Journal of Computers Vol. 36 No. 5, October 2025, pp. 201-218
https://doi.org/10.63367/199115992025103605014

A Priority-Based Dynamic Rescheduling Offloading Strategy for Dependent Task Offloading in the Internet of Vehicles

Zhongyu Ma[1], Jianbing Pu[1], Zhirong Liu[1], Ning Zhang[1],
Zhanjuna Hao[1], and Qun Guo[2*]

[1] College of Computer Science and Engineering, Northwest Normal University,
Lanzhou 730070, China
{mazybg, 2022222311, 2023212049, 2023222071, haozhj}@nwnu.edu.cn

[2] College of Electrical and Information Engineering, Lanzhou University of Technology,
Lanzhou, 730050, China
guoqun@lut.edu.cn

Received 5 August 2024; Revised 26 December 2024; Accepted 25 January 2025

Abstract. The problem of offloading computational tasks in the Internet of Vehicles (IoV) has received academic attention in recent years. A key aspect of this study is reducing task scheduling latency. Among these tasks, there is a class of dependent tasks. Offload scheduling of these tasks is complex and they are usually represented by Directed Acyclic Graphs (DAG). This paper explores the problem of dependent task offloading in IoV. Firstly, we model the problem as a Mixed Integer Nonlinear Programming (MINLP) problem. Because this problem is difficult to solve, we decompose it into two subproblems. Secondly, we design the Critical Path based Prioritisation Determination (CPPD) algorithm and the Priority-based Computing Device Allocation (PCDA) algorithm to solve these subproblems, respectively. These two algorithms together form the Priority-based Dynamic Rescheduling Offloading Strategy (PDROS), which can dynamically adjust the unloading scheme according to the changes of the information of the connected vehicle system. Finally, we compare the PDROS algorithm with three existing dependency task offloading algorithms from recent years. Under our experimental settings, the performance of our algorithm is at least 16.39% better than other algorithms when the system status changes.

Keywords: Internet of Vehicles, MultiInternet of Vehicles-Access Edge Computing, dependent task, computing devices

1 Introduction

In recent years, the rapid development of the Internet of Vehicles (IoV) and autonomous driving technologies has gradually become a crucial component of modern transportation systems. IoV connects vehicles, infrastructure, and user devices, enabling real-time information sharing and communication, thereby improving traffic management efficiency and driving safety. However, with the continuous emergence of various in-vehicle applications, the number and complexity of computing tasks have rapidly increased, posing higher demands on computing resources and energy consumption. The limited computing capability of existing vehicles cannot meet the low-latency requirements of these applications. Offloading computing tasks generated by in-vehicle applications to cloud servers can lead to unacceptable transmission delays [1, 2].

To address this issue, the academic community has introduced the Multi-access Edge Computing (MEC) paradigm as a solution [3]. MEC deploys computing servers and storage resources at the network edge, such as base stations (BS) and access points (AP), to handle computing tasks near the user. Users can upload computing tasks to nearby Roadside Units (RSU), which then offload the tasks to MEC servers for processing. The computing results are subsequently returned to the user devices. Additionally, RSUs can transmit road information and nearby computing device information to vehicles, aiding them in making better decisions. Due to the varying data sizes and computational complexities of each task, selecting the most suitable offloading device for each computational task is a significant research problem. It requires a comprehensive consideration of both the task information and the computing device information within the Internet of Vehicles (IoV). Unreasonable offloading decisions

* Corresponding Author

may lead to excessive task offloading delays and waste computational resources [4-6]. Therefore, finding the optimal offloading strategy from numerous decisions to minimize the total computational delay is crucial.

Among these computational tasks, there is a special category known as dependent tasks [7-10]. Dependent tasks typically consist of multiple subtasks, where each subtask relies on the result of the previous subtask to proceed. The offloading delay of a subtask is influenced not only by its own offloading decision but also by the decisions made for preceding tasks. In dependent tasks, subtasks can be offloaded to different computing devices. Due to the dependency constraints, suboptimal offloading decisions can increase the waiting time for subtasks and the data transmission time between computing devices, significantly affecting the total computational delay. As the number of subtasks and computing devices increases, exhaustively searching all possible offloading decisions becomes impractical.

Furthermore, Additionally, the mobility of vehicles makes the task offloading problem more complex and challenging. As vehicles move, their positions constantly change, leading to rapid changes in network topology and wireless channel conditions [11]. This dynamic nature requires task vehicles to frequently collect and update system information to ensure the effectiveness and accuracy of the computation offloading strategy. If the system information is outdated, it can result in poor offloading decisions, thereby increasing the task computation delay. To address these changes, task vehicles must continuously monitor and update the surrounding network environment and resource status. However, frequent data collection and updates introduce additional time costs and energy consumption. Each information update not only occupies computational resources but also consumes a significant amount of energy, especially in complex environments where such resource consumption is even more pronounced. Therefore, the mobility of vehicles significantly increases the complexity of the task offloading problem. This complexity requires systems to make quick and accurate decisions in a dynamically changing environment while also balancing resource consumption and transmission delay.

Existing offloading scheduling algorithms often fail to fully consider the characteristics of fast-moving vehicles. For dependent tasks, all subtask offloading schemes are typically determined at the beginning. However, in reality, since dependent tasks consist of multiple subtasks, the channel environment at the time of offloading the first subtask and the last subtask often differ. When the vehicle's position changes, the distance between the vehicle and the MEC servers changes, and the channel state also changes. At this point, the offloading scheme for the remaining unscheduled subtasks should be adjusted accordingly. Additionally, most offloading scheduling algorithms overlook the data transmission time between subtasks. In scenarios with a large number of subtasks, this oversight can significantly impact the performance of the scheme.

The key research problem of this work is how to offload and schedule dependent tasks generated by high-speed moving vehicles in the IoV in the shortest possible time. To address this, this paper proposes a Priority-based Dynamic Rescheduling Offloading Strategy (PDROS). The strategy detects changes in system status after each subtask upload and updates system information and adjusts the remaining subtask offloading strategy only when significant changes occur, such as the addition or removal of computing devices. Specifically, this paper first prioritizes the subtasks in dependent tasks using a priority sorting algorithm and then determines the optimal offloading devices for each subtask based on their priority. After each subtask is uploaded, the system status is checked for updates, and if needed, the offloading scheduling plan is adjusted accordingly. This process is repeated until all tasks are offloaded. Simulation experiments demonstrate that, compared to existing algorithms, the proposed algorithm's priority sorting strategy is more precise and effective, resulting in shorter offloading times.

The main contributions are summarized as follows:

- Firstly, we formulate the problem of minimizing the total time spent on task offload scheduling in the IoV as a Mixed Integer Nonlinear Programming (MINLP) problem, denoted as problem P0. Due to the complexity of solving this problem directly, we decompose it into sub-problems P1 and P2. Sub-problem P1 involves determining the upload sequence of sub-tasks, while sub-problem P2 focuses on selecting the offloading devices for these sub-tasks.
- Secondly, using critical path theory, we propose a new Priority-based Dynamic Rescheduling Offloading Strategy (PDROS) aimed at reducing the time spent on task offload scheduling in IoV. The proposed PDROS consists of CPPD and PCDA algorithms, and dynamically adjusts the offloading scheme throughout the task offloading process in response to changes in the system environment, ensuring completion of the entire dependency task offload. Subsequently, we analyze the algorithm's complexity.
- Finally, we compare it with three existing dependency task offloading algorithms from recent years to validate the performance of the proposed algorithm. Results demonstrate that our proposed algorithm outperforms the comparison algorithms. Specifically, under our experimental settings, our algorithm reduces the total time spent on task offload scheduling by 16.39%.

The remainder of this paper is structured as follows. Section 2 reviews related research, Section 3 models the studied problem as a mathematical model, Section 4 proposes a solution algorithm for the problem, Section 5 compares the algorithm with the existing algorithms through simulation experiments, and Section 6 concludes the paper.

2 Related Work

2.1 Computational Task Offload

With the rapid advancement of communication and computing technologies, more and more mobile applications and services have emerged. The demand for computational resources by these mobile applications has also increased significantly, making the computational task offload scheduling problem the focus of many experts. This research focuses on different aspects of MEC systems, such as latency [12], energy consumption [13], caches [14], throughput [15], and commercial revenue [16], among others. Huang et al. [17] propose a deep reinforcement learning model to solve the edge computing task offloading problem, but the considered network scenarios are such that each task can only choose to compute locally or offload to a specific deterministic edge server, without considering multiple edge servers. Yang et al. [18] proposed a distributed algorithm to solve the computation offloading problem, which includes two optimization algorithms for computational resource allocation and offloading policy. Sun et al. [19] proposed a learn-while-offloading strategy, where the task vehicle can learn the latency performance of the system while offloading the task.

2.2 Dependent Task Offload

Compared with general computing tasks, dependent tasks are difficult to apply general offloading algorithms due to the complex dependencies between sub-tasks, and many scholars have studied the dependent task offloading problem separately. Mo et al. [20] proposed a graph convolutional neural network model to optimize the total offloading delay in the dependent task offloading problem. Hou et al. [21] investigated the optimal offloading scheme considering memory space by finding the optimal offloading sequence of dependent tasks and thus optimizing memory space usage. Nan et al. [22] partitioned a single task into multiple sequential sub-tasks and then offloaded them to the base station or auxiliary nodes, with the aim of optimizing the energy usage of the offloading nodes and auxiliary nodes. Lv et al. [23] proposed a heuristic algorithm based on the Time Consumption Table (TCT), by predicting the offloading time of each sub-task and recording it in the TCT to select a more optimal scheduling scheme. The research shortcomings are that the algorithm's complexity is high, making it difficult to adapt to the complex scenario of multiple nodes. Wang et al. [24] proposed a meta-reinforcement learning model for solving the dependency task offloading problem. This model learns the offloading strategy through interactions between the user side and the MEC server, demonstrating faster adaptation to new system environments compared to conventional reinforcement learning models. Chai et al. [25] proposed a dynamic priority-based computation scheduling and offloading strategy, which can continuously update sub-task priorities during the offloading process.

2.3 Task Offload During Movement

Various approaches have been proposed to address the computational offloading challenge in mobile applications. Thananjeyan et al. [26] formulated the energy optimization problem to minimize the total energy consumption of the MEC system, taking into account application requirements such as delay constraints and user mobility. Li et al. [27] proposed a multi-hop task offloading decision model to improve the efficiency of task execution. Li et al. [28] used particle swarm optimization to solve the offloading problem. In order to meet the fast response requirements, a large number of studies have been devoted to minimizing the total execution time. Song et al. [29] proposed a Multi-Objective Evolutionary Algorithm/Distributed (MOEA/D) to solve the multi-objective computational offloading problem. Deng et al. [30] proposed offloading partitioning methods to reduce the cost. Taken together, existing studies have investigated optimization strategies for computational offloading from multiple perspectives, but further research is needed to deeply consider offloading decisions in dynamic and complex environments to achieve adaptation to environmental uncertainty.

Although there is a large amount of literature on the dependency task offloading problem, many of these studies do not consider the fluctuations in system networks and the high-speed movement of vehicles. In this paper, based on previous studies, we propose a PDROS algorithm to solve the problem of scheduling dependent tasks in IoV. Compared with the existing dependency task offloading algorithms, it reduces the delay consumption and improves the efficiency of computing device usage.

3 System Model

3.1 Channel Modelling

We define the set of server vehicles that can act as computing devices as $\mathcal{V} = \{1, 2, ..., V\}$, and the set of MEC servers as $S = \{1, 2, ..., S\}$, and the task vehicle itself acts as a computing device denoted as V_0. Therefore, the set of all computing devices is $C = \{V_0\} \cup \mathcal{V} \cup S = \{0, 1, 2, ..., V, V{+}1, ..., C\}$. The general system model diagram is shown in Fig. 1.

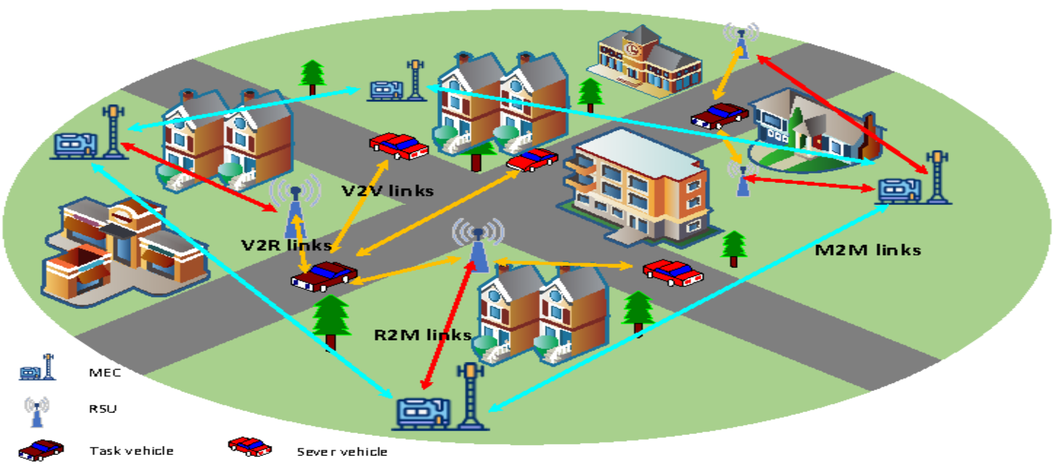


Fig. 1. Schematic diagram of task offloading in IoV

We use the binary variable x to indicate whether or not the computing devices $c \in \mathcal{C}$ and $c' \in \mathcal{C}$ are connected at time t, as shown in equation (1),

$$x_{c,c'}(t) = \begin{cases} 1, & c \text{ is associated with } c' \text{ at } t \text{ time}, \\ 0, & \text{else}. \end{cases} \quad (1)$$

Using the Non-Orthogonal Frequency Division Multiple Access (NOMA) channel model, the transmission rate $R_{c,c'}$ when transmitting data from computing device c to c' can be expressed as,

$$R_{c,c'} = B \log_2 \left(1 + \frac{x_{c,c'}(t) p_c h_{c,c'} (l_{c,c'}(t))^{-\alpha}}{\sum_{s \in C, s \neq c} x_{s,c'}(t) p_s h_{s,c'} (l_{s,c'}(t))^{-\alpha} + \sigma^2}\right), \quad (2)$$

where B indicates system bandwidth, $l_{c,c'}$ is the distance between computing device c and c' at time t, p_c is the transmit power of computing device c, h denotes the channel gain, and σ^2 denotes additive Gaussian white noise.

3.2 DAG Task Modelling

Dependent tasks are usually represented by a Directed Acyclic Graph (DAG), where each node represents a sub-task and each edge signifies a dependency relationship between sub-tasks as shown in Fig. 2. Each sub-task mainly considers three attributes (ω, d^{up}, d^{down}), where ω represents the computational workload of the sub-task, specifically expressed as the number of CPU cycles required for the computation. d^{up} represents the size of uploaded data and d^{down} denotes the size of the result data passed between sub-tasks. When sub-task n_j requires the execution result of sub-task n_i before it can start execution, we call n_i a predecessor sub-task of n_j. At the same time, n_j is also a successor sub-tasks of n_i. Dependencies between sub-tasks are given by edges in the DAG, pointing from a predecessor sub-task to a successor sub-task. The sets of predecessor sub-tasks and successor sub-tasks are denoted by *pred*(i) and *succ*(i).

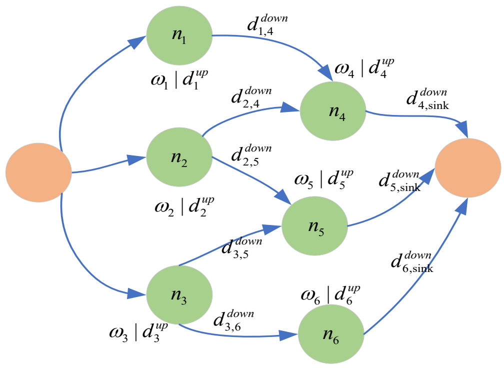


Fig. 2. Example diagram of a DAG task

For ease of representation, we denote the dependencies between sub-tasks by a matrix $\boldsymbol{M}$. The representation of element m in $\boldsymbol{M}$ is shown in equation (3),

$$m_{i,j} = \begin{cases} 1, \text{ if } i \neq j, n_j \text{ dependent on } n_i, \\ 0, \text{ else.} \end{cases} \tag{3}$$

Therefore, the sets of predecessor sub-task and successors sub-tasks for sub-task n_i are represented by equations (4) and (5), respectively,

$$pred(i) = \{n_j \mid m_{j,i} = 1, n_j \in N\}, \tag{4}$$

$$succ(i) = \{n_j \mid m_{i,j} = 1, n_j \in N\}. \tag{5}$$

It is assumed that each computing device can execute only one task at a time, and non-preemptive scheduling is implemented, meaning that computational resources allocated to a sub-task are not preempted by other sub-tasks until the former is completed. The offload device assignment matrix of sub-tasks is represented by the binary matrix $\boldsymbol{Y}$. Specifically, for each element y in $\boldsymbol{Y}$ there are,

$$y_{i,c} = \begin{cases} 1, \text{ if } n_i \text{ is offloaded to } c, \\ 0, \text{ else.} \end{cases} \tag{6}$$

Since servers have limited computational and storage resources, each sub-task can be offloaded to only one server. Each sub-task must be executed and should not be executed more than once, i.e.,

$$\sum_{c\in\mathcal{C}} y_{i,c} = 1,\ \forall n_i \in N, c \in C, \tag{7}$$

The upload time required to offload the sub-task i to the computer c is

$$t_i^{up} = \begin{cases} \sum_{c\in\mathcal{C}} y_{i,c} \dfrac{d_i^{up}}{R_{0,c}},\ \text{if } c \neq 0, \\ 0,\ \text{else.} \end{cases} \tag{8}$$

The computation time of sub-task i on device c is shown in the following equation (9),

$$t_i^{comp} = \sum_{c\in\mathcal{C}} \frac{y_{i,c}\omega_i}{f_c},\ \forall n_i \in \mathcal{N}, c \in \mathcal{C}, \tag{9}$$

Assuming that sub-task n_i has a predecessor sub-task n_j, that n_i is computed at device c and that n_j is computed at device c', the time for the transfer of result data from sub-task n_j to sub-task n_i can be expressed as equations (10),

$$t_{j,i}^{down} = \begin{cases} \sum_{c\in\mathcal{C}} y_{i,c} \sum_{c'\in\mathcal{C}} y_{j,c'} \dfrac{d_{j,i}^{down}}{R_{cc'}},\ \text{if } c' \neq c, \\ 0,\ \text{else.} \end{cases} \tag{10}$$

3.3 Time Slot Modelling

The starting moment of the task must be later than the moment of availability of the computing device, it satisfies,

$$T_i^{start} \geq \sum_{c\in\mathcal{C}} y_{i,c} T_c^{ava}, \tag{11}$$

where T_i^{start} denotes the time slot when sub-task n_i starts computing, T_c^{ava} denotes the time slot when the computing device c is idle and available.

In addition, the start time T_i^{start} of sub-task n_i must be later than the return time slot of the calculation results of all its preceding sub-tasks $n_j \in pred(i)$,

$$T_i^{start} \geq \max_{j\in pred(i)} \left\{ T_j^{finish} + t_{j,i}^{down} \right\}, \tag{12}$$

where T_j^{finish} denotes the finish time of sub-task n_j, which is calculated as,

$$T_i^{finish} = T_i^{start} + t_i^{comp},\ \forall n_i \in \mathcal{N}. \tag{13}$$

The available time T_c^{ava} of the computing device c is set to coincide with the finish time T_i^{finish} of the last sub-task n_i offloaded to the device, and the available time T_c^{ava} of the computing device c is updated whenever a new sub-task n_i is offloaded to the c, i.e.,

$$T_c^{ava} = \max_{i\in N}(y_{i,c} T_i^{finish}),\ \forall c \in \mathcal{C}. \tag{14}$$

We specify that the task vehicle can only offload one sub-task in the same time slot. As shown in Fig. 3, Take the DAG task in Fig. 2 as an example, assuming that sub-task n_1 is offloaded to computing device $c1$, when the task vehicle uploads the sub-task to $c1$, the task vehicle's transmission resources are busy, and therefore, it cannot transmit any other tasks. Furthermore, when n_1 is uploaded, the task vehicle can start transmitting subsequent sub-tasks. When sub-task n_4 is uploaded to $c3$, $c3$ cannot start calculating n_4 immediately, but only after the calculation results of its predecessor sub-tasks n_1 and n_2 have been transferred to $c3$.

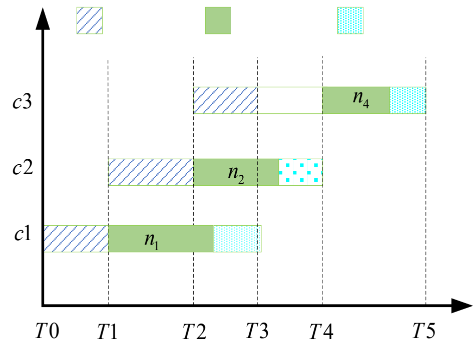


Fig. 3. Example diagram of time slot modelling

Therefore, the start time slot of sub-task n_i satisfies the constraint of equation (15):

$$T_i^{start} \geq \max_{j \in pred(i)} \{\sum_{c \in \mathcal{C}} y_{i,c} T_c^{ava},\ T_j^{finish} + t_{j,i}^{down},\ T^{trans} + t_i^{up}\}, \tag{15}$$

where T^{trans} denotes the time slot when the task vehicle has finished uploading the previous sub-task.

3.4 Problem Formulation

In summary, we want to optimize the total time for dependency task offloading, so the problem can be modelled as problem,

$$\begin{aligned}
&\text{P0}: \quad \min T_{\text{sink}}^{\text{finish}} \\
&\text{s.t.C1}: \sum_{c \in \mathcal{C}} y_{i,c} = 1,\ \forall n_i \in N, c \in C, \\
&\text{C2}: t_i^{up} = \begin{cases} \sum_{c \in \mathcal{C}} y_{i,c} \dfrac{d_i^{up}}{R_{0,c}}, \text{ if } c \neq 0, \\ 0, \text{ else}, \end{cases} \\
&\text{C3}: t_i^{comp} = \sum_{c \in \mathcal{C}} \frac{y_{i,c}\omega_i}{f_c},\ \forall n_i \in \mathcal{N}, c \in \mathcal{C}, \\
&\text{C4}: t_{j,i}^{down} = \begin{cases} \sum_{c \in \mathcal{C}} y_{i,c} \sum_{c' \in \mathcal{C}} y_{j,c'} \dfrac{d_{j,i}^{down}}{R_{cc'}}, \text{ if } c' \neq c, \\ 0, \text{ else}, \end{cases} \\
&\text{C5}: T_c^{ava} = \max_{i \in N}(y_{i,c} T_i^{finish}),\ \forall c \in \mathcal{C}, \\
&\text{C6}: T_i^{start} \geq \max_{j \in pred(i)} \{\sum_{c \in \mathcal{C}} y_{i,c} T_c^{ava},\ T_j^{finish} + t_{j,i}^{down},\ T^{trans} + t_i^{up}\}, \\
&\text{C7}: T_i^{finish} = T_i^{start} + t_i^{comp},\ \forall n_i \in \mathcal{N}, \\
&\text{C8}: x_{c,c'},\ y_{i,j},\ m_{i,j} = \{0,\ 1\},\ \forall n_i,\ n_j \in \mathcal{N},\ c \in \mathcal{C}.
\end{aligned} \tag{16}$$

The objective function aims to minimize the total offload scheduling time in problem P0. C1 imposes constraints on the use of system resources. C2-C4 are constraints on the time required for sub-task transmission and computation. C5~C7 are constraints on specific moments of state change of sub-tasks and offloading devices. C8 is a constraint related to binary indicative variables. From the modeling of the problem, it can be seen that this problem is a MINLP problem, i.e., an NP-hard problem [31], which is difficult to solve directly. In the next section, we will propose an algorithm to solve it.

4 Algorithm Design

4.1 Problem Breakdown

The offloading scheme sought in problem P0 can be divided into two parts, the choice of upload order and the allocation of computing devices. On the one hand, if the offloading devices are uncertain, the time required to offload the sub-tasks is unknown, making it difficult to determine the priority of the tasks. On the other hand, if the upload order of the sub-tasks is unknown, the uncertainty of the start time of the sub-tasks makes it difficult to determine the allocation scheme of the offloading devices.

Therefore, we use the coordinate descent method [32] to decompose problem P0 into two subproblems P1 and P2, where subproblem P1 assumes that the offload device allocation scheme $\boldsymbol{Y}$ is known to solve for the sub-task upload order $\boldsymbol{\Phi}$, and subproblem P2 assumes that the sub-tasks upload order $\boldsymbol{\Phi}$ is known to solve for the offload device allocation scheme $\boldsymbol{Y}$.

$$
\begin{aligned}
&\text{P1:}\quad \min T_{\text{sink}}^{\text{finish}}(\boldsymbol{Y})\\
&\text{s.t.C5:} T_c^{ava} = \max_{i\in N}(y_{i,c}T_i^{finish}),\ \forall c\in\mathcal{C},\\
&\quad \text{C6:} T_i^{start} \geq \max_{j\in pred(i)}\{\sum_{c\in\mathcal{C}} y_{i,c}T_c^{ava},\ T_j^{finish}+t_{j,i}^{down},\ T^{trans}+t_i^{up}\},\\
&\quad \text{C7:} T_i^{finish} = T_i^{start}+t_i^{comp},\ \forall n_i\in\mathcal{N},\\
&\quad \text{C8:} x_{c,c'},\ y_{i,j},\ m_{i,j}=\{0,\ 1\}, \forall n_i,\ n_j\in\mathcal{N},\ c\in\mathcal{C}.
\end{aligned}
\tag{17}
$$

$$
\begin{aligned}
&\text{P2}\quad \min T_{\text{sink}}^{\text{finish}}(\boldsymbol{\Phi})\\
&\text{s.t.C1}: \sum_{c\in\mathcal{C}} y_{i,c}=1,\ \forall n_i\in N, c\in C,\\
&\quad \text{C2:} t_i^{up} = \begin{cases}\sum_{c\in\mathcal{C}} y_{i,c}\dfrac{d_i^{up}}{R_{0,c}},\ \text{if}\ c\neq 0,\\ 0,\ \text{else},\end{cases}\\
&\quad \text{C3:} t_i^{comp} = \sum_{c\in\mathcal{C}}\frac{y_{i,c}\omega_i}{f_c},\ \forall n_i\in\mathcal{N}, c\in\mathcal{C},\\
&\quad \text{C4:} t_{j,i}^{down} = \begin{cases}\sum_{c\in\mathcal{C}} y_{i,c}\sum_{c'\in\mathcal{C}} y_{j,c'}\dfrac{d_{j,i}^{down}}{R_{cc'}},\ \text{if}\ c'\neq c,\\ 0,\ \text{else},\end{cases}\\
&\quad \text{C8:} x_{c,c'},\ y_{i,j},\ m_{i,j}=\{0,\ 1\},\ \forall n_i,\ n_j\in\mathcal{N},\ c\in\mathcal{C}.
\end{aligned}
\tag{18}
$$

In this section, we propose a Priority-based Dynamic Rescheduling Offloading Strategy (PDROS) for offloading DAG tasks. The strategy consists of two algorithms, the Critical Path based Prioritisation Determination (CPPD) algorithm and the Priority-based Computing Device Allocation (PCDA) algorithm. The CPPD algorithm generates the scheduling priorities of sub-tasks based on the computational resource allocation scheme in the IoV network. The PCDA algorithm schedules tasks based on the sub-task scheduling priority generated by the CPPD algorithm. The offloading scheme is then dynamically adjusted according to the real-time changes in the IoV.

4.2 Critical Path based Prioritisation Determination (CPPD) Algorithm

The critical path refers to a path that takes the longest time from start to finish of a task, and prioritizing the execution of sub-tasks on the critical path can reduce the overall execution time of the task. Based on the idea of the critical path, we sort the sub-tasks according to their required computation offload time, and the longer the time spent on a path, the higher the priority of the sub-tasks on that path. Also considering the constraints of the dependent tasks, we let the priority of each sub-task add the maximum priority of the successor sub-tasks, so that the priority of the successor sub-tasks can be passed on to the predecessor sub-tasks, while ensuring that the predecessor sub-tasks can be offloaded earlier. Hence, the priority of each sub-task represents the maximum predicted completion time from its start to the end of the task, it can be expressed as,

$$L(n_i) = T^{trans} + \hat{t}_i^{up} + t_i^{comp} + \max_{j \in succ(i)} (L(n_j) + t_{i,j}^{down}), \ \forall n_i \in \mathcal{N}, \tag{19}$$

where $(\hat{t}^{up}, t^{comp}, t^{down})$ is the predicted values of time parameters.

The exact flow of the CPPD is shown in **Algorithm 1**. In **Algorithm 1**, the first inputs are the task dependency matrix $\boldsymbol{M}$, the offload device allocation scheme $\boldsymbol{Y}$ and the set of sub-tasks to be uploaded $\mathcal{U}$. When the algorithm is run for the first time, all sub-tasks need to be uploaded, so $\mathcal{U} = \mathcal{N}$, and all sub-tasks in the offload matrix $\boldsymbol{Y}$ are designated as locally offloaded, i.e. $y_{i,0} = 1$, $\forall n_i \in \mathcal{N}$. The set of pending sub-tasks $\mathcal{Q}$ is initialised in order to distinguish whether the priority of sub-tasks has already been calculated or not, so that repeated calculations can be avoided.

Algorithm 1. CPPD algorithm for subproblem P1

Input:= $\boldsymbol{M}$, $\boldsymbol{Y}$, $\mathcal{C}$, $\mathcal{U}$
Output:= $\boldsymbol{\Phi}$
Initialization:= $\mathcal{Q} = \{n_{sink}\}$, $T^{trans} = 0$, $PV(n_i) = 0$, $\forall n_i \in N$
1. repeat:=
2.   for= n_i= $\mathcal{Q}$= do
3.   if $succ(i) \cup \mathcal{Q} = \varnothing$ then
4.   for $n_j \in succ(i)$ do
5. $temp = L(n_j) + \hat{t}_{i,j}^{down}$;
6. $L(n_i) \leftarrow max\{L(n_i), temp\}$;
7.   end for
8.   Calculate $L(n_i)$ by equation (19);
9. $\mathcal{Q} \leftarrow \mathcal{Q} - \{n_i\}$;
10.   for $n_j \in pred(i)$ do
11.   if $succ(j) \cup \mathcal{Q} = \varnothing$ then
12. $\mathcal{Q} \leftarrow \mathcal{Q} + \{n_j\}$;
13.   end if
14.   end for
15.   end if
16.   end for
17. until $\mathcal{Q} = \varnothing$;
18. The sub-task sorting is denoted as $\boldsymbol{\Phi}$;
19. return $\boldsymbol{\Phi}$;

To facilitate traversing the sub-tasks, we use a recursive approach to calculate the priority L for each sub-task. Specifically, line 1 of the algorithm is used to ensure that the priority of all pending sub-tasks have been computed. Lines 2 to 3 traverse the set of pending sub-tasks $\mathcal{Q}$, searching for those whose priority L has not been computed, but the priority of the subsequent sub-task has been computed. Lines 4 to 8 are used to calculate the priority L of the sub-task. Lines 9 to 12 update $\mathcal{Q}$ to avoid repeated calculations. Finally, line 18 of the algorithm sorts all the sub-tasks according to their priority to obtain the priority sequence $\boldsymbol{\Phi}$.

4.3 Priority-based Computing Device Allocation (PCDA) Algorithm

For the computational device selection problem, we propose a Priority-based Computing Device Allocation

(PCDA) algorithm. After obtaining the priority sequence Φ of the sub-tasks using Algorithm 1, the problem is transformed into a linear dependency task offloading problem. This involves traversing all sub-tasks according to Φ, calculating the time parameter of each sub-task to be offloaded to each computing device, and selecting the device with the shortest computational offloading time for each sub-task. After traversing all the sub-tasks, the offloading scheme Y is generated. The exact flow of the PCDA algorithm is shown in **Algorithm 2**.

Algorithm 2. PCDA algorithm for subproblem P2

Input: $\boldsymbol{M}$, $\mathcal{C}$, $\mathcal{N}$, $\boldsymbol{\Phi}$
Output: $\boldsymbol{Y}$, T_{sink}^{finish}
Initialization: $T^{final} = \infty$, $T^{ava} = 0$
1.for $n_i \in \boldsymbol{\Phi}$ do
2.   for $c \in C$ do
3.   Calculate t_i^{up} and t_i^{comp}= by equation (8) and (9);
4. $= T_i^{start} = \max(T_c^{ava}, T^{trans} + t_i^{up})$;
5.   for $n_j \in pred(i)$ do
6.   Calculate $t_{j,i}^{down}$ by (10);
7. $T_i^{start} \leftarrow \max(T_j^{finish} + t_{j,i}^{down}, T_i^{start})$;
8.   end for
9. $temp = T_i^{start} + t^{comp}$;
10.   if $T_i^{finish} > temp$ then
11. $T_i^{finish} = temp$;
12. $y_{i,c} = 1, \forall y_{i,c'} = 0, c' \in C, c' \neq c$;
13. $= T^{trans} \leftarrow T^{trans} + t_i^{up}$;
14. $T_c^{ava} \leftarrow T_i^{finish}$;
15.   end if
16.   end for
17.end for

Specifically, lines 1 to 2 iterate through all combinations of sub-tasks and computing devices, lines 3-12 identify the least time-consuming combinations, and lines 13-14 update the time slot status.

4.4 Priority-based Dynamic Rescheduling Offloading Strategy (PDROS) Algorithm

Considering the real-time changes in system information and the rapid movement of vehicles in IoV, we propose the Priority-based Dynamic Rescheduling Offloading Strategy (PDROS) algorithm. This algorithm detects changes in system information after each sub-task upload. For instance, when a new DAG task is generated in the system, a new idle server vehicle joins, or the computing device of an original server vehicle is about to quit or close its computing function, it triggers a system information update. As a result, both the DAG task and the computing device information are updated. In addition, due to the constant movement of the task vehicle during this process, updates to the vehicle position and channel parameters are required as they affect the computation results. The exact flow of the PDROS algorithm is shown in **Algorithm 3**.

Algorithm 3. PDROS algorithm

Input: $\boldsymbol{M}$, $\mathcal{C}$, $\mathcal{N}$
Output: $\boldsymbol{\Phi}$, $\boldsymbol{Y}$, T_{sink}^{finish}
Initialization: $\mathcal{U} = \mathcal{N}, y_{i,0} = 1, y_{i,c} = 0, \forall n_i = N, \forall c \in C, c \neq 0$
repeat:
   if the system status change then
       Updating vehicle V_0 locations;
       Updating computing devices $\mathcal{C}$;
       Updating channel parameters $\boldsymbol{H}$;

```
        input Y to Algorithm 1 and output Φ for subproblem P1;
        input Φ to Algorithm 2 and output Y for subproblem P2;
    end if
    U ← U − {n_i};
    Φ ← Φ − {n_i};
until U = ∅;
return Φ;
```

After updating the system state, **Algorithms 1** and **2** are repeatedly executed to generate new computational offloading scheduling decisions $\boldsymbol{Y}$ and $\boldsymbol{\Phi}$. Sub-tasks that have already been uploaded are excluded from the decision reformulation to avoid duplicate scheduling.

4.5 Algorithmic Complexity Analysis

For Algorithm 1, the loop (line 2) will continue executing until all pending sub-tasks have been processed. In each iteration of the loop, at least one sub-task is processed. The inner loop (lines 5-8) handles successor sub-tasks for each sub-task, calculating priority. The complexity primarily depends on the depth and breadth of the DAG. For each node, it is necessary to traverse all its successor nodes, hence the time complexity approaches $O(N)$, where N is the number of nodes in the DAG.

For Algorithm 2, each execution iterates over all pending sub-tasks (line 2), iterates over all computing devices for each sub-task (line 3), and calculates relevant latency parameters. The inner loop (lines 6-9) traverses all predecessor sub-tasks of the sub-task to determine their arrival times on each computing device. Therefore, the algorithm's complexity is $O(CN)^2$, where C is the number of computing devices.

For Algorithm 3, when the system is not updated, **Algorithms 1** and **2** are invoked only once until all sub-tasks are uploaded, the time complexity is $O(CN^2)$. When the system is updated, the complexity of the algorithm becomes $O(CN^3)$.

5 Simulation and Analysis

In this section, we verify the performance of the algorithm through simulation analysis. The simulation setup is detailed in subsection 5.1, while the experimental analysis is presented in subsection 5.2.

5.1 Simulation Settings

The parameter settings for the simulation experiment are shown in Table 1.

Table 1. Simulation parameter settings

Description	Parameters
Number of MEC servers	$S = 5 \sim 7$
Number of server service vehicles	$V = 2 \sim 10$
Number of sub-tasks in DAG tasks	$N = 100 \sim 500$
Density of directed edges for DAG tasks	$0.2 \sim 0.6$
CPU frequency of MEC servers /GHz	$f_s = 8 \sim 10$
CPU frequency of vehicles /GHz	$f_v = 1.3 \sim 2$
Sub-task upload data size /MB	$d^{up} = 10 \sim 20$
Sub-task result download data size /MB	$d^{down} = 2 \sim 5$
System Bandwidth /GHz	$B = 20$
MEC servers and RSU transmit power /dBm	$p_s = 30$
Vehicles transmit power /dBm	$p_v = 20$

The paper sets up three comparison algorithms selected from existing dependent task scheduling algorithms used in previous research. The comparison algorithms are set up as follows,

1. **Priority-Based Task Scheduling Algorithm (PBTSA):** Proposed by Wang et al. [33] in 2023. The algorithm traverses all sub-tasks in reverse according to the DAG graph, calculates the priority values of all sub-tasks using a recursive approach, and subsequently offloads all sub-tasks according to a greedy strategy.
2. **Priority-Based Level Algorithm (PBLA):** Proposed by literature Zheng et al. [34] in 2022. The algorithm first calculates three quotient values, which are combined into a final sub-tasks priority using a decision tree. The disadvantage is that it only considers the topology of the DAG tasks without considering the workload and data volume of the tasks.
3. **Priority and dependency-based DAG tasks offloading algorithm (PDAGTO):** Proposed by literature Fu et al. [35] in 2021. The algorithm first calculates the priority of sub-tasks according to the average value of each parameter of the system, and subsequently iterates through all the combinations of sub-tasks and computing devices in turn.

5.2 Result Analysis

Fig. 4 shows the performance differences of each algorithm as the number of sub-tasks varies. In this scenario, the DAG directed edge density is set to 0.2, with 5 MEC servers and 5 server vehicles. The performance of these algorithms is assessed using two metrics: the total time spent on offloading scheduling of DAG tasks and the average usage efficiency of the computing devices. The average usage efficiency of the computing devices is obtained by dividing the time spent by the computing device on computation by its total usage time, as expressed in equation (20). The total usage time of the computing device includes sub-task upload time, waiting time for the uploaded sub-tasks, sub-task computation time, and result return time. The higher the average usage efficiency of the computing devices in the algorithm, the greater the percentage of time that the device is involved in computation. This results in a lower percentage of time spent on task queuing and transmission, proving that the task offloading algorithm is more efficient.

$$E = \frac{\sum_{i \in N} y_{i,c} t_i^{comp}}{T_c^{ava}} \tag{20}$$

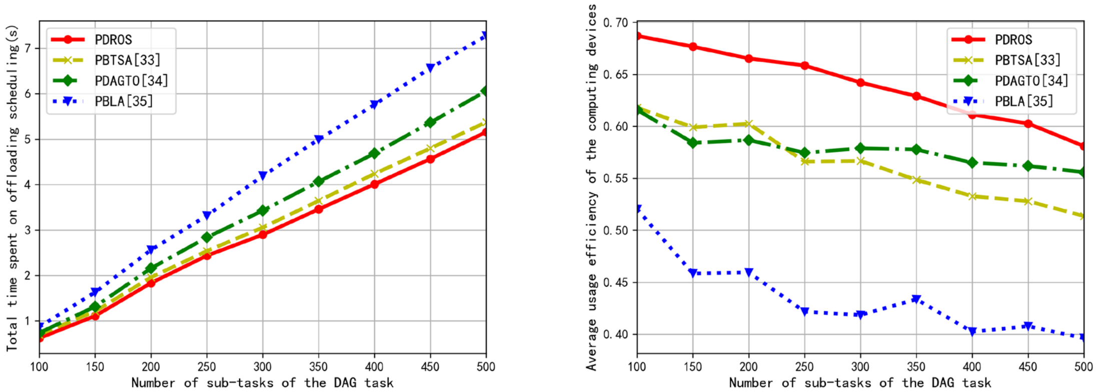
Fig. 4(a) shows the graph of the total time spent on offloading scheduling of a DAG task as the number of sub-tasks varies for each algorithm. It can be seen that the total time spent by the PDROS algorithm is always the least, while the PBLA algorithm spends the longest time. When the number of sub-tasks of a DAG task reaches 500, the time spent by the PDROS algorithm is 5.15s, while the times spent by the PBLA and PBTSA algorithms are 7.27s and 5.37s, respectively. The algorithmic performance of PDROS is improved by 29.16% and 4.10% compared to the PBLA and PBTSA algorithms, respectively. In addition, it can be seen that as the number of sub-tasks grows, the scheduling time spent by each algorithm grows and the value of the difference between the algorithms' time spent increases. This is because as the number of sub-tasks increases, the space for scheduling optimization of the tasks also increases, and therefore the difference between different scheduling strategies also increases.

Fig. 4(b) demonstrates the graph of the average computational device usage efficiency of each algorithm with the number of sub-tasks. As can be seen from the graph, with the increase of the number of sub-tasks, the computational efficiency of the PDROS algorithm is always the highest. When the number of sub-tasks of a DAG task reaches 500, the computational efficiency of the PDROS, PBTSA, PDAGTO, and PBLA algorithms are 58.08%, 51.36%, 55.58%, and 39.62%, respectively. The algorithmic performance of PDROS is improved by 6.72%, 2.50%, and 18.46% in terms of computational device efficiency compared to the other algorithms.

Fig. 5. shows a comparison of the performance of the algorithms as the directed edges density of DAG changes, where the DAG directed edge density denotes the ratio of the actual number of directed edges in the DAG to the maximum possible number of directed edges. The number of sub-tasks is fixed to 100 in Fig. 5, the number of MEC servers is 5 and the number of idle vehicles is 5.

From Fig. 5(a), it is observed that as the DAG directed edge density increases, the total time spent on offloading scheduling also increases. This is because as the density of directed edge increases, the dependencies between tasks become tighter and more complex, resulting in an increase in the queuing and waiting time for sub-tasks. When the DAG directed edge density is 0.2, the PDROS algorithm spends 0.63s, the PBLA algorithm spends 0.84s, and the PBTSA algorithm spends 0.68s. The performance of PDROS is improved by 25.00% and 7.35% compared to the PBLA and PBTSA algorithms, respectively.

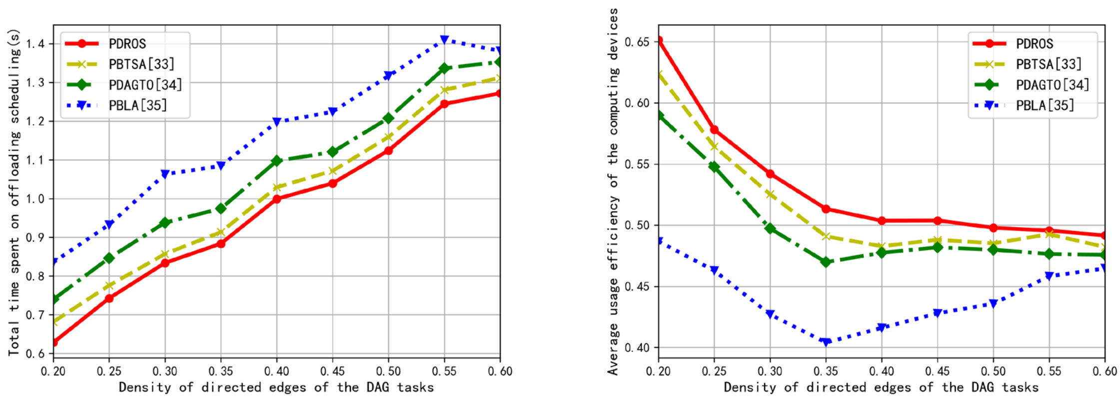
Fig. 5(b) shows the average usage efficiency of the computing devices of each algorithm with the DAG directed edge density. It can be seen that the average usage efficiency of the computing devices of the PDROS algorithm gradually decreases as the density of the DAG directed edges increases and eventually approaches that of the three compared algorithms. This is because the higher the DAG directed edge density, the stronger the dependency between sub-tasks. The larger the offload queuing time and transmission time spent in order to take into account the sub-tasks' dependency accounts for the proportion of the total time spent, and thus the lower the computational efficiency. In addition, as the DAG directed edge density increases, the performance difference between the algorithms decreases. Too large a DAG directed edge density leads to a reduction in the optimization space of the algorithms, and when the DAG directed edge density reaches 1, the DAG task can be transformed into a linear dependency task, at which point all the results of the sub-task sorting algorithms will be the same.



(a) Total offloading scheduling time variation with the number of sub-tasks

(b) Average computational device usage efficiency variation with the number of sub-tasks

Fig. 4. Performance variation of different algorithms with the number of sub-tasks



(a) Total offloading scheduling time variation in the density of DAG directed edges

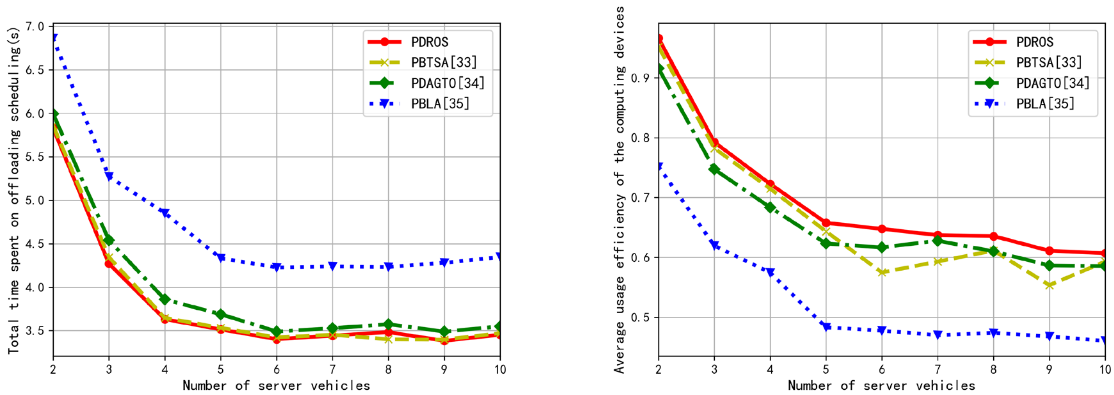
(b) Average computational device usage efficiency variation with DAG directed edge density

Fig. 5. Performance variation of different algorithms with the density of DAG directed edges

Fig. 6. shows a comparison of the algorithms' performance with the number of vehicles, with the number of sub-tasks fixed at 100, the DAG directed edge density set to 0.2, and the number of MEC servers set to 0. From

Fig. 6(a) it can be observed that the PDROS algorithm consistently spends the least amount of time. Also, as the number of vehicles increases, the total task scheduling time initially decreases rapidly and then levels off. This is because when the number of vehicles is small, the number of sub-tasks that can be unloaded exceeds the number of vehicles, resulting in the sub-task queuing time constituting a larger proportion of the total time spent. As the number of vehicles increases, the capacity to offload tasks simultaneously also increases, and the sub-task queuing time decreases. At this point, the total time spent is primarily affected by task dependency. Further increasing the number of vehicles does not significantly impact the task unloading time, causing the curves of each algorithm to flatten out.

Fig. 6(b) shows the trend of the average usage efficiency of the computing devices with the number of vehicles, when the number of vehicles is too small, all the tasks are offloaded to a small number of vehicles and therefore the vehicles are used efficiently. As the number of vehicles increases, the computing devices become more selective and the tasks are spread out across multiple offloading devices. This results in the vehicles needing to wait for the computation results of the front-loaded tasks to be returned from the other vehicles, therefore decreasing the average efficiency of computing device usage.



(a) Total offloading scheduling time variation with the number of vehicles

(b) Average computational device usage efficiency variation with the number of vehicles

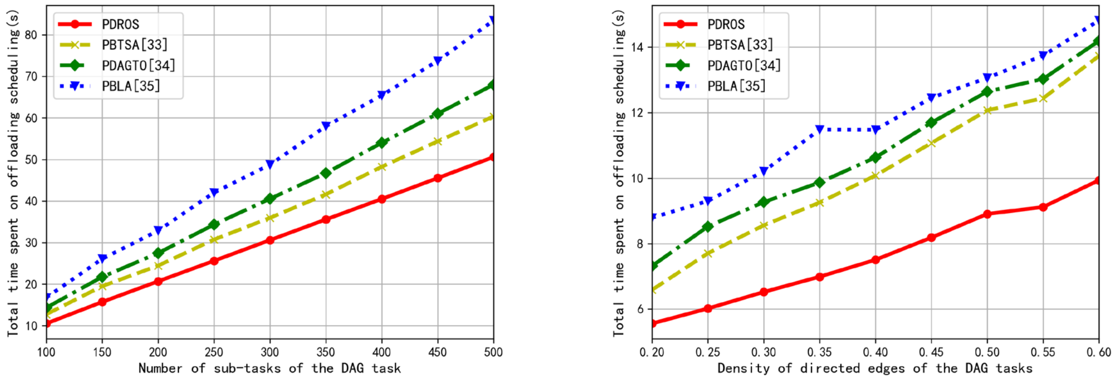
Fig. 6. Performance variation of different algorithms with the number of service vehicles

Fig. 7. shows the difference in the performance of the algorithms when the system state changes. In this scenario, the number of total DAG tasks is 10, the number of sub-tasks is set to 100, the DAG directed edge density is set to 0.2, the number of MEC servers is 5, and the number of server vehicles is 5. To simulate a change in the system state, we assume that the number of MEC servers and vehicles each increase by 1 during offloading.

Fig. 7(a) demonstrates the total time spent on task offload scheduling for each algorithm as the number of subtasks changes when the system state changes. It can be seen that when the number of sub-tasks is increased to 500, the time spent by the PDROS algorithm is 5.05s, and the time spent by the PBTSA, PDAGTO, and PBLA algorithms are 6.04s, 6.80s, and 8.34s, respectively. The performance of the PDROS algorithm is improved by 16.39%, 25.74%, and 39.45% compared to the PBTSA, PDAGTO, and PBLA algorithms, respectively.

Fig. 7(b) demonstrates the total time spent on task offload scheduling for each algorithm with the change of DAG directed edge density when the system state changes. It can be seen that when the DAG directed edge density reaches 0.6, the time spent by the PDROS algorithm is 0.99s, while the times spent by the PBTSA, PDAGTO, and PBLA algorithms are 1.37s, 1.41s, and 1.48s, respectively. The performance of the PDROS algorithm is improved by 27.73%, 29.79%, and 33.10% compared to the PBTSA, PDAGTO, and PBLA algorithms, respectively. It can be observed that the performance gap between the PDROS algorithm and the comparison algorithms is greater compared to the case where there is no change in the system state.

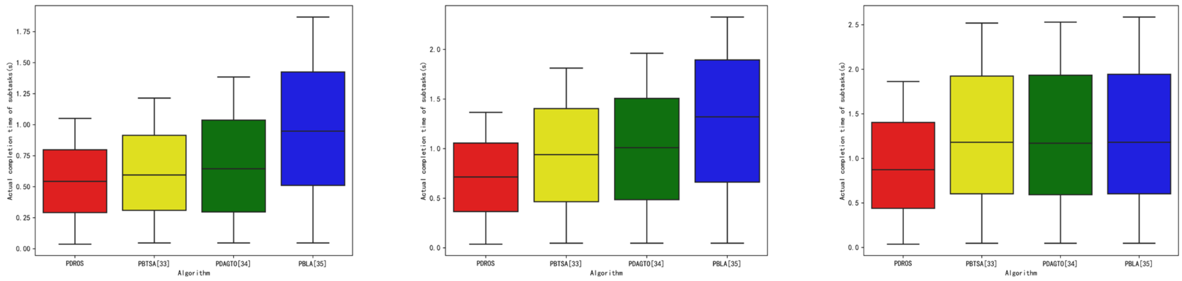
Fig. 8 presents the box plots of the completion times for each subtask under different densities of directed edges in the DAG, with the vertical axis representing the actual completion time for each subtask and the horizontal axis representing the data for the four algorithms. The scenario is set with 5 MEC servers, 5 service vehicles, and 100 sub-tasks. From the fig, it can be observed that the completion times of sub-tasks using the PDROS algorithm are shorter than those of the comparison algorithms. As the DAG directed edge density increases, the completion times for all algorithms increase. When the DAG density reaches 0.6, the optimization space for task scheduling becomes limited, and the differences between the algorithms correspondingly decrease. However, even at this point, the PDROS algorithm still significantly outperforms the comparison algorithms.



(a) Total offloading scheduling time variation with the number of sub-tasks during system state changes

(b) Total offloading scheduling time variation with DAG directed edge density during system state changes

Fig. 7. Performance variation of algorithms with the task parameters when system state changes



(a) DAG directed edge density is 0.2 (b) DAG directed edge density is 0.4 (c) DAG directed edge density is 0.6

Fig. 8. Sub-tasks offloading scheduling time variation with DAG directed edge density during system state changes

6 Conclusion and Future Work

This paper proposes a priority-based dynamic rescheduling offloading strategy to minimize the scheduling delay of related tasks in vehicular networks. First, tasks are represented in the form of a DAG and an adjacency matrix, and a MINLP model for the task offloading problem is established. Subsequently, we decompose the problem into two sub-problems. To solve these sub-problems, we design the CPPD and PCDA algorithms, respectively. Finally, through simulation experiments, we compare the proposed algorithms with existing algorithms. The results show that our proposed algorithms effectively reduce offloading delays and outperform existing algorithms in terms of computational resource utilization and algorithm execution time.

However, there are some limitations to the proposed algorithms. Firstly, the computational complexity of the algorithms is relatively high when dealing with large-scale tasks, which may lead to prolonged execution times. Secondly, the algorithms mainly focus on scheduling delays in offloading decisions and do not fully consider other factors such as energy consumption and load balancing. Additionally, the algorithms' performance in practical applications may be affected by changes in channel conditions and vehicle mobility, which could reduce their effectiveness.

In the future, we will optimize offloading decisions by considering various server resources, including memory, storage space, CPU, and bandwidth, making the algorithms more applicable to real-world scenarios. Furthermore, we plan to incorporate factors such as energy consumption and load balancing to further enhance the algorithms' performance. To address the issue of computational complexity, we will explore more efficient computational methods to improve the practical value of the algorithms.

7 Acknowledgements

This work was supported in part by National Natural Science Foundation of China (Grant No. 62261050), Key R. & D. Foundation of Gansu Province (Grant No. 23YFGA0079), 'FeiTian Scholar' Supporting Program of Gansu Province (Grant No. 2024FTQN-020), Youth Doctoral Support Program of Gansu Universities (Grant No. 2024QB-014), Young Teachers' Scientific Research Ability Promotion Foundation of Northwest Normal University (Grant No. NWNU-LKQN2021-04) and Graduate Research Funding Project of Northwest Normal University in 2023 (Grant No. 2023-126).

References

[1] W. Shi, Q. Li, Q. Yu, F. Wang, G. Shen, Y. Jiang, Y. Xu, L. Ma, G.-M. Muntean, A Survey on Intelligent Solutions for Increased Video Delivery Quality in Cloud–Edge–End Networks, IEEE Communications Surveys & Tutorials 27(2) (2025) 1363–1394.
https://doi.org/10.1109/COMST.2024.3427360

[2] Y. Zhang, T. Gao, C. Li, C.W. Tan, Coded Federated Learning for Communication-Efficient Edge Computing: A Survey, IEEE Open Journal of the Communications Society 5(2024) 4098–4124.
https://doi.org/10.1109/OJCOMS.2024.3423362

[3] Q.-V. Pham, F. Fang, V. N. Ha, M.J. Piran, M. Le, L.B. Le, W.-J. Hwang, Z. Ding, A Survey of Multi-Access Edge Computing in 5G and Beyond: Fundamentals, Technology Integration, and State-of-the-Art, IEEE Access 8(2020) 116974-117017.
https://doi.org/10.1109/ACCESS.2020.3001277

[4] S. Abdelhamid, H.S. Hassanein, G. Takahara, Vehicle as a resource (VaaR), IEEE Network 29(1)(2015) 12–17.
https://doi.org/10.1109/MNET.2015.7018198

[5] S. Zhou, Y. Sun, Z. Jiang, Z. Niu, Exploiting Moving Intelligence: Delay-Optimized Computation Offloading in Vehicular Fog Networks, IEEE Communications Magazine 57(5)(2019) 49–55.
https://doi.org/10.1109/MCOM.2019.1800230

[6] S. Andreev, V. Petrov, K. Huang, M. A. Lema, M. Dohler, Dense Moving Fog for Intelligent IoT: Key Challenges and Opportunities, IEEE Communications Magazine 57(5)(2019) 34–41.
https://doi.org/10.1109/MCOM.2019.1800226

[7] M. Verucchi, A comprehensive analysis of DAG tasks: solutions for modern real-time embedded systems, [doctoral dissertation], Modena (Italy): University of Modena and Reggio Emilia, 2020.

[8] M. Verucchi, I.S. Olmedo, M. Bertogna, A survey on real-time DAG scheduling, revisiting the Global-Partitioned Infinity War, Real-Time Systems 59(3)(2023) 479–530.
https://doi.org/10.1007/s11241-023-09403-3

[9] S. Baruah, Scheduling DAGs When Processor Assignments Are Specified, in: Proc. 28th International Conference on Real-Time Networks and Systems, 2020.
https://doi.org/10.1145/3394810.3394813

[10] S. Chang, J. Sun, Z. Hao, Q. Deng, N. Guan, Computing exact WCRT for typed DAG tasks on heterogeneous multi-core processors, Journal of Systems Architecture 124(2022) 102385.
https://doi.org/10.1016/j.sysarc.2021.102385

[11] X. Cheng, C.-X. Wang, B. Ai, H. Aggoune, Envelope Level Crossing Rate and Average Fade Duration of Nonisotropic Vehicle-to-Vehicle Ricean Fading Channels, IEEE Transactions on Intelligent Transportation Systems 15(1)(2014) 62–72.
https://doi.org/10.1109/TITS.2013.2274618

[12] Q. Luo, C. Li, T.H. Luan, W. Shi, W. Wu, Self-Learning Based Computation Offloading for Internet of Vehicles: Model and Algorithm, IEEE Transactions on Wireless Communications 20(9)(2021) 5913–5925.
https://doi.org/10.1109/TWC.2021.3071248

[13] L. Zhao, Y. Han, A. Hawbani, S. Wan, Z. Guo, M. Guizani, MEDIA: An Incremental DNN Based Computation Offloading for Collaborative Cloud-Edge Computing, IEEE Transactions on Network Science and Engineering 11(2) (2024) 1986–1998.
https://doi.org/10.1109/TNSE.2023.3335345

[14] Z. Chen, W. Yi, A.S. Alam, A. Nallanathan, Dynamic Task Software Caching-Assisted Computation Offloading for Multi-Access Edge Computing, IEEE Transactions on Communications 70(10)(2022) 6950–6965.
https://doi.org/10.1109/TCOMM.2022.3200109

[15] G. Tian, Y. Ren, C. Pan, Z. Zhou, X. Wang, Asynchronous Federated Learning Empowered Computation Offloading in Collaborative Vehicular Networks, in: Proc. 2022 IEEE Wireless Communications and Networking Conference, 2022.
https://doi.org/10.1109/WCNC51071.2022.9771736

[16] K. Zheng, H. Meng, P. Chatzimisios, L. Lei, X. Shen, An SMDP-Based Resource Allocation in Vehicular Cloud Computing Systems, IEEE Transactions on Industrial Electronics 62(12)(2015) 7920–7928.
https://doi.org/10.1109/TIE.2015.2482119
[17] L. Huang, S. Bi, Y.-J.A. Zhang, Deep Reinforcement Learning for Online Computation Offloading in Wireless Powered Mobile-Edge Computing Networks, IEEE Transactions on Mobile Computing 19(11)(2020) 2581–2593.
https://doi.org/10.1109/TMC.2019.2928811
[18] B. Yang, X. Cao, J. Bassey, X. Li, L. Qian, Computation Offloading in Multi-Access Edge Computing: A Multi-Task Learning Approach, IEEE Transactions on Mobile Computing 20(9)(2021) 2745–2762.
https://doi.org/10.1109/TMC.2020.2990630
[19] Y. Sun, S. Zhou, Z. Niu, Distributed Task Replication for Vehicular Edge Computing: Performance Analysis and Learning-Based Algorithm, IEEE Transactions on Wireless Communications 20(2)(2021) 1138–1151.
https://doi.org/10.1109/TWC.2020.3030889
[20] C.-T. Mo, J.-H. Chen, W. Liao, Graph Convolutional Network Augmented Deep Reinforcement Learning for Dependent Task Offloading in Mobile Edge Computing, in: Proc. 2023 IEEE Wireless Communications and Networking Conference, 2023.
https://doi.org/10.1109/WCNC55385.2023.10119034
[21] C. Hou, Q. Zhao, Optimal Task-Offloading Control for Edge Computing System With Tasks Offloaded and Computed in Sequence, IEEE Transactions on Automation Science and Engineering 20(2)(2023) 1378–1392.
https://doi.org/10.1109/TASE.2022.3176745
[22] Z. Nan, S. Zhou, Y. Jia, Z. Niu, Joint Task Offloading and Resource Allocation for Vehicular Edge Computing With Result Feedback Delay, IEEE Transactions on Wireless Communications 22(10)(2023) 6547–6561.
https://doi.org/10.1109/TWC.2023.3244391
[23] X. Lv, H. Du, Q. Ye, TBTOA: A DAG-Based Task Offloading Scheme for Mobile Edge Computing, in: Proc. ICC 2022 - IEEE International Conference on Communications, 2022.
https://doi.org/10.1109/ICC45855.2022.9838987
[24] J. Wang, J. Hu, G. Min, A.Y. Zomaya, N. Georgalas, Fast Adaptive Task Offloading in Edge Computing Based on Meta Reinforcement Learning, IEEE Transactions on Parallel and Distributed Systems 32(1)(2021) 242–253.
https://doi.org/10.1109/TPDS.2020.3014896
[25] R. Chai, M. Li, T. Yang, Q. Chen, Dynamic Priority-Based Computation Scheduling and Offloading for Interdependent Tasks: Leveraging Parallel Transmission and Execution, IEEE Transactions on Vehicular Technology 70(10)(2021) 10970–10985.
https://doi.org/10.1109/TVT.2021.3110401
[26] S. Thananjeyan, C.A. Chan, E. Wong, A. Nirmalathas, Application Based Energy Optimization for Computation Offloading in Hierarchical MEC Network, in: Proc. 2021 10th International Conference on Information and Automation for Sustainability, 2021.
https://doi.org/10.1109/ICIAfS52090.2021.9605862
[27] C. Li, Q. Cai, Y. Luo, Low-latency edge cooperation caching based on base station cooperation in SDN based MEC, Expert Systems with Applications 191(2022) 116252.
https://doi.org/10.1016/j.eswa.2021.116252
[28] G. Li, M. Zeng, D. Mishra, L. Hao, Z. Ma, O.A. Dobre, Latency Minimization for IRS-Aided NOMA MEC Systems With WPT-Enabled IoT Devices, IEEE Internet of Things Journal 10(14)(2023) 12156–12168.
https://doi.org/10.1109/JIOT.2023.3240395
[29] F. Song, H. Xing, S. Luo, D. Zhan, P. Dai, R. Qu, A Multiobjective Computation Offloading Algorithm for Mobile-Edge Computing, IEEE Internet of Things Journal 7(9)(2020) 8780–8799.
https://doi.org/10.1109/JIOT.2020.2996762
[30] T. Deng, Y. Chen, G. Chen, M. Yang, L. Du, Task offloading based on edge collaboration in MEC-enabled IoV networks, Journal of Communications and Networks 25(2)(2023) 197–207.
https://doi.org/10.23919/JCN.2023.000004
[31] M.R. Garey, D.S. Johnson, "Strong" NP-Completeness Results: Motivation, Examples, and Implications, Journal of the ACM 25(3)(1978) 499–508.
https://doi.org/10.1145/322077.322090
[32] J.C. Bezdek, R.J. Hathaway, R.E. Howard, C.A. Wilson, M.P. Windham, Local convergence analysis of a grouped variable version of coordinate descent, Journal of Optimization Theory and Applications 54(3)(1987) 471–477.
https://doi.org/10.1007/BF00940196
[33] Z. Wang, G. Sun, H. Su, H. Yu, B. Lei, M. Guizani, Low-Latency Scheduling Approach for Dependent Tasks in MEC-Enabled 5G Vehicular Networks, IEEE Internet of Things Journal 11(4)(2024) 6278–6289.
https://doi.org/10.1109/JIOT.2023.3309940
[34] W. Zheng, C. Wang, Z. Chen, D. Zhang, A Priority-Based Level Heuristic Approach for Scheduling DAG Applications with Uncertainties, in: Proc. 2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design, 2022.
https://doi.org/10.1109/CSCWD54268.2022.9776105

[35] X. Fu, B. Tang, F. Guo, L. Kang, Priority and Dependency-Based DAG Tasks Offloading in Fog/Edge Collaborative Environment, in: Proc. 2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design, 2021.
https://doi.org/10.1109/CSCWD49262.2021.9437784