Received December 12, 2019, accepted December 31, 2019, date of publication January 10, 2020, date of current version January 17, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2965620

Joint Optimization of Computation Offloading and Task Scheduling in Vehicular Edge Computing Networks

JIANAN SUN[1,2], QING GU[3], (Member, IEEE), TAO ZHENG[2], (Member, IEEE), PING DONG[2], (Member, IEEE), ALVIN VALERA[4], (Member, IEEE), AND YAJUAN QIN[2]

[1]State Key Laboratory of Rail Traffic Control and Safety, Beijing Jiaotong University, Beijing 100044, China
[2]School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China
[3]School of Mechanical Engineering, University of Science and Technology Beijing, Beijing 100083, China
[4]School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand

Corresponding author: Tao Zheng (zhengtao@bjtu.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFC0810500, and in part by the Key Project of the State Key Lab of Rail Traffic Control and Safety under Grant RCS2018ZZ007.

ABSTRACT Resource-intensive applications on smart vehicles is posing difficulties to the use of traditional cloud computing for computation offloading in vehicular networks. In particular, the long transmission distance between the vehicles and the cloud center can cause high latency and poor reliability which may degrade application performance and quality of service. As an integration of mobile edge computing and vehicular networks, vehicular edge computing is a promising paradigm that aims to improve vehicular services by performing computation offloading in close proximity to vehicles. In this paper, the task offloading algorithm that efficiently optimizes task delay and computing resource consumption in multi-user, multi-server vehicular edge computing scenarios is studied. The offloading algorithm not only determines where the tasks are performed, but also indicates the execution order of the tasks on the server. In order to reduce the time complexity, this paper proposes a hybrid intelligent optimization algorithm based on partheno genetic algorithm and heuristic rules. Extensive simulations are conducted, and the results show that compared with the baseline algorithms, the proposed algorithm effectively improves the offloading utility of the VEC system and is suitable for task offloading in various situations.

INDEX TERMS Computation offloading, Internet of Things, mobile edge computing, task scheduling, vehicular networks.

I. INTRODUCTION

With the development of the Internet of Things (IoT) and wireless technologies, cutting edge applications such as traffic cognition, automatic driving and augmented reality are emerging — aimed at improving the efficiency and safety of transportation systems. One of the key challenges posed by these new class of futuristic applications is their requirement to analyze large volumes of data to make appropriate and timely decisions [1]. This entails tremendous demand for computing power, a major challenge to the limited computing resources on vehicles. In recent years, cloud computing has been used, wherein the resource-consuming tasks are offloaded to a powerful cloud center which performs all the necessary computation and returns the results to vehicles [2]. At present, cloud computing is still the mainstream offloading method to alleviate the heavy computational burden on vehicles. However, the long transmission distance between the vehicles and the cloud center can cause high latency and poor reliability which may degrade application performance and quality of service. Moreover, extensive offloading may result in backbone network congestion [3].

The drawbacks of cloud computing in the context of computation offloading have led to the introduction of mobile edge computing (MEC). Unlike cloud computing where the cloud servers are placed far away from mobile users, MEC

The associate editor coordinating the review of this manuscript and approving it for publication was Markus Rupp.

migrates the computing resources to the edge of radio access networks. The mobile users' computation tasks are performed by local MEC servers, thereby avoiding delay fluctuations and capacity limitations of the transmissions on backhaul networks. Furthermore, MEC can provide personalized services based on surrounding contextual information, and its network resource allocation can be optimized [4].

Vehicular edge computing (VEC) is a promising computing paradigm that combines MEC and vehicular networks. In VEC networks, lightweight MEC servers are deployed along with infrastructure like road side units (RSUs) [5]. Thus, the quality of vehicular service can be greatly enhanced by offloading computation-intensive applications from the resource-constrained vehicles to the MEC servers. The efficiency of an edge computing network depends on the computation task offloading algorithm it employs. Although several excellent works have been proposed on task offloading in MEC networks, research on VEC networks is still sparse.

In this paper, we consider a VEC scenario in which multiple adjacent MEC servers provide computation offloading services for passing vehicles, and each server needs to serve multiple vehicles. Then we study the task offloading algorithm in this scenario. Usually, the tasks of different users are independent of each other. Based on this, we give a new perspective that is different from previous studies, that is, the task offloading algorithm here not only determines whether a task is offloaded and which MEC server performs the task, but also determines the order in which the tasks are executed by the MEC server. In addition, the algorithm comprehensively considers the costs of both the supplier and demander, that is, the task delay of the vehicles and the computing resource consumption of the MEC servers, in order to achieve the optimal system offloading utility.

The main contributions of this paper are summarized as follows:

(i) We construct the task offloading model for multi-user, multi-server VEC scenarios and define the *offloading utility* of the vehicles, which consists of task delay and computing resource consumption;
(ii) We formulate the *joint offloading decision and task scheduling problem* as a mixed integer non-linear programming problem with the goal of maximizing the system offloading utility;
(iii) We propose a *hybrid intelligent optimization algorithm* that combines partheno genetic algorithm (PGA) and heuristic rules to obtain an approximate optimal solution to the problem with low time complexity;
(iv) Extensive simulations are conducted and the results prove the accuracy of the proposed algorithm. Compared with the baseline algorithms, the algorithm effectively improves the offloading utility of the VEC system and is suitable for task offloading in various situations.

The remainder of this paper is organized as follows: In Section II, we provide an overview of the related work. In Section III, we present the VEC system architecture and the formulation of the task offloading problem. In Section IV, we introduce a hybrid intelligent optimization algorithm based on PGA and heuristic rules. In Section V, we present the results and analysis of the performance evaluation and finally in Section VI, we conclude the paper.

II. RELATED WORK

By deploying computing and storage resources at the edge of radio access networks, MEC can handle delay-sensitive and computation-intensive applications. As one of the most important problems in MEC, task offloading optimization has been investigated from different perspectives. Dinh *et al.* [6] studied the computational offloading from a single mobile device to multiple edge devices. They proposed a framework that minimizes the tasks' execution latency and the mobile device's energy consumption, by jointly optimizing the task allocation decision and the mobile device's central process unit (CPU) frequency. Wang *et al.* [7] studied the joint optimization of computation offloading decision, resource allocation and content caching in heterogeneous wireless cellular networks with MEC, and applied an alternating direction method of multipliers-based distributed solution to tackle this problem. Tran and Pompili [8] studied the joint optimization of task offloading decision, users' uplink transmission power, and MEC servers' computing resource allocation in a multi-cell MEC network. They addressed the resource allocation problem using quasi-convex and convex optimization techniques, and proposed a heuristic algorithm to solve the task offloading problem. Xu *et al.* [9] studied an energy harvesting MEC system that makes offloading and edge server provisioning decisions based on unique information such as available battery power and anticipated renewable power arrival. They proposed a post-decision state based reinforcement learning algorithm to learn on-the-fly the optimal offloading and auto-scaling policy. Yu *et al.* [10] considered a scenario where multiple mobile users offload duplicated tasks to the edge server and share the computation results. They proposed a fine-grained collaborative offloading strategy with caching enhancement scheme to minimize the execution delay of mobile users. Chen *et al.* [11] investigated computation peer offloading in MEC-enabled small-cell networks. They proposed a online peer offloading framework to maximize the long-term system-wide performance under limited energy resources committed by individual small-cell base stations. However, these works are mainly aimed at scenarios where the users are stationary or moving slowly within a relatively small area, without considering the impact of mobility on the users' MEC server selection.

As a special case of data offloading in vehicular networks, there have been some works on task offloading in VEC. Zhou *et al.* [12] focused on reducing the completion time of virtual reality (VR) application in vehicular networks. They divided the VR task into two sub-tasks so that the vehicle can be engaged in task computation with the MEC server. Then they proposed an algorithm to jointly optimize the offloading proportion, communication resource and computation resource allocation. Cui *et al.* [13] proposed

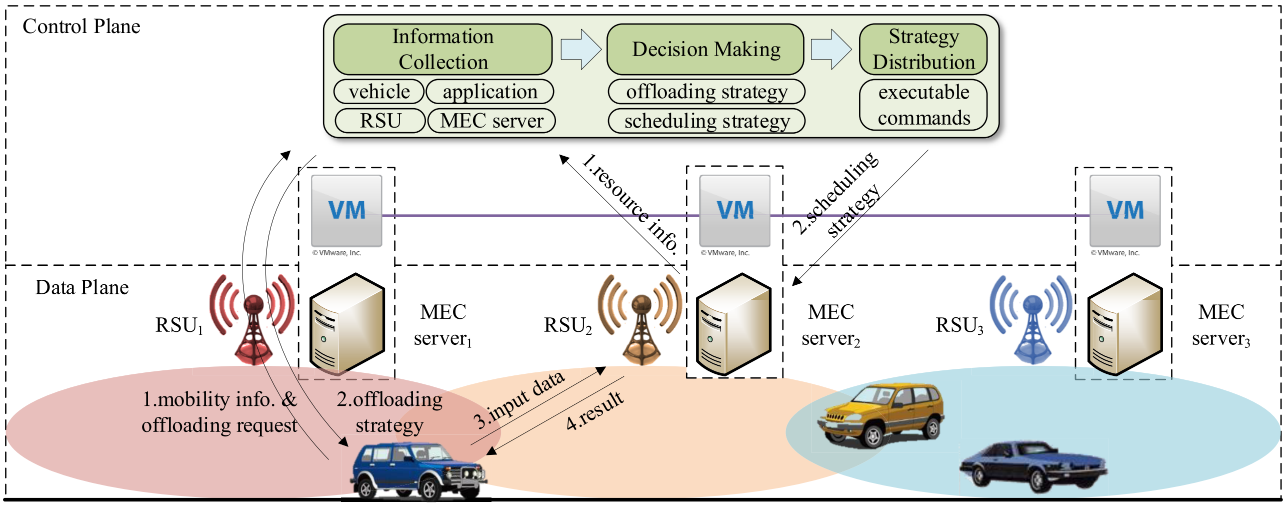


FIGURE 1. Vehicular edge computing system architecture.

a multi-platform intelligent offloading and resource allocation algorithm. They use the K-nearest neighbor algorithm to determine the suitable task offloading platform and use the reinforcement learning algorithm to solve the resource allocation problem. Qi *et al.* [14] proposed a knowledge-driven service offloading framework by exploring a deep reinforcement learning model. The framework considers the future data dependency of the following tasks when making a decision for a current task, so that the optimal offloading policy can be obtained directly from the environment without complicated computation. Dai *et al.* [15] proposed integrating load balancing with offloading in VEC networks, and developed a low-complexity algorithm to jointly optimize server selection, offloading ratio, and computation resource. Liu *et al.* [16] studied the task offloading problem from a matching perspective. They treated the vehicles and the RSUs as the two sides of a matching problem, transforming the offloading problem into one-to-one matching and matching with quota, respectively. Then they proposed a matching-based task offloading algorithm to minimize the network delay. Sun *et al.* [17] studied the task offloading among vehicles, i.e., the tasks of some vehicles are offloaded to other vehicles. They proposed an adaptive learning-based task offloading algorithm based on multi-armed bandit theory, which enables vehicles to learn the delay performance of their neighboring vehicles in a distributed manner. However, the above mentioned works focus on minimizing the cost of task delay from the perspective of the vehicle, and neglect the cost of the computing resource provider.

Over this background, this paper studies the task offloading algorithm for a VEC scenario in which multiple adjacent MEC servers provide computation offloading services for moving vehicles. Different from previous works that allocate computing resources in a coarse-grained manner, this paper takes the execution order of tasks on the MEC server into consideration, and jointly optimizes the offloading decision and task scheduling. In addition, the optimization objective of this paper takes into account the different needs of both the supplier and demander, i.e., the task delay of the vehicles and the computing resource consumption of the MEC servers.

III. SYSTEM OVERVIEW AND PROBLEM FORMULATION

In this section, we first introduce the VEC system architecture. Then we present the task offloading model and formulate the task offloading problem.

A. SYSTEM ARCHITECTURE

As illustrated in Fig. 1, we consider a VEC system where MEC servers are deployed on adjacent RSUs. With massive multiple-input multiple-output (MIMO) technology [18] [19], the RSUs can exchange data with multiple vehicles at the same time. The MEC servers have moderate computing capabilities and can only serve one computation task at a time. The MEC servers can communicate with the vehicles through the wireless channels of the RSUs to provide computation offloading services. For ease of presentation, each MEC server and its connected RSU is defined as a service node.

These distributed infrastructures can provide low-latency and ubiquitously available computation offloading services for the vehicles. However, these infrastructure are heterogeneous. The MEC servers have different computing capabilities, and the RSUs have different wireless coverage areas due to the assortment of radio power and communication environment. In order to efficiently utilize network resources and improve system performance, a software defined network (SDN) architecture is employed to support cooperation between the infrastructures. The data plane includes the radio access networks and the underlying infrastructure. A virtual machine is deployed on each MEC server, and together they form the control plane at the edge of the access network. The proximity of the control plane can support real-time monitoring of the network environment and quick response

TABLE 1. Notation definitions.

Notation	Description
N, n	set and number of vehicles
v_i	computation task of vehicle i
d_i, w_i, p_i^d, e_i	input data, workload, unit delay cost and expected delay of task v_i
f_i	local computing capability of vehicle i
M, m	set and number of service nodes
S_j	RSU/MEC server (service node) j
B_j, F_j, p_j^c	uplink rate per vehicle, computing capability and unit operating cost of service node S_j
t_{i0}	local execution time of task v_i
$t_{ij}^a, t_{ij}^u, t_{ij}^w, t_{ij}^e$	access time, upload time, wait time and execution time of task v_i on service node S_j
t_{ij}	completion time of offloading task v_i to S_j
a_{ij}	availability of service node S_j to vehicle i
U_{ij}	offloading utility of offloading task v_i to S_j
C_i^L	local execution cost of task v_i
C_{ij}^O	offloading cost of offloading task v_i to S_j

to the vehicle's offloading request. Several key functional modules are embedded in the control plane.

Information collection: this module collects essential information such as the offloading requests of on-vehicle applications and the available resources of service nodes. It also senses the location, mobility and accessible RSUs of each vehicle in real time.

Decision making: based on the latest perceived information, this module determines the vehicles' task offloading strategies and the service nodes' task scheduling strategies. The method of obtaining the strategies will be described in detail in Section IV.

Strategy distribution: this module is responsible for converting the strategies into executable commands and sending them to the corresponding vehicles and service nodes.

The interactions of the various components of the VEC system are depicted in Fig. 1.

B. TASK OFFLOADING MODEL

We now describe the task offloading model and define the offloading utility. For ease of reference, a summary of the key notations are presented in Table 1.

n vehicles on the road make computation offloading requests to the VEC infrastructure. The computation task of vehicle i is denoted by v_i, $i \in N$, $N = \{1, 2, \cdots, n\}$, which is atomic and cannot be divided into subtasks. Each task v_i is characterized by a tuple $\{d_i, w_i, p_i^d, e_i\}$, in which d_i is the size of the input data necessary for computation, w_i is the workload, i.e., the amount of computing resources required to accomplish the task, p_i^d is a vehicle parameter that specifies vehicle i's sensitivity to the increase of task delay [20], and e_i is the expected delay of the task. A task that is completed within the expected time e_i incurs a delay cost of 0.

Each RSU is equipped with an MEC server and is defined as a service node $S_j, j \in M$, $M = \{1, 2, \cdots, m\}$. The resource status of S_j is characterized by a tuple $\{B_j, F_j, p_j^c\}$, in which B_j is the uplink rate that the RSU assigns to each access vehicle, F_j is the computing capability of the MEC server, and p_j^c is the operating cost for S_j providing a unit computing resource.

Each vehicle can perform the task locally using its own device. Let f_i denote the local computing capability of vehicle i. Then the time for executing task v_i at vehicle i, denoted by t_{i0} is given by

$$t_{i0} = \frac{w_i}{f_i}. \tag{1}$$

By offloading a computation task to MEC servers, a vehicle can achieve shorter task execution time. Since the vehicle informs the control plane of its mobility, the control plane can know whether a particular service node is available to the vehicle. Let a_{ij} denote the availability of service node S_j to vehicle i. We have $a_{ij} = 1$ if vehicle i can access S_j, i.e., it is within the coverage of S_j. Otherwise, $a_{ij} = 0$. Hence, service node S_j can be selected to perform task v_i on condition that $a_{ij} = 1$. At this point, the completion time includes:

(1) t_{ij}^a: the time for vehicle i to access service node S_j. For S_j that has not established a communication connection with vehicle i, this time is usually affected by the relative position and speed between the two, with signal strength as an intermediate variable;

(2) t_{ij}^u: the time to upload the input data of task v_i to service node S_j through the uplink channel, given by

$$t_{ij}^u = \frac{d_i}{B_j}; \tag{2}$$

(3) t_{ij}^w: the time that task v_i waits for execution on service node S_j; and

(4) t_{ij}^e: the time for service node S_j to perform task v_i, given by

$$t_{ij}^e = \frac{w_i}{F_j}. \tag{3}$$

Note that we ignore the time required for the MEC servers to return the output results to the vehicles. This is because the size of the output result is much smaller than the input data in most cases, and the data rate of the downlink is much higher than that of the uplink [21] [22]. These factors essentially yield negligible time for transmitting the results.

Now, the completion time of offloading task v_i to service node S_j is simply the sum of the four time components, or

$$t_{ij} = t_{ij}^a + t_{ij}^u + t_{ij}^w + t_{ij}^e. \tag{4}$$

The corresponding offloading utility is defined as:

$$U_{ij} = p_i^d(t_{i0} - e_i) - (p_i^d(t_{ij} - e_i)^+ + p_j^c w_i), \tag{5}$$

where, $(a)^+ = max(a, 0)$. The first term refers to the delay cost for vehicle i to execute its computation task v_i, the value of which only depends on vehicle i. Thus, we denote it by C_i^L. The second term represents the sum of the delay cost and the operating cost for vehicle i to offload its computation task v_i to service node S_j, which we denote by C_{ij}^O.

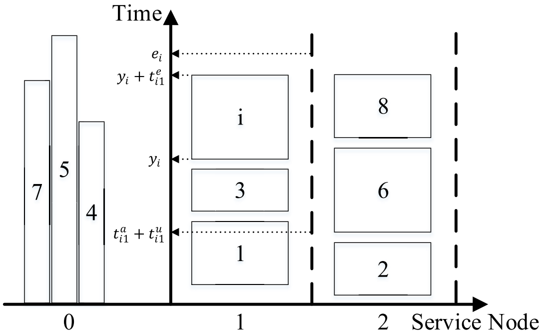


FIGURE 2. Schematic diagram of offloading scheme.

C. PROBLEM FORMULATION

To gain better insight into the task assignment and scheduling problem, we use a schematic diagram (see Fig. 2) to represent the offloading scheme. In the figure, the vertical axis represents time while the horizontal axis represents service node. We consider all vehicles as a single service node S_0. Then the local execution time t_{i0} of task v_i can be denoted as $t_{ij}, j = 0$. The service node set M is expanded to $M' = M + \{0\} = \{0, 1, \cdots, m\}$. Obviously, $t_{i0}^a = t_{i0}^u = t_{i0}^w = 0, p_0^c = 0$.

Each rectangle represents a computation task, with the number in a rectangle representing the task number. The height of a rectangle represents the time it takes to perform a task, depending on the workload of the task and the computing capability of the service node. It can be seen that the abscissa of a rectangle represents the service node performing the task, and the ordinates of the upper and lower sides are the start and completion times of the task execution, respectively. Therefore, given the position of each rectangle on the two-dimensional plane, the service node and start time corresponding to each task are determined, and its utility can be deduced. Assuming task v_i is executed by S_1, its corresponding arrival time, start time, completion time and expected delay are shown in the diagram. It should be noted that the tasks on S_0 are performed by different vehicles, and there is no dependence between these tasks, so the corresponding rectangles can be placed side by side. Other service nodes can only perform at most one task at a time, so the rectangles on the same service node must be placed sequentially. Thus, the task offloading problem in this paper is equivalent to solving the following special two-dimensional bin-packing problem: *How to arrange the rectangles of the tasks on the time-node plane while ensuring that the rectangles do not overlap each other, so that the total offloading utility of all tasks is maximized.*

We now formulate the task offloading problem. The following are the decision variables for the problem:

x_{ij}: if task v_i is assigned to service node S_j and $a_{ij} = 1$, then $x_{ij} = 1$, otherwise $x_{ij} = 0$; In particular, $a_{i0} = 1$ is always true;

y_i: the start time of task v_i; and

g_{ij}: if the start time of task v_j is greater than the completion time of task v_i, then $g_{ij} = 1$, otherwise $g_{ij} = 0$.

Then the total offloading utility becomes

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=0}^{m} x_{ij}U_{ij} &= \sum_{i=1}^{n} C_i^L \sum_{j=0}^{m} x_{ij} - \sum_{i=1}^{n}\sum_{j=0}^{m} x_{ij}C_{ij}^O \\ &= \sum_{i=1}^{n} C_i^L - \sum_{i=1}^{n}\sum_{j=0}^{m} x_{ij}(p_i^d(y_i + t_{ij}^e - e_i)^+ + p_j^c w_i). \quad (6)\end{aligned}$$

It can be seen that maximizing the total offloading utility is equivalent to minimizing the total offloading cost. Thus, the model Ω for the task offloading problem is:

(Ω) :

$$\min f = \sum_{i=1}^{n}\sum_{j=0}^{m} x_{ij}(p_i^d(y_i + t_{ij}^e - e_i)^+ + p_j^c w_i)$$

$$\text{s.t. C1}: \sum_{j=0}^{m} x_{ij} = 1, \forall i \in N$$

$$\text{C2}: y_i \geqslant \sum_{j=0}^{m} x_{ij}(t_{ij}^a + t_{ij}^u), \quad \forall i \in N$$

$$\text{C3}: y_i + \sum_{k=1}^{m} x_{ik}t_{ik}^e \leqslant y_j + L(1 - g_{ij}), \quad \forall i, j \in N, i \neq j$$

$$\text{C4}: x_{ik} + x_{jk} - g_{ij} - g_{ji} \leqslant 1, \quad \forall i, j \in N, i \neq j, k \in M$$

$$\text{C5}: x_{ij} \in \{0, a_{ij}\}, \quad \forall i \in N, j \in M'$$

$$\text{C6}: g_{ij} \in \{0, 1\}, \quad \forall i, j \in N, i \neq j$$

The objective function f represents the total offloading cost. Constraint C1 indicates that each task must be executed on exactly one service node. Constraint C2 ensures that the task can only be executed after the service node has obtained its input data, whereas constraint C3 stipulates that the start time of task v_j is later than the completion time of task v_i when $g_{ij} = 1$, where L is a large positive number. Constraint C4 ensures that the rectangles on the same service node (except S_0) do not overlap with each other, that is, when both x_{ik} and x_{jk} are 1, one of g_{ij} and g_{ji} must be 1. Finally, constraints C5 and C6 are the value constraints of the decision variables.

IV. OPTIMIZATION SOLUTION

By analyzing problem model Ω, we can see that two main steps are required to get a complete offloading scheme: 1) Determine the set of computation tasks performed by each service node; 2) Determine the execution order of the computation tasks on each service node. In our envisioned scenario, the number of possible offloading schemes is $\sum_{i=0}^{n} \binom{n}{i} i!(i+1)^{m-1}$. Therefore, the time complexity of problem Ω is $O(n!(n+1)^{m-1})$. Obviously, as the scale of the problem increases, the time to seek the global optimal solution increases sharply. Such long computation delays can affect the timeliness of the solution rendering them unusable.

In this paper, we propose a joint offloading decision and task scheduling algorithm (JODTS), a hybrid intelligent optimization algorithm combining partheno genetic algorithm (PGA) and heuristic rules. Unlike the traditional genetic algorithm, PGA does not use crossover operators, but instead performs genetic operations such as gene mutation only on one chromosome. Thus the genetic operation process is simplified, resulting in improvements in computational efficiency. In addition, PGA does not require the diversity of the initial population which effectively avoids premature convergence [23]. Compared with the standard PGA, JODTS has the following three main features: (i) In neighborhood search, the heuristic rules are used to construct neighborhood structures; (ii) Only a portion of the chromosome needs to be analyzed in decoding, reducing the decoding complexity; and (iii) An annealing selection method is introduced to enhance the local search ability of the algorithm.

A. CHROMOSOME CODING

A chromosome represents a task assignment scheme that determines the task set for each service node. Based on the characteristics of the problem, JODTS uses natural number coding method. The coding steps of a chromosome are as follows:

Step 1: Randomly assign each task to a service node that it can access. For example, if $a_{ij} = 1$, task v_i can be assigned to service node S_j (that is, set $x_{ij} = 1$);

Step 2: Make a sequence of service nodes for all tasks as an initial chromosome.

B. NEIGHBORHOOD SEARCH

For a given chromosome, a new neighborhood can be constructed by using genetic operators such as gene mutation. Gene mutation purports to change the gene value of one position in the chromosome to another value, that is, transferring the task from one service node (transfer-out node) to another (transfer-in node). However, constructing the neighborhood using random transformation will result in a large number of unreasonable task transfers, thus reducing the search efficiency. Therefore, we use heuristic rules to construct neighborhood. By setting the probability of each node being the transfer node, the sampling of more promising neighborhood is enhanced.

Let ξ_j and η_j denote the sum of the delay costs and the operating costs of the tasks assigned to service node S_j, respectively (for short, delay cost and operating cost of S_j). Then we have

$$\xi_j = \sum_{i=1}^{n} x_{ij} p_i^d (t_{ij} - e_i)^+; \tag{7}$$

$$\eta_j = p_j^c \sum_{i=1}^{n} x_{ij} w_i. \tag{8}$$

Let σ_{ij} denote the operating cost of executing task v_i on service node S_j. For a task assignment scheme, the higher the cost of a service node, the more feasible it is to optimize the scheme by transferring tasks on that node to other service nodes. It is easier to optimize the scheme by transferring tasks from the node with large cost to the node with small cost, than to the node with similar cost. Based on this intuitive understanding, the following heuristic neighborhood construction method is designed:

Step 1: Calculate ξ_j and η_j, $\forall j \in M'$;

Step 2: Calculate the probability p_j^{out} of service node S_j being the transfer-out node,

$$p_j^{out} = \frac{\xi_j + \eta_j}{\sum_{k=0}^{m}(\xi_k + \eta_k)}, \quad \forall j \in M'; \tag{9}$$

Step 3: Use roulette selection to select a service node and mark it as transfer-out node a. Randomly select a task v_i from node a, and calculate σ_{ij}, $\forall j \in M'$;

Step 4: Calculate the cost difference δ_{ij},

$$\delta_{ij} = \begin{cases} (\xi_a + \sigma_{ia} - \xi_j - \sigma_{ij})^+, & a_{ij} = 1 \\ 0, & a_{ij} = 0, \end{cases} \quad \forall j \in M'. \tag{10}$$

If δ_{ij} of all service nodes is 0, return to Step 3;

Step 5: Calculate the probability p_j^{in} of service node S_j being the transfer-in node,

$$p_j^{in} = \frac{\delta_{ij}}{\sum_{k=0}^{m} \delta_{ik}}, \quad \forall j \in M'. \tag{11}$$

Use roulette selection to select a service node and mark it as transfer-in node b;

Step 6: Change the service node for task v_i from a to b, thus obtaining a new neighborhood of the chromosome.

C. CHROMOSOME DECODING

The chromosome determines the task set for each service node, so their corresponding operating costs can be deduced. At this point, each service node can be seen as a single machine scheduling problem with the goal of minimizing the delay cost of the node. Let V_k denote the task set of service node S_k. The sub-problem model corresponding to S_k (denoted as Ω_k) has an objective function of:

$$\min f_k = \sum_{i \in V_k} p_i^d (y_i + t_{ik}^e - e_i)^+.$$

Its constraints can be obtained by simplifying the constraints of model Ω: ① Remove constraints related to service node selection, i.e., constraints C1 and C5; ② Remove useless variables and subscripts and replace N with V_k; Constraints C2 $\sim$ C4 are modified to:

$$y_i \geqslant t_{ik}^a + t_{ik}^u, \quad \forall i \in V_k$$
$$y_i + t_{ik}^e \leqslant y_j + L(1 - g_{ij}), \quad \forall i, j \in V_k, i \neq j$$
$$g_{ij} + g_{ji} = 1, \quad \forall i, j \in V_k, i \neq j$$

③ Constraint C6 is unchanged.

Through analysis, it can be seen that the time complexity of sub-problem Ω_k is $O(n_k!)$, where n_k is the number of elements in V_k. However, except for the initial chromosome, it is not necessary to solve all the sub-problems. Two properties are stated and proved in the appendix, based on which only a portion of the chromosome needs to be analyzed. Thus, the decoding process is as follows:

(1) In each neighborhood search, only the task sets of two service nodes (i.e., transfer-out node a and transfer-in node b) have changed. According to Property 1, only the task scheduling on these two nodes needs to be considered, while the other nodes can directly inherit the decoding scheme of the previous chromosome;

(2) If node a or node b is S_0, the decoding scheme of the node can be quickly obtained by reducing or adding the delay cost of transfer task v_i;

(3) For node b that is not S_0, its time-node plane is constructed based on the decoding scheme of the previous chromosome. It is judged whether there is a feasible position on the plane to place the rectangle corresponding to the transfer task and the delay time is 0. If there is, place the rectangle in that position. According to Property 2, the modified time-node plane is the decoding scheme of node b.

(4) For node a that is not S_0 and node b with no feasible position in (3), solve the sub-problem using a branch-and-bound algorithm such as [24].

D. ANNEALING SELECTION

The annealing selection refers to setting the probability of a child individual replacing its parent individual. Let A denote the parent individual, B denote the child individual, and T denote the temperature, then the replacement probability is:

$$p^r = \begin{cases} 1, & f(B) \leqslant f(A) \\ exp(\dfrac{f(A) - f(B)}{T}), & f(B) > f(A). \end{cases} \tag{12}$$

An initial temperature T_0 is set at the beginning, and the temperature is changed in each iteration using

$$T \leftarrow \alpha T,$$

where α is the attenuation coefficient of temperature, and $0 < \alpha < 1$. It can be seen that as the iteration number increases, the temperature gradually decreases, so the replacement probability in the second case also decreases. In the later stages, the replacement probability of a worse child individual is almost zero. The annealing selection not only accepts better individuals but also worse ones. This helps JODTS to jump out of local optimal solution traps and to obtain the global optimal solution.

E. ALGORITHM

Algorithm 1 shows a listing of the JODTS optimization process which integrates the various aspects described in this section. The algorithm terminates when no individual replacement occurs in successive l_0 iterations or the temperature drops to T_l. The solving time of the branch-and-bound algorithm for chromosome decoding is sensitive to the position of the optimal solution in the state space tree.

Algorithm 1 Joint Offloading Decision and Task Scheduling (JODTS) Algorithm

Input: task $\{v_i, i \in N\}$, service node $\left\{S_j, j \in M'\right\}$

Perform chromosome coding to generate initial individual A;
for *each service node* S_j **do**
  Count task set V_j and solve sub-problem Ω_j;
end
Get decoding scheme Ψ_A and objective value $f(A)$;
$T \leftarrow T_0, k \leftarrow 0$;
while $l < l_0 \& T > T_l$ **do**
  Determine the transfer-out node, transfer-in node and transfer task;
  Adjust the task assignment to obtain child individual B;
  Based on Ψ_A, partially decode B to obtain Ψ_B and $f(B)$;
  Calculate replacement probability p^r;
  Generate random number *rand*;
  if $rand \leqslant p^r$ **then**
    $A \leftarrow B, \Psi_A \leftarrow \Psi_B, f(A) \leftarrow f(B), l \leftarrow 0$;
  else
    $l \leftarrow l + 1$;
  end
  $T \leftarrow \alpha T$;
end

Output: offloading scheme Ψ_A

Therefore, in the worst case, the time complexity of JODTS is $O(n!)$.

V. SIMULATION EVALUATION

In this section, we implement the proposed JODTS on NS3, evaluate its performance in various situations and compare it with the baseline algorithms. The experiments are conducted on a personal computer with 3.1 GHz CPU and 4 GB memory.

A. SIMULATION SCENARIO

We simulate a road topology consisting of a two-lane road of length 1000 m. Five service nodes are distributed on both sides of the road, and the neighboring service nodes are set 200 m apart from each other. A random number of vehicles are evenly distributed on the road, and they travel towards the end at a speed of 100 km/h. The IEEE 802.11p standard is employed for the uplink data transmission, with a bandwidth of 10 MHz per channel. The vehicles send task data at the lowest data rate of 3 Mbps, as this provides the best communication reliability [25]. The maximum transmission range of each vehicle is set to 200 m. The detailed parameters and values used in the simulation environment are summarized in Table 2.

For the performance comparison, we consider four other algorithms as follows:

TABLE 2. Simulation parameter setting.

Parameter		Value	Description
n		10∼40	number of vehicles
m		5	number of service nodes
v_i	d_i	3 Mb	input data
	w_i	5 gigacycles	workload
	e_i	5 s	expected delay
	f_i	0.5 GHz	local computing capability
	p_i^d	1∼3/s	unit delay cost
S_j	B_j	3 Mbps	uplink rate per vehicle
	F_j	5 GHz	computing capability
	p_j^c	0.1∼0.5/gigacycle	unit operating cost

SCIP: this algorithm uses the mathematical solver SCIP [26], based on branch-and-bound, to solve for the global optimal solution of the task offloading problem model Ω.

Nearby offloading (NO): each vehicle offloads its task to the service node it is currently connected to, and each service node performs the arriving tasks using the traditional first come first service (FCFS) strategy.

Node selection optimization (NSO): an online scheduling algorithm that selects the service node that can maximize the offloading utility of the vehicle in the worst case (i.e., when the MEC server preferentially performs other assigned tasks).

Nearby offloading optimization (NOO): each vehicle offloads its task to the service node it is currently connected to. Unlike *NO* that uses the FCFS task execution strategy, the service nodes in *NOO* independently optimize the offloading decision and task scheduling, as in [27].

B. PERFORMANCE COMPARISON

1) ACCURACY AND CONVERGENCE

We randomly generate simulation cases for different numbers of vehicles and use *SCIP* and *JODTS* to solve them. Table 3 shows the offloading utility, relative error, and computation time of the two algorithms. The relative error is the percentage of the offloading utility of *JODTS* lower than that of *SCIP*.

Theoretically, the global optimal solution of the task offloading problem can be directly obtained by using *SCIP* to solve the model Ω. As can be seen from Table 3, for small-scale problems ($n \leqslant 20$), *SCIP* still takes a few seconds. The solution time increases dramatically as the problem scale increases. For example, the solution time for 35 vehicles is 2004.39 s. What's worse, when the number of vehicles reaches 40, *SCIP* cannot find a feasible solution due to memory overflow caused by excessive memory consumption. Therefore, *SCIP* is not suitable for making task offloading decisions in actual VEC systems. In contrast, *JODTS* finds the approximate optimal solution to the task offloading problem. The relative error between its solution and the optimal solution is less than 2%, showing a high solving accuracy. More importantly, although the solution time of *JODTS* also increases with the number of vehicles, it is only about 1 second at most. Its advantages of high accuracy and low complexity can meet the needs of practical applications.

In order to analyze the convergence of *JODTS*, Fig. 3 shows how the offloading utility of the algorithm changes

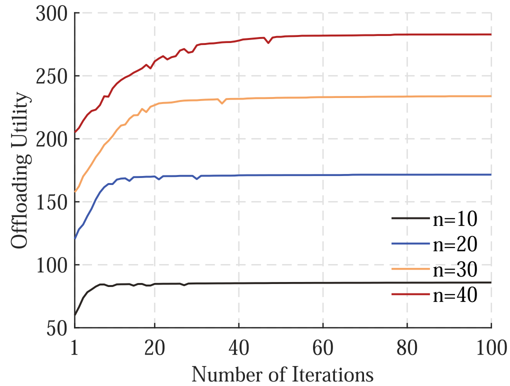


FIGURE 3. Convergence curves of JODTS.

with the number of iterations when the number of vehicles is 10, 20, 30 and 40, respectively. It can be seen that the offloading utility increases with the number of iterations. Due to the influence of annealing selection, fluctuations appear on each curve. After dozens of iterations, the offloading utility eventually stabilizes at a certain value. The more vehicles, the more iterations are required. For example, it takes about 40 iterations for 20 vehicles while 70 iterations for 40 vehicles. The results indicate that *JODTS* has good convergence properties in various situations.

2) EFFECT OF NUMBER OF VEHICLES

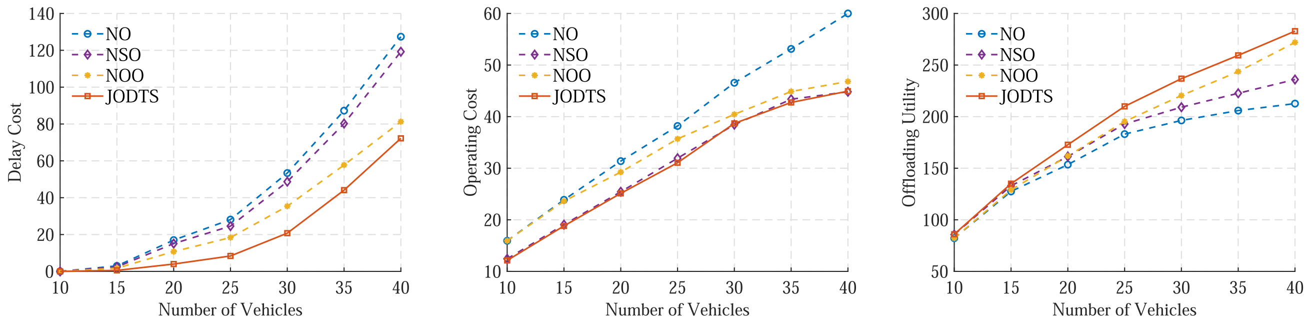
Now we evaluate the impact of the number of vehicles on system performance. We vary the number of vehicles from 10 to 40. The delay cost, operating cost, and offloading utility of *JODTS* and three other algorithms (*NO*, *NSO* and *NOO*) are shown in Fig. 4.

Fig. 4a shows that when the number of vehicles is small (less than 15), the delay cost of each algorithm is almost zero. This is because most tasks can be completed within their expected delay by using the service nodes. However, as the number of vehicles increases, the delay cost increases dramatically. This is due to the fact that as more vehicles offload tasks to the service nodes, the average task completion time becomes longer. Dominated by both the number of vehicles and the completion time of tasks, the delay cost increases exponentially. In terms of the operating cost, we can see in Fig. 4b that the operating cost of each algorithm increases almost linearly. When the number of vehicles exceeds 30, the growth rate of algorithms other than *NO* gradually decreases. This is because *NO* offloads all tasks to the service nodes, while other algorithms let some vehicles perform their own tasks locally. In Fig. 4c, the offloading utility of all algorithms increases with the number of vehicles. The service nodes can help most vehicles significantly reduce their task completion time. Therefore, despite the monetary cost of using the computing resources of service nodes, the vehicles will benefit more from task offloading. However, the rate of utility increase diminishes for all the algorithms as the number of vehicles increases. The increase in delay cost reduces the offloading utility that each vehicle can get.

Since *NO* and *NSO* do not optimize the execution order of tasks, tasks with higher unit delay costs may be executed

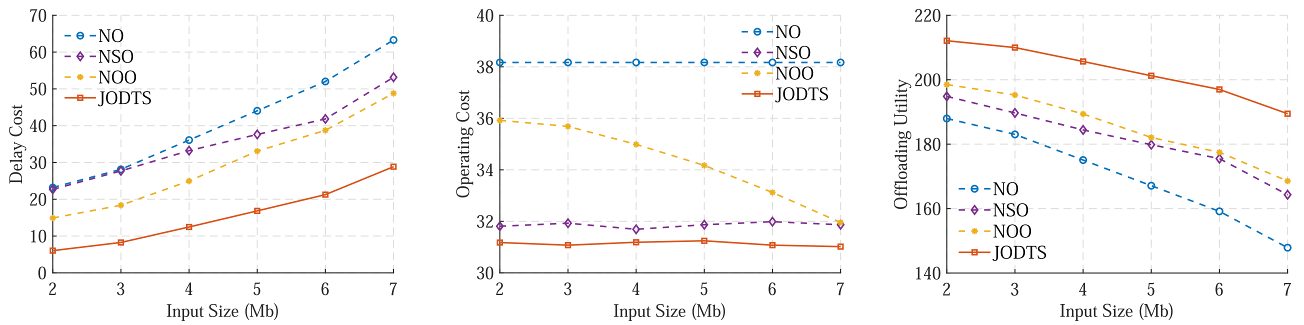
TABLE 3. SCIP vs JODTS.

n	SCIP		JODTS		
	Offloading Utility	Computation Time (s)	Offloading Utility	Computation Time (s)	Relative Error
10	86.00	1.15	85.82	0.15	0.21%
15	135.01	4.42	134.46	0.26	0.41%
20	172.75	14.50	171.54	0.41	0.70%
25	209.94	74.62	207.62	0.62	1.11%
30	236.81	362.72	233.82	0.86	1.26%
35	259.40	2004.39	255.69	1.17	1.43%
40	-	-	277.84	1.53	-



(a) Task delay cost. (b) MEC server operating cost. (c) Offloading utility.

FIGURE 4. The task offloading under different numbers of vehicles.



(a) Task delay cost. (b) MEC server operating cost. (c) Offloading utility.

FIGURE 5. The task offloading under different task input sizes.

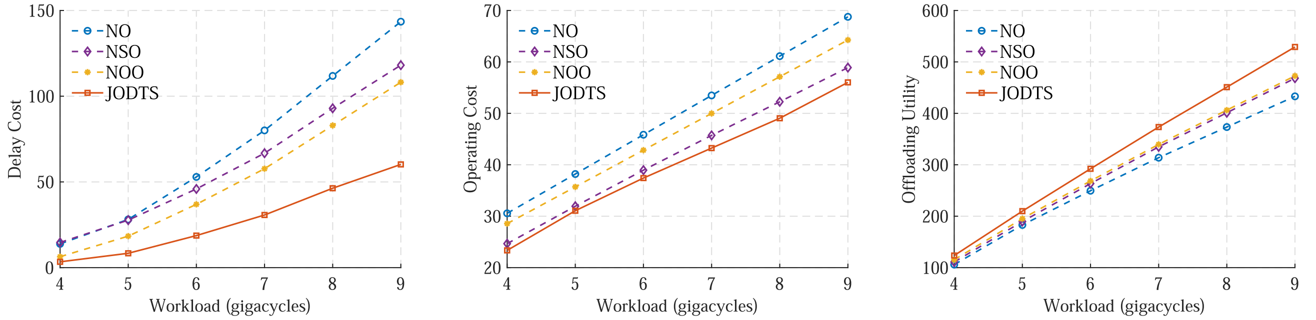
after those with lower costs, resulting in higher total delay costs. It can be seen that *NO* always performs the worst, with the highest delay cost and operating cost, and the lowest offloading utility. By properly offloading some tasks and arranging their execution order, *NOO* achieves a lower delay cost than *NSO*, but is not as good as the latter in terms of operating cost. When the number of vehicles is large (greater than 25), the offloading utility of *NOO* is higher than that of *NSO*, because in *NSO*, the significant increase in delay cost cancels the savings in operating cost. Jointly considering the offloading and node decision of computation tasks and the task execution order of MEC servers, *JODTS* achieves the lowest delay cost and operating cost, so its offloading utility is the highest.

3) EFFECT OF TASK PROFILE

Here, we evaluate the system performance under different task profiles in terms of input size d_i and workload w_i. The number of vehicles is 25. Fig. 5 and Fig. 6 show the delay cost, operating cost, and offloading utility of the four algorithms with different values of d_i and w_i, respectively.

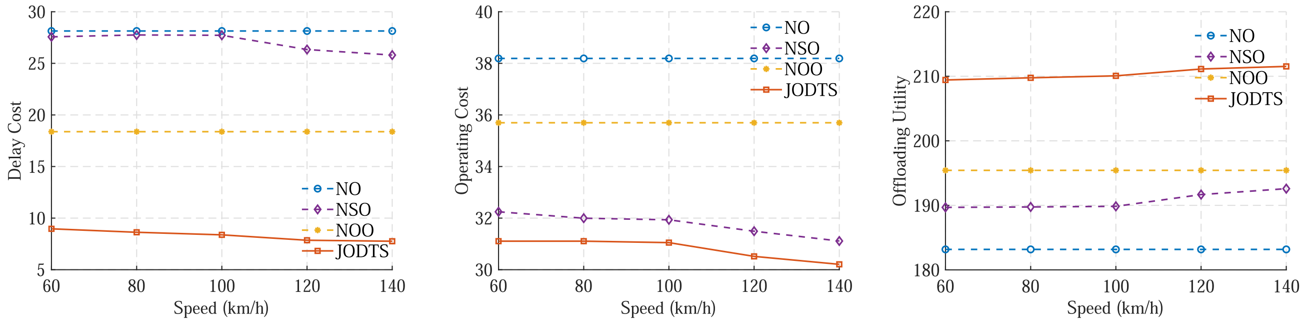
Fig. 5a shows that the delay costs of *JODTS* and the other algorithms increase with the task input size due to the fact that a larger input size entails a longer upload time, resulting in a longer task completion time. However, the growth rate of the delay costs is relatively stable. This is because the upload of one task and the execution of another task can be performed at the same time, so the increase in upload time only affects the first few tasks on the service nodes. In Fig. 5b, since each task is offloaded to its currently connected service node and the workload remains the same, the operating cost of *NO* has not changed. The operating cost of *NOO* decreases as the input size increases, because the increase in task completion time causes the service nodes to reject offloading requests for more vehicles. The operating costs of *NSO* and *JODTS* are kept at a low level with little change. In Fig. 5c, the offloading utility of each algorithm decreases as the task input size increases, as the consequence of rising delay cost.

Fig. 6a shows that the delay cost of each algorithm increases with the task workload. The higher the workload of a task, the longer it takes to perform the task on the vehicle or on a service node. Moreover, each task has to wait for



(a) Task delay cost. (b) MEC server operating cost. (c) Offloading utility.

FIGURE 6. **The task offloading under different task workloads.**



(a) Task delay cost. (b) MEC server operating cost. (c) Offloading utility.

FIGURE 7. **The task offloading under different vehicle speeds.**

its predecessor to complete before starting execution, so an increase in the execution time of a task affects not only the completion time of the task itself, but also the completion time of its subsequent tasks. As a result, the delay cost increases exponentially. In Fig. 6b, the operating cost of each algorithm increases linearly due to the fact that higher workloads lead to more computing resource consumption on the service nodes. Although both the delay cost and the operating cost are increasing, the offloading utility in Fig. 6c rises dramatically. This is because a higher workload entails a longer local execution time, allowing the vehicle to benefit more from offloading its task to the service nodes.

From the results shown in Fig. 5c and Fig. 6c, we can draw the following conclusion: *computation tasks with small input sizes and high workloads are more preferable to be offloaded than those with large input sizes and low workloads*. The task offloading varies more with the workload than with the input size.

4) EFFECT OF VEHICULAR MOBILITY

Next, we examine the impact of vehicle speed on system performance. The number of vehicles is 25. Fig. 7 shows the delay cost, operating cost, and offloading utility of the four algorithms with different speeds. It can be seen that when the vehicles move faster, the delay costs and operating costs of *JODTS* and *NSO* decrease, and the offloading utility increases. A vehicle can only reliably access a service node when the radio signal strength is strong enough. At low speeds, it takes a long time for a vehicle to approach a distant service node, so its task is only offloaded to the service node it is currently connected to. As the speed increases, the vehicle's access time to distant service nodes gradually decreases. Therefore, these two algorithms can make better offloading decisions because they have more service node options for offloading tasks. Since *JODTS* fully considers the task offloading decision and server execution time allocation, its performance is significantly better than *NSO* which only considers the selection of service nodes. On the other hand, *NO* and *NOO* are not sensitive to the increment of speed because they always offload each task to the service node it is currently connected to.

TABLE 4. **The ratio of offloading utility to local execution cost in different situations.**

Ratio (%) / Algorithm / Situation	NO	NSO	NOO	JODTS
number of vehicles (30)	66.27	70.59	74.41	80.00
input size (6 Mb)	63.83	70.39	71.18	80.01
workload (8 gigacycles)	68.34	73.42	74.36	82.54
speed (120 km/h)	73.42	76.82	78.33	84.62

5) EFFECT COMPARISON

In order to show the effects of *JODTS* and other algorithms more clearly, Table 4 lists the evaluation results of each algorithm in some simulation experiments, where the evaluation index is the ratio of the offloading utility of the algorithm to the local execution cost of the tasks. It can be seen that task offloading reduces the computational burden on the vehicles. Consistent with the previous simulation results, *JODTS* performs best in all situations. It can reduce the execution cost by more than 80%, which is about 15% higher than baseline algorithms.

VI. CONCLUSION

In this paper, we consider multi-user, multi-server VEC scenarios and study the task offloading algorithm that jointly optimizes the offloading decision of computation tasks and the execution order of MEC servers. We construct the task offloading model and define the offloading utility of the vehicles. Then we formulate the task offloading problem as a mixed integer non-linear programming problem with the goal of maximizing the system offloading utility. As the problem is hard to solve, we propose a hybrid intelligent optimization algorithm that combines PGA and heuristic rules to obtain an approximate optimal solution with low time complexity. The simulation results prove the accuracy of the proposed algorithm. Compared with the baseline algorithms, the algorithm effectively improves the offloading utility of the VEC system and is suitable for task offloading in various situations.

In this paper, we focus on the impact of task offloading decision and execution scheduling on computation offloading services, while neglecting other factors such as outdated information and wireless channel fading. In our future work, we will extend the proposed algorithm to handle computation offloading in environments where the characteristic parameters of the tasks and VEC network are uncertain. In addition, we also plan to investigate the use of relay nodes for transmitting task data and results in a reliable and timely manner.

APPENDIX

Property 1: For a given task offloading problem, Ψ_A^* is an optimal scheme for code A. If the task sets of service nodes $S_{M_1}, S_{M_2}, \cdots, S_{M_k} (1 \leqslant k \leqslant m+1)$ in another code B are the same as those in code A, then there is at least one optimal scheme Ψ_B^{**} for code B, in which the scheduling of service nodes $S_{M_1}, S_{M_2}, \cdots, S_{M_k}$ is the same as in scheme Ψ_A^*.

Proof: Let Ψ_B^* be an optimal scheme for code B. The delay cost of service node S_{M_k} in scheme Ψ_A^* and Ψ_B^* is marked as $\xi_{AM_k}^*$ and $\xi_{BM_k}^*$, respectively. In Ψ_A^* and Ψ_B^*, the delay cost of each service node is minimized. Thus, when the task sets of service node S_{M_k} in code A and code B are the same, $\xi_{AM_k}^* = \xi_{BM_k}^*$. Therefore, by replacing the scheduling of service nodes $S_{M_1}, S_{M_2}, \cdots, S_{M_k}$ in Ψ_B^* with the scheduling in Ψ_A^*, another optimal scheme Ψ_B^{**} for code B can be obtained. □

Property 2: For a given task offloading problem, Ψ_A^* is an optimal scheme for code A. The task sets of service node S_k in code A and code B are V_{Ak} and V_{Bk}, respectively. If ① $V_{Ak} \subset V_{Bk}$; ② there is a feasible scheme Ψ_B' for code B, in which the start time of the tasks belonging to V_{Ak} is the same as in scheme Ψ_A^*, and the delay time of the tasks belonging to $V_{Bk} \backslash V_{Ak}$ is 0; then there is at least one optimal scheme Ψ_B^{**} for code B, in which the scheduling of service node S_k is the same as in scheme Ψ_B'.

Proof: Let Ψ_B^* be an optimal scheme for code B. The delay cost of service node S_k in scheme Ψ_A^* and Ψ_B^* is marked as ξ_{Ak}^* and ξ_{Bk}^*, respectively, and in scheme Ψ_B' it is marked as ξ_{Bk}'. Then we prove the following conclusions: ① $\xi_{Bk}^* \geqslant \xi_{Bk}'$; ② $\xi_{Bk}^* \leqslant \xi_{Bk}'$.

① The task sets of service node S_k in scheme Ψ_B' is V_{Bk}. $V_{Ak} \subset V_{Bk}$, so there are two kinds of tasks in V_{Bk}: the tasks belonging to V_{Ak}, and the tasks belonging to $V_{Bk} \backslash V_{Ak}$. For the former, the start time of these tasks is the same as in scheme Ψ_A^*, so the total delay cost is also the same. For the latter, the delay time of these tasks is 0, so the total delay cost is 0. Add them together and we can see that the total delay cost of the tasks belonging to V_{Bk} in scheme Ψ_B' is equal to that of the tasks belonging to V_{Ak} in scheme Ψ_A^*, that is, $\xi_{Bk}' = \xi_{Ak}^*$. From $V_{Ak} \subset V_{Bk}$, it is known that $\xi_{Bk}^* \geqslant \xi_{Ak}^*$, so $\xi_{Bk}^* \geqslant \xi_{Bk}'$.

② Ψ_B^* is an optimal scheme for code B, and Ψ_B' is a feasible scheme. Obviously, $\xi_{Bk}^* \leqslant \xi_{Bk}'$.

From the conclusions ① and ②, it can be shown that $\xi_{Bk}^* = \xi_{Bk}'$. Therefore, by replacing the scheduling of service node S_k in Ψ_B^* with the scheduling in Ψ_B', another optimal scheme Ψ_B^{**} for code B can be obtained. □

REFERENCES

[1] H. Sedjelmaci, I. B. Jemaa, M. Hadji, and A. Kaiser, "Security framework for vehicular edge computing network based on behavioral game," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Abu Dhabi, United Arab Emirates, Dec. 2018, pp. 1–6.

[2] L. Li, H. Zhou, S. X. Xiong, J. Yang, and Y. Mao, "Compound model of task arrivals and load-aware offloading for vehicular mobile edge computing networks," *IEEE Access*, vol. 7, pp. 26631–26640, 2019.

[3] Q. Liu, Z. Su, and Y. Hui, "Computation offloading scheme to improve QoE in vehicular networks with mobile edge computing," in *Proc. 10th Int. Conf. Wireless Commun. Signal Process. (WCSP)*, Hangzhou, China, Oct. 2018, pp. 1–5.

[4] L. Tang and S. He, "Multi-user computation offloading in mobile edge computing: A behavioral perspective," *IEEE Netw.*, vol. 32, no. 1, pp. 48–53, Jan. 2018.

[5] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint offloading and resource allocation in vehicular edge computing and networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Abu Dhabi, United Arab Emirates, Dec. 2018, pp. 1–7.

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

[7] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 16, no. 8, pp. 4924–4938, Aug. 2017.

[8] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.

[9] J. Xu, L. Chen, and S. Ren, "Online learning for offloading and autoscaling in energy harvesting mobile edge computing," *IEEE Trans. Cogn. Commun. Netw.*, vol. 3, no. 3, pp. 361–373, Sep. 2017.

[10] S. Yu, R. Langar, X. Fu, L. Wang, and Z. Han, "Computation offloading with data caching enhancement for mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 11098–11112, Nov. 2018.

[11] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619–1632, Aug. 2018.

[12] J. Zhou, F. Wu, K. Zhang, Y. Mao, and S. Leng, "Joint optimization of offloading and resource allocation in vehicular networks with mobile edge computing," in *Proc. 10th Int. Conf. Wireless Commun. Signal Process. (WCSP)*, Hangzhou, China, Oct. 2018, pp. 1–6.

[13] Y. Cui, Y. Liang, and R. Wang, "Resource allocation algorithm with multi-platform intelligent offloading in D2D-enabled vehicular networks," *IEEE Access*, vol. 7, pp. 21246–21253, 2019.

[14] Q. Qi, J. Wang, Z. Ma, H. Sun, Y. Cao, L. Zhang, and J. Liao, "Knowledge-driven service offloading decision for vehicular edge computing: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4192–4203, May 2019.

[15] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint load balancing and offloading in vehicular edge computing and networks," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4377–4387, Jun. 2019.

[16] P. Liu, J. Li, and Z. Sun, "Matching-based task offloading for vehicular edge computing," *IEEE Access*, vol. 7, pp. 27628–27640, 2019.

[17] Y. Sun, X. Guo, J. Song, S. Zhou, Z. Jiang, X. Liu, and Z. Niu, "Adaptive learning-based task offloading for vehicular edge computing systems," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3061–3074, Apr. 2019.

[18] L. N. Ribeiro, S. Schwarz, M. Rupp, and A. L. F. De Almeida, "Energy efficiency of mmWave massive MIMO precoding with low-resolution DACs," *IEEE J. Sel. Top. Signal Process.*, vol. 12, no. 2, pp. 298–312, May 2018.

[19] S. Schwarz, M. Rupp, and S. Wesemann, "Grassmannian product codebooks for limited feedback massive MIMO with two-tier precoding," *IEEE J. Sel. Topics Signal Process.*, vol. 13, no. 5, pp. 1119–1135, Sep. 2019.

[20] K. Zhang, Y. Mao, S. Leng, S. Maharjan, and Y. Zhang, "Optimal delay constrained offloading for vehicular edge computing networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Paris, France, May 2017, pp. 1–6.

[21] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, Mar. 2018.

[22] S.-W. Ko, K. Han, and K. Huang, "Wireless networks for mobile edge computing: Spatial modeling and latency analysis," *IEEE Trans. Wireless Commun.*, vol. 17, no. 8, pp. 5225–5240, Aug. 2018.

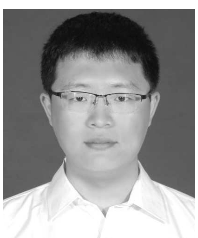
[23] H. Zhou, M. Song, and W. Pedrycz, "A comparative study of improved GA and PSO in solving multiple traveling salesmen problem," *Appl. Soft Comput.*, vol. 64, pp. 564–580, Mar. 2018.

[24] J. M. Valente and R. A. Alves, "An exact approach to early/tardy scheduling with release dates," *Comput. Oper. Res.*, vol. 32, no. 11, pp. 2905–2917, Nov. 2005.

[25] F. Lyu, N. Cheng, H. Zhou, W. Xu, W. Shi, J. Chen, and M. Li, "DBCC: Leveraging link perception for distributed beacon congestion control in VANETs," *IEEE Internet Things J.*, vol. 5, no. 6, pp. 4237–4249, Dec. 2018.

[26] A. Gleixner, L. Eifler, and T. Gally, "The SCIP optimization suite 5.0," Zuse Inst. Berlin, Berlin, Germany, ZIB-Rep. 17-61, Dec. 2017. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:0297-zib-66297

[27] Z. Cai, Z. Zhang, X. Chen, and W. Wang, "Task assignment with precedence constraint over networks: A case study of computation-communication convergence," in *Proc. 10th Int. Conf. Wireless Commun. Signal Process. (WCSP)*, Hangzhou, China, 2018, pp. 1–6.



JIANAN SUN received the B.S. degree in communications engineering from Beijing Jiaotong University, Beijing, China, in 2013, where he is currently pursuing the Ph.D. degree with the School of Electronic and Information Engineering. His research interests include software-defined networking, mobile edge computing, and vehicular networks.

QING GU (Member, IEEE) received the Ph.D. degree in intelligent traffic engineering from Beijing Jiaotong University, Beijing, China, in 2014. She is currently an Assistant Professor with the School of Mechanical Engineering, University of Science and Technology Beijing, Beijing. Her research interests focus on systems modeling, control, and optimization with application in intelligent transportation systems.

TAO ZHENG (Member, IEEE) received the B.S. degree in communications engineering and the Ph.D. degree in communication and information systems from Beijing Jiaotong University, Beijing, China, in 2006 and 2014, respectively. He was a Visiting Researcher with the ABB Corporate Research, Sweden, from September 2012 to October 2013, where he joined the industrial communication and embedded systems. He is currently an Associate Professor with Beijing Jiaotong University. His specific areas of research interest mainly focus on wireless communication technology in industrial network and the Internet of Things. He is also a member of the IEEE Industrial Electronics Society. He also serves as a Technical Reviewer for several journals, including the IEEE Transactions on Industrial Electronics (TIE) and the IEEE Transactions on Industrial Informatics (TII).

PING DONG (Member, IEEE) received the Ph.D. degree in communication and information systems from Beijing Jiaotong University, Beijing, China, in 2009. He has worked as a Visiting Scholar with Temple University, USA, for one year, from 2015 to 2016. He currently is currently a Professor with the School of Electronic and Information Engineering, Beijing Jiaotong University. He has published more than 50 research articles in the areas of computer networks. His research interests include the mobile Internet, software-defined networking, and network security. He also serves as a Technical Reviewer for several journals, including the IEEE Transactions on Vehicular Technology and the IEEE Wireless Communications.

ALVIN VALERA (Member, IEEE) received the B.Sc. degree in computer engineering from the University of the Philippines, in 1998, and the M.Sc. degree in computer science and Ph.D. degree in electrical and computer engineering from the National University of Singapore, in 2003 and 2015, respectively. He is currently a Senior Lecturer with the School of Engineering and Computer Science, Victoria University of Wellington. His current research interests are in the development of protocols and architectures for the Internet of Things (IoT), edge/fog computing, and resource-constrained information-centric networks.

YAJUAN QIN received the B.S. and M.S. degrees in radio technology from the University of Electronic Science and Technology of China, in 1985 and 1988, respectively, and the Ph.D. degree in communications engineering from the Beijing University of Posts and Telecommunications, in 2003. She was a Research Associate with CRL, Japan, in 2002. She joined Beijing Jiaotong University, in 2004, where she is currently a Professor. Her research interests are in the areas of computer networks and wireless communications.

• • •