Low-Latency Scheduling Approach for Dependent Tasks in MEC-Enabled 5G Vehicular Networks

Zhiying Wang, Gang Sun, *Senior Member, IEEE*, Hanyue Su, Hongfang Yu, *Member, IEEE*, Bo Lei, and Mohsen Guizani, *Fellow, IEEE*

***Abstract*—With the development of the Internet of Vehicles (IoV), multiaccess edge computing (MEC) technology places computing resources closer to users at edge nodes, enabling faster, more reliable, and secure computing services. In the MEC-enabled IoV networks, task offloading scheduling, as an effective method to alleviate the computational burden on vehicles, is gaining increasing attention. However, with the intelligent and networked development of vehicles, the complex data dependency between in-vehicle tasks brings challenges to offloading scheduling. In contrast to many existing methods that solely address individual tasks, there is a growing need to tackle interrelated tasks within the IoV framework. This includes tasks like processing vehicle sensor data, gathering and analyzing road condition information, facilitating collaborative decision making among vehicles, and optimizing traffic signal systems. Our objective is to address the broader challenge of offloading dependent tasks, as this closely aligns with real-world scenes and requirements. In this article, we propose a priority-based task scheduling algorithm (PBTSA) to minimize processing delay when the tasks are interdependent. PBTSA proposes a method that can better measure the data transmission and calculation delay of the IoV networks. We first model dependent tasks as a directed acyclic graph (DAG) and then use the reverse breadth-first search (RBFS) algorithm to generate the priority of each subtask, and finally according to the priority with low complexity to offload subtasks greedily to minimize task processing delay. We compare the PBTSA with the other two existing algorithms through simulations. The results show that the PBTSA can effectively reduce the task processing delay and can reach close to 10%.**

***Index Terms*—5G, dependent tasks, Internet of Vehicles (IoV), multiaccess edge computing (MEC), resource allocation.**

I. Introduction

With the advancement of Internet of Vehicles (IoV) [1], the number of intelligent vehicles has skyrocketed, leading to the emergence of various in-vehicle applications. However, due to the limited computing and storage resources available in vehicles, there is a lack of capability to effectively run these applications. As a result, the performance of these applications suffers, and the quality of service is compromised [2]. To address this issue, the integration of multiaccess edge computing (MEC) into vehicular networks has been recognized as an effective solution. MEC involves deploying servers on network edge devices like roadside units (RSUs) and base stations (BSs), allowing vehicle users to access computing services with low latency and high reliability.

In the early stages of the IoV, in-vehicle applications were relatively small and monolithic, with the primary focus of computing offloading being the entire application. Researchers primarily investigated whether the application should be offloaded and where it should be offloaded. However, previous algorithms have shown limitations in making effective offloading decisions when there are dependencies among subtasks. With the advancement of vehicle intelligence, more computationally complex applications have emerged, such as path planning, autonomous driving, and augmented reality [3]. These applications often consist of multiple interdependent tasks that work together to meet intricate functional requirements. To improve task execution parallelism and reduce application completion latency, each task within the application can be offloaded to a different edge server for execution. However, the execution order of tasks is constrained by data dependencies, and transferring data between servers can introduce additional communication overhead. As a result, effectively managing intertask dependencies has become a key focus in offloading scheduling research.

The scheduling problem for applications with dependent tasks can be represented as a directed acyclic graph (DAG), where the tasks are depicted as vertices and the task dependencies are represented as edges. In this model, each task is associated with a specific number of CPU cycles required for its execution and obtains the result upon completion. The problem arises when mapping the dependent tasks to available resources in edge–cloud scenes. As the number of tasks and resources increases, determining the optimal schedule from numerous task-resource mapping combinations becomes challenging. Consequently, efficiently making offloading decisions that consider task dependencies in mobile-edge computing systems becomes an NP-hard problem [4]. The primary challenge is to minimize the overall application completion time while ensuring task dependencies are met in resource-constrained mobile-edge computing systems.

Manuscript received 25 June 2023; revised 28 July 2023; accepted 27 August 2023. Date of publication 30 August 2023; date of current version 6 February 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2019YFB1802800. *(Corresponding author: Gang Sun.)*

Zhiying Wang, Gang Sun, Hanyue Su, and Hongfang Yu are with the Key Laboratory of Optical Fiber Sensing and Communications, Ministry of Education, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: wzy1211893610@163.com; gangsun@uestc.edu.cn; 442205331@qq.com; yuhf@uestc.edu.cn).

Bo Lei is with the Network Research Institute, China Telecom Corporation Limited Research Institute, Beijing 100045, China (e-mail: leibo@chinatelecom.cn).

Mohsen Guizani is with the Machine Learning Department, Mohamed Bin Zayed University of Artificial Intelligence, Abu Dhabi, UAE (e-mail: mguizani@ieee.org).

Digital Object Identifier 10.1109/JIOT.2023.3309940

2327-4662 © 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on March 12,2024 at 07:10:23 UTC from IEEE Xplore. Restrictions apply.

There have been many studies on offloading-dependent tasks. Sundar and Liang [5] proposed a heuristic algorithm termed individual time allocation with greedy scheduling (ITAGS) to achieve an efficient solution, aiming at minimizing the execution cost of applications under the application completion time constraint. Zhao et al. [6] modeled the ODT problem with service caching in a homogeneous MEC scene. And they presented a favorite successor-based heuristic algorithm to efficiently offload dependent tasks to edge nodes with limited (and predetermined) service caching and minimize the application completion time. Although there have been many studies on dependent task scheduling, previous researches have never investigated the scheduling strategy for dependent tasks in the IoV context.

The existing algorithms often have high computational complexity, leading to significant task processing delays and execution time for the decision algorithm when the number of tasks increases. Therefore, there is a need for a low-complexity algorithm to handle dependent task scheduling in IoV networks. Building upon previous research, this article proposes priority-based task scheduling algorithm (PBTSA) to solve the dependent subtask scheduling problem in the IoV. Compared with existing methods, PBTSA has performance in measuring data transmission and computing delays in the IoV. In addition, the algorithm also optimizes the calculation formula of subtask priority, which improves the accuracy and effectiveness of the assignment.

The main contributions of this article are as follows.

1) We propose a new approach to solve the resource scheduling problem for dependent tasks in the IoV. This method effectively captures the transmission delays that occur during data exchange within the IoV network. By accurately assessing these latencies, we can better evaluate the overall performance of the system.
2) We enhance the calculation formula used for subtask prioritization. By optimizing this formula, we improve the accuracy and effectiveness of assigning priorities to individual subtasks within the IoV system. Additionally, we propose the reverse breadth-first search (RBFS) algorithm to generate prioritization orders for each subtask. This approach ensures that subtasks are sequenced appropriately, taking into consideration their interdependencies and system requirements.
3) We propose a low-complexity greedy approach PBTSA to offload subtasks based on their priorities. Leveraging the prioritization generated by the RBFS algorithm, we design an efficient offloading strategy that minimizes both task processing latency and the execution time of the decision algorithm. This approach allows for optimized resource allocation and effective utilization of available computing resources within the IoV system.

The remainder of this article is organized as follows. Section II reviews the related work and shows the motivation of our research. Section III presents the system model, which consists of the system model, physical network model, and the task model and establishes the optimization objectives. Section IV introduces the proposed algorithm based on priority greedy offloading to solve the studied problem. In Section V, we conduct extensive simulations and analyze the results. Finally, Section VI concludes this article.

II. Related Work

With the advancement of the IoV, vehicles increasingly require computing resources and reduced latency. In this context, MEC emerges as a promising solution to tackle computation offloading challenges in mobile applications that are highly sensitive to latency and computationally intensive [7], [8], [9], [10], [11], [12], [13]. Zhang et al. [7] proposed an iterative algorithm to obtain the optimal solution and a two-stage heuristic algorithm to obtain an approximate optimal solution by offloading tasks to edge servers or vehicles and allocating wireless resources of base stations and computing resources of servers to minimize task processing delays for all devices. Similarly, Thananjeyan et al. [8] formulated a problem of allocating edge storage and computing resources using Lyapunov optimization theory, matching theory, and other optimization methods to jointly optimize service caching, service request scheduling, and resource allocation, thus minimizing the average response delay of services. MEC's effectiveness is limited in areas with poor server coverage, and there are many idle computing resources in peripheral vehicles. The author proposes a distributed multihop task offloading decision model that includes a candidate vehicle selection mechanism and a task offloading decision algorithm to improve task execution efficiency [12]. The authors propose a strategy to tackle energy consumption and computation costs in vehicular networks by employing traffic offloading and scheduling. Their proposed approach utilizes quantum particle swarm optimization (PSO) to optimize the joint offloading strategy in mobile-edge computing-enabled vehicular networks, thereby enhancing the Quality of Experience (QoE) [13].

In recent years, there has been a rapid development of the IoV, leading to the emergence of complex applications that were not seen before. These applications often consist of interdependent subtasks, and efficient resource scheduling and task offloading in the presence of task dependencies has become a hot research topic. Several researchers have made significant contributions to optimizing various aspects of application execution in the IoV [14], [15], [16], [17], [18], [19], [20], [21]. One notable contribution in this field is the joint-dependent task offloading and flow scheduling heuristic (JDOFH) proposed by Sahni et al. [14]. This heuristic takes into account task dependencies represented by a DAG and the start time of network flows. The goal is to optimize the overall execution time of the application. Another related work by Maray et al. [15] presents a heuristic algorithm that aims to reduce the completion time of dependent tasks while ensuring that application deadline constraints are satisfied. Their algorithm focuses on minimizing the time required for executing interdependent tasks. Additionally, Mahmoodi et al. [16] proposed a wireless-aware scheduling and computation offloading algorithm that parallelizes appropriate tasks to shorten the overall execution time of the application. This approach considers the wireless

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on March 12,2024 at 07:10:23 UTC from IEEE Xplore. Restrictions apply.

communication aspects and takes advantage of parallel processing to optimize the performance. In order to address both application completion time and energy consumption, Peng et al. [19] introduced a task scheduling method based on the whale optimization algorithm (WOA). The objective of this method is to minimize both completion time and energy consumption by effectively allocating tasks to available resources. Moreover, Huang et al. [20] took into account security, energy consumption, and application completion time. They utilized a genetic algorithm (GA) to minimize the energy consumption of mobile devices while satisfying the application deadline constraints. Similarly, Xie et al. [21] proposed a directional and nonlocal-convergent PSO algorithm to optimize the completion time and cost of applications. Their algorithm aims to find an optimal solution considering the completion time and associated cost.

Applications in the IoV often require fast responses, and there are numerous studies currently being conducted with the objective of minimizing the total execution time (TET) [22], [23], [24], [25], [26], [27], [28], [29], [30], [31]. To minimize the completion time of the entire application, Mo et al. [22] utilized graph convolutional networks (GCNs) to capture task dependencies and transform the problem of offloading dependent tasks into a node classification problem, effectively addressing non-Euclidean data. Song et al. [23] investigated the multiobjective computation offloading problem, aiming to minimize the application completion time and energy consumption of mobile devices (MDs). They proposed a multiobjective evolutionary algorithm based on decomposition (MOEA/D) to tackle this problem. In another study [24], two algorithms have been proposed: one based on a GA and the other based on conflict graph models. Zhang and Wen [25] formulated application execution as a delay-constrained workflow scheduling problem and proposed a one-climb policy and Lagrange relaxation algorithm to minimize MDs' energy consumption, considering execution restrictions. To address the tradeoff between latency and energy consumption, Deng et al. [31] presented a formulation that treats task offloading and splitting as an optimization problem. The objective is to minimize the overall cost, which encompasses both latency and energy consumption. This is achieved by simultaneously optimizing the task splitting ratio and the uplink transmit power of the vehicle terminal (VT). Although there are numerous studies on reducing network latency for MEC scenes, there is currently no method specifically tailored to the task-dependent reduction of task completion time in the context of MEC-enabled IoV. Given the time-sensitive nature IoV, finding a solution to this problem has become urgent. Therefore, we propose a more realistic approach considering both data transmission and computation delays in vehicular networks for effectively reducing the latency of task processing.

III. System Model and Problem Formulation

A. System Model

A three-tier network model (as shown in Fig. 1) is built for the task offloading scheduling problem in IoV.

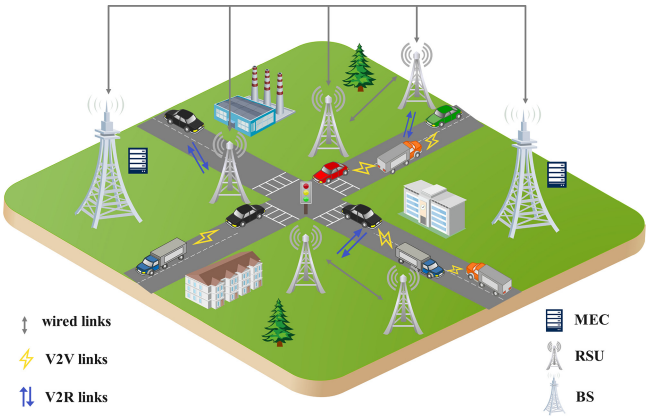

Fig. 1. MEC-enabled 5G IoV model.

1) The bottom layer of the model consists of vehicle nodes. For any two vehicles, they can communicate directly with a vehicle-to-vehicle (V2V) link if the distance between them does not exceed the wireless signal coverage.
2) The middle layer of the model consists of RSUs. RSU has two communication connections: a) wired connection to other RSUs and the BSs to realize the connection between MEC and vehicles and b) Vehicle-to-RSU (V2R) wireless link used to communicate with vehicles within the wireless signal coverage.
3) The upper layer of the model consists of BSs and a MEC server is deployed on each BS.

When a vehicle generates a computational task, it can choose to offload it locally, using its own computing resources for execution. The vehicle can also offload the task using V2V communication, transferring the computational task to other vehicles and utilizing their computing resources for execution. Alternatively, the vehicle can offload the task to the MEC on the BSs through RSUs and utilizing the MEC's computing resources for execution.

B. Physical Network Model

The abstraction of MEC-enabled 5G IoV can be represented as $W = (V, M, L)$. Here, $V = 1, 2, \ldots, v, \ldots, |V|$ represents the set of vehicles with their corresponding low-frequency computing resources f_v. The MEC set is denoted as $M = 1, 2, \ldots, m, \ldots, |M|$. The MEC is generally integrated with communication base stations, offering high-frequency computing resources f_m. The union of V and M is represented as $C = V \cup M = 1, 2, \ldots, |v|, |v| + 1, \ldots, |V| + |M|$, encompassing all computational resource devices. The physical environment may have tasks currently being executed, so t^c_{remain} is used to represent the cumulative execution time of all tasks on device c, indicating the time it takes for device c to transition from an active task state to an idle state after waiting for t^c_{remain}. L represents the set of network links, including V2V communication links L_V, cellular network communication links between vehicles and MEC L_{VM}, and fiber-optic links between MECs L_M. Therefore, $L = L_V \cup L_M \cup L_{VM}$. Additionally, the definitions of the variables involved in this section are summarized in Table I.

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on March 12,2024 at 07:10:23 UTC from IEEE Xplore. Restrictions apply.

TABLE I
KEY NOTATIONS USED IN THIS ARTICLE

Notations	Descriptions
W	MEC-enabled 5G IoV networks
M	Set of MECs
V	Set of vehicles
C	Set of all computational devices
f_c	Computational frequency of device c
t^c_{remain}	Duration of executing all local tasks on device c
L	Set of communication links
L_V	Set of V2V communication links
L_M	Set of fiber optic communication links between MECs
L_{VM}	Cellular links between MECs and vehicles
K	Subchannel set
$R_{cc'}$	Communication rate from device c to device c'
G	DAG task
N	Set of subtasks in a DAG task
E	Set of all data dependencies in a DAG task
ω_i	Computational resource requirement of subtask n_i
$d_{i,j}$	Data transmitted from subtask n_i to n_j after execution
$succ(n_i)$	Set of immediate successor tasks of subtask n_i
n_{src}	Source subtask
n_{sink}	Sink subtask
$t^{comp}_{i,c}$	Computation time of subtask n_i on device c
$t^{trans(c_1,c_2)}_{i,j}$	Transmission delay of data from device c_1 to device c_2
Y	Subtask offloading decision matrix
$y_{i,c}$	Decision variable for subtask offloading to device c
o_i	Execution time of subtask n_i
$g_{i,j}$	Average transmission delay from device c_i to device c_j
$EST(n_i,c)$	Earliest start time for subtask n_i executed on device c
$EFT(n_i,c)$	Earliest finish time for subtask n_i on device c
$T_{avail}(c)$	Earliest time when c has computing resources for n_i
$AST(n_i,c)$	Actual start time for subtask n_i on device c
$AAT(n_i,c)$	Actual arrival time of subtask n_i offloaded to device c
$AFT(n_i,c)$	Actual finish time for subtask n_i on device c
n^c_{arrive}	Pending offloaded subtasks on device c
n^c_{ready}	Set of subtasks in a ready state on device c
$PV(n_i)$	Priority value of subtask n_i
$\overline{o_i}$	average processing delay of subtask n_i
$\overline{g_{i,j}}$	Average transmission time of data dependency (n_i, n_j)

In this article, the wireless communication in the 5G IoV network is modeled based on the nonorthogonal multiple access (NOMA) communication protocol, where the set of subchannels is denoted as $K = \{1, 2, \ldots, k, \ldots, |K|\}$. The communication transmission rate from vehicle v to vehicle v' can be obtained as

$$R_{vv'} = B_{vv'} \log_2 \left(1 + \frac{x_{v,k} P_v g_{vv'}}{\sum_{s \in C} x_{s,k} P_s g_{s,v'} + \sigma^2}\right) \quad (1)$$

where $x_{v,k}$ denotes whether vehicle v occupies subchannel k. If $x_{v,k} = 1$, it indicates that vehicle v utilizes subchannel k for communication transmission. P_v represents the transmission power of vehicle v. $g_{vv'}$ represents the channel gain between vehicle v and vehicle v'. σ^2 denotes the additive white Gaussian noise on the subchannel.

Similarly, the communication transmission rates from vehicle v to MEC m and from MEC m to vehicle v are, respectively

$$R_{vm} = B_{vm} \log_2 \left(\frac{x_{v,k} P_v g_{vm}}{\sum_{s \in V} x_{s,k} P_s g_{s,m} + \sigma^2}\right) \quad (2)$$

$$R_{mv} = B_{mv} \log_2 \left(\frac{x_{m,k} P_m g_{mv}}{\sum_{s \in C} x_{s,k} P_s g_{s,v} + \sigma^2}\right). \quad (3)$$

MECs communicate with each other using fiber-optic links, and the transmission rate is represented as (4), where $B_{mm'}$ is the bandwidth of channel between MECs, $D_{mm'}$ is the distance between MECs, p_r is the power due to receiving task, and N_o is the noise power

$$R_{mm'} = B_{mm'} \log_2 \left(1 + \frac{p_r}{D_{mm'} N_o}\right). \quad (4)$$

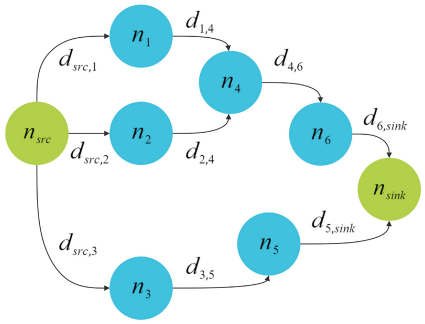

Fig. 2. Example of DAG task.

C. Task Model

When dependent tasks are generated, subtasks with preconditions must wait for the completion of the preceding tasks and obtain their execution results before they can start executing. Additionally, some of the computing resources in the physical network may be occupied by other tasks. In cases where computing resources are limited, it is necessary to consider how to schedule dependent subtasks. Without loss of generality, we assume that vehicle v has a computational task to be completed, which can be represented by a DAG denoted as $G = (N, E)$, as shown in Fig. 2. Here, N represents the set of subtasks, and each subtask in the DAG task is atomic and noninterruptible during computation. E represents the set of directed edges that describe the dependencies between subtasks. The total number of CPU clock cycles required to complete subtask n_i is denoted by the corresponding node weight ω_i, which can vary based on different computing requirements, data sizes, algorithmic complexities, or resource availability for each subtask.

Each edge (n_i, n_j) represents an execution dependency or data dependency between subtasks n_i and n_j, where subtask n_j can only be executed after subtask n_i is completed. Subtask n_i is a direct predecessor of subtask n_j, and subtask n_j is a direct successor of subtask n_i. The amount of data that needs to be transferred from subtask n_i to subtask n_j after the execution of n_i is represented by the edge weight (n_i, n_j), denoted as $d_{i,j}$.

The sets $\text{pred}(n_i) = n_j \mid (n_j, n_i) \in E$ and $\text{succ}(n_i) = n_j \mid (n_i, n_j) \in E$ represent the immediate predecessor and successor subtask sets of subtask n_i, respectively. For example, in Fig. 2, $\text{pred}(n_4) = n_1, n_2$ and $\text{succ}(n_{\text{src}}) = n_1, n_2, n_3$. If $\text{pred}(n_i) = \emptyset$, then n_i is the source subtask n_{src}; if $\text{succ}(n_i) = \emptyset$, then n_i is the sink subtask n_{sink}.

Therefore, the computation time for subtask n_i executed on device c is

$$t^{\text{comp}}_{i,c} = \frac{\omega_i}{f_c} \quad \forall i \in N, c \in C. \quad (5)$$

Assuming that subtask n_i is executed at c_1 and subtask n_j is executed at c_2, the transmission delay for c_1 to transmit the

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on March 12,2024 at 07:10:23 UTC from IEEE Xplore. Restrictions apply.

execution result of subtask n_i to c_2 for the execution of n_j is

$$t_{i,j}^{\text{trans}(c_1,c_2)} = \begin{cases} \frac{d_{i,j}}{R_v}, & c_1, c_2 \in V, c_1 \neq c_2, (n_i, n_j) \in E \\ \frac{d_{i,j}}{R_m}, & c_1, c_2 \in M, c_1 \neq c_2, (n_i, n_j) \in E \\ \frac{d_{i,j}}{R_{vm}}, & c_1 \in V, c_2 \in M, (n_i, n_j) \in E \\ \frac{d_{i,j}}{R_{mv}}, & c_1 \in M, c_2 \in V, (n_i, n_j) \in E \\ 0, & c_1 = c_2, c_1, c_2 \in C. \end{cases} \tag{6}$$

D. Problem Formulation

Assuming that MEC enables nonpreemptive scheduling of subtasks in 5G IoV networks. A computing device can only execute one subtask, and as long as a subtask is not completed, its execution process cannot be interrupted, and the computing resources used by it cannot be preempted by other subtasks. Each subtask can be executed by a computing device within the physical network range, and the offloading decision of subtasks is represented by a binary matrix $Y = |V| \times |C|$. If subtask n_i is offloaded to computing device c, then $y_{i,c} = 1$, otherwise $y_{i,c} = 0$. Each subtask in the task should be executed and can only be executed once by a computing device, i.e.,

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

$$y_{i,c} \in \{0, 1\} \ \forall i \in N, c \in C. \tag{8}$$

Therefore, after the offloading by decision Y, the execution time of subtask n_i is

$$o_i = \sum_{c \in C} \frac{y_{i,c}\omega_i}{f_c} \quad \forall i \in N, c \in C. \tag{9}$$

The transmission delay of the data dependency $(n_i, n_j) \in E$ between subtasks n_i and n_j is

$$g_{i,j} = \sum_{c \in C} \sum_{c' \in C} y_{i,c} y_{j,c'} \frac{d_{i,j}}{R_{cc'}} \quad \forall \ n_i, n_j \in N, \ (n_i, n_j) \in E, \ c, c' \in C. \tag{10}$$

Assuming that subtask n_i is offloaded to computing device c, i.e., $y_{i,c} = 1$, the earliest start time (EST) and earliest finish time (EFT) for subtask n_i on device c are as follows:

$$\text{EST}(n_i, c) = \max \left\{ T_{\text{avail}}(c), \max_{n_j \in \text{pred}(n_i), \ y_{j,c'}=1} \left\{ \text{AFT}(n_j) + \frac{d_{j,i}}{R_{c'c}} \right\} \right\} \ \forall i \in N, c \in C \tag{11}$$

$$\text{EFT}(n_i, c) = \text{EST}(n_i, c) + \frac{\omega_i}{f_c} \ \forall i \in N, c \in C. \tag{12}$$

Due to the offloading decision of subtasks, the actual finish time (AFT) $\text{AFT}(n_j)$ of a subtask n_j may be earlier than the EFT $\text{EFT}(n_j)$. $T_{\text{avail}}(c)$ represents the earliest time when device c has sufficient computing resources to execute subtask n_i. $\max_{n_j \in \text{pred}(n_i), \ y_{j,c'}=1}\{\text{AFT}(n_j) + ([d_{j,i}]/[R_{c'c}])\}$ represents the sum of the latest AFT among the predecessor tasks of subtask n_i and the time taken to transmit the execution result to device c. The necessary condition for subtask n_i to start execution is that device c has sufficient computing resources and the execution completion results of all its predecessor tasks have arrived at device c.

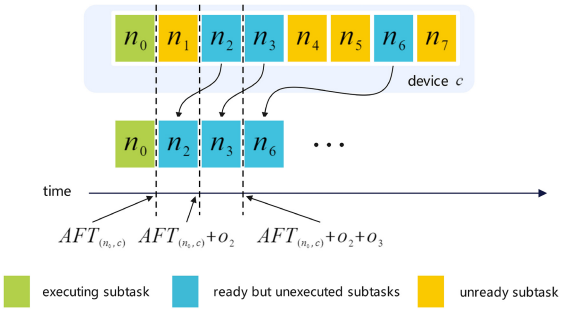

Fig. 3. DAG subtask offloading queue of device c.

Assuming that multiple subtasks are offloaded to device c and the set of subtasks that have not yet started execution is denoted as n_{arrive}^c or $\{n_{\text{arrive}}^c \mid y_{i,c} = 1 \ \forall n_i \in n_{\text{arrive}}^c\}$, the actual start time (AST) of subtask n_i is subject to the following constraints:

$$\begin{aligned} &\text{AST}(n_i, c) \geq \text{AST}(n_j, c) \\ \text{if} \ \ &\text{AAT}(n_i, c) \geq \text{AAT}(n_j, c) \ \text{ and} \\ &\text{EST}(n_i, c) = \text{EST}(n_j, c) = T_{\text{avail}}(c) \\ &\forall \ n_i, n_j \in n_{\text{arrive}}^c, \ \ c \in C. \end{aligned} \tag{13}$$

The actual arrive time (AAT) represents the actual time when a subtask arrives at the device after offloading. Constraint (13) ensures the first-in–first-out (FIFO) characteristic of the task queue in the device. When the EST of two subtasks n_i and n_j in the device is equal, i.e., $\text{EST}(n_i, c) = \text{EST}(n_j, c)$, the subtask that arrives first at device c is given priority for execution.

As shown in Fig. 3, subtasks n_2 and n_3 are offloaded and executed on device c, with subtask n_2 arriving earlier, i.e., $n_2, n_3 \in n_{\text{arrive}}^c$ and $\text{AAT}(n_2, c) > \text{AAT}(n_3, c)$. The data dependencies of n_2 and n_3 have been transmitted to device c, but their execution has not yet started, i.e., $\text{EST}(n_2, c) = \text{EST}(n_3, c) = T_{\text{avail}}(c) = \text{AFT}(n_0)$. Due to the FIFO characteristic of the device's task queue, subtask n_2 is given priority over n_3, resulting in $\text{AST}(n_2, c) > \text{AST}(n_3, c)$.

Therefore, the AST of the subtask n_i executed on device c is

$$\begin{aligned} &\text{AST}(n_i, c) = \text{AFT}(n_0, c) + \sum_{j=1}^{i-1} o_j \\ \text{if} \ \ &\text{EST}(n_i, c) = \text{EST}(n_j, c) = T_{\text{avail}}(c) \ \text{ and } \ y_{k,c} = 1 \\ &\forall \ 1 \leq j \leq i-1, \ \ n_i, n_j, n_k \in n_{\text{ready}}^c, \ \ c \in C \end{aligned} \tag{14}$$

$$n_{\text{ready}}^c = \left\{ n_{\text{ready}}^c \mid \text{EST}(n_i, c) = T_{\text{avail}}(c), \ y_{i,c} = 1 \ \forall n_i \in n_{\text{ready}}^c \right\} \tag{15}$$

where n_{ready}^c represents the queue of subtasks that are ready for execution on device c, following the FIFO principle. As long as computing resources are available, subtasks can enter the execution state. n_0 represents the subtask that is currently being executed or the previously completed subtask.

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on March 12,2024 at 07:10:23 UTC from IEEE Xplore. Restrictions apply.

Therefore, (14) represents that the AST of subtask n_i on device c, $\mathrm{AST}(n_i, c)$, is the sum of $\mathrm{AFT}(n_0, c)$ and the execution time of all preceding ready subtasks before n_i. As shown in Fig. 3, device c is executing subtask n_0, and the ready subtasks offloaded to device c are in the order of $[n_2, n_3, n_6]$. Therefore, the AST of subtask n_6 is $\mathrm{AST}(n_6, c) = \mathrm{AFT}(n_0) + o_2 + o_3$.

Hence, the AFT of subtask n_i is

$$\mathrm{AFT}(n_i) = \sum_{c \in C} y_{i,c} \ \mathrm{AST}(n_i, c) + o_i \ \forall i \in N, c \in C. \quad (16)$$

The optimization objective of this article is to obtain reasonable task offloading decisions that minimize the task completion time, while satisfying the aforementioned constraints

$$\begin{aligned} \min: \quad & \mathrm{AFT}(n_{\mathrm{sink}}) \quad (17) \\ \text{s.t.} \quad & \text{Constraints (7)–(16).} \end{aligned}$$

This problem can be solved by verifying the solution in polynomial time, but no efficient algorithm has been found to solve it. Therefore, problem (17) is an integer linear programming problem with NP property, which is an NP-complete problem [32]. In the next section, an algorithm will be proposed to solve it.

IV. Algorithm Design

The previous section provided a detailed explanation of the physical network model and DAG task model in the context of 5G IoV networks. From a service perspective, minimizing task latency, as stated in the optimization objective (17), is crucial for user experience. Therefore, we present a PBTSA called PBTSA. The algorithm consists of two phases. The first phase involves generating task scheduling priorities based on the MEC-enabled 5G IoV network computing resource environment $W = (V, M, L)$. The second phase utilizes the priorities generated in the first phase to schedule tasks.

A. Subtask Priorities

In the task scheduling process, the use of subtask priorities is crucial to determine the order in which subtasks are executed and to make informed offloading decisions. By incorporating subtask priorities, the scheduling algorithm can effectively manage resource allocation, reduce task latency, and improve the overall efficiency of task execution in the MEC-enabled 5G IoV networks.

In this section, the subtask priorities are obtained based on the recursive best-first search algorithm, ensuring that the priority of a subtask is always greater than the priority of its immediate successor tasks.

The priority value (PV) for each subtask in the MEC-enabled 5G IoV network environment is recursively obtained from the subtask n_{sink} using the following equation:

$$PV(n_i) = \max_{n_j \in \mathrm{succ}(n_i)} \left\{ PV(n_j) + \overline{o_i} + \overline{g_{i,j}} \right\} \ \forall \ i \in N \quad (18)$$

$$\overline{o_i} = \frac{1}{|C|} \sum_{c \in C} \frac{\omega_i}{f_c} \quad \forall \ i \in N \quad (19)$$

Algorithm 1 Priority Generation Algorithm Based on RBFS

Input: $G = (N, E)$, $W = (V, M, L)$
Output: Priority values $PV(n_i)$ for all subtasks

1: Initialization: $G^T = (N, E^T)$, $visited = \emptyset$, $pending = \{n_{\mathrm{sink}}\}$, $\mathrm{pred}(n_i)$, $\mathrm{succ}(n_i)$.
2: **while** $len(pending) > 0$ **do**
3: **if** $n_i \in \{n_i \mid pred(n_i) \subset visited \ \forall n_i \in pending\}$ **then**
4: $pending \leftarrow pending - \{n_i\}$;
5: $cost \leftarrow 0$;
6: **for** $n_j \in pred(n_i)$ **do**
7: $temp \leftarrow PV(n_j) + \overline{o_i} + \overline{g_{j,i}}$;
8: $cost \leftarrow \max\{cost, temp\}$;
9: **end for**
10: $PV(n_i) \leftarrow cost$;
11: $visited \leftarrow visited \cup \{n_i\}$;
12: **for** $n_j \in succ(n_i)$ **do**
13: **if** $n_j \notin visited$ **then**
14: $pending \leftarrow pending \cup \{n_j\}$;
15: **end if**
16: **end for**
17: **end if**
18: **end while**
19: **return** $\{PV(n_i) \ \forall n_i \in N\}$.

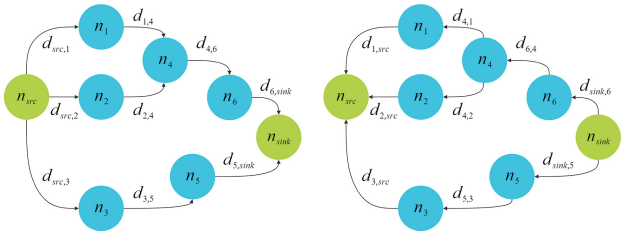

Fig. 4. Get the reversed DAG task. (a) DAG task. (b) Reversed DAG task.

$$\overline{g_{i,j}} = \frac{|C|(|C|-1)d_{i,j}}{\sum_{c \in C} \sum_{\substack{c' \in C \\ c' \neq c}} R_{cc'}} \quad \forall \ (n_i, n_j) \in E. \quad (20)$$

If $\mathrm{succ}(n_i) = \emptyset$, then we define $PV(n_i) = 0$. Typically, there is only one sink subtask, denoted as n_{sink}. Equation (19) calculates the average processing delay $\overline{o_i}$ for subtask n_i in the MEC-enabled 5G IoV network environment, while (20) calculates the average transmission time $\overline{g_{i,j}}$ for the data dependency (n_i, n_j).

More specifically, Algorithm 1 provides a detailed description. The algorithm takes the DAG task $G = (N, E)$ and the MEC-enabled 5G IoV network environment $W = (V, M, L)$ as inputs to obtain the PV $PV(n_i)$ for each subtask n_i. Initialization is performed in line 1. The basic idea of Algorithm 1 is to use the RBFS algorithm. So in the initialization, the reversed tasks G^T of the current DAG task are obtained by reversing all edges $(n_i, n_j) \rightarrow (n_j, n_i)$, as shown in Fig. 4. $\mathrm{succ}(n_i)$ and $\mathrm{pred}(n_i)$ represent the set of predecessor tasks and successor tasks for each subtask n_i. $visited = \emptyset$ initializes the set of visited subtasks, which is used to record subtasks for which the PV has been generated ($visited = n_i \mid PV(n_i) \neq \mathrm{None} \ \forall n_i \in N$). $pending = n_{\mathrm{sink}}$

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on March 12,2024 at 07:10:23 UTC from IEEE Xplore. Restrictions apply.

represents the set of pending subtasks, which is used to record subtasks that may have priority values generated (pending $= n_i \mid \text{pred}(n_i) \cap \text{visited} \neq \emptyset \ \forall n_i \in N, n_i \notin \text{visited}$). Line 2 iterates for each subtask n_i in the pending set to generate the PV $PV(n_i)$. Line 3 represents visiting subtasks n_i whose predecessor tasks have all been visited ($n_i \in n_i \mid \text{pred}(n_i) \subset \text{visited} \ \forall n_i \in \text{pending}$). Line 4 represents removing the currently visited subtask n_i from the pending set. Lines 5–10 represent the PV for subtask n_i. Line 11 adds the subtask n_i to the visited set. Lines 12–16 represent adding the unvisited successor tasks $n_j \in n_j \mid n_j \notin \text{visited} \ \forall n_j \in succ(n_i)$ of the currently visited subtask n_i to the pending set. Finally, line 19 returns all the priority values for subtasks in the MEC-enabled 5G IoV network environment, $PV(n_i) \ \forall n_i \in N$.

B. Subtask Offloading

In the second stage of the algorithm, a greedy approach is used to make offloading decisions. All subtasks $\{n_i \ \forall i \in N\}$ are sorted in descending order based on their priority values. For each subtask n_i, a greedy offloading decision is made to minimize the AFT $\text{AFT}(n_i)$, and the offloading matrix Y is updated based on the offloading decision for subtask n_i. Thus, we propose Algorithm 2.

Algorithm 2 takes the DAG task, the initiating vehicle, and the current execution status of all computing devices as inputs and provides the offloading decisions for all subtasks and the AFT of the task.

Line 1 represents the initialization process. In line 1, the AFT for each subtask is set to $\text{AFT}(n_i) = None$. $\text{pred}(n_i)$ and $\text{succ}(n_i)$ represent the sets of direct predecessors and successors of subtask n_i in $G = (N, E)$. We use Algorithm 1 to obtain the priority values for the subtasks and sort them in descending order to obtain the subtask scheduling list $Order$. We initialize the offloading decision matrix Y as a zero matrix.

Line 2 uses a loop to traverse the subtask scheduling list. Line 3 represents the offloading decision for subtask n_i. Line 5 initializes the AFT of subtask n_i to infinity. Lines 7–11 represent the selection of the device $device$ that minimizes the AFT $\text{AFT}(n_i)$ from all computing devices. Lines 9–11 represent the calculation of the EST of the data dependencies of the subtask's direct predecessors on device c. Line 12 obtains the EST for the subtask on device c considering the device's computational resource constraint. Line 13 obtains the earliest completion time of the subtask on device c. Lines 14–17 represent recording the optimal offloading device selection among all devices. Line 19 updates the usage of computing resources on device c. Line 20 updates the offloading decision by offloading subtask n_i to device $device$. Line 22 returns the offloading decisions for all subtasks and the AFT $\text{AFT}(n_{\text{sink}})$ of the convergence subtask, which represents the completion time of the DAG task.

C. Algorithm Complexity Analysis

The time complexity analysis of the proposed algorithms in this section is as follows.

Algorithm 2 PBTSA

Input: $G = (N, E)$, (V, M, L), v_{init}, $\{t^c_{\text{remain}} \ \forall c \in C\}$
Output: Y, $AFT(n_{\text{sink}})$

1: Initialization: $AFT(n_i) = None$, $\text{pred}(n_i)$, $\text{succ}(n_i)$, $Order$, $Y = \{y_{i,c} = 0 \ \forall n_i \in N, c \in C\}$.
2: **while** $len(Order) > 0$ **do**
3: $n_i \leftarrow Order[0]$;
4: $Order \leftarrow Order - \{n_i\}$;
5: $AFT(n_i) \leftarrow \infty$;
6: $device \leftarrow None$;
7: **for** $c \in C$ **do**
8: $EST(n_i) \leftarrow 0$;
9: **for** $n_j \in pred(n_i)$ **do**
10: $EST(n_i) \leftarrow \max\{EST(n_i), AFT(n_j) + g_{j,i}\}$;
11: **end for**
12: $EST(n_i) \leftarrow \max\{EST(n_i), t^c_{\text{remain}}\}$;
13: $EFT(n_i) \leftarrow EST(n_i) + \omega_i / f_c$;
14: **if** $EFT(n_i) < AFT(n_i)$ **then**
15: $device \leftarrow c$;
16: $AFT(n_i) \leftarrow EFT(n_i)$;
17: **end if**
18: **end for**
19: $t^{device}_{\text{remain}} \leftarrow AFT(n_i)$;
20: $y_{i,device} \leftarrow 1$;
21: **end while**
22: **return** Y *and* $AFT(n_{\text{sink}})$.

First, in the generation of subtask priorities using Algorithm 1, we need to access each subtask and its predecessors' priorities in the DAG task. Therefore, the complexity is $O(|N| + |E|)$. Next, in the greedy offloading of each subtask using Algorithm 2, the complexity is $O(|N|)$ as we try to offload each subtask to the device that minimizes the AFT. Considering that we try all devices, the complexity becomes $O(|N||C|)$. Thus, the complexity of Algorithm 2 is $O(|N|^2|C|)$.

V. Simulation Results and Analysis

This section begins by introducing the setting of key parameters and then proceeds to compare the proposed algorithm with existing algorithms. Zheng et al. [33] proposed a PBLA algorithm. This algorithm computes three parameters for each subtask and then uses these parameters to sort all subtasks. The sorted results are then used for greedy task offloading. Another study by [34] introduced a TBTOA algorithm. This algorithm determines the priority of each subtask on each computing device and performs greedy task offloading based on these priorities. The following section will present and analyze the performance comparison results.

A. Simulation Environment and Parameters

In the simulation experiments, we compare the PBTSA algorithm proposed in this article with PBLA and TBTOA. We use Python 3.11.2 to build a simulation platform to test and verify the performance of the comparison algorithms. The communication-related parameters used in the experiments are taken from [35], while the task, MEC, vehicle

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on March 12,2024 at 07:10:23 UTC from IEEE Xplore. Restrictions apply.

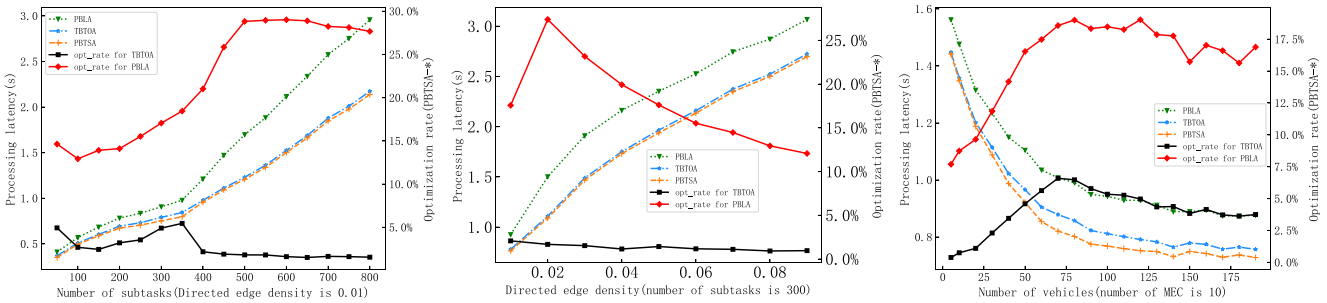

Fig. 5. Variations in processing latency and optimization rate under different experimental parameters. (a) Results for different number of subtasks. (b) Results for different directed edge densities. (c) Results for different number of vehicles.

computing frequency, and other computation-related parameters are derived from [36].

B. Simulation Results and Analysis

This section primarily focuses on comparing the performance metrics and optimization rates of the same tasks under different algorithms (PBTSA, PBLA, and TBTOA). The performance comparison and optimization rates are obtained by averaging the metrics over 50 repeated experiments. Taking the PBLA algorithm as an example, the optimization rate formula of the PBTSA algorithm is

$$\text{opt\_rate} = \frac{\text{Value}_{\text{PBTSA}} - \text{Value}_{\text{PBLA}}}{\text{Value}_{\text{PBLA}}}. \tag{21}$$

Fig. 5 shows variations in processing latency and optimization rate under different experimental parameters. Fig. 5(a) illustrates the processing latency and optimization rates of the three algorithms when executing the same DAG task. Directed edge density refers to the ratio between the actual number of edges and the possible number of edges in a directed graph. For example, a directed edge density of 0.01 means that the number of edges in the DAG is 0.01 times the maximum possible number of edges. A higher directed edge density indicates a greater interdependence among the nodes in the graph, implying stronger dependencies among the subtasks within the application. From the figure, we can observe that as the number of subtasks in the DAG task increases, the computation time of the task also increases, resulting in an increasing processing latency for all three algorithms. The PBTSA algorithm performs better than the PBLA and TBTOA algorithms. The optimization rate of the PBTSA algorithm relative to the PBLA algorithm remains above 10% and increases as the number of subtasks in the DAG task increases. When the number of subtasks reaches 500, the optimization rate stabilizes at around 25%–30%. For subtask numbers ranging from 10 to 350, the optimization rate of the PBTSA algorithm relative to the TBTOA algorithm increases from 3% to 6%. However, as the number of subtasks further increases, the optimization rate decreases to around 2%. These results demonstrate that the proposed PBTSA algorithm can achieve a more optimal subtask scheduling order, leading to shorter processing latency for the DAG task.

Fig. 5(b) shows the processing latency and optimization rates of the three algorithms for a DAG task with 300 subtasks and different directed edge densities. It can be observed that as the directed edge density increases, the time for data dependency transmission also increases, resulting in an increase in the processing latency for all three algorithms. The PBTSA algorithm performs better than the PBLA and TBTOA algorithms. As the density of the DAG task increases from 0.01 to 0.09, the optimization rates of the PBTSA algorithm relative to the PBLA and TBTOA algorithms decrease. Therefore, the proposed PBTSA algorithm achieves better subtask scheduling optimization for DAG tasks with lower densities. This is because as the density of the DAG task increases, the number of directed edges and data dependencies also increases. Since the greedy algorithm schedules subtasks based on their priority order, the more data dependencies there are, the more fixed the priority order of the subtasks becomes.

To further investigate the impact of the number of vehicles in MEC-enabled 5G IoV networks on the processing latency of DAG tasks, experiments are conducted with the number of vehicles as a variable. The simulation results are shown in Fig. 5(c). It can be observed that as the number of vehicles increases, the computing resources in the network environment increase, resulting in a decrease in the processing latency of DAG tasks. However, due to the data dependencies in DAG tasks, the decrease in processing latency occurs at a slower rate as computing resources continue to increase. When the number of vehicles is between 5 and 80, the optimization rate of the PBTSA algorithm relative to the PBLA algorithm increases from 7.5% to 18%, and relative to the TBTOA algorithm, it increases from around 0% to 6%. As the number of vehicles continues to increase, the optimization rate of the PBTSA algorithm stabilizes at above 15% relative to the PBLA algorithm and decreases from 6% to 3% relative to the TBTOA algorithm. Therefore, as the number of vehicles initially increases, the optimization rate of the PBTSA algorithm relative to the PBLA and TBTOA algorithms also increases, as the increase in computing resources leads to a higher optimization rate for the PBTSA algorithm. However, after reaching a critical point in the number of vehicles, the optimization rate starts to decrease, which is due to the nature of data dependencies in DAG tasks.

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on March 12,2024 at 07:10:23 UTC from IEEE Xplore. Restrictions apply.

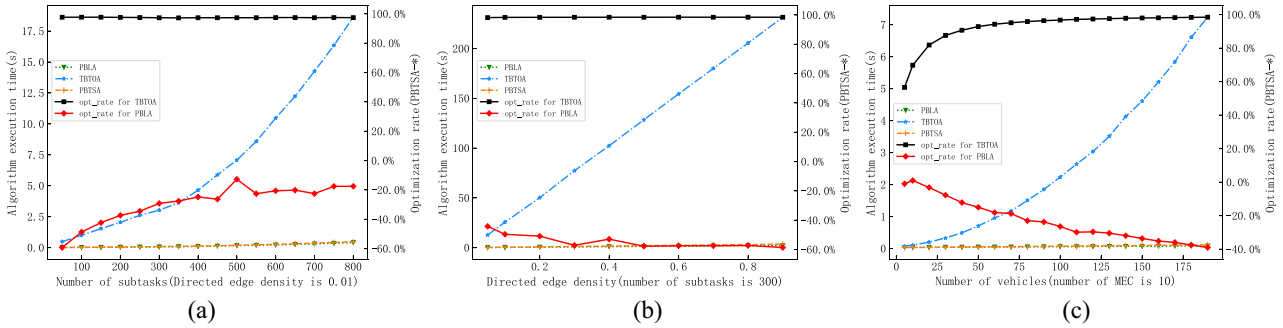

Fig. 6. Variations in algorithm execution time and optimization rate under different experimental parameters. (a) Results under different number of subtasks. (b) Results under different directed edge densities. (c) Results under different number of vehicles.

Fig. 6 shows variations in algorithm execution time and optimization rate under different experimental parameters. Fig. 6(a) illustrates the execution time of algorithms and the optimization rate of the PBTSA algorithm under different number of subtasks. It can be observed that the TBTOA algorithm shows the fastest growth in execution time. On the other hand, the execution time of the PBLA and PBTSA algorithms grows at a much slower rate. Therefore, in the same environment and with the same task, the PBLA algorithm shows the shortest execution time, followed by the PBTSA algorithm. The TBTOA algorithm has the longest execution time, significantly longer than the execution times of the PBLA and PBTSA algorithms.

Fig. 6(b) shows the execution time of algorithms and the optimization rate of the PBTSA algorithm under different directed edge densities. From the figure, it can be observed that as the number of directed edges in the DAG tasks increases, the execution time of the PBTSA, PBLA, and TBTOA algorithms all increase. Among them, the TBTOA algorithm shows the fastest growth in execution time. On the other hand, the execution time of the PBLA and PBTSA algorithms increases at a much slower rate. When the directed edge density ranges from 0.05 to 0.9, the execution time of both algorithms remains between 0 and 1 s. The optimization rate of the PBTSA algorithm relative to the TBTOA algorithm consistently approaches 100%. However, when the directed edge density ranges from 0.05 to 0.5, the optimization rate of the PBTSA algorithm relative to the PBLA algorithm decreases from -50% to -60%. As the directed edge density continues to increase, the optimization rate stabilizes at around -60%.

Fig. 6(c) illustrates the execution time of algorithms and the optimization rate of the PBTSA algorithm under different number of vehicles. From the figure, it can be seen that as the number of vehicles in the MEC-enabled 5G IoV networks increase, the execution time of the PBTSA, PBLA, and TBTOA algorithms all increase. Among them, the TBTOA algorithm shows the fastest growth in execution time. When the number of vehicles ranges from 5 to 200, the execution time of the TBTOA algorithm increases from around 0 to 7 s. On the other hand, the execution time of the PBLA and PBTSA algorithms increases at a much slower rate. When the number of vehicles ranges from 5 to 200, the execution time of both algorithms remains between 0 and 0.2 s. When the number of vehicles ranges from 5 to 200, the optimization rate of the PBTSA algorithm relative to the TBTOA algorithm increases from 60% to nearly 100%, while the optimization rate of the PBTSA algorithm relative to the PBLA algorithm decreases from 0% to -40%.

The average utilization of computing resources for algorithms in MEC-enabled 5G IoV networks is also an important metric. Fig. 7 shows the variations in average utilization of computing resources and optimization rate under different experimental parameters. Fig. 7(a) shows the average utilization of computing resources and optimization rate for different number of subtasks. From the figure, it can be observed that as the number of subtasks increases, the average utilization of computing resources for the PBTSA, TBTOA, and PBLA algorithms also increases because the increased number of subtasks leads to higher utilization of computing resources in vehicular networks. However, when the number of subtasks reaches 350, the average utilization of computing resources for the three algorithms starts to decrease due to the fixed computing resources in vehicular networks and the data dependency of tasks. The average utilization of computing resources for the PBTSA algorithm is significantly better than that of the PBLA and TBTOA algorithms. The results indicate that the PBTSA algorithm can effectively improve the average utilization of resources.

Fig. 7(b) illustrates the average utilization of computing resources and optimization rate for tasks with different directed edge densities. It can be observed that as the directed edge density increases, the average utilization of computing resources for the PBTSA, PBLA, and TBTOA algorithms decreases. This is because the increase in the density of directed edges in tasks leads to a decrease in the number of subtasks that can be executed in parallel, resulting in idle computing resources. The PBTSA algorithm has a slightly better average utilization of computing resources compared to the TBTOA algorithm, but it generally outperforms the PBLA algorithm. As the density of directed edges in the DAG tasks increases, the difference in the average utilization of computing resources among the three algorithms gradually decreases. This is also because the increase in the number of data dependencies in the DAG tasks leads to a convergence in the execution order of subtasks for the PBTSA, PBLA, and TBTOA algorithms.

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on March 12,2024 at 07:10:23 UTC from IEEE Xplore. Restrictions apply.

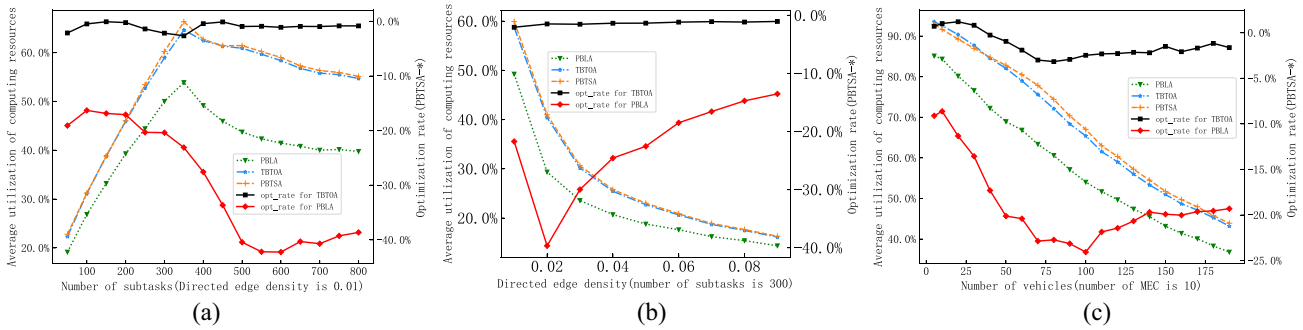

Fig. 7. Variations in average utilization of computing resources and optimization rate under different experimental parameters. (a) Results for different number of subtasks. (b) Results for tasks with different directed edge densities. (c) Results for different number of vehicles.

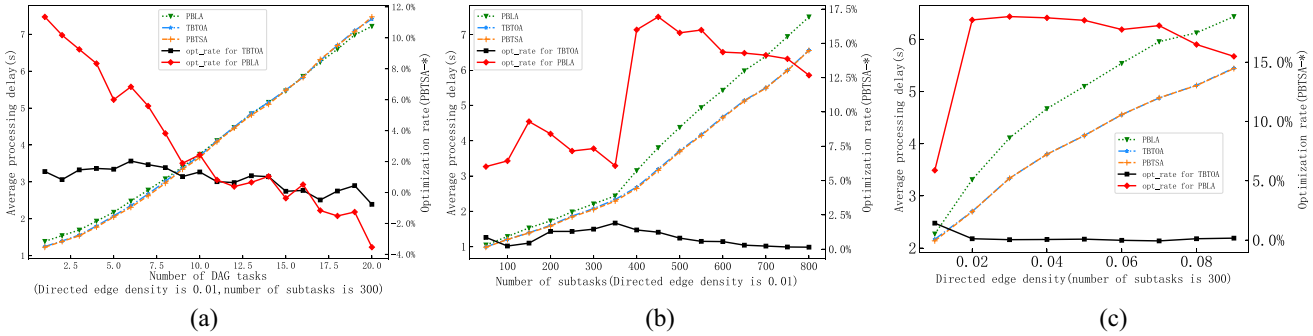

Fig. 8. Variations in average processing delay and optimization rate under different experimental parameters. (a) Results for different number of DAG tasks. (b) Results for different number of subtasks. (c) Results for different directed edge densities.

Fig. 7(c) presents the average utilization of computing resources and optimization rate for different number of vehicles. From the figure, it can be observed that as the number of vehicles increases, the average utilization of computing resources decreases for all three algorithms. This is because with an increased number of vehicles, computing resources increase, but the DAG tasks remain the same. Additionally, due to the data-dependency nature of the DAG tasks, the average utilization of computing resources decreases. As the number of vehicles increases, the PBTSA algorithm shows a higher average utilization of computing resources compared to the PBLA algorithm and the TBTOA algorithm. This indicates that the PBTSA algorithm achieves lower task processing delays with higher average utilization of computing resources.

In order to further investigate the scene of handling multiple identical DAG tasks within a short period of time, this section conducts experiments to obtain the average processing delay of DAG tasks. First, the three algorithms are used to obtain the priority order for each DAG task during offloading, and then the offloading decisions are made based on the earliest scheduling time of each subtask. For example, at time t_A, DAG task A is generated, and at time t_B, DAG task B is generated. Assuming the PBTSA algorithm is used, the priority order of subtasks is given as $\text{order}(A) = [n_{A,0}, n_{A,1}, \ldots]$ and $\text{order}(B) = [n_{B,0}, n_{B,1}, \ldots]$. The earliest scheduling time refers to the earliest time at which all preceding tasks of a subtask have been executed. The initial task scheduling queue is $[(n_{A,0}, t_A), (n_{B,0}, t_B)]$. Assuming $t_A < t_B$, the subtask $n_{A,0}$ of DAG task A is prioritized for greedy offloading. Then, the task scheduling queue becomes $[(n_{A,1}, t_{A,1}), (n_{B,0}, t_B)]$. If $t_{A,1} > t_B$, then the subtask $n_{B,0}$ of DAG task B is greedily offloaded, and the task scheduling queue becomes $[(n_{A,1}, t_{A,1}), (n_{B,1}, t_{B,1})]$. This process continues until all subtasks are offloaded, and relevant metrics are recorded for performance comparison among the three algorithms.

Fig. 8(a) shows the average processing delay and optimization rate for different number of DAG tasks after being randomly offloaded within the time interval (0, 1) s, using the three algorithms. From the figure, it can be observed that as the number of randomly arriving DAG tasks increases, the average processing delay of DAG tasks under the three algorithms also increases. This is because the computing resources are limited and the number of tasks increases, resulting in an increase in computation time. When the number of randomly arriving DAG tasks is between 1 and 20, the optimization rate of the PBTSA algorithm relative to the PBLA algorithm decreases from above 11% to around −4%, and the optimization rate of the PBTSA algorithm relative to the TBTOA algorithm fluctuates between (−1%, 1%). This is because as the number of DAG tasks increases while the computing resources remain unchanged, the optimization range of the PBTSA algorithm also narrows, leading to a decrease in the optimization rate.

Fig. 8(b) shows the average processing delay of DAG tasks after handling multiple randomly arriving DAG subtasks, with the number of subtasks as the variable. In each experiment, the number of DAG tasks is set to 5, and the DAG's directed edge density is 0.01. From the figure, it can be observed that as the

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on March 12,2024 at 07:10:23 UTC from IEEE Xplore. Restrictions apply.

number of subtasks increases, the average processing delay of DAG tasks under the three algorithms also increases. When the number of subtasks is between 50 and 350, the optimization rate of the PBTSA algorithm relative to the PBLA algorithm fluctuates between 5% and 10%. When the number of subtasks is between 350 and 800, the optimization rate stabilizes at 12% or above. When the number of subtasks is between 50 and 800, the optimization rate of the PBTSA algorithm relative to the TBTOA algorithm remains steady between 0% and 1%.

Fig. 8(c) shows the average processing delay of DAG tasks after multiple random DAG subtasks arrive, with the directed edge density of the DAG tasks as the variable. From the figure, it can be observed that as the directed edge density of the subtasks increases, the average processing delay of the DAG tasks under the three algorithms increases. When the directed edge density varies from 0.1 to 0.9, the optimization rate of the PBTSA algorithm relative to the PBLA algorithm gradually decreases from 16% to close to 0%, and the average processing delay of the PBTSA algorithm is similar to that of the TBTOA algorithm. When the directed edge density varies from 0.01 to 0.09, the optimization rate of the PBTSA algorithm relative to the PBLA algorithm is generally greater than 15%, and the average processing delay of the PBTSA algorithm is generally better than that of the TBTOA algorithm, but the optimization rate is low.

VI. Conclusion

This article proposes an online scheduling algorithm for minimizing latency in dependent task scheduling. In the context of 5G-enabled IoV, where vehicles initiate DAG tasks, the challenge lies in scheduling subtasks to fully utilize the computing resources within the vehicular network and achieve minimal task processing delay. This article establishes a mathematical model for DAG task offloading and defines the optimization objective. Based on the established model, a novel online DAG task scheduling algorithm called PBTSA is proposed. This algorithm utilizes the RBFS algorithm to generate priorities for each subtask, resolves data dependencies within the DAG tasks, and greedily offloads subtasks in descending priority order, aiming to minimize subtask execution delay and achieve the optimization objective. Additionally, the PBTSA algorithm is compared with two existing algorithms through simulation experiments to evaluate its performance. The results demonstrate that the PBTSA algorithm effectively reduces DAG task processing delay and shows advantages in terms of algorithm decision execution time and computational resource utilization.

In real-world scenes, multiple DAG tasks are often executed concurrently within the MEC-enabled 5G IoV networks. Therefore, our future research can focus on investigating unified subtask scheduling for multiple identical DAG tasks and multiple distinct types of DAG tasks, respectively. By addressing this issue, a more comprehensive understanding of subtask scheduling in the context of multiple concurrent DAG tasks can be achieved.

References

[1] P. Zhou, W. Zhang, T. Braud, P. Hui, and J. Kangasharju, "ARVE: Augmented reality applications in vehicle to edge networks," in *Proc. Workshop Mobile Edge Commun.*, 2018, pp. 25–30.

[2] J. Lin, L. Huang, H. Zhang, X. Yang, and P. Zhao, "A novel latency-guaranteed based resource double auction for market-oriented edge computing," *Comput. Netw.*, vol. 189, pp. 1–11, Apr. 2021.

[3] Z. Zhou, P. Liu, J. Feng, Y. Zhang, S. Mumtaz, and J. Rodriguez, "Computation resource allocation and task assignment optimization in vehicular fog computing: A contract-matching approach," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3113–3125, Apr. 2019.

[4] M. Kaur, "FastPGA based scheduling of dependent tasks in grid computing to provide QoS to grid users," in *Proc. Int. Conf. Internet Things Appl. (IOTA)*, 2016, pp. 418–423.

[5] S. Sundar and B. Liang, "Offloading dependent tasks with communication delay and deadline constraint," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 37–45.

[6] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading dependent tasks in mobile edge computing with service caching," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 1997–2006.

[7] K. Zhang et al., "Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.

[8] S. Thananjeyan, C. A. Chan, E. Wong, and A. Nirmalathas, "Application based energy optimization for computation offloading in hierarchical MEC network," in *Proc. 10th Int. Conf. Inf. Autom. Sustainabil. (ICIAfS)*, 2021, pp. 129–134.

[9] W. Wang, H. Li, Y. Liu, W. Cheng, and R. Liang, "Files cooperative caching strategy based on physical layer security for air-to-ground integrated IoV," *Drones*, vol. 7, no. 3, pp. 1–18, 2023.

[10] J. Huang, J. Wan, B. Lv, Q. Ye, and Y. Chen, "Joint computation offloading and resource allocation for edge-cloud collaboration in Internet of Vehicles via deep reinforcement learning," *IEEE Syst. J.*, vol. 17, no. 2, pp. 2500–2511, Jun. 2023.

[11] S. Zhou, W. Jadoon, and I. A. Khan, "Computing offloading strategy in mobile edge computing environment: A comparison between adopted frameworks, challenges, and future directions," *Electronics*, vol. 12, no. 11, p. 2452, 2023.

[12] C. Li, C. Qianqian, and Y. Luo, "Low-latency edge cooperation caching based on base station cooperation in SDN based MEC," *Expert Syst. Appl.*, vol. 191, pp. 1–14, Apr. 2022.

[13] G. Li, M. Zeng, D. Mishra, L. Hao, Z. Ma, and O. A. Dobre, "Latency minimization for IRS-aided NOMA MEC systems with WPT-enabled IoT devices," *IEEE Internet Things J.*, vol. 10, no. 14, pp. 12156–12168, Jul. 2023.

[14] Y. Sahni, J. Cao, L. Yang, and Y. Ji, "Multihop offloading of multiple DAG tasks in collaborative edge computing," *IEEE Internet Things J.*, vol. 8, no. 6, pp. 4893–4905, Mar. 2021.

[15] M. Maray, A. Jhumka, A. Chester, and M. Younis, "Scheduling dependent tasks in edge networks," in *Proc. IEEE 38th Int. Perform. Comput. Commun. Conf. (IPCCC)*, 2019, pp. 1–4.

[16] S. E. Mahmoodi, R. N. Uma, and K. P. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 301–313, Apr.-Jun. 2019.

[17] X. Bi, X. Sun, Z. Lyu, B. Zhang, and X. Wei, "A back adjustment based dependent task offloading scheduling algorithm with fairness constraints in VEC networks," *Comput. Netw.*, vol. 223, pp. 1–15, Mar. 2023.

[18] Z. Tang, J. Lou, F. Zhang, and W. Jia, "Dependent task offloading for multiple jobs in edge computing," in *Proc. 29th Int. Conf. Comput. Commun. Netw. (ICCCN)*, 2020, pp. 1–9.

[19] H. Peng, W.-S. Wen, M.-L. Tseng, and L.-L. Li, "Joint optimization method for task scheduling time and energy consumption in mobile cloud computing environment," *Appl. Soft Comput.*, vol. 80, pp. 534–545, Jul. 2019.

[20] B. Huang et al., "Security modeling and efficient computation offloading for service workflow in mobile edge computing," *Future Gener. Comput. Syst.*, vol. 97, pp. 755–774, Aug. 2019.

[21] Y. Xie et al., "A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud–edge environment," *Future Gener. Comput. Syst.*, vol. 97, pp. 361–378, Aug. 2019.

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on March 12,2024 at 07:10:23 UTC from IEEE Xplore. Restrictions apply.

[22] C.-T. Mo, J.-H. Chen, and W. Liao, "Graph convolutional network augmented deep reinforcement learning for dependent task offloading in mobile edge computing," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2023, pp. 1–6.

[23] F. Song, H. Xing, S. Luo, D. Zhan, P. Dai, and R. Qu, "A multiobjective computation offloading algorithm for mobile-edge computing," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8780–8799, Sep. 2020.

[24] A. A. Al-Habob, O. A. Dobre, A. G. Armada, and S. Muhaidat, "Task scheduling for mobile edge computing using genetic algorithm and conflict graphs," *IEEE Trans. Veh. Technol.*, vol. 69, no. 8, pp. 8805–8819, Aug. 2020.

[25] W. Zhang and Y. Wen, "Energy-efficient task execution for application as a general topology in mobile cloud computing," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 708–719, Jul.-Sep. 2018.

[26] M. Kaur, S. Kadam, and N. Hannoon, "Multi-level parallel scheduling of dependent-tasks using graph-partitioning and hybrid approaches over edge-cloud," *Soft Comput. Fusion Found. Methodol. Appl.*, vol. 26, no. 11, pp. 5347–5362, 2022.

[27] M. Asim, W. K. Mashwani, and A. A. Abd El-Latif, "Energy and task completion time minimization algorithm for UAVs-empowered MEC SYSTEM," *Sustain. Comput. Inform. Syst.*, vol. 35, pp. 1–9, sep. 2022.

[28] T. Liu, D. Guo, Q. Xu, H. Gao, Y. Zhu, and Y. Yang, "Joint task offloading and dispatching for MEC with rational mobile devices and edge nodes," *IEEE Trans. Cloud Comput.*, vol. 11, no. 3, pp. 3262–3273, Jul.-Sep. 2023.

[29] I. Dias, L. Ruan, C. Ranaweera, and E. Wong, "From 5G to beyond: Passive optical network and multi-access edge computing integration for latency-sensitive applications," *Opt. Fiber Technol.*, vol. 75, pp. 1–12, Jan. 2023.

[30] F. Spinelli and V. Mancuso, "Toward enabled industrial verticals in 5G: A survey on MEC-based approaches to provisioning and flexibility," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 1, pp. 596–630, 1st Quart., 2021.

[31] T. Deng, Y. Chen, G. Chen, M. Yang, and L. Du, "Task offloading based on edge collaboration in MEC-enabled IoV networks," *J. Commun. Netw.*, vol. 25, no. 2, pp. 197–207, Apr. 2023.

[32] M. R. Garey and D. S. Johnson, ""Strong" NP-completeness results: Motivation, examples, and implications," *J. ACM*, vol. 25, no. 3, pp. 499–508, 1978.

[33] W. Zheng, C. Wang, Z. Chen, and D. Zhang, "A priority-based level heuristic approach for scheduling DAG applications with uncertainties," in *Proc. IEEE 25th Int. Conf. Comput. Supported Cooperat. Work Design*, 2022, pp. 1022–1027.

[34] X. Lv, H. Du, and Q. Ye, "TBTOA: A DAG-based task offloading scheme for mobile edge computing," in *Proc. IEEE Int. Conf. Commun.*, 2022, pp. 4607–4612.

[35] H. Wang, Z. Lin, K. Guo, and T. Lv, "Computation offloading based on game theory in MEC-assisted V2X networks," in *Proc. IEEE Int. Conf. Commun. Workshops*, 2021, pp. 1–6.

[36] X. Dai et al., "Task co-offloading for D2D-assisted mobile edge computing in Industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 19, no. 1, pp. 480–490, Jan. 2023.

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on March 12,2024 at 07:10:23 UTC from IEEE Xplore. Restrictions apply.