Minimizing the Delay and Cost of Computation Offloading for Vehicular Edge Computing

Quyuan Luo®, Changle Li®, Senior Member, IEEE,
Tom H. Luan®, Senior Member, IEEE, and Weisong Shi®, Fellow, IEEE

Abstract—The development of autonomous driving poses significant demands on computing resource, which is challenging to resource-constrained vehicles. To alleviate the issue, Vehicular edge computing (VEC) has been developed to offload real-time computation tasks from vehicles. However, with multiple vehicles contending for the communication and computation resources at the same time for different applications, how to efficiently schedule the edge resources toward maximal system welfare represents a fundamental issue in VEC. This article aims to provide a detailed analysis on the delay and cost of computation offloading for VEC and minimize the delay and cost from the perspective of multi-objective optimization. Specifically, we first establish an offloading framework with communication and computation for VEC, where computation tasks with different requirements for computation capability are considered. To pursue a comprehensive performance improvement during computation offloading, we then formulate a multi-objective optimization problem to minimize both the delay and cost by jointly considering the offloading decision, allocation of communication and computation resources. By applying the game theoretic analysis, we propose a particle swarm optimization based computation offloading (PSOCO) algorithm to obtain the Pareto-optimal solutions to the multi-objective optimization problem. Extensive simulation results verify that our proposed PSOCO outperforms counterparts. Based on the results, we also present a comprehensive analysis and discussion on the relationship between delay and cost among the Pareto-optimal solutions.

Index Terms—Vehicular edge computing, computation offloading, multi-objective optimization, Pareto optimality, particle swarm

1 INTRODUCTION

The connected and automated vehicles (CAVs) have recently attracted increasing interests from both academia and industry [1], [2], [3]. By integrating the computation and communication, CAVs can support a variety of novel vehicular applications, such as autonomous driving, precise fleet management and real-time video analytics, which plays a crucial role toward a safer and more convenient road experience to people [4]. However, the powerful and resource-hungry applications always require intensive computation, which poses significant challenges on resource-constrained CAVs [5]. For example, 4 TB of data per day will be generated for the autonomous driving application according to Intel [6]. For Level-5 autonomous driving, 500+ TOPS¹ of processing capability are required [7].

The Vehicular Edge Computing (VEC) represents a practical and effective approach to support the large-scale CAVs [8], [9]. By offloading the computation-intensive tasks to roadside units (RSUs) equipped with edge servers (RES) [10], the VEC can significantly save the computation workload of vehicles yet reduce the processing latency of the computation tasks toward more efficient CAV applications. However, note that an RSU needs to serve multiple vehicles at the same time [11], how to effectively and economically use the limited edge resources and provide the maximal system welfare is a key issue.

A number of works have been developed on the allocation of edge resources in CAVs [11], [5], [10], [12], [13]. In the above works, the computation offloading is typically formulated as an optimization problem to either minimize the total processing delay or energy consumption or maximize the system utility. Distributed resource allocation methods (such as game-theoretic approach) or centralized resource allocation methods using optimization or heuristic algorithms are developed. More recently, artificial intelligence or deep reinforcement learning-based methods are proposed to solve the problems [14], [15]. The existing works mainly focus on one performance index. However, facing diverse applications, different requirements and system performance indexes during the offloading should be jointly considered. For example, to reduce delay, more cost would also be produced if tasks are offloaded to RSU to be processed, including the communication cost and computation cost. In this regard, it is imperative for CAVs to consider comprehensive performance and achieve multi-objective optimization.

As motivated, we aim to provide a detailed analysis of the delay and cost of computation offloading for VEC and

1 TOPS (Tera Operations Per Second) is the unit of processor computing capability. 1 TOPS represents one trillion (10¹²) operations per second by the processor.
minimize the delay and cost from the perspective of multi-objective optimization. To this goal, we first develop an offloading framework with communication and computation for VEC. Then, we formulate the joint resource allocation problem as a multi-objective optimization problem, considering optimizing both delay and cost during computation offloading. The formulated problem is also a mixed-integer non-linear programming (MINLP) problem, which cannot be solved effectively by traditional optimization methods. By utilizing the concept of Pareto optimality [16], we propose a particle swarm optimization based computation offloading (PSOCO) algorithm to solve the multi-objective optimization problem and obtain the Pareto-optimal solutions. The contributions of this paper are summarized as follows.

1) Model: We establish an offloading framework with communication and computation, where tasks with different computation capability requirements are considered. Under the framework, we elaborate on the detailed delay and cost of computation offloading.

2) Multi-Objective Optimization: Considering the comprehensive performance for CAVs, we formulate a multi-objective optimization problem to minimize both the delay and cost, where the offloading decision, local processing capability, communication resource, and RES processing capability are jointly considered.

3) Algorithm Design: To solve the formulated multi-objective optimization problem, which is also a MINLP problem, we introduce the concept of Pareto optimality. Motivated by the computational intelligence, we propose a particle swarm optimization based computation offloading (PSOCO) algorithm to obtain the Pareto-optimal solutions.

4) Validation and Discussion: Based on the real-world vehicular trace, extensive simulation results are provided to demonstrate the effectiveness of our proposed PSOCO over counterparts. Based on the obtained Pareto-optimal solutions, we provide a comprehensive analysis and discussion on the relationship between delay and cost among the Pareto-optimal solutions.

The remainder of this paper is organized as follows. We present the related work in Section 2. The system model is depicted in Section 3. The proposed PSOCO algorithm is presented in Section 4. Extensive simulation results are discussed in Section 5. We conclude this paper in Section 6.

2 RELATED WORK

In this section, we survey the existing literature on the allocation of communication and computation resources during computation offloading.

We first present the literature specifically on resource allocation for vehicles. Du et al. in [5] exploit Lyapunov optimization theory and propose a DDORV algorithm to minimize the cost on the vehicle side and the RSU side, respectively. Considering the mobility of vehicles, Zhang et al. in [10] propose a predictive-mode transmission scheme to minimize offloading cost, by focusing on both edge server selection and transmission management. In [13], they further propose to use backup computing servers to assist the mobile edge computing (MEC) server. And a Stackelberg game-based method is adopted to maximize the utilities on both the vehicle side and the MEC server side. To reduce latency, Liu et al. in [12] propose a distributed computation offloading scheme through formulating the computation offloading decision-making problem as a multi-user game. The Nash equilibrium of the game is further proved.

More recently, some marvellous works adopt machine learning-based methods in this area. Dai et al. [1] propose an architecture that dynamically allocates computation and caching resources. Based on the architecture, a deep reinforcement learning-based method is exploited to maximize system utility. Zhang et al. in [15] utilize the cognitive radio (CR) to alleviate the spectrum scarcity problem during computation offloading. To reduce transmission costs among vehicle-to-infrastructure (V2I) and vehicle-to-vehicle (V2V) communications, they propose a deep Q-learning method to schedule the communication modes and resources. Different from traditional works that study communication, caching, and computation technologies separately, He et al. in [14] propose an integrated framework that can orchestrate the three aspects dynamically. To solve the joint optimization problem, they utilize a deep reinforcement learning method to maximize the reward function, which is defined as the comprehensive revenue from communication, caching, and computing.

In addition to literature specifically for vehicles, other researches on MEC resource allocation are also emerging. To reduce energy consumption during computation offloading, Zhang et al. in [17] propose a three-stage energy-efficient computation offloading scheme through priority assignment and type classification. You et al. in [18] consider the edge cloud with both infinite and finite computation capacities respectively and the access mode with both TDMA and OFDMA. Wang et al. in [19] introduce the wireless power transfer (WPT) method and propose a unified MEC-WPT design. Dinh et al. in [20] formulate a distributed computation offloading problem based on game theory and propose a model-free reinforcement learning method. To minimize both delay and energy consumption, Messous et al. in [21] and Dinh et al. in [22] both transfer the multi-objective optimization problem into a single-objective optimization problem by weighting coefficients. To minimize the costs on both the user side and service provider side, Kim et al. in [23] propose a dual-side optimization algorithm for MEC. To maximize the total revenue, Wang et al. in [24] formulate an optimization problem by jointly considering the offloading decision, resource allocation, and caching in heterogeneous wireless cellular networks and propose a distributed algorithm based on alternating direction method of multipliers (ADMM).

All those works above are marvellous solutions. They often try to formulate a single-objective optimization problem such as minimizing delay, cost, or energy consumption. As for the multi-objective optimization problem, they often adopt a weighted sum method and transfer it to a single single-objective optimization problem. Moreover, most of the existing computation offloading solutions do not consider the differential requirements of computation tasks. In light
This can be easily achieved by integrating network function virtualization (NFV) and software defined networking (SDN) technologies, by which various radio spectrum resources can be abstracted and sliced upon demand, and various physical resources can share spectrum resources. This represents the total bandwidths of uplink and downlink of V2I channels, respectively. By leveraging the non-orthogonal multiple access (NOMA) technique, which has been considered as a key enabling technique for 5G networks due to its potentially superior spectral efficiency [26], the communication resource is divided into resource blocks (RBs), expressed as $B_1 = \{1, 2, \ldots, L\}$ for uplink and $B_2 = \{1, 2, \ldots, M\}$ for downlink. It is noting that there is a control channel [15].

For ease of analysis, we consider the system to be quasi-static so that the topology and wireless channels keep unchanged during the task processing period [5]. We define $\lambda_n (0 \leq \lambda_n \leq 1)$ as the offloading decision variable of task $T_n$, which stands for the ratio of the amount of bits offloaded to RSU $\text{D}_n$. Accordingly, the amount of bits that would be offloaded to RSU is $\lambda_n D_n$ bits and that would be processed locally is $(1 - \lambda_n) D_n$ bits. In the following, we will elaborate on the local processing part and offloading part, respectively.

### 3.2 Task Processed Locally

We use $f_{n}^l$ $(0 \leq f_{n}^l \leq F_n)$ to denote the processing capability (in CPU cycles/s) at vehicle $n$ assigned for local computation, where $F_n$ is the maximum processing capability of vehicle $n$. The power consumption of vehicle $n$ is then

$$P_n = f_{n}^l * \rho_n * \frac{1}{\theta},$$

where $\rho_n$ is the price of transmission per bit data in uplink and downlink, $\theta$ is the path loss exponent, and $D_n$ is the distance between vehicle $n$ and RSU.

### 3.1 Offloading Framework With Communication and Computation for VEC

Fig. 1 shows the offloading framework for VEC. The road is partitioned into segments, and each is covered by a roadside unit (RSU) with a roadside edge server (RES). In this paper, we consider a coverage area of one RSU and a set of $N = \{1, 2, \ldots, N\}$ CAVs (hereinafter referred to as vehicle for short). Various task data would be generated from the onboard applications of vehicles for entertainment (e.g., face recognition and augmented reality) or safety (e.g., LiDAR and high-definition camera) purpose [25]. We denote the number of task types by $K$. The RSU can provide powerful computing capability due to the deployed RES. Each vehicle $n$ ($n \in N$) has a computation task $T_n$ to be processed. We use four items to describe $T_n$ as $T_n = \{D_n, \gamma_n, c_n, I_n\}$, where $D_n$ stands for the input data size of $T_n$, $\gamma_n$ is a ratio of output data size to input data size, $c_n$ stands for the processing density of task $T_n$, and $I_n$ is an indicator that stands for the type of task $T_n$. We assume that different types of tasks have different processing densities.

### 3.3 Computation Offloading

We use four items to describe $T_n$ as $T_n = \{D_n, \gamma_n, c_n, I_n\}$, where $D_n$ stands for the input data size of $T_n$, $\gamma_n$ is a ratio of output data size to input data size, $c_n$ stands for the processing density of task $T_n$, and $I_n$ is an indicator that stands for the type of task $T_n$. We assume that different types of tasks have different processing densities.

### 3.3 Computation Offloading

We use four items to describe $T_n$ as $T_n = \{D_n, \gamma_n, c_n, I_n\}$, where $D_n$ stands for the input data size of $T_n$, $\gamma_n$ is a ratio of output data size to input data size, $c_n$ stands for the processing density of task $T_n$, and $I_n$ is an indicator that stands for the type of task $T_n$. We assume that different types of tasks have different processing densities.

### 3.3 Computation Offloading

We use four items to describe $T_n$ as $T_n = \{D_n, \gamma_n, c_n, I_n\}$, where $D_n$ stands for the input data size of $T_n$, $\gamma_n$ is a ratio of output data size to input data size, $c_n$ stands for the processing density of task $T_n$, and $I_n$ is an indicator that stands for the type of task $T_n$. We assume that different types of tasks have different processing densities.

### 3.3 Computation Offloading

We use four items to describe $T_n$ as $T_n = \{D_n, \gamma_n, c_n, I_n\}$, where $D_n$ stands for the input data size of $T_n$, $\gamma_n$ is a ratio of output data size to input data size, $c_n$ stands for the processing density of task $T_n$, and $I_n$ is an indicator that stands for the type of task $T_n$. We assume that different types of tasks have different processing densities.

### 3.3 Computation Offloading

We use four items to describe $T_n$ as $T_n = \{D_n, \gamma_n, c_n, I_n\}$, where $D_n$ stands for the input data size of $T_n$, $\gamma_n$ is a ratio of output data size to input data size, $c_n$ stands for the processing density of task $T_n$, and $I_n$ is an indicator that stands for the type of task $T_n$. We assume that different types of tasks have different processing densities.

### 3.3 Computation Offloading

We use four items to describe $T_n$ as $T_n = \{D_n, \gamma_n, c_n, I_n\}$, where $D_n$ stands for the input data size of $T_n$, $\gamma_n$ is a ratio of output data size to input data size, $c_n$ stands for the processing density of task $T_n$, and $I_n$ is an indicator that stands for the type of task $T_n$. We assume that different types of tasks have different processing densities.

### 3.3 Computation Offloading

We use four items to describe $T_n$ as $T_n = \{D_n, \gamma_n, c_n, I_n\}$, where $D_n$ stands for the input data size of $T_n$, $\gamma_n$ is a ratio of output data size to input data size, $c_n$ stands for the processing density of task $T_n$, and $I_n$ is an indicator that stands for the type of task $T_n$. We assume that different types of tasks have different processing densities.

### 3.3 Computation Offloading

We use four items to describe $T_n$ as $T_n = \{D_n, \gamma_n, c_n, I_n\}$, where $D_n$ stands for the input data size of $T_n$, $\gamma_n$ is a ratio of output data size to input data size, $c_n$ stands for the processing density of task $T_n$, and $I_n$ is an indicator that stands for the type of task $T_n$. We assume that different types of tasks have different processing densities.

### 3.3 Computation Offloading

We use four items to describe $T_n$ as $T_n = \{D_n, \gamma_n, c_n, I_n\}$, where $D_n$ stands for the input data size of $T_n$, $\gamma_n$ is a ratio of output data size to input data size, $c_n$ stands for the processing density of task $T_n$, and $I_n$ is an indicator that stands for the type of task $T_n$. We assume that different types of tasks have different processing densities.

### 3.3 Computation Offloading

We use four items to describe $T_n$ as $T_n = \{D_n, \gamma_n, c_n, I_n\}$, where $D_n$ stands for the input data size of $T_n$, $\gamma_n$ is a ratio of output data size to input data size, $c_n$ stands for the processing density of task $T_n$, and $I_n$ is an indicator that stands for the type of task $T_n$. We assume that different types of tasks have different processing densities.
3.3 Task Offloaded to RSU

For the offloaded part of task $T_n$, there exist three procedures to accomplish the task computing, that is, $\lambda_n D_n$-bits of task $T_n$ should be first transmitted to RSU through V2I channels, then computed by RSU, and finally, RSU will return the result to vehicle $n$.

3.3.1 Task Transmission

We denote $\alpha_i (i \in B_1)$ as the uplink binary RB allocation indicator, where $\alpha_i = 1$ means $i$ is allocated to vehicle $n$, and $\alpha_i = 0$ otherwise. We denote $p_i$ as the uplink transmission power of vehicle $n$ over RB $l$, $h_i$ as the power gain between vehicle $n$ and RSU over RB $l$. Accordingly, the uplink data rate of vehicle $n$ over RB $l$ after performing successive interference cancellation (SIC)$^3$ can be formulated as $r_i = \frac{W}{T} \log_2 (1 + \frac{\alpha_i p_i h_i^2}{\sigma^2 + d_i^2})$, where $\sigma^2$ denotes the White Gaussian noise power over RB $l$ [30], $\xi_i = \sum_{i \in A \setminus \{n\}} h_i^2 < h_i^2$, and $d_i$ are the distance from vehicle $n$ to RSU and the path loss exponent, respectively. Through SIC technology, the interference signal from vehicle $i \in A \setminus \{n\}$ will be decoded and removed at the RSU side if $h_i^2 < h_i^2$ [31]. The uplink data rate from vehicle $n$ is then formulated as $R_i = \sum_{l \in B_1} r_i$. Accordingly, the uplink transmission delay and energy consumption for offloading $\lambda_n D_n$-bits of task $T_n$ are obtained by $t_i^{up} = \frac{\lambda_n D_n}{R_i}$ and $E_i^{up} = \sum_{l \in B_1} p_i^{up}$, respectively.

3.3.2 Task Computed by RSU

After $\lambda_n D_n$-bits of task $T_n$ is transmitted from vehicle $n$ to RSU, it will be computed by the deployed RES. We use $f_c (0 \leq f_c \leq F_c)$ to denote the assigned processing capability (in CPU cycles/s) from RES, where $F_c$ denotes the maximum processing capability of RSU. The power consumption of RES is then calculated as $p_n = k_e (f_c)^3$, where $k_e$ is a coefficient related to power in RES. For the $\lambda_n D_n$-bits of task $T_n$, its processing time is expressed as $t_i^{up} = \frac{\lambda_n D_n}{F_c}$. And the energy consumption of RES for computing the offloaded part of task $T_n$ is expressed as $E_v = k_e c_n \lambda_n D_n (f_c)^2$.

3.3.3 Result Return

We denote $\beta_n (m \in B_2)$ as the downlink binary RB allocation indicator, where $\beta_m = 1$ means RB $m$ is allocated to vehicle $n$, and $\beta_m = 0$ otherwise. We denote $t_n^{off}$ as the downlink transmission power of RSU sending back the computation result to vehicle $n$ over RB $m$, $h_n^m$ as the power gain between vehicle $n$ and RSU over RB $m$. Accordingly, the downlink data rate of vehicle $n$ for result return from RSU over RB $m$ can be formulated as $r_n^m = \frac{W}{T} \log_2 (1 + \frac{\alpha_n p_n h_n^m}{\sigma^2 + d_n^2})$, where $\sigma^2$ denotes the White Gaussian noise power over RB $m$, $\alpha_n = \sum_{i \not\in n \cap B_2} h_n^m < h_n^m$, $\theta_n = \frac{\alpha_n p_n h_n^m}{\sigma^2 + d_n^2}$ denotes the interference signal power from other vehicles over RB $m$. The downlink data rate from RSU is then formulated as $R_n^m = \sum_{m \in B_2} r_n^m$. Accordingly, the downlink transmission delay and energy consumption for result return can be obtained by $t_n^{off} = \frac{\lambda_n D_n}{R_n^m}$ and $E_n^{off} = \sum_{m \in B_2} p_n^{off}$, respectively.

Therefore, the total latency and energy consumption for the offloaded part of task $T_n$ are expressed as

$$t_n^{off} = t_i^{up} + t_n^{on} + t_n^{off} = \frac{\lambda_n D_n}{R_i} + \frac{\lambda_n D_n}{R_n^m} + \frac{\lambda_n D_n}{R_n^m}, \quad (1)$$

and

$$E_n^{off} = E_i^{up} + E_n^{off} + E_n^{off} = \sum_{l = 1}^L p_n^{up} + k_e c_n \lambda_n D_n (f_c)^2 + \sum_{m = 1}^M p_n^{off}, \quad (2)$$

respectively.

3.4 Problem Formulation

For a given task $T_n$, delay and cost would be produced to process it. For the delay aspect, it is determined by both the delays of processing the local part and the offloaded part. Accordingly, the total delay of processing $T_n$ can be expressed as $t_n = \max (t_i^{up}, t_n^{off})$. For the cost part, it is also determined by both the cost of processing the local part and the offloaded part. The former only includes the energy consumption for local computing while the latter includes three aspects: a) the energy consumption of transmitting and computing task $T_n$; b) the communication cost for using RBs; and c) the computing cost for RES computing the offloaded part of task $T_n$. Accordingly, the cost for processing task $T_n$ can be expressed as

$$U_n = U_i^{up} + U_n^{off} = \xi E_i^{up} + \xi (E_i^{up} + E_n^{off} + E_n^{off}) + \mu \lambda_n D_n + \nu r_n^{on} \lambda_n D_n + \rho c_n \lambda_n D_n,$$

where $\xi$ is a weighting coefficient indicating the energy consumption cost of one unit energy during task computing and transmission [23], $\mu$ and $\nu$ are coefficient $s$ indicating the communication cost required to transmit one unit of task data by using uplink and downlink RBs, respectively. And $\rho$ is a coefficient indicating the computing cost to execute one CPU cycle.

In this paper, we consider minimizing the delay and cost of all vehicles under the computation capability and communication resource limitations. To this end, the offloading decision variable, the local processing capability, the RES processing capability, the uplink binary RB allocation indicator, and the downlink binary RB allocation indicator need to be optimized. We denote $\Lambda = \{\lambda_1, \ldots, \lambda_N\}$, $f^c = \{f_1^c, \ldots, f_N^c\}$, $f^r = \{f_1^r, \ldots, f_N^r\}$, $\alpha = \{\alpha_1^r, \ldots, \alpha_N^r, \alpha_1^c, \ldots, \alpha_N^c\}$, $\beta = (\beta_1, \ldots, \beta_N)$.
\( \{p_1^1, \ldots, p_M^1, \ldots, p_1^N, \ldots, p_M^N \} \). Thus, the multi-objective optimization problem is formulated as

\[
\begin{align*}
\min_{\lambda, f^1, f^2, \alpha, \beta} & \quad t = \sum_{n=1}^{N} t_n \\
\min_{\lambda, f^1, f^2, \alpha, \beta} & \quad U = \sum_{n=1}^{N} U_n \\
\text{s.t.} & \quad 1-C1 : 0 \leq \lambda_n \leq 1, \forall n \in \mathcal{N}, \\
& \quad 1-C2 : 0 \leq f_n^1 \leq F_n, \forall n \in \mathcal{N}, \\
& \quad 1-C3 : 0 \leq f_n^2 \leq F_e, \forall n \in \mathcal{N}, \\
& \quad 1-C4 : \sum_{n=1}^{N} f_n^e \leq F_e, \\
& \quad 1-C5 : \alpha_n^l, \beta_n^l \in \{0, 1\}, \forall n \in \mathcal{N}, l = l_1, m = m_2, \\
& \quad 1-C6 : \sum_{n=1}^{N} \alpha_n^l \leq 1, \sum_{n=1}^{N} \beta_n^l \leq 1, \\
& \quad \\forall l \in \mathcal{L}_1, m \in \mathcal{L}_2,
\end{align*}
\]

(4)

where (1-C1) is the constraints on offloading decision variable; (1-C2), (1-C3) and (1-C4) are the processing capability constraints for vehicles and RES, where \( F_e \) denotes the maximum processing capability of RES at the RSU; (1-C5) are the binary constrains on uplink and downlink RB allocation indicators; (1-C6) ensures that each uplink RB and each downlink RB can be allocated to at most one vehicle.

Since the delay and energy consumption in downlink is much less than in uplink for most traditional computation-intensive tasks [18], [32]. Accordingly, for simplicity, we ignore the delay and energy consumption in downlink. Then Formulas (1) and (2) can be rewritten as

\[
t^\text{off}_n = t^\text{up}_n + t^e_n = \frac{\lambda_n D_n}{R_n^1} + \frac{e_n m_n D_n}{f_n^1},
\]

(5)

and

\[
E^\text{off}_n = E^\text{up}_n + E^e_n = \sum_{l=1}^{L} f_n^l p_n^l + k_e c_e \lambda_n D_n \left( f_n^l \right)^2,
\]

(6)

respectively. Accordingly, the problem in Formula (4) can be reformulated as

\[
\begin{align*}
\min_{\lambda, f^1, f^2, \alpha, \beta} & \quad t = \sum_{n=1}^{N} \max \left\{ \frac{c_n (1 - \lambda_n) D_n}{f_n^1}, \lambda_n D_n \left( \frac{1}{R_n^1} + \frac{c_n}{f_n^1} \right) \right\} \\
\min_{\lambda, f^1, f^2, \alpha, \beta} & \quad U = \sum_{n=1}^{N} \xi D_n \left\{ k_n c_n (1 - \lambda_n) \left( f_n^l \right)^2 + \lambda_n \sum_{l=1}^{L} p_n^l \right. \\
& \quad + k_e c_e \lambda_n \left( f_n^l \right)^2 \left. \right\} + \lambda_n D_n \left\{ \mu + v \gamma_n + p \epsilon_n \right. \\
& \quad \text{s.t.} \quad 2-C1 : 0 \leq \lambda_n \leq 1, \forall n \in \mathcal{N}, \\
& \quad 2-C2 : 0 \leq f_n^1 \leq F_n, \forall n \in \mathcal{N}, \\
& \quad 2-C3 : 0 \leq f_n^2 \leq F_e, \forall n \in \mathcal{N}, \\
& \quad 2-C4 : \sum_{n=1}^{N} f_n^e \leq F_e, \\
& \quad 2-C5 : \alpha_n^l \in \{0, 1\}, \forall n \in \mathcal{N}, l = \mathcal{L}_1, \\
& \quad 2-C6 : \sum_{n=1}^{N} \alpha_n^l \leq 1, \forall l \in \mathcal{L}_2,
\end{align*}
\]

where \( R_n^e \) denotes the uplink data rate from from vehicle \( n \) and is formulated as

\[
R_n^e = \sum_{l=1}^{L} \frac{W_1}{L} \log_2 \left( 1 + \frac{\alpha_n^l p_n^l h_n^l \sigma^l(d_n) \gamma}{\sum_{i \in \mathcal{N}, i \neq n} \alpha_i^l p_i^l h_i^l + \sigma^l(d_n) \gamma} \right) .
\]

(8)

### 4 Multi-Objective Optimization and Computation Offloading Algorithm

#### 4.1 Multi-Objective and Pareto Optimization

The purpose of Formula (7) is to minimize the cost and delay simultaneously, which is a multi-objective optimization problem. The improvement in the performance of one objective may cause a decrease in the performance of another objective during the optimization. Therefore, the two objectives cannot achieve their optimal values at the same time, a trade-off and compromise should be obtained between them. There is no optimal solution to Formula (7) but a set of optimal solutions, the elements of which are called Pareto-optimal solutions or non-dominated solutions [33].

**Definition 1.** Consider an optimization of a multi-criteria decision-making problem aiming to minimize \( \kappa \) objective functions \( f_i(x) \) \( (q = 1, 2, \ldots, \kappa) \), where \( x = [x_1, x_2, \ldots, x_t]^T \) is a vector of \( t \) decision variables. The set of all feasible solutions is denoted as \( \Omega \). Assume \( S_1 \in \Omega \) and \( S_2 \in \Omega \), it is said that solution \( S_1 \) dominates the other solution \( S_2 \) (denoted by \( S_1 > S_2 \)) if and only if \( \forall q : f_q(x_{S_1}) \leq f_q(x_{S_2}) \) and \( \exists k : f_k(x_{S_1}) < f_k(x_{S_2}) \), where \( f_q(x) \) are the objective functions and \( x_{S_1}, x_{S_2} \) are vectors of \( t \) variables representing \( S_1 \) and \( S_1 \), respectively [34].

**Definition 2.** All feasible solutions in search space \( \Omega \), which are not dominated (see Definition 1) by any other solution in the search space are called Pareto-optimal solutions. They form the so-called Pareto-optimal set (or Pareto-optimal front).

Based on Definitions 1 and 2, this paper aims to find the Pareto-optimal solutions for the formulated delay and cost minimization problem in Formula (7). Since the problem formulated in Formula (7) is a MINLP problem, it is hard to find the Pareto-optimal solutions through traditional optimization methods [35]. Inspired by the swarm intelligence, we resort to the particle swarm optimization (PSO) [36] method and propose a PSO based computation offloading (PSOCO) algorithm. In the following, we will give a brief...
introduction about PSO and elaborate on the PSOCO algorithm.

4.2 Particle Swarm Optimization

As a computational intelligence method, PSO is inspired by the swarm behavior of birds to search for food, in which each bird changes its search pattern by learning from its own and others’ experience [33]. PSO has been widely used in optimization problems in various fields due to its good ability to solve complex problems in multi-dimensional complex space [37]. Compared with other swarm intelligence algorithms, PSO has fewer parameters and is very suitable for multi-objective optimization problems [38]. More importantly, the PSO has a faster convergence speed than other population-based stochastic optimization methods (e.g., genetic algorithms) [39]. To describe the proposed PSOCO explicitly, we first introduce some terms about PSO.

1) **Particle**: The search space for solving optimization problems is compared to the flight space of birds during the foraging behavior, and each bird is abstracted into a particle without mass or volume. In this paper, a particle denotes a candidate solution of Formula (7).

2) **Particle swarm**: A particle swarm consists of several particles, and the number of particles denotes the size of the particle swarm.

3) **Solution encoding**: Solution encoding means how to represent the variables to be solved in Formula (7) by the particle. In this paper, we adopt the real encoding method.

4) **Fitness**: Fitness indicates the suitability of a particle for the solution and is presented by the objective function value. In this paper, fitness means the optimization objectives in Formula (7), i.e., the delay and cost.

5) **Fitness evaluation**: Fitness evaluation represents to calculate the fitness value (i.e., the value of the two objectives in Formula (7)) for each particle.

6) **Swarm updating**: Based on the best positions of individuals and swarm, particles update their speeds and positions. The swarm updating is completed after all particles are updated. Through swarm updating, the evolution of candidate solutions can be achieved.

7) **Boundary condition processing**: When the position and speed exceed the set value, the boundary condition processing can restrict particle’s position to a feasible space, which avoids the expansion and divergence of the swarm, and also avoids the blindly searching in a large range, thus improving the search efficiency.

4.3 Proposed PSOCO Algorithm

1) **Initialization**: Initialize iteration \( g = 0 \) and particle swarm \( S(g) \) randomly as \( S(g) = \{S_1(g), S_2(g), \ldots, S_s(g)\} \), where \( s \) denotes the swarm size, \( S_i(g) \) \( (1 \leq i \leq s) \) denotes a particle. Specifically, \( S_i(g) \) is represented by a set of \( \lambda, \alpha, f^l \) and \( f^s \), which can be defined as an array \( S_i(g) = [\lambda, \alpha, f^l, f^s] \), where \( d \) is the size of the array and is defined as \( d = 3 \times N + L \). In each \( S_i(g) \), the size of \( \lambda, f^l \) and \( f^s \) is \( N \), the size of \( \alpha \) is \( L \). And \( \lambda \) \( f^l \) and \( f^s \) are real coded. It is worth noting that the elements in \( \alpha \) are different from that in constraints (3-C5) and (3-C6). Here, the \( \alpha \) is defined as \( \alpha = \{\alpha' | l \in B_1, 1 \leq \alpha' \leq N\} \), where \( \alpha' \) is an integer to indicate the vehicle which obtains the RB \( l \).

In addition, we define four memory units \( P^{\text{indv}}, O^{\text{indv}}, P^{\text{glb}} \) and \( O^{\text{glb}} \) to represent the best positions of individuals, the best objectives of individuals, the best positions of swarm, and the best objectives of swarm, respectively. Specifically, \( P^{\text{indv}} \) and \( O^{\text{indv}} \) are expressed by two cell arrays with the size of \( s \), where each element in the two cell arrays represents the Pareto-optimal set and corresponding optimal objectives of a particle, respectively. \( P^{\text{indv}} \) and \( O^{\text{indv}} \) are formulated as

\[
P^{\text{indv}} = \{P_1^{\text{indv}}, P_2^{\text{indv}}, \ldots, P_s^{\text{indv}}\},
\]

\[
O^{\text{indv}} = \{O_1^{\text{indv}}, O_2^{\text{indv}}, \ldots, O_s^{\text{indv}}\}.
\]

\( P^{\text{glb}} \) and \( O^{\text{glb}} \) are two arrays, where each element in the two arrays represents one of the optimal Pareto-optimal set and corresponding optimal objectives of the swarm, respectively. For the sake of iteration, we assign the value of the initialized \( S(g) \) to \( P^{\text{indv}} \) and randomly select an element from \( P^{\text{indv}} \) and assign it to \( P^{\text{glb}} \). For the initialization of \( O^{\text{indv}} \) and \( O^{\text{glb}} \), we assign some values that are big enough (e.g., infinity) to them. We also initialize each particle’s speed \( V_i \), as \( V_i = \{v_{i1}, v_{i2}, \ldots, v_{id}\}, \forall i \leq t \leq s \).

2) **Mapping of particle representation to optimization variables**: This step is also called solution encoding. Map each element value \( e_{l} \) \( (1 \leq l \leq 3 \times N + L) \) of each \( S_i(g) \) in the particle swarm to the variables to be solved in Formula (7). Specifically, since the elements in \( \alpha, f^l \) and \( f^s \) are real coded, they can directly represent the variables of \( \alpha_n, f^l_n \) and \( f^s_n \) \( (n \in N) \). For the element \( \alpha^l \) in \( \alpha \), the corresponding relationship with \( \alpha^l_i \) in Formula (7) is expressed as

\[
\alpha^l_i = \begin{cases} 1, & \alpha^l = n, \forall i \in B_1, n \in N \smallskip \end{cases}
\]

\[0, \text{ otherwise}.\]

3) **Fitness evaluation**: Since we aim to minimize the cost and delay simultaneously, we regard both the cost and delay as the fitness functions. In order to find the Pareto-optimal solutions, we first calculate the fitness of each particle \( S_i(g) \) \( (1 \leq i \leq s) \) in \( S(g) \). Specifically, we use the variable values from the particle representation in step 2) to calculate the objectives of delay and cost in Formula (7), and obtained the fitness values \( O^{\text{delay}}(s) \) \( (1 \leq i \leq s) \) and \( O^{\text{cost}}(g_s) \), we use \( O_{g_s} \) to represent the pair of objectives. It can be defined as

\[
O_{g_s} = \{O^{\text{delay}}, O^{\text{cost}}\}.
\]

4) **Recording the best individuals and swarm**: After the two fitness values of each particle \( S_i(g) \) are obtained by step 3), we then compare them with the optimal objectives \( O^{\text{indv}} \) and \( O^{\text{glb}} \), respectively.

For the recording of the best individuals, we first calculate the size of \( P^{\text{indv}} \) by \( s^{\text{indv}} \), i.e., the number of optimal objectives of particle \( S_i(g) \). Then, for any Pareto-optimal solution \( P^{\text{indv}}_{i,k} \) \( (1 \leq k \leq s^{\text{indv}}) \) in \( P^{\text{indv}}_{i,k} \),
and its corresponding optimal objective $Q_{i,k}^{ind}$ in $O_{i}^{ind}$, we judge whether $S_i(g)$ (1 ≤ i ≤ s) is dominated by $P_{i,k}^{ind}$ (1 ≤ k ≤ s) according to $O_{i,j}^{ind}$ and $Q_{i,k}^{ind}$. Based on Definition 1, if $\exists k : P_{i,k}^{ind} \succ S_i(g)$, then the current solution $S_i(g)$ is not the Pareto-optimal solution; otherwise (i.e., $S_i(g)$ is not dominated by any solution $P_{i,k}^{ind}$ (1 ≤ k ≤ s)) in $P_{i}^{ind}$), add $S_i(g)$ to the Pareto-optimal set $P_{i}^{ind}$ and add $O_{i,j}$ to optimal objectives $O_{i}^{ind}$.

For the recording of the best swarm, we first calculate the size of $P_{b}^{gb}$ by $s_{gb}^{b}$, then for any Pareto-optimal solution $P_{b}^{gb}$ (1 ≤ $\xi$ ≤ $s_{gb}^{b}$) in $P_{b}^{gb}$ and its corresponding optimal objective $Q_{b}^{gb}$ in $O_{b}^{gb}$, judge whether $S_i(g)$ (1 ≤ i ≤ s) is dominated by $P_{b}^{gb}$ according to $O_{i,b}^{gb}$ and $Q_{b}^{gb}$.

Based on Definition 1, if $\exists \xi : P_{b}^{gb} \succ S_i(g)$, then the current solution $S_i(g)$ is not the Pareto-optimal solution; otherwise (i.e., $S_i(g)$ is not dominated by any solution $P_{b}^{gb}$ (1 ≤ $\xi$ ≤ $s_{gb}^{b}$) in $P_{b}^{gb}$), add $S_i(g)$ to the Pareto-optimal set $P_{b}^{gb}$ and add $O_{i,b}$ to optimal objectives $Q_{b}^{gb}$.

Judgement of termination condition: If reach the termination iteration $g_{max}$, map the particle representation in Pareto-optimal set $P_{b}^{gb}$ to the variables in Formula (7); Otherwise, go to step 6).

**Algorithm 1. Particle Swarm Optimization Based Computation Offloading (PSOCO) Algorithm**

1: Initialize $S(g)$, $P_{i}^{ind}$, $O_{i}^{ind}$, $P_{b}^{gb}$, $Q_{b}^{gb}$, $V_i$, (1 ≤ i ≤ s)
2: for iteration $g = 0, 1, 2, \ldots, g_{max}$
3:     for each particle:
4:         Map $S_i(g)$ to the variables to be solved in Formula (7)
5:         Calculate the fitness values $O_{i,j}^{ind}$ and $Q_{i,k}^{ind}$ according to Formula (7), and define $O_{i,j} = \{O_{i,j}^{ind}, O_{i,j}^{gb}\}$
6:         Calculate the size of $P_{i}^{ind}$ by $s_{ind}^{i}$
7:         for each Pareto-optimal solution $P_{i,k}^{ind}$ in $P_{i}^{ind}$:
8:             if $S_i(g)$ is not dominated by $P_{i,k}^{ind}$ then
9:                 Add $S_i(g)$ to set $P_{i,k}^{ind}$
10:                Add $O_{i,j}$ to set $O_{i,k}^{ind}$
11:         end for
12:     end for
13:     for each Pareto-optimal solution $P_{b}^{gb}$ in $P_{b}^{gb}$:
14:         if $S_i(g)$ is not dominated by $P_{b}^{gb}$ then
15:             Add $S_i(g)$ to set $P_{b}^{gb}$
16:             Add $O_{i,j}$ to set $O_{b}^{gb}$
17:         end for
18:     Randomly choose one Pareto-optimal solution from $P_{i}^{ind}$ and $P_{b}^{gb}$, denoted as $P_{i}^{ind}(g)$ and $P_{b}^{gb}$ respectively
19:     Each particle updates its speed and position according to Formulas (16) and (17)
20:     for each element $s_{i,j}(g)$ in $S_i(g)$:
21:         if $s_{i,j}(g)$ is beyond the position ranges then
22:             Re-initialize $s_{i,j}(g)$
23:         end for
24:     end for

6) Position and speed updating of individuals: This step is also called swarm updating. Since there are multiple Pareto-optimal solutions of a particle in $P_{i}^{ind}$ (1 ≤ i ≤ s) and Pareto-optimal solutions of the swarm in $P_{b}^{gb}$, we select randomly select one Pareto-optimal solution from each of $P_{i}^{ind}$ and $P_{b}^{gb}$, denoted by $P_{b}^{ind}(g)$ and $P_{b}^{gb}(g)$, respectively. Now, for each particle, we have the current position $S_i(g)$, the current speed $V_i(g)$, the randomly selected Pareto-optimal solutions $P_{b}^{ind}(g)$ and $P_{b}^{gb}(g)$, formulated as

$$S_i(g) = \{s_{i,1}(g), s_{i,2}(g), \ldots, s_{i,d}(g)\},$$

$$V_i(g) = \{v_{i,1}(g), v_{i,2}(g), \ldots, v_{i,d}(g)\},$$

$$P_{b}^{ind}(g) = \{P_{b,1}^{ind}(g), P_{b,2}^{ind}(g), \ldots, P_{b,d}^{ind}(g)\},$$

$$P_{b}^{gb}(g) = \{P_{b,1}^{gb}(g), P_{b,2}^{gb}(g), \ldots, P_{b,d}^{gb}(g)\}.$$ (15)

Then, each particle updates speed and position as

$$v_{i,x}(g + 1) = \omega v_{i,x}(g) + \delta_1 \theta_1 (P_{b,i}^{ind}(g) - s_{i,x}(g)) + \delta_2 \theta_2 (P_{b,i}^{gb}(g) - s_{i,x}(g)), \forall e \in \{1, \ldots, d\},$$

$$s_{i,x}(g + 1) = s_{i,x}(g) + v_{i,x}(g + 1), \forall e \in \{1, \ldots, d\},$$

where $\omega$ denotes an inertia weight factor between 0.8 and 1.2, and can be dynamically adjusted according to the linear decrement strategy, shown as

$$\omega = \omega_{max} - \frac{(\omega_{max} - \omega_{min}) \times g}{g_{max}}.$$ (18)

And $\delta_1$ and $\delta_2$ denote learning factor, also named acceleration constant. $\theta_1$ and $\theta_2$ are two random numbers uniformly distributed between 0 and 1. After step 6), the swarm is denoted as

$$S'(g) = \{S'_1(g), S'_2(g), \ldots, S'_d(g)\}.$$ (19)

7) Boundary condition processing: For each particle, judge whether $s_{i,x}(g + 1)$ is beyond the position ranges defined by constrains (3-C1)~(3-C6) in Formula (7). If $s_{i,x}(g + 1)$ is beyond the position ranges, re-initialize it according to step 1). After step 7), the swarm is denoted as

$$S''(g) = \{S''_1(g), S''_2(g), \ldots, S''_d(g)\}.$$ (20)

Then update the iterative number $g := g + 1$ and return to step 2).

We summarize the PSOCO algorithm in Algorithm 1. It is worth noting that the PSOCO algorithm can be also applied to other distributed RUs and edge servers. It is also worth noting that vehicles may enter and leave the coverage of the RSU during task offloading, leading to a failed result reception from RSU and a higher delay and cost. To address this issue, we can first adopt a duration prediction method to evaluate the link duration between vehicles and RSU based on the current position and speed of vehicles, and the position of RSU. Then, a threshold duration is set, guaranteeing the task processing result can
be returned before the vehicle leaves the coverage of the RSU. Only the vehicles whose predicted link duration is longer than the threshold duration can participate in the task offloading process. Another way is utilizing the cooperation between adjacent RSUs. The remaining task data can be offloaded to the next RSU if the vehicle leaves the coverage of the current RSU during task offloading, or can be migrated from the current RSU to the next RSU if the vehicle leaves the coverage of the current RSU during task processing. Also, the vehicles that will enter the coverage of the RSU during the process of self-learning of a vehicle can also participate in the task offloading process in the current RSU. For simplicity, in this paper, we assume that all vehicles can receive the results before they leave the coverage of the RSU.

5 SIMULATION RESULTS AND DISCUSSIONS

5.1 Simulation Setup

We consider a two-way two-lane road with a length of 1000 m. The width of each lane is 4 m. And one RSU is deployed in the middle of the roadside and its coverage radius is 500 m. Vehicles that are on two different lanes are keeping moving back and forth along their lanes. For the behavior of vehicles, we use part of the GAIA Open Dataset containing mobility traces of DiDi Express in Xi’an China [40]. We randomly choose 20 60 traces in our simulation. The data size is randomly distributed between 0.1 and 1 MB. The processing density is set from {10, 100, 1000} for type 1, 2, and 3 tasks, respectively. We present the parameters setting in Table 2.

For the simulation environment, we use a GPU-based server, where the CPU is Intel Xeno(R) E5-2690v4 with 64 GB memory. The software environment is Python 3.7 on Ubuntu16.04.6 LTS.

5.2 Simulation Results

We consider the following schemes as benchmarks to evaluate our proposed PSOCO: 1) Offload-Comp-Only (OCO), where all vehicles offload their computation tasks to RSU to be processed; 2) Local-Comp-Only (LCO), where all vehicles compute their computation tasks locally; 3) GACO, a genetic algorithm-based computation offloading scheme.

5.2.1 Effectiveness

We first conduct simulations to verify the effectiveness of the proposed PSOCO. As a comparison, the GACO scheme is evaluated under the same conditions. In this set of simulations, we set N = 20. As shown in Fig. 2, we give the results for delay and cost varying with iterations. It is worth noting that we optimize the two objectives separately. From the figure, we can see that both PSOCO and GACO can converge to optimal solutions for delay and cost. Specifically, for PSOCO, its delay converges at about the 300th iteration and its cost converges at the 180th iteration. However, for the GACO scheme, its delay converges at about the 350th iteration and its cost converges at about the 310th iteration. Both results verify the effectiveness of our proposed PSOCO and show that PSOCO has a faster convergence than the benchmark GACO.

![Fig. 2. Convergence of delay and cost.](image-url)
5.2.2 Pareto-Optimal Solutions

To find the Pareto-optimal solutions and the Pareto-optimal front, we implement the simulations with different number of vehicles in Figs. 3 and 5.

The simulation results of OCO are presented in the top of Figs. 3 and 5a. It can be easily observed from the top of Fig. 3 that the objectives for the delay and the cost run in the opposite direction, which means that the improvement of one objective may lead to the decline of the other. This is because that more communication and computation resources will be allocated to a certain task for transmitting and processing if a lower delay is preferred, which results in a higher cost for the utilization of communication and computation resources, and vice versa. When putting all five curves in the top of Fig. 3 together, we depict Fig. 5a. It is shown that both the delay and cost increase with the increase of \( N \). This is because more vehicles lead to more tasks to be processed, which further results in higher delay and cost due to the limited communication and computation resources of RSU.

The simulation results of LCO are presented in the middle of Figs. 3 and 5b. It is shown from the middle of Fig. 3 that the objectives for the delay and the cost also run in the opposite direction. This is because more computing resources of a certain vehicle will be allocated to process its task if a lower delay is preferred, which will result in a higher cost for the utilization of computing resources, and vice versa. When putting all five curves in the middle of Fig. 3 together, we depict Fig. 5b. It is shown that both the delay and cost increase with the increase of \( N \). The reason is that the communication resource and computing resource of RSU within one road segment are limited, each vehicle will be allocated less communication and computation resources from RSU with the increase of \( N \). Therefore, a higher delay and hence a higher cost will be consumed to complete the tasks of vehicles.

What’s more, we depict Fig. 4 to show the performance comparison among OCO, LCO, and PSOCO under different number of vehicles. It is shown that the Pareto-optimal solutions among OCO, LCO, and PSOCO vary greatly. For OCO, the delay is very low while the cost is very high. This is because more communication and computation resources will be allocated to a certain task for transmitting and processing if a lower delay is preferred, which results in a higher cost for the utilization of communication and computation resources, and vice versa. When putting all five curves in the bottom of Fig. 3 together, we depict Fig. 5c. It is shown that both the delay and cost increase with the increase of \( N \). The reason is that more tasks will be offloaded to RSU for processing to reduce the processing time due to the higher processing capability of RSU, which may lead to a cost increase due to the cost of uplink communication resource and the cost for RSU to process tasks. When putting all five curves in the bottom of Fig. 3 together, we depict Fig. 5c. It is shown that both the delay and cost increase with the increase of \( N \). The reason is that the communication resource and computing resource of RSU within one road segment are limited, each vehicle will be allocated less communication and computation resources from RSU with the increase of \( N \). Therefore, a higher delay and hence a higher cost will be consumed to complete the tasks of vehicles.

What’s more, we depict Fig. 4 to show the performance comparison among OCO, LCO, and PSOCO under different number of vehicles. It is shown that the Pareto-optimal solutions among OCO, LCO, and PSOCO vary greatly. For OCO, the delay is very low while the cost is very high. This

![Fig. 3. Pareto-optimal solutions and Pareto-optimal front of OCO, LCO, and PSOCO under different number of vehicles.](image1)

![Fig. 4. Comparison of Pareto-optimal solutions among OCO, LCO, and PSOCO under different number of vehicles.](image2)
because the cost of uplink communication resource and the cost for RSU to process tasks are high. For LCO, the cost is very low while the delay is very high. This is because the limited processing capability of vehicles will result in a severe delay for some compute-intensive tasks since all tasks are processed locally. Different from the OCO and LCO, the tasks can be partially offloaded to RSU in our proposed PSOCO, which jointly allocates the communication and computation resources of vehicles and RSU to obtain the optimal delay and cost. Moreover, the delay and cost of PSOCO are between that of OCO and LCO. A trade-off between delay and cost is also obtained, which can guide the allocation of communication and computation resources for different types of tasks.

5.2.3 Allocation of Communication and Computation

To elaborate on the communication and computation resource allocation, we select two Pareto-optimal solutions, i.e., A and B (hereinafter referred to as Pareto-A and Pareto-B, respectively) from Fig. 5c. We first depict the task attribute distribution about $c_n$, $g_n$, and $\mu_n$ in Fig. 6. It is shown that 4 tasks are with the processing density of 1000 cycles/bit, 7 tasks are with the processing density of 100 cycles/bit, 9 tasks are with the processing density of 10 cycles/bit. And all $g_n$ of tasks are randomly distributed between 0 and 0.2. Based on the 20 tasks, we present in the following how the communication and computation resources are allocated under Pareto-optimal solutions A and B.

1) Allocation of offloading decision $\lambda_n$: For the accuracy of the statistical results, we evaluate the average offloading decision, which is defined as the average $\lambda_n$ of tasks with the same processing density. Fig. 7 shows that the average offloading decision increases with the increase of processing density. This is because a higher processing density needs more computing cycles, which results in that more task data bits should be offloaded to RSU for processing. More importantly, the average offloading decision of Pareto-A is higher than that of Pareto-B, which reflects the preference for delay objective of Pareto-A. A lower delay is realized by offloading more task data bits to RSU since the higher computing capability of RSU.

2) Allocation of local processing capability $f_n$: For the accuracy of the statistical results, we evaluate the average local processing capability, which is defined as the average $f_n$ of tasks with the same processing density. It can be shown from Fig. 8 that the average local processing capability increases with the increase of processing density. This is because more local processing capabilities should be allocated to process
more task data bits due to the increased processing density to obtain an optimal delay. And the average local processing capability of Pareto-B is higher than that of Pareto-A. The reason is that more task data bits are processed locally for Pareto-B, with the result that more local processing capabilities are allocated to process the tasks.

3) Allocation of communication resource: We evaluate the allocation of communication resource by the allocated uplink communication resource. As shown in Fig. 9, the horizontal axis and vertical axis denote the offloaded data size and allocated uplink communication resource, respectively. The offloaded data size refers to the size of the data that will be offloaded to RSU for processing. It shows that the allocated uplink communication resource increases with the increase of offloaded data size. What’s more, less communication resource is allocated to transmit offloading tasks for Pareto-A when the offloaded data size is small. When the offloaded data size reaches a threshold, as point C in Fig. 9, the allocated communication resource of Pareto-A is higher than that of Pareto-B. This is because the local computing capabilities of vehicles are enough to process the tasks when data size is small. Offloading tasks may cause extra delay, which is against the purpose of Pareto-A. With the increase of the offloaded data size, more communication resource is allocated to transmit the tasks to RSU. This is because the RES enabled RSU has powerful computing capability toward a lower processing delay, which is consistent with the purpose of Pareto-A.

4) Allocation of RES computation resource: Fig. 10 reflects the relationship between the allocated RES processing capability and the computing amount of offloaded tasks for Pareto-optimal solutions A and B. It is shown that the allocated RES processing capability increases with the increase of the computing amount of offloaded task. And the allocated RES processing capability of Pareto-A is higher than that of Pareto-B. The reason is that the purpose of Pareto-B is obtaining a lower delay in task processing, which needs more RES processing capabilities to achieve this goal. On the contrary, since the Pareto-B prefers a low cost, less RES processing capabilities should be allocated due to the high cost of RES resource.

6 CONCLUSION

In this paper, we have investigated the computation offloading problem in a VEC network through jointly considering the allocation of communication and computation resources, aiming at providing a detailed analysis of the delay and cost of computation offloading for VEC and minimizing the delay and cost from the perspective of multi-objective optimization. To this end, we first established an offloading framework with communication and computation for VEC, where tasks with different requirements for computation capability are considered. To consider comprehensive performance, we then formulated a multi-objective optimization problem to minimize both the delay and cost during computation offloading. To solve the formulated optimization problem, which is also a MINLP problem, we introduced the concept of Pareto optimality and proposed a particle swarm optimization based computation offloading (PSOCO) algorithm to obtain the Pareto-optimal solutions. Finally, based on the real-world vehicular trace, we implemented extensive simulation to verify the performance of PSOCO. Moreover, we also presented a comprehensive analysis and discussion on the relationship between delay and cost among the Pareto-optimal solutions.

In the future, we will consider continuous tasks, the mobility of vehicles, and the handover between different RSUs for a more practical VEC scenario. And we will consider developing real prototypes to conduct field tests of the proposed algorithm.

ACKNOWLEDGMENTS

This work was supported in part by the scholarship from China Scholarship Council, in part by the National Natural Science Foundation of China under Grant U1801266.

REFERENCES


Tom H. Luan (Senior Member, IEEE) received the BE degree from Xi’an Jiaotong University, Xi’an, China, in 2004, the MPhil degree from the Hong Kong University of Science and Technology, Kowloon, Hong Kong, in 2007, and the PhD degree from the University of Waterloo, Waterloo, ON, Canada, in 2012. He has been a lecturer with the School of Information Technology, Deakin University, Burwood, VIC., Australia from 2013 to 2017 and is currently a professor with the School of Cyber Engineering, Xidian University. His current research interests include the content distribution in vehicular networks, mobile cloud computing, and fog computing.

Weisong Shi (Fellow, IEEE) received the BS degree in computer engineering from Xidian University, Xi’an, China, in 1995, and the PhD degree in computer engineering from the Chinese Academy of Sciences, Beijing, China, in 2000. He is a Charles H. Gershenson distinguished faculty fellow and a professor of computer science with Wayne State University, Detroit, MI, USA. His current research interests include edge computing, computer systems, energy-efficiency, and wireless health. He was a recipient of the National Outstanding PhD Dissertation Award of China and the NSF CAREER Award. He is an ACM distinguished scientist.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.