Journal of Computer Science and Cybernetics, V.38, N.4 (2022), 347–364
DOI no. 10.15625/1813-9663/38/4/17012

EVOLUTIONARY ALGORITHM FOR TASK OFFLOADING IN VEHICULAR FOG COMPUTING

DO BAO SON[1,2], VU TRI AN[2], HIEP KHAC VO[3], PHAM VU MINH[2], NGUYEN QUANG PHUC[2], TA HUU BINH[2], NGUYEN PHI LE[2], BINH MINH NGUYEN[2], HUYNH THI THANH BINH[2,*]

[1] *University of Transport Technology, Ha Noi, Viet Nam*
[2] *School of Information Communication and Technology, Ha Noi University of Science and Technology, Vietnam*
[3] *University of Technology Sydney, Australia*

Abstract. Internet of Things technology was introduced to allow many physical devices to connect over the Internet. The data and tasks generated by these devices put pressure on the traditional cloud due to high resource and latency demand. Vehicular Fog Computing (VFC) is a concept that utilizes the computational resources integrated into the vehicles to support the processing of end-user-generated tasks. This research first proposes a bag of tasks offloading framework that allows vehicles to handle multiple tasks and any given time step. We then implement an evolution-based algorithm called Time-Cost-aware Task-Node Mapping (TCaTNM) to optimize completion time and operating costs simultaneously. The proposed algorithm is evaluated on datasets of different tasks and computing node sizes. The results show that our scheduling algorithm can save more than 60% of monetary cost than the Particle Swarm Optimization (PSO) algorithm with competitive computation time. Further evaluations also show that our algorithm has a much faster learning rate and can scale its performance as the number of tasks and computing nodes increases.

Keywords. Evolutionary algorithm; Task offloading; Vehicular fog computing.

1. INTRODUCTION

In recent years, promising advancements in fields such as artificial intelligence, machine learning, and the emergence of technologies like 5G has brought about rapid development in the Internet of Things (IoT). International Data Corporation estimated that by 2025, there would be 41.6 billion devices connected to IoT that will generate 79.4 Zettabytes of data [13]. The ability of IoT to provide sensor information and enable device-to-device communication is driving a wide range of applications. An example of these applications is IoT in the automotive industry, which results in designs of vehicles that see a performance increase, reduced cost, and facilitated quality control. Furthermore, IoT connectivity enables solutions for fleet management that aid fleet operators by monitoring driver performance and

*Corresponding author.
E-mail addresses: sondb@utt.edu.vn (D.B.Son), an.vt212432m@sis.hust.edu.vn (V.T.An), hiep.k.vo@student.uts.edu.au (H.K.Vo), vuminhph@gmail.com (P.V.Minh), phuc.nq173304@sis.hust.edu.vn (N.Q.Phuc), binh.th190094@sis.hust.edu.vn (T.H.Binh), lenp@soict.hust.edu.vn (N.P.Le), minhnb@soict.hust.edu.vn (B.M.Nguyen), binhht@soict.hust.edu.vn (H.T.T.Binh)

optimizing maintenance and logistics. Users also benefit by having a better in-car experience with infotainment applications and more safety and efficiency overall. Ensuring the Quality of Service (QoS) of these applications often requires intense computational power and constant data processing, which are difficult considering intrinsic limitations of IoT devices such as low battery life, computing capabilities, and storage. Mobile cloud computing (MCC) [8] is introduced as a solution to alleviate this problem by utilizing the cloud to handle resource-demanding tasks offloaded by IoT devices. However, the centralized nature of the cloud leads to high transmission delay and cost.

Multi-access edge computing (MEC) [18] is a distributed computing paradigm that aims to extend the cloud by moving the computing traffic and services to the edge of the network and closer to the generation source, thus, reducing latency and bringing real-time performance to high-bandwidth applications [3,27]. Moreover, the MEC network can be integrated alongside cloud platforms to take advantage of the cloud's scalability and on-demand, pay-as-you-go services, which further improves MEC's robustness and resilience [2, 16]. Vehicle Fog Computing (VFC) is another promising solution that can expand MEC's resources by exploiting idle computing power of vehicles [11,21,22], therein, vehicles on streets, roadways, and parking lots will be treated as computing nodes. The owners of these vehicles could rent out excess onboard resources and benefit economically.

The main contributions of this paper are as follows:

- Develop a two-phase task offloading framework in VFC which allows each vehicle to process multiple tasks in batches while minimizing offloading time and operation cost.
- Formulate problem in each phase as task-node mapping problem and resource allocation problem, respectively.
- Propose a Time-Cost-aware Task-Node Mapping algorithm based evolutionary strategy to solve the task-node mapping problem and utilize an exact polynomial-time-complexity algorithm for resource allocation problem.
- Evaluate and compare the performance of the proposed algorithm against PSO algorithm.

The organization of this paper is as follows. In Section 2, we discuss the related works of this paper. Section 3 provides the formal definition of the network model and problem formulation. Details about our proposed solution are presented in Section 4. In Section 5, simulation results are analyzed to verify the performance of the proposed algorithms. Finally, Section 6 concludes the paper.

2. RELATED WORK

For several years, the task offloading problem in fog computing have gotten a lot of attention. Many studies have been published on this topic, especially since VFC [12,17] was introduced. These works propose solutions and algorithms for problems with one or multiple goals simultaneously. The objectives and strategies used to solve the problems are presented below.

Despite their popularity, some vehicular applications are computationally complex, time-consuming, and real-time [5]. Taking too long to process the application's actions can have a negative impact on performance, data validity, and even the safety of passengers on the

vehicle. The main goal of the computation offloading technique is to reduce application response time (also known as task processing time and computation overhead). However, sending tasks to be processed on other network devices sometimes be challenging. It can also cause transmission and processing delays (e.g., sending tasks to overloaded devices would be a bad decision). Rahman et al. [20] proposed an offloading scheme to address transmission failures and offloading delays by allowing vehicles to take the idle resources from nearby vehicles. The authors in [23] proposed a mechanism to determine where and when to offload tasks to ensure the delivery of high-reliability and low-latency computing services to users. The work in [26] proposed a federated offloading scheme to achieve the lowest total latency possible. Another goal mentioned in some papers is to lower the financial cost of offloading. Vehicles, for example, may be charged for computation and communication services. The vehicle may be charged for data transmission and reception using cellular communications. The vehicular server may charge a fee to handle the end-user tasks in computing. From the server's perspective, providing computing services can be profitable. However, it may be required to pay the grid operator for electricity and the network operator for wireless bandwidth rental [7]. In [24], the authors focused on effectively using the vehicle's resources and performing optimal resource allocation for cost savings based on application context. The research in [10] reduced the financial cost by managing the content caching, computing, and networking at vehicular networks. The cost in the above studies may vary depending on hardware, vehicle location, and energy consumption.

Following are some strategies for solving task offloading problems. The study in [7] proposed a task offloading and resource allocation problem as linear programming and used Lyapunov optimization to minimize the cost of vehicle and fog servers. The authors in [29] designed a collaborative task offloading and resource allocation scheme. Then, they used the game theory approach to take offloading decisions and Lagrangian relaxation for resource allocation. Joint load balancing and offloading with the vehicle is the subject of [4] to maximize system utility under latency threshold. The problem is formulated as a mixed-integer non-linear programming problem. A greedy approach called JSCO (Joint Optimization for Selection, Computation, and Offloading algorithm) is used to solve this problem. Intending to maximize the network utility, the authors consider the computational offloading and resource allocation problem in vehicular edge computing networks using time-varying channels. They proposed a linearization-based branch-and-bound (LBB) algorithm and a closest rounding integer (CRI) algorithm to solve the static and dynamic problems. In [19], the authors proposed a scalable vehicle-assisted MEC (SVMEC) scheme to extend the computing resources. The problem is formulated as a mixed-integer non-linear programming (MINLP) problem and split into sub-problems: resource allocation problem and Node selection problem. Karush-Kuhn-Tucker (KKT) conditions are used to obtain an optimal resource allocation plan. Meanwhile, the branch-and-bound algorithm determines which task will be executed on which node.

Most of the above-related studies consider the problem of task offloading in Vehicular Edge Computing (VEC) environments, where there are two main types of computing resources: vehicular server and edge server. Accordingly, computational tasks can be offloaded to neighboring vehicles or edge servers. In this environment, powerful cloud servers are not selected to perform tasks. As a result, computing capacity is limited to edge servers and a few vehicles. Unlike the mentioned studies, our study focuses on task offloading in VFC,

where tasks are not only performed at the edge of the network but can also be sent to the remote cloud. The model in [19] is similar to VFC. However, the authors considered offloading one task to one vehicle, so the maximum number of tasks that can be offloaded is equal to the number of vehicles. While the number of tasks can be up to thousands, tens of thousands, the actual number of vehicles is much lower. In addition, because each vehicle can only perform one task, it is impossible to utilize the vehicle's idle resources fully. This study proposes a task offloading scheme that can process tasks in batches to fully use the vehicle's resources. Then, we implement algorithms based on the evolutionary strategy to achieve an optimal trade-off between processing delay and cost.

3. SYSTEM MODEL

3.1. Network model

We consider the system consisting of three main components: the Stationary Fog, the Cloud, and the Vehicular Fog Servers (VFS). The Stationary Fog includes a base station, a resource controller, and the local servers. IoT devices operating in the range of the base station send their jobs to the Stationary Fog via nearby access points. The resource controller is responsible for keeping track of available resources in the whole system. Upon receiving a task, the resource controller extracts that task's information and identifies resources needed for processing. It then delegates the task to be executed on either the remote Cloud, the vehicular mounted Fog servers, or the local server.

The Cloud is introduced to the system to mitigate the computational strain put on the Stationary Fog by utilizing the Cloud's vast resources, which significantly improves services, but it is expensive. Moreover, communicating with the Cloud server takes time. The network also consists of several vehicular fog servers, aiming to exploit the vehicle's idle resources. Buses are ideal for applying this model due to their fixed routes and predictable span inside the base station's range. A fee is rewarded to the bus owners on each task completion to incentivize resource sharing. In [19], the authors only consider sending a single task to each vehicle, therefore at any given moment, the number of tasks processed by the vehicular fog is bounded by the number of vehicles in the communication range of the base station. In our model, each vehicle receives tasks depending on its capability and routes. The set of tasks is stored on a queue, and, at each instance, a task is processed.

3.2. Computing node and task model

Tasks are sent to the network in cycles. Each task t_i consists of three components: $\{d_{req_i}, w_i, d_{res_i}\}$, where d_{req_i} is the size of the computational task requested by IoT devices, w_i is the number of CPU cycles required to complete, and d_{res_i} represents the size of return data containing the results after the task is finished. The tasks considered in this paper are indivisible. Consequently, each computation task can only be executed on the Cloud, the MEC server, or a vehicular fog node. Let $\mathcal{T} = \{\tau_1, \tau_2, \tau_3, ..., \tau_n\}$ denote the set of tasks in each cycle that the MEC provider has to complete while keeping track of available resources and communication quality of the systems.

In each cycle, the set of vehicles operating in range of the MEC provider is denoted by $\{p_1, ..., p_m\}$, which is subset of $\mathcal{P} = \{p_{-1}, p_0, p_1, ..., p_m\}$, representing all computing nodes

in the system. Note that computing node $p_j \in \mathcal{P}$ holds two special values, -1 and 0. Here, $j = -1$ implies that the computing node is the MEC server, while $j = 0$ indicates the Cloud server. The strategy of processing tasks includes two phases, each of them has an optimization problem to solve:

1. Task-node mapping problem: select computing node to which tasks are sent to be executed, i.e., the remote Cloud, local MEC server, or vehicular nodes.
2. Resource allocation problem: allocate computing resources for the Cloud and the MEC server.

We utilize different approaches for task-node mapping problem, which is formulated in Subsection 3.4. Time-Cost-aware Task-Node Mapping, our proposed algorithms is presented in Subsection 4.1. Optimal solution for resource allocation problem, given a feasible solution after solving task-node mapping problem, is mathematically proved in Subsection 4.2, and the algorithm are polynomial time complexity.

3.3. Computing model

3.3.1. Tasks computation on local MEC server

Computing tasks locally on the MEC server results in no monetary setbacks as the MEC provider owns the computing resources. The time it takes the MEC server to complete a task τ_i is expressed as

$$t_{i,-1} = \frac{w_i}{f_{i,-1}}, \tag{1}$$

where w_i, as stated in the prior section, is the computational intensity of task τ_i and $f_{i,-1}$ is the computational resource (CPU cycles per second) that the MEC server allocates to task τ_i. Under the position of MEC service provider, the fee of renting MEC server is 0

$$c_{i,-1} = 0. \tag{2}$$

3.3.2. Tasks computation on remote Cloud server

The Cloud's completion time for a given task τ_i is the sum of the transmission time between the Cloud and local MEC server, which includes time sending task information and receiving the result from the Cloud and the task's computation time

$$t_{i,0} = \frac{d_{req_i}}{R_0} + \frac{w_i}{f_{i,0}} + \frac{d_{res_i}}{R_0}, \tag{3}$$

where, R_0 is the communication rate between MEC and Cloud server, and $f_{i,0}$ is the computational resource allocated by the Cloud for processing task τ_i.

The service cost of using the Cloud's resources is

$$c_{i,0} = \delta_0^{comp} w_i, \tag{4}$$

where, δ_0^{comp} is the unit cost of computation on the Cloud.

3.3.3. Tasks computation on vehicular fog nodes

There are two main components of a base station to take into account regarding its placement: The geographic coordinate (x_0, y_0) and the maximum communication radius D_r. Due to the bus routes' rigidity, estimating each vehicle's span within the communication range of the base station is relatively simple. Denote Δt_m the duration that vehicle node p_j $(j = \overline{1,m})$ operates within range of the MEC base station and $I_j = \{(x_1, y_1), (x_2, y_2), ..., (x_k, y_k)\}$ are it's discrete coordinates during that duration. Thus, the Euclidean distance has the following constraint

$$d_{j_i} = \sqrt{(x_i - x_0)^2 + (y_i - y_0)^2} \leq D_r,\ i = 1, 2, 3, ...k. \tag{5}$$

The transmission rate at each coordinate (x_i, y_i) is given by

$$R_{j_i} = W \log_2 \left(1 + \frac{P_{j_i} d_{j_i}^{-\eta} |h_0|^2}{N_0}\right), \tag{6}$$

where W is the channel bandwidth, P_{j_i} is the transmission power of base station to coordinate (x_i, y_i), h_0 is the complex Gaussian channel coefficient which follows complex normal distribution $CN(0, 1)$ and N_0 is the additive white Gaussian noise (AWGN) at the bus receivers.

Since W, h_0, and N_0 are constants and P_{j_i} is proportional to $d_{j_i}^{-\eta}$. The access point (AP) needs to increase its power to maintain the transmission rate as the bus gets further.

To sustain transmission rate R_j^* at every time step i, transmission power has to follow the formula

$$P_{j_i} = \frac{(2^{R_j^*/W} - 1)N_0}{d_j |h_0|^2}. \tag{7}$$

The average transmission power is calculated as follows

$$P_j = \frac{\sum_{i=1}^{k} P_{j_i}}{k},\ i = 1, 2, ..., k. \tag{8}$$

In addition, the vehicles also have to vary their transmission power every time step to maintain a stable communication rate, $R_j^{response}$, which is smaller than that of the base station. Hence, $R_j^{response} < R_j^*$.

The computation time of a task executed on vehicular fog server is given by

$$t_{i,j} = \frac{d_{req_i}}{R_j^*} + \frac{w_i}{f_j} + \frac{d_{res_i}}{R_j^{response}}, \tag{9}$$

where f_j is the computational capacity of vehicle node j.

The computation cost comprises of two factors: The monetary incentives paid to the bus owners in exchange for their resources and the energy cost of maintaining communication with the base station. Define δ_m^{comp} and β the unit cost of service and power consumption, respectively. The computation cost can be obtained by

$$c_{i,m} = \delta_j^{comp} w_i + \beta P_j \frac{d_{req_i}}{R_j^*},\ \forall p_j\ (j = \overline{1,m}). \tag{10}$$

3.4. Task-node mapping problem formulation

The task offloading problem in Vehicular Fog Computing can be formulated as follows:

Input:

$\mathcal{T} = \{t_1, t_2, t_3, ..., t_n\}$: a set of n independent tasks

$\mathcal{P} = \{p_{-1}, p_0, p_1, p_2, ..., p_m\}$: a set of processing nodes including Cloud server, local server, and m Vehicular Fog Servers, respectively

Output:

$X = \left\{x_{i,j} \in \{0,1\} | t_i \in T, p_j \in P, \sum_{j=-1}^{m} x_{i,j} = 1\right\}$: an assignment of each task executed into processing nodes, where $x_{i,j} = 1$ if the decision of offloading task i^{th} to server j^{th}, $x_{i,j} = 0$ otherwise.

Objective:

Minimize the utility function

$$\begin{aligned} &\min \sum_{j=-1}^{m} \sum_{i=1}^{n} x_{i,j}(\gamma * t_{i,j} + (1-\gamma) * c_{i,j}), \\ \text{s.t.} \quad &x_{i,j} = \{0,1\}, \\ &\sum_{j=-1}^{m} x_{i,j} = 1, \ \forall i = \overline{1,n}. \end{aligned} \tag{11}$$

4. PROPOSAL

In this section, we consider and give the solution to the problem of each phase as mentioned in Subsection 3.2.

(i) Task-node mapping problem.

(ii) Resource allocation problem.

4.1. Time-cost-aware task-node mapping (TCaTNM) algorithm

Since Task-node mapping problem is NP-Hard [9], we propose the Time-cost-aware task-node mapping algorithm, which is based on an evolutionary algorithm called the Genetic algorithm (GA), as shown below.

4.1.1. Chromosomes encoding

A task-node mapping solution can be represented by a chromosome sample in GA. An array of n genes corresponding to a sequence of n tasks is used to encode the chromosomes. Each gene holds an integer k ranging $[1; m]$ representing the node that task is assigned to, with m is the total number of nodes.

For example, a chromosome sample $[3, 1, 3, 2, 1, 2, 2, 3, 1, 3]$ means Node 1 is responsible for tasks $\{2, 5, 9\}$, Node 2 handles tasks $\{4, 6, 7\}$, and tasks $\{1, 3, 8, 10\}$ are assigned to Node 3.

This chromosome encoding method makes it easier to perform genetic operators, such as crossover and mutation over the solution search space to create new offsprings that inherit

the quality gene segments of the parents. Here, the tasks' order is not considered because it has no effect on the makespan and the total cost objectives.

In our model, suppose there is a task-node mapping option X then each vehicular fog server p_j will be assigned a set of tasks T^j. According to Equation (10), the total cost of renting vehicular fog is constant. We will prioritize the least demanding task in T^j first to minimize the objective function.

Based on the preceding sections, we assume an optimal node selection X_0 and resource assignment F^* already existed. We now consider the problem of task-node mapping in each vehicular fog server's queue to minimize the time cost. Since vehicular servers can process only one task at any given moment, we utilize a wait queue $\mathcal{Q}$ for tasks' ordering. The time cost is the total wait time of all tasks in the queue. Furthermore, we formulate the wait time of an arbitrary k-th task in $\mathcal{Q}$ as the total computation time of all its preceding tasks

$$T_{W_k} = \begin{cases} \sum_{q=1}^{k-1} t_{i,m}^q & \text{if } \ 2 \leq k \leq |\mathcal{Q}|, \\ 0 & \text{otherwise,} \end{cases} \tag{12}$$

where $t_{i,m}^q$ is the computation time of task i, the q-th task to be computed on bus m, which can be derived from (9). Since an optimal resource assignment, F^*, has already been established, we can easily calculate the computation time of each task in $\mathcal{Q}$. Therefore, we can set a task processing schedule by applying the Shortest-Job-First method, which sorts the queue based on the ascending order of computation time.

4.1.2. Population initialization

The set of initial population is a set of scheduling solutions (chromosomes) generated at random. This population has a size of $\mathbb{N}$ and ensures the diversity of the first generation. To find the optimal solution, we perform some genetic operations on the initial population to generate better solution generations.

4.1.3. Fitness function

The fitness function evaluates the quality of an individual in the population. A good solution should have a high fitness value estimated by the utility function shown in Equation (11). Through each generation, individuals can be kept or eliminated under consideration of their fitness values.

4.1.4. Genetic operators

a. Crossover operator

Two-point crossover operation is applied to produce offspring with good genes from parents in the population. In this operation, two crossover points are randomly chosen, dividing the gene sequence of each parent into three segments. The first parent swaps the middle gene segment with the second parent while preserving the rest to shape a new individual. In the crossover stage over the population, parental selection has a significant impact on the algorithm's performance. Each individual has a crossover rate γ. Everyone in

the population is likely to become the first parent with a probability of γ, and then the second parent is chosen with the roulette wheel technique. This selection technique encourages good quality individuals with a larger fitness value to have a higher probability of being selected, supporting the preservation of good genes.

b. Mutation operator

After crossover and selection process, a new population with N individuals is formed. Each individual is likely to engage in a one-point mutation with the probability of γ. The location of the mutant is randomly determined, and the corresponding gene is substituted by a different value; thus, another node is allocated for the chosen task's processing.

Mutation helps to overcome the limitation of crossover operator by escaping from local extreme values to explore the other region in the solution space or finding the optimum when individuals are vicious around the extremes.

According to the proposed TCaTNM algorithm, some modifications are applied to the original GA as follows:

- Parental selection: In the crossover process, each individual in the population participates as the first parent, and the roulette wheel technique is used to find the second parent.
- Greedy selection strategy: Only one offspring is formed in each crossover operation, and it remains in the next-generation population if it is better than its first parent or perishes otherwise.

c. Selection strategy

The selection operation is conducted to determine the individuals of the next-generation population, based on Darwin's theory of natural selection. After the crossover process, the fitness value is calculated for each new individual, and then used for comparison between offspring and its parent. If the offspring is superior to the parent, it is kept for the next generation. Otherwise, the parent will survive, and the offspring will die out in the next generation.

4.2. Resource allocation solution

Assuming that a tasks-to-computing-nodes mapping has been found, we consider the problem of allocating system resources to minimize the computing cost on each nodes. We denote $\mathcal{U}_j = \{u \in \mathbb{Z} | 1 \leq u \leq n,\ x_{u,j} = 1\}$ with each $j \in \{-1, 0\}$, the resource allocation profile as $F = \{f_{i,j} | j \in \{-1, 0\},\ i \in \mathcal{U}_j\}$, the maximum computing resource capacity of the local MEC server and the remote Cloud server as F_{-1} and F_0, respectively. The objective is now to minimize computation time in the local MEC server and the remote Cloud server

$$\min_{F} K(F) = \sum_{j \in \{-1,0\}} \sum_{i \in \mathcal{U}_j} t_{i,j}, \tag{13a}$$

$$\text{s.t. } f_{i,j} > 0,\ \forall j \in \{-1, 0\},\ i \in \mathcal{U}_j, \tag{13b}$$

$$\sum_{i \in \mathcal{U}_0} f_{i,0} \leq F_0, \tag{13c}$$

$$\sum_{i \in \mathcal{U}_{-1}} f_{i,-1} \leq F_{-1}. \tag{13d}$$

Note that with $j \in \{-1, 0\},\ i \in \mathcal{U}_j$, the value $t_{i,j}$ depends on F (equation (1), (3)). The similar approach to solve the Resource Allocation problem can be found in Subsection 4.1 of [19] and section V-B of [25], but we include our solution for the reader's convenience.

We calculate the first-order derivatives $\partial K(F)$. Its elements are as follows

$$\frac{\partial K(F)}{\partial f_{i,j}} = -\frac{w_i}{f_{i,j}^2},\ \forall j \in \{-1, 0\},\ i \in \mathcal{U}_j. \tag{14}$$

We calculate the second-order derivatives $\partial^2 K(F)$. Its elements are as follows

$$\frac{\partial^2 K(F)}{\partial^2 f_{i,j}} = \frac{2w_i}{f_{i,j}^3} > 0,\ \forall j \in \{-1, 0\},\ i \in \mathcal{U}_j. \tag{15}$$

$$\frac{\partial^2 K}{\partial f_{i,j} \partial f_{m,n}} = 0,\ \forall (i, j) \neq (m, n),\ j, n \in \{-1, 0\},\ i \in \mathcal{U}_j,\ m \in \mathcal{U}_n. \tag{16}$$

From (15) and (16), the Hessian matrix is diagonal with all entries being strictly positive, thus, it is positive definite. Accordingly, the objective function is a convex function. Moreover, the inequality constraint functions are linear functions. Therefore, the problem in (13) is a convex optimization problem. We can obtain its optimal solution by using Karush-Kuhn-Tucker (KKT) conditions. Let $\upsilon = \{\upsilon_{-1}, \upsilon_0\}$ be the Lagrange multiplier vector associated with constraints (13c), (13d). The Lagrange function is as follows

$$L(F, \upsilon) = \sum_{j \in \{-1,0\}} \sum_{i \in \mathcal{U}_j} t_{i,j} + \sum_{j \in \{-1,0\}} \upsilon_j (\sum_{i \in \mathcal{U}_j} f_{i,j} - F_j). \tag{17}$$

From the KKT conditions, we have

$$\partial L(F, \upsilon) = 0, \tag{18}$$

$$\sum_{i \in \mathcal{U}_j} f_{i,j} - F_j = 0,\ \forall j \in \{-1, 0\}. \tag{19}$$

From (18), the first-order derivative of $L(F, \upsilon)$ with respect to $f_{i,j}$ is 0, the optimal $f_{i,j}^*$ then is as

$$f_{i,j}^* = \sqrt{\frac{w_i}{\upsilon_j}}. \tag{20}$$

From (19), $\forall j \in \{-1, 0\}$

$$\begin{aligned} F_j &= \sum_{i \in \mathcal{U}_j} f_{i,j}^* \\ &= \sum_{i \in \mathcal{U}_j} \sqrt{\frac{w_i}{\upsilon_j}} \\ &= \frac{\sum_{i \in \mathcal{U}_j} \sqrt{w_i}}{\sqrt{\upsilon_j}}. \end{aligned} \tag{21}$$

Therefore

$$\sqrt{\upsilon_j} = \frac{\sum\limits_{i \in \mathcal{U}_j} \sqrt{w_i}}{F_j}. \tag{22}$$

Table 1: Characteristics of processing nodes

Parameter	Cloud	Local Fog	Vehicular	Unit
Number of nodes	1	1	20	node
Computational capacity	10	4	$\{0.5, 0.8, 1.0\}$	Ghz
Resource usage cost	0.9	0	0.5	$/gigacycles

Table 2: Parameters concerning the tasks

Properties	Values	Units
Input data size	[1000,2000]	Kb
Output data size	[100,200]	Kb
Number of CPU cycles	[500,1500]	Megacycles

From (20) and (22), the optimal computational resource allocated to task i by node j ($j \in \{-1, 0\},\ i \in \mathcal{U}_j$) is given by

$$f^*_{i,j} = \frac{F_j \sqrt{w_i}}{\sum\limits_{i=\mathcal{U}_j} \sqrt{w_i}}. \tag{23}$$

5. EXPERIMENTS

In this section, we present the simulation setup and evaluation of the proposed algorithm to solve the problem in equation (11) based on the results of conducted tests.

5.1. Experimental settings

For the routes of vehicles, we employ the Seattle bus trace [15], which has been used in several recent studies [6, 28, 30]. The trace shows the actual movement data of about 1200 buses on their normal routes in Seattle, USA for several weeks, a sampled map from the dataset from 08:00 a.m to 08:10 p.m on October 31th, 2003 for our experiment. Each record contains the bus route ID, as well as the location in terms of x- and y-coordinates, as well as the time. The base station is assumed to be in the center of the sampled dataset's map, with a communication coverage of 2 kilometers. Since the trajectory of the bus is fixed, the number and duration of the bus within the base station's communication range within the cycle time can be estimated.

Each node in system is assumed to have its own processing capacity, as measured by Ghz, as well as resource usage costs. The system was built with twenty two processing nodes, which have the characteristics listed in Table 1. Cloud node are much faster than other nodes in terms of computation capacity. On the other hand, using resources in the Cloud is more expensive than using resources in vehicular nodes. Due to the perspective of the fog service provider, this study considers the price when using local fog server to be zero.

For computation task, we use five datasets with size from 100 to 500 tasks for our simulation. Each computation task has some attributes: input data size, output data size, the number of CPU cycles. Each attributes was generated randomly following Table 2. Different from the previous study in [19] when the authors used a set of tasks with the same properties, our experiment could cover a variety of scenarios, with some requiring a lot of processing and others requiring bandwidth usage because many different types of tasks were created.

Table 3: Parameters of algorithm

Parameter	**TCaTNM**	**MPSO**
Running times	30	30
Population size	200	200
Crossover rate	90%	c_1=1.5, c_2=1
Mutation rate	10%	$w = 0.5$
Number of Generations	1000	1000

We use the orthogonal frequency-division multiple access (OFDMA) scheme for wireless communication from base station to vehicular server [1]. The base station's transmission power (P_{m_i}) is 46 dBm, and the channel bandwidth (W) is 10 MHz. Furthermore, we assume there is no interference, the path loss exponent (η) is 4, and the background noise power ($|h_0|^2$) is -100 dBm.

We compare the results achieved by the TCaTNM algorithm with those obtained by Particle Swarm Optimization (PSO) [14]. We set the balance coefficient (γ) in equation (11) is 0.5, that means time and cost have the same priority in fitness function. The parameters of the two algorithms are shown in the Table 3.

5.2. Experimental results

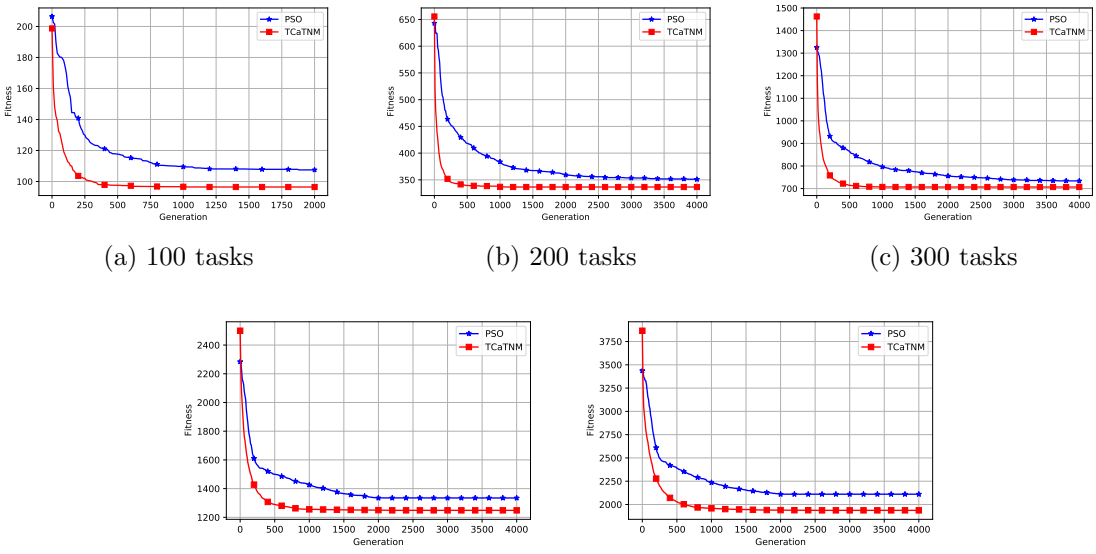
We compare TCaTNM against PSO for convergence analysis by their fitness value under multiple experiment settings - increasing the number of tasks and the number of vehicles. The experimental results for each setting include a line chart of fitted values across training generations, a bar chart of converged fitted values, and a table of converged time and cost. When compare the increasing number of tasks, we fixed the number of vehicles at 20 and when compare the increasing number of vehicles, the number of tasks is set at 100.

5.2.1. Increasing number of tasks

Figure 1 shows the two algorithms' fitness value under five datasets: 100 tasks, 200 tasks, 300 tasks, 400 tasks, and 500 tasks across training. For these experiments, we record and visualize the fitness value of every ten generations.

In terms of the algorithm's learning rate, overall, TCaTNM outperforms PSO in all datasets across all training generations in Figure 1. With a dataset of 100 tasks, Figure 1a shows a faster convergence of TCaTNM from 400 generations, which is 60% faster than that of PSO. As the tasks population increases, the learning rate gap also increases. In specific, TCATNM manages to converge early at 500^{th} generation for 200 tasks and at 1000^{th} generation for 300 tasks which are from 70% to 80% faster than PSO. For 400 tasks and 500 tasks, PSO converges at 2000^{th} generation, 2 times slower than TCATNM. This experimental result shows the stability of our algorithm fast learning rate as the size of the environment scales compared to PSO.

Figure 2a presents the two algorithms' converged fitness values for the number of tasks of 100, 200, 300, 400, and 500. Figure 2a shows that on every experimental data set of the task, the fitness value of TCaTNM is from 6% to 10% better than that of PSO. According to both evolutionary methods, the fitness value increases as the number of tasks increases. It can be explained that as the data set grows, so does the search space, and the two methods



(a) 100 tasks (b) 200 tasks (c) 300 tasks

(d) 400 tasks (e) 500 tasks

Figure 1: Convergence comparison of TCaTNM, PSO

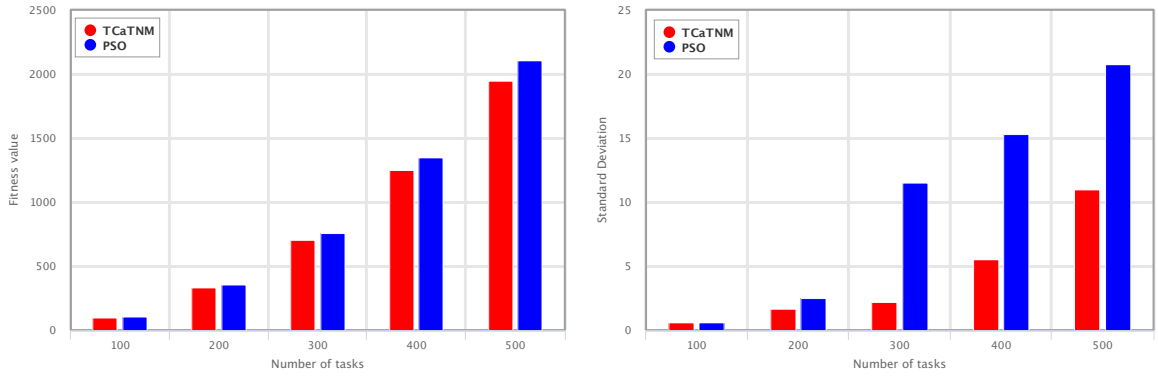
may not be able to converge in a limited amount of iterations or examine the entire huge space with a small population. Furthermore, figure 2a also shows the scalability potential of TCaTNM as the performance gap between TCaTNM and PSO gets wider as the number of tasks increases. This result shows that TCaTNM algorithm can adapt better to more demanding environments. Not only that, in Figure 2b, we can see the increasing trend in the deviation of both TCaTNM and PSO since the more number of tasks there are, the bigger the search space becomes. It can also be seen clearly that the deviation of TCaTCM is much lower than PSO.

Table 4 shows the significant difference in execution time and processing cost between the TCaTNM method and the PSO algorithm. The processing cost of the TCaTNM's solution is 2.48 times lower than that of PSO for the 100 tasks dataset (59.62% improved). As the number of tasks increases, the difference becomes more considerable to nearly 2.89 times cheaper cost for TCaTNM (65.37% improved). This result shows that the TCaTNM algorithm can greatly reduce the computation cost compared to PSO under different datasets and TCaTNM can scale better with task population demand.

Considering the processing time aspect, when the number of tasks is small (100), TCaTNM still holds a competitive processing time (only 8.98%) worse than PSO. The *Improved* metric is the worst in the 200-task scenario. Starting from that, as the number of tasks increases, TCaTNM begins to improve the processing time at a faster rate. In specific, the processing time gap slowly decreases, and in the case of 500 tasks, it remains only 6.55%. Table 4 illustrates that there can be a trade-off between processing time and cost in some scenarios. However, the cost benefit can greatly outweighs the drawbacks of processing time.

5.2.2. Increasing number of vehicles

Figure 3 shows the fitness value of TCaTNM and PSO over 200 generations in different number-of-vehicles settings. Here, TCaTNM only underperforms when there are too few



(a) Fitness (b) Standard deviation

Figure 2: Fitness and standard deviation comparison of TCaTNM, PSO with different number of tasks.

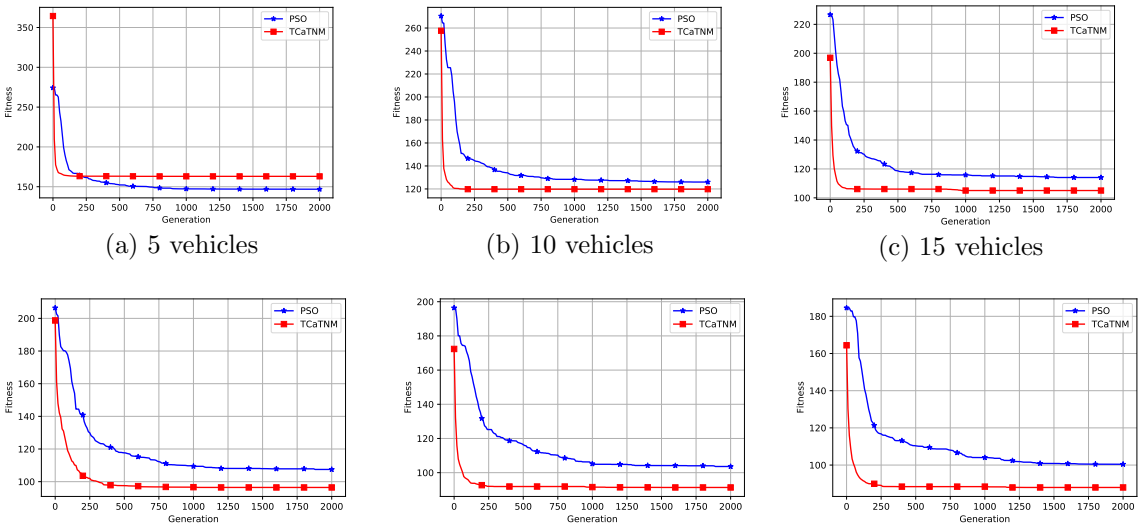
Table 4: Time and Cost between TCaTNM and PSO with different number of tasks

Number of tasks	Time			Cost		
	TCaTNM	**PSO**	**Improved**	**TCaTNM**	**PSO**	**Improved**
100	168.67	**154.77**	-8.98%	**23.89**	59.16	59.62%
200	620.08	**559.95**	-10.74%	**52**	138.93	62.57%
300	1334.96	**1231.19**	-8.43%	**78.69**	221.61	64.49%
400	2388.98	**2227.84**	-7.23%	**108.38**	309.92	65.03%
500	3753.49	**3522.85**	-6.55%	**140.35**	405.32	65.37%

number of vehicles (5 vehicles). As the number of vehicles increases, the target fitness value of TCaTNM gradually becomes better, and the performance gap between the two algorithms also widens. In specific, in cases of 25 - 30 vehicles, TCaTNM keeps the fitness score below 50 while PSO has much slower learning rate and converges at above 100. This result proves that TCaTNM algorithm can scales its performance with more computing nodes.

Figure 4a compares the converged fitness value between TCaTNM and PSO for the number of vehicles increasing from 5 to 30. Compared to the scenario in Subsection 5.2.1, the trend of fitness value is decreasing instead of increasing. More available computing vehicles can save both computing time and cost. Overall, the TCaTNM algorithm is superior to PSO regardless of the number of vehicles. Furthermore, TCaNM scales the improved fitness value when the number of vehicles increases - 1.5% better fitted-score at five vehicles to 12.3% better fitted-score at 30 vehicles than PSO. This is because the GA-based TCaTNM method effectively discovers the whole search space to identify the globally optimal solution, whereas PSO is prone to the local extreme. However, the deviations of performances of both algorithms seem not depends too much on the number of vehicles as can be seen in Figure 4b.

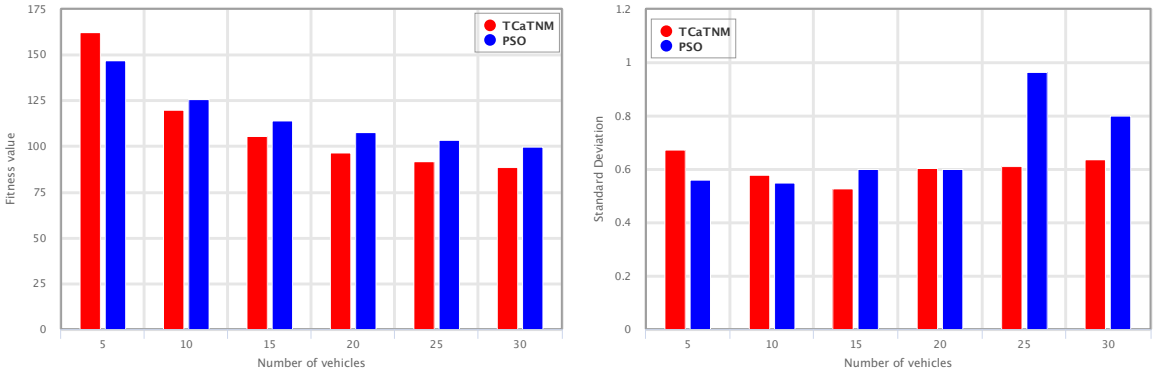
Table 5 presents the processing cost and execution time of two algorithms when the number of processing nodes changes. TCaTNM saves from 41.35% to 61.02% of the monetary cost compared to PSO. In terms of processing time, TCaTNM algorithm maintains a competitive performance against PSO with around 10% difference in cases of more than 5 vehicles. Both algorithms effectively decrease the processing time when there are more computing nodes to process tasks. Table 5 shows that the cost-benefit of TCaTNM still outweighs the difference in processing time.



(a) 5 vehicles (b) 10 vehicles (c) 15 vehicles

(d) 20 vehicles (e) 25 vehicles (f) 30 vehicles

Figure 3: Convergence comparison of TCaTNM, PSO



(a) Fitness (b) Standard deviation

Figure 4: Fitness and standard deviation comparison of TCaTNM, PSO with different number of vehicles.

Table 5: Time and Cost between TCaTNM and PSO with different number of vehicles

Number of vehicles	Time			Cost		
	TCaTNM	PSO	Improved	TCaTNM	PSO	Improved
5	297.64	**247.13**	-20.44%	**27.33**	46.6	41.35%
10	214.23	**197.86**	-8.27%	**25.3**	53.88	53.04%
15	186.33	**169.89**	-9.68%	**25.15**	57.92	56.58%
20	168.67	**155.24**	-8.65%	**23.89**	60.04	60.21%
25	159.72	**146.06**	-9.35%	**24.06**	60.85	60.46%
30	152.89	**138.25**	-10.59%	**23.91**	61.34	61.02%

6. CONCLUSIONS

In this work, we focus on the task-node mapping problem for the Vehicular Fog Computing environment. The proposed evolutionary algorithm TCaTNM is introduced and evaluated in various experiments. Compared with PSO, the proposed algorithm outperforms over 5 datasets of tasks and 6 datasets of vehicles in terms of the trade-off between execution time and processing cost (TCaTNM can save up over 60% of the monetary cost while only taking up to less than 10% longer to process). TCaTNM also has a much faster learning rate in all instances and has better scaling potential as the environment becomes more task demanding and when there are more computing nodes. We will research, improve, and apply more algorithms to solve the bag of task offloading problems in the future. In addition, we plan to expand the problem by focusing on optimizing many other objectives, such as time, transmission costs, computing resources, energy consumption to satisfy users. Budget, deadline, and resource limitation constraints can be added for greater practicality.

ACKNOWLEDGMENT

This research is funded by Ministry of Education and Training of Vietnam under grant number B2020-BKA-13.

REFERENCES

[1] A. Bazzi, A. Zanella, and B. M. Masini, "An ofdma-based mac protocol for next-generation vanets," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 9, pp. 4088–4100, 2014.

[2] H. T. T. Binh, T. T. Anh, D. B. Son, P. A. Duc, and B. M. Nguyen, "An evolutionary algorithm for solving task scheduling problem in cloud-fog computing environment," in *Proceedings of the Ninth International Symposium on Information and Communication Technology*, 2018, pp. 397–404.

[3] H. T. T. Binh, N. P. Le, N. B. Minh, T. T. Hai, N. Q. Minh, and D. B. Son, "A reinforcement learning algorithm for resource provisioning in mobile edge computing network," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–7.

[4] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint load balancing and offloading in vehicular edge computing and networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4377–4387, 2018.

[5] A. B. de Souza, P. A. L. Rego, P. H. G. Rocha, T. Carneiro, and J. N. de Souza, "A task offloading scheme for wave vehicular clouds and 5g mobile edge computing," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.

[6] D. S. Dias, L. H. M. Costa, and M. D. de Amorim, "Data offloading capacity in a megalopolis using taxis and buses as data carriers," *Vehicular Communications*, vol. 14, pp. 80–96, 2018.

[7] J. Du, F. R. Yu, X. Chu, J. Feng, and G. Lu, "Computation offloading and resource allocation in vehicular networks based on dual-side cost minimization," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1079–1092, 2018.

[8] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84–106, 2013.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability.* Freeman San Francisco, 1979, vol. 174.

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

[11] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, "Vehicular fog computing: A viewpoint of vehicles as the infrastructures," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 6, pp. 3860–3873, 2016.

[12] C. Huang, R. Lu, and K.-K. R. Choo, "Vehicular fog computing: architecture, use case, and security and forensic challenges," *IEEE Communications Magazine*, vol. 55, no. 11, pp. 105–111, 2017.

[13] IDC, "The growth in connected iot devices is expected to generate 79.4zb of data in 2025, according to a new idc forecast." 2020. [Online]. Available: https://perma.cc/RNG6-HVJV

[14] H. Izakian, B. Tork Ladani, K. Zamanifar, and A. Abraham, "A novel particle swarm optimization approach for grid job scheduling," in *International Conference on Information Systems, Technology and Management.* Springer, 2009, pp. 100–109.

[15] J. G. Jetcheva, Y.-C. Hu, S. PalChaudhuri, A. Kumar, S. David, and B. Johnson, "Design and evaluation of a metropolitan area multitier wireless ad hoc network architecture," 2003.

[16] B. M. Nguyen, H. Thi Thanh Binh, T. The Anh, and D. Bao Son, "Evolutionary algorithms to optimize task scheduling problem for the iot based bag-of-tasks application in cloud–fog computing environment," *Applied Sciences*, vol. 9, no. 9, p. 1730, 2019.

[17] Z. Ning, J. Huang, and X. Wang, "Vehicular fog computing: Enabling real-time traffic management for smart cities," *IEEE Wireless Communications*, vol. 26, no. 1, pp. 87–93, 2019.

[18] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal *et al.*, "Mobile-edge computing introductory technical white paper," *White Paper, Mobile-Edge Computing (MEC) Industry Initiative*, pp. 1089–7801, 2014.

[19] X.-Q. Pham, T.-D. Nguyen, V. Nguyen, and E.-N. Huh, "Joint node selection and resource allocation for task offloading in scalable vehicle-assisted multi-access edge computing," *Symmetry*, vol. 11, no. 1, p. 58, 2019.

[20] A. U. Rahman, A. W. Malik, V. Sati, A. Chopra, and S. D. Ravana, "Context-aware opportunistic computing in vehicle-to-vehicle networks," *Vehicular Communications*, vol. 24, p. 100236, 2020.

[21] D. B. Son, V. T. An, T. T. Hai, B. M. Nguyen, N. P. Le, and H. T. T. Binh, "Fuzzy deep q-learning task offloading in delay constrained vehicular fog computing," in *2021 International Joint Conference on Neural Networks (IJCNN).* IEEE, 2021, pp. 1–8.

[22] D. B. Son, T. H. Binh, H. K. Vo, B. M. Nguyen, H. T. T. Binh, and S. Yu, "Value-based reinforcement learning approaches for task offloading in delay constrained vehicular edge computing," *Engineering Applications of Artificial Intelligence*, vol. 113, p. 104898, 2022.

[23] 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 Transactions on Vehicular Technology*, vol. 68, no. 4, pp. 3061–3074, 2019.

[24] M. Tahmasebi and M. R. Khayyambashi, "An efficient model for vehicular cloud computing with prioritizing computing resources," *Peer-to-Peer Networking and Applications*, vol. 12, no. 5, pp. 1466–1475, 2019.

[25] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, 2018.

[26] H. Wang, X. Li, H. Ji, and H. Zhang, "Federated offloading scheme to minimize latency in mec-enabled vehicular networks," in *2018 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2018, pp. 1–6.

[27] Y. Xiao and Chao Zhu, "Vehicular fog computing: Vision and challenges," in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2017, pp. 6–9.

[28] D. Ye, M. Wu, S. Tang, and R. Yu, "Scalable fog computing with service offloading in bus networks," in *2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud)*. IEEE, 2016, pp. 247–251.

[29] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 7944–7956, 2019.

[30] H. Zheng, W. Chang, and J. Wu, "Traffic flow monitoring systems in smart cities: Coverage and distinguishability among vehicles," *Journal of Parallel and Distributed Computing*, vol. 127, pp. 224–237, 2019.

Received on March 22, 2022
Accepted on December 20, 2022