static energy consumption. The benefits of task caching have been summarized in the previous sections, such as response latency reduction and QoE improvement. We also mentioned that task caching may render more energy consumptions especially for the fine-granularity task caching. Therefore, we conduct the last set of experiments to investigate the influence of the static energy consumptions (i.e., γ_i defined in Eq. 5). For simplification, we assume that all the γ_i is the same. In other words, the static power consumption for one unit data/task caching is the same for all the RUSs in $\mathcal{R}$. As the equation denotes, when the value of γ increases, the corresponding energy consumption also increases. In the experiment, the number of RSUs is 5 and the number of offloading requests for each RSU is 20. Each vehicle randomly requests the computation offloading to its serving RSU, i.e., offloading the task from $\mathcal{T}$ in a random way. The simulation results are shown in Fig. 11, where γ varies from 0.1 to 0.9.

From this figure, we can see that the optimal values of the three strategies all increase as the value of γ increases, which is reasonable since γ is the unit cost for task caching in terms of energy consumptions. Still, GATC averagely achieves the best performance w.r.t. the optimal values and the random approach averagely achieves the worst performance among the three approaches. Interestingly, the random approach achieves the best performance when the value of γ is 0.2, compared to the greedy approach and GATC. The reason is similar to the case shown in Fig. 10 where the random approach is better than the greedy approach with the number of requests equal to 16. Specifically, the random placement of task results at RSUs caters to the randomly generated offloading requests much more than GATC and the greedy approach.

6 Conclusion

Extensive attention has been paid to task offloading in VEC system recently. As a new computing paradigm, VEC can undertake the computation offloaded from the vehicles in its serving area. To further enhance the performance of VEC w.r.t. response latency reduction, task oriented caching strategy has been applied to VEC. However, some issues that revolved around caching enabled task offloading in VEC still need to be addressed. In this paper, we have proposed a general caching-enabled VEC scheme. For example, we consider not only caching results placement at RSUs but also the caching results delivery in VEC. For the problem formulation, both the response latency and energy consumption are taken into consideration, by jointly optimizing the task caching and computation offloading in VEC. A genetic algorithm-based approach is put forward to minimize the weighted sum of the service time and energy consumption for all the offloading requests. Simulation results have shown its advantages over the benchmark algorithms. For the future work, we plan to design a more efficient algorithm for this issue.

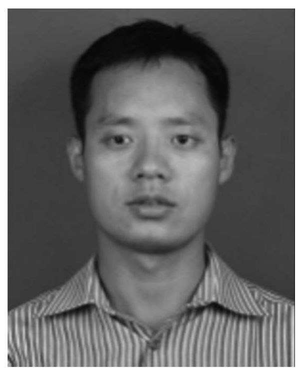
Acknowledgements This work is supported by the National Natural Science Foundation of China (61801325 and 62071327). The authors are really grateful to the editor and anonymous reviewers for their professional comments and helpful suggestions.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Tang C, Zhu C, Wei X, Wu H, Li Q, Rodrigues JJPC (2020) Intelligent resource allocation for utility optimization in rsu-empowered vehicular network. IEEE Access 8:94453–94462
2. Feng J, Liu Z, Wu C, Ji Y (2019) Mobile edge computing for the internet of vehicles: Offloading framework and job scheduling. IEEE Veh Technol Mag 14(1):28–36
3. Laroui M, Nour B, Moungla H, Afifi H, Cherif MA (2020) Mobile vehicular edge computing architecture using rideshare taxis as a mobile edge server. In: IEEE 17th Annual Consumer Communications & Networking Conference, CCNC 2020, Las Vegas, NV, USA, January 10-13, 2020, IEEE pp. 1–2
4. Zhang J, Guo H, Liu J, Zhang Y (2020) Task offloading in vehicular edge computing networks: A load-balancing solution. IEEE Trans Veh Technol 69(2):2092–2104
5. Xu J, Chen L, Zhou P (2018) Joint service caching and task offloading for mobile edge computing in dense networks. In: 2018 IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, April 16-19, 2018, IEEE pp. 207–215
6. Yao Y, Xiao B, Wang W, Yang G, Zhou X, Peng Z (2020) Real-time cache-aided route planning based on mobile edge computing. IEEE Wirel Commun 27(5):155–161
7. Avino G, Malinverno M, Malandrino F, Casetti C, Chiasserini C (2017) Characterizing docker overhead in mobile edge computing scenarios. In: Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems, HotConNet@SIGCOMM 2017, Los Angeles, CA, USA, August 25, 2017, ACM pp. 30–35
8. Cha N, Wu C, Yoshinaga T, Ji Y, Yau KA (2021) Virtual edge: Exploring computation offloading in collaborative vehicular edge computing. IEEE Access 9:37739–37751
9. Gu X, Zhang G (2021) Energy-efficient computation offloading for vehicular edge computing networks. Comput Commun 166:244–253
10. Hu J, Chen C, Cai L, Khosravi MR, Pei Q, Wan S (2021) Uav-assisted vehicular edge computing for the 6g internet of vehicles: Architecture, intelligence, and challenges. IEEE Commun Stand Mag 5(2):12–18
11. Wu C, Liu Z, Liu F, Yoshinaga T, Ji Y, Li J (2020) Collaborative learning of communication routes in edge-enabled multi-access vehicular environment. IEEE Trans Cogn Commun Netw 6(4):1155–1165
12. Zhao L, Yang K, Tan Z, Li X, Sharma S, Liu Z (2021) A novel cost optimization strategy for sdn-enabled uav-assisted vehicular computation offloading. IEEE Trans Intell Transp Syst 22(6):3664–3674
13. Sonmez C, Tunca C, Ozgovde A, Ersoy C (2021) Machine learning-based workload orchestrator for vehicular edge computing. IEEE Trans Intell Transp Syst 22(4):2239–2251
14. Wang S, Ye D, Huang X, Yu R, Wang Y, Zhang Y (2021) Consortium blockchain for secure resource sharing in vehicular edge computing: A contract-based approach. IEEE Trans Netw Sci Eng 8(2):1189–1201
15. Islam S, Badsha S, Sengupta S, La HM, Khalil I, Atiquzzaman M (2021) Blockchain-enabled intelligent vehicular edge computing. IEEE Netw 35(3):125–131
16. Hao Y, Chen M, Hu L, Hossain MS, Ghoneim A (2018) Energy efficient task caching and offloading for mobile edge computing. IEEE Access 6:11365–11373
17. Xing H, Cui J, Deng Y, Nallanathan A (2019) Energy-efficient proactive caching for fog computing with correlated task arrivals. In: 20th IEEE International Workshop on Signal Processing Advances in Wireless Communications, SPAWC 2019, Cannes, France, July 2-5, 2019, IEEE pp. 1–5
18. Javed MA, Zeadally S (2021) Ai-empowered content caching in vehicular edge computing: Opportunities and challenges. IEEE Netw 35(3):109–115
19. Qiao G, Leng S, Maharjan S, Zhang Y, Ansari N (2020) Deep reinforcement learning for cooperative content caching in vehicular edge computing and networks. IEEE Internet Things J 7(1):247–257
20. Tan LT, Hu RQ, Hanzo L (2019) Twin-timescale artificial intelligence aided mobility-aware edge caching and computing in vehicular networks. IEEE Trans Veh Technol 68(4):3086–3099
21. Wang S, Zhang Z, Yu R, Zhang Y (2017) Low-latency caching with auction game in vehicular edge computing. In: 2017 IEEE/CIC International Conference on Communications in China, ICCC 2017, Qingdao, China, October 22-24, 2017, IEEE pp. 1–6
22. Tang C, Zhu C, Wei X, Wu H, Rodrigues JJPC (2021a) Task offloading and caching for mobile edge computing. In: 2021 IEEE 17th Int. Wireless Communications & Mobile Computing Conference-, IWCMC, Harbin, China, June 28 - July 2, 2021, IEEE pp. 1–5
23. Tang C, Xia S, Li Q, Chen W, Fang W (2021b) Resource pooling in vehicular fog computing. J Cloud Comput 10(1):19



Chaogang Tang received his B.S. degree from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, and Ph.D. degree from the School of Information Science and Technology, University of Science and Technology of China, Hefei, China, and the Department of Computer Science, City University of Hong Kong, under a joint Ph.D. Program, in 2012. He is now with the China University of Mining and Technology. His research interests include mobile cloud computing, fog computing, Internet of Things and big data.

Huaming Wu received the B.E. and M.S. degrees from Harbin Institute of Technology, China in 2009 and 2011, respectively, both in electrical engineering. He received the Ph.D. degree with the highest honor in computer science at Free University of Berlin, Germany in 2015. He is currently an associate professor in the Center for Applied Mathematics, Tianjin University. His current research interests include mobile and cloud computing, edge computing, wireless and mobile network systems, internet of things (IoTs), and deep learning.