Dependency-Aware Computation Offloading for Mobile Edge Computing with Edge-Cloud Cooperation

Long Chen, Jigang Wu, *Member, IEEE,* Jun Zhang, *Senior Member, IEEE,* Hong-Ning Dai *Senior Member, IEEE,* Xin Long and Mianyang Yao

***Abstract*—Most of existing Multi-access edge computing (MEC) studies consider the remote cloud server as a special edge server, the opportunity of edge-cloud collaboration has not been well exploited. We propose a dependency-aware offloading scheme in MEC with edge-cloud cooperation under task dependency constraints. Each mobile device has a limited budget and has to determine which sub-task should be computed locally or should be sent to the edge or remote cloud. To address this issue, we divide the offloading problem into two application finishing time minimization sub-problems with two different cooperation modes, both of which are proved to be NP-hard. We then devise one greedy algorithm with approximation ratio of $1+\epsilon$ for the first mode with edge-cloud cooperation but no edge-edge cooperation. Then we design an efficient greedy algorithm for the second mode, considering both edge-cloud and edge-edge co-operations. Extensive simulation results show that for the first mode, the proposed greedy algorithm achieves near optimal performance for typical task topologies. On average, it outperforms the modified Hermes benchmark algorithm by about $23\% \sim 43.6\%$ in terms of application finishing time with given budgets. By further exploiting collaborations among edge servers in the second cooperation mode, the proposed algorithm helps to achieve over 20.3% average performance gain on the application finishing time over the first mode under various scenarios. Real-world experiments comply with simulation results.**

***Index Terms*—Edge Computing, Offloading, Task Dependency, Graph, Cooperation**

I. Introduction

Various computation-intensive applications such as mobile gaming and augmented reality are usually delay-sensitive and require high computing resources such as power, memory and battery life. However, due to the small physical size, mobile devices are usually constrained by limited computing power, which has become one of the most challenging issues.

Dr. Long Chen, Prof. Jigang Wu, Mr. Xin Long and Mr. Mianyang Yao are with the School of Computer Science and Technology, Guangdong University of Technology, Guangzhou, Guangdong, 510006. The Co-corresponding authors are Jigang Wu, Long Chen.
E-mail(s):asjgwucn@outlook.com, lonchen@mail.ustc.edu.cn

Prof. Jun Zhang is with the Department of Electronic and Information Engineering, the Hong Kong Polytechnic University, HKSAR, Hong Kong, China.
Email:jun-eie.zhang@polyu.edu.hk

Prof. Hong-Ning Dai is with the Faculty of Information Technology, Macau University of Science and Technology, Macau, China.
Email:hndai@ieee.org

Manuscript received Sept. XXXX, 2019;

Although mobile cloud computing (MCC) enables convenient access to a pool of computation resources in the cloud, offloading tasks to the remote cloud would result in large transmission latencies, which degrades users' quality of experience (QoE). Multi-access edge computing (MEC) [1] [2] has emerged as a remedy to the above limitations. By deploying edge or cloudlet servers that are closer to mobile users, mobile users can easily access abundant computing resources [3]. In this way, the transmission delay can be reduced while meeting the computation resource demands of mobile devices. For example, in the heterogeneous wireless network [4], small cell base stations can be equipped with edge servers to serve local mobile users.

While MEC demonstrates its potential to improve the QoE of mobile users by bringing services close to users, emerging applications anticipate efficient edge-cloud cooperation. For example, when burst user traffic arrives, edge servers on their own may not possess sufficient resources to serve. With 5G, industrial internet of things devices can use cloud servers to execute heavy tasks while running light tasks in edge servers [5]. As pointed out in [6], the edge-cloud architecture is able to solve the edge resource contention problem. Meanwhile, load balance can be achieved for online games with edge-cloud cooperation.

II. Motivation and Contributions

Most of existing studies on computation offloading with MEC only consider task offloading among mobile devices [7] or between mobile devices and the MEC servers [8], [9]. The remote cloud such as Amazon AWS, Windows Azure and Google compute engine with great computing capabilities are neglected to some extent. As cloud providers are heavily investing to expand data centers, the accessible cloud resources, while still far away compared with MEC, can be utilized with the deployment of high speed optical fiber networks. The opportunity of edge-cloud collaboration brings many advantages. For example, in the area of software engineering, the heavy load training sub-tasks using bug repositories [10] can be offloaded to the remote cloud server while running bug reporting and matching sub-tasks on edge servers to reduce the total execution time of bug classification [11]. For object detection in vehicular networks, the cooperation between edge-cloud can improve the detection accuracy [12]. This can be done by balancing the tasks of global model cloud execution

and geographic context updating on edge servers. Moreover, when some edge servers are idle, they can be further exploited to alleviate the burden of the remote cloud by incorporating collaboration between neighboring edge servers. Through edge offloading, the total cost of users can be reduced because of the cheaper service price on edge servers over the cloud [13].

In MEC networks, most of existing works assume edge servers and the cloud server always possess abundant resources and they are willing to help. In fact, edge servers and the cloud servers are operated by companies, the revenue of servers should not be negative when providing services. In real-world scenarios, an edge server at local access point may experience poor link connectivity because of link congestion and may be unable to connect to the remote cloud server. Edge servers must then collaborate with each other via neighboring edge servers. An edge server must handle tasks from its local users and the tasks from neighboring edge servers, which complicates the offloading problem. Different from the most similar work [14] that investigates task finishing time minimization problem with budget constraints, the utilities of edge and cloud servers are ensured in this paper. To solve the above challenges, in this paper, we try to fill the gap by exploring the collaborations among both edge-edge and edge-cloud, which haven't been properly investigated yet. The main contributions are summarized as follows:

- We consider the joint task offloading and computation service coordination problem in a three-layer wireless access network, *i.e.*, remote cloud, edge servers co-located at access points (APs) and mobile devices. In this scenario, not only the cooperation between the edge and remote cloud, but also the collaborations among edge servers are taken into account.
- We present the system model in terms of mobile devices' energy consumptions and limited budgets. The dependency-aware offloading problem is formulated as a task completion time minimization problem under two cooperation modes, both of which are proved to be NP-hard: one with only edge-cloud cooperation, and the other with both edge-cloud and edge-edge co-operations.
- Since it is difficult to solve the NP-hard problem, we devise one greedy algorithm with approximation ratio of $1+\epsilon$ for the first mode with edge-cloud cooperation but no edge-edge cooperation. Then we design an efficient greedy algorithm for the second mode considering both edge-cloud and edge-edge co-operations.
- Simulation results show that for the first mode, the proposed greedy algorithm outperforms the modified benchmark algorithm by about $23\% \sim 43.6\%$ on the application finishing time averagely. For the second mode, by further exploiting collaborations among edge servers, the proposed algorithm helps to achieve over 20.3% average performance gain over the first mode under various cases. Real-world experiments comply with simulation results. All the codes can be obtained via the following link https://github.com/Derfei/Dependency-Aware-Computation-Offloading-for-Mobile-Edge-Computing-with-Edge-Cloud-Cooperation.

The remainder of the paper is organized as follows. Related works on offloading with MEC are presented in Section III. Section IV presents the system and computational models. Problem formulation is described in Section V. Section VI presents the proposed algorithms. Performance evaluation is presented in Section VII and Section VIII concludes this paper with future remarks.

III. Related Works

For MEC, computation offloading can be classified into two types, *i.e. full offloading* [15] and *partial offloading* [8]. For *full offloading*, the whole computation task is offloaded and processed by the edge server. Chen et al. in [15] proposed a game theoretic offloading scheme in the multi-channel wireless contention environment, minimizing the number of cloud computing users with interference constraints. For *partial offloading*, part of the computation tasks are processed locally on the mobile devices while the rest are offloaded to the MEC. In [8], by using convex optimization, You et al. presented multi-user offloading algorithms to reduce the energy consumption of mobile devices with delay constraints. In [9], Mao et al. developed a stochastic optimization algorithm for joint radio and computation resource management in multi-user MEC systems with partial offloading. *However, these works all focus on how much workload should be distributed to the MEC server without considering task dependency of an application*.

Although there have been recent works [16], [17] on computation offloading with task dependency in MEC, *they failed to address budgets constraints of mobile users* [18]. In [16], under the multi-tenant cloud computing environment, Rimal et al. designed a few algorithms to schedule workflows with flow delay constraints. In [17], sub-tasks of an application can be offloaded to mobile device and the cloud with application finishing time constraint. In [19], authors regarded the user budgets as peak transmission power and peak CPU frequency, but they assume edge servers are willing to offer offloading services. In fact, the utilities of the edge and cloud servers should be guaranteed.

In this work, we investigate partial offloading with joint consideration of computation costs of mobile devices and the edge servers. Computation offloading that takes both cost and delay into consideration in mobile cloud computing has not been extensively studied. To our best knowledge, there are only a few works that have addressed the computation offloading problem in MEC considering the cost of edge servers such as energy consumption during computation and transmission. In [20], Deng et al. investigated the power consumption and delay trade-off with the objective to minimize total power consumption of edge servers and remote cloud servers. Chen et al. [21] jointly optimized the offloading decisions of all user tasks and the allocation of computation resources with the objective to minimize the total cost. Compared with our work, *they failed to address task dependency and cost budgets of mobile devices, which are critical factors that affect users' performance*.

TABLE I
NOTATIONS FOR SYSTEM MODEL

Notations	Meaning
$\mathcal{M}$	The set of mobile devices, where $\|\mathcal{M}\| = M$
$\mathcal{N}$	The set of sub-tasks, where $\|\mathcal{N}\| = N$
m	Mobile device m, where $m \in \mathcal{M}$
$G(V, E)$	Task graph of an application with $\|V\| = N$ and E is the edge set
$\mathcal{K}_e, \mathcal{K}_c$	Edge server set, cloud server set, respectively
$x^e_{m,n,p}$	Offloading strategy variable when $x^e_{m,n,p} = 1$ meaning that mobile device m chooses to offload sub-task n to pth edge server
$x^c_{m,n,q}$	Offloading strategy variable when $x^c_{m,n,q} = 1$ meaning that sub-task n is further offloaded to the qth remote cloud server
$\tau^l_{m,n}$, $E^l_{m,n}$	Completion time, energy consumption of local execution of sub-task n on mobile device m, respectively
$\tau^e_{m,n}$, $E^e_{m,n}$	Computation time, energy consumption for sub-task n of mobile user m on the edge server, respectively
$\tau^t_{m,n}$, $E^t_{m,n}$	Transmission time, energy consumption between device and its corresponding access point, respectively
$\tau^c_{m,n,q}$, $E^c_{m,n}$	Remote execution time, energy consumption for sub-task n of device m on remote cloud server q, respectively
$w_{m,n}$, $d_{m,n}$, $C_{m,n}$	Workload, data size, cost for sub-task n of mobile device m, respectively
T_m	Task finishing time of mobile device m

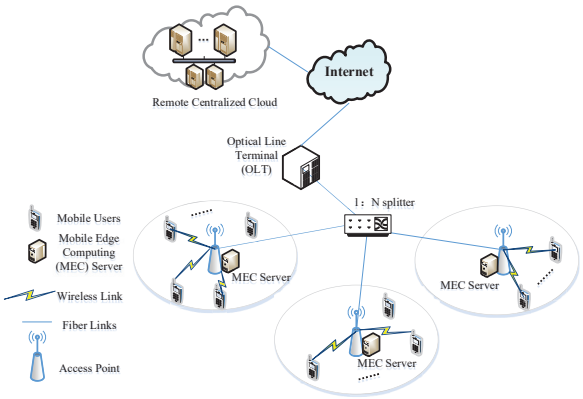


Fig. 1. System Architecture

IV. System Model and Computational Model

A. System Model

As shown in Fig. 1, there is one optical fiber line splitter connecting multiple APs with an optical line terminal through high speed fiber links. Usually, the optical fiber line splitter is connected with a central controller e.g., a software defined network controller, to collect necessary network information. The APs can be long time evolution, 5G base stations and WiFi access points. A control server at the splitter can be used to schedule computation tasks. The remote cloud center is linked to the optical line terminal via the Internet. Each AP that can be a WiFi or a small cell base station in a HetNet is equipped with an edge server. We assume that a group of mobile devices denoted as $\mathcal{M} = \{1, 2, \cdots, M\}$ are located in the vicinity of their corresponding APs. A mobile application is partitioned into a set of N sub-tasks denoted by $\mathcal{N} = \{1, 2, \cdots, N\}$. The application addressed in this paper can be represented by a directed acyclic graph (DAG) with one entry node and one exit node. The entry node is the root on the spanning tree of the graph, which represents the beginning sub-task. The exit node represents the sub-task where the whole application terminates. Let $\mathcal{K}_e = \{1, 2, \cdots, P\}$ and $\mathcal{K}_c = \{1, 2, \cdots, Q\}$ denote the set of edge servers and the set of cloud servers, respectively. There is a set of $\mathcal{K} = \mathcal{K}_c \cup \mathcal{K}_e$ servers in total[1]. Hence, $|\mathcal{K}| = P+Q$. The application is modeled as a weighted DAG denoted by $G(V, E)$, where V denotes a set of sub-tasks, each of which is v_i and $i = 1, 2, \cdots, N$ and $|V| = N$. The term E denotes a set of edges, each of which e_{ij} represents precedence relation such that task v_i should complete its execution before task v_j starts, where $(i, j) \in \{1, 2, \cdots, N\} \times \{1, 2, \cdots, N\}$ and $|E| = e$. The notations are summarized in Table I.

B. Channel Model

The channel between mobile device $m \in \mathcal{M}$ and AP $p \in \mathcal{K}_e$ follows quasi-static block fading. We assume that orthogonal frequency division multiplexing (OFDM) access with the corresponding interference cancellation algorithms is adopted so that interference between any two adjacent cells can be properly handled [4]. For each cell, a particular OFDM sub-channel is used by at most one mobile device in that cell. Let $x^e_{m,n,p} \in \{0, 1\}$ and $x^c_{m,n,q} \in \{0, 1\}$ represent the computation offloading strategies to edge and cloud servers, respectively, where $p \in \mathcal{K}_e$ and $q \in \mathcal{K}_c$. Particularly, $x^e_{m,n,p} = 1$ means that mobile device m chooses to offload sub-task n to the pth edge server while $x^e_{m,n,p} = 0$ means otherwise. Similarly, $x^c_{m,n,q} = 1$ means that mobile device m decides to offload sub-task n to the qth remote cloud server and $x^c_{m,n,q} = 0$ otherwise. We can compute the uplink data rate denoted by $R_{m,n,p}$, for wireless communication between device m and access point p when transmitting the nth sub-task as

$$R_{m,n,p} = W \log_2 \left(1 + \frac{P^{tx}_{m,n,p} G_{m,n,p}}{I + \sigma^2_{m,p}}\right), p \in \mathcal{K}_e, \quad (1)$$

where $P^{tx}_{m,n,p}$ is the transmission power of mobile device m to offload its sub-task n to AP p, which is set as 27dBm by default [22]. The channel gain from the mth device to the pth AP is $G_{m,n,p}$ when transmitting sub-task n and $G_{m,n,p} = (d_{m,p})^{-\eta} |h_{m,p}|^2$, where $d_{m,p}$ is the Euclidean distance between the mth mobile device and the pth edge server, $h_{m,p}$ is the corresponding Rayleigh fading channel coefficient and η is a constant set as 4 [23]. The channel bandwidth is W, the interference from neighboring cells is denoted by I and the surrounding noise power at the receiver with the transmission link (m, p) is $\sigma^2_{m,p}$.

C. Computation Models

Let $\tau^l_{m,n}$ be the completion time of local execution of sub-task n on device m. Let $\tau^t_{m,n}$ be the transmission time between

[1]The same as most existing studies, e.g., [21], we treat cloud servers as a whole by setting $Q = 1$. Because one dedicated high performance cloud server has the strong computation capability, equivalent to a number of servers with inferior computation capability.

the mobile device and its corresponding AP, and $\tau^e_{m,n}$ be the execution time at edge server. Denote the transmission time between the edge servers and the remote cloud server q by $\tau^r_{m,n,q}$. Denote the execution time of sub-task n of device m on remote cloud server q by $\tau^c_{m,n,q}$. Next, we present the computation overhead on energy consumption, task completion time as well as the coordination between edge servers and remote cloud.

1) Local Computing: Let f^l_m be the computation capability, *i.e.,* the average CPU clock speed (Giga Cycles/second) of mobile devices. Note that different mobile devices may have different CPU clock speeds. The computation execution time of sub-task n on mobile device m is then calculated as

$$\tau^l_{m,n} = \frac{w_{m,n}}{f^l_m}, \tag{2}$$

and the energy consumption of mobile device m for the corresponding sub-task n is given by

$$E^l_{m,n} = \kappa w_{m,n} {f^l_m}^2, \tag{3}$$

where κ is set to be 10^{-11} according to [24] and $w_{m,n}$ is the workload of sub-task n of mobile device m.

2) Edge and Cloud Computing: Similar to [24], we ignore the time and energy consumption that the cloud takes in returning the computation outcome because of the small size. Let f^e_p be the computation ability of edge server p, *i.e.,* the average machine CPU frequency. Then, the computation execution time is given by

$$\tau^e_{m,n} = \frac{w_{m,n}}{f^e_p}, \tag{4}$$

and the energy consumption of the edge server is given by

$$E^e_{m,n} = \left(\alpha_e \left(f^e_p\right)^\epsilon + \beta_e\right) \tau^e_{m,n}, \tag{5}$$

where both α_e and β_e are positive constants which can be obtained by offline power fitting and the value ϵ ranges from 2.5 to 3 according to [25]. Similarly, the energy consumption of the remote cloud server for sub-task n of mobile device m is given by

$$E^c_{m,n} = \left(\alpha_c \left(f^c_q\right)^\epsilon + \beta_c\right) \tau^c_{m,n,q}, \tag{6}$$

where the remote execution time $\tau^c_{m,n,q}$ is calculated as

$$\tau^c_{m,n,q} = \frac{w_{m,n}}{f^c_q}. \tag{7}$$

Similarly, f^c_q denotes the average CPU frequency of the qth remote cloud server and both α_c and β_c are also positive constants.

3) Data Transferring Cost: Given the nth sub-task size of mobile device m, including the input data *i.e.* $d_{m,n}$, $\tau^t_{m,n}$ can be expressed as

$$\tau^t_{m,n} = \frac{d_{m,n}}{R_{m,n,p}}. \tag{8}$$

Then the energy cost when transferring the data to AP is given by

$$E^t_{m,n} = P^{tx}_{m,n,p} \tau^t_{m,n}. \tag{9}$$

Furthermore, we can get the data transfer delay if the remote cloud server is employed by the corresponding edge server as

$$\tau^r_{m,n,q} = \frac{d_{m,n}}{\omega}, \tag{10}$$

where ω is average transfer rate or bandwidth between the edge server and the corresponding remote cloud server. The energy cost of the edge server during the offloading to the remote server is denoted by $E^s_{m,n}$, which is given by

$$E^s_{m,n} = P_0 \cdot \tau^r_{m,n,q}, \tag{11}$$

where P_0 is the amount of additional power when performing data transfer per unit time from edge server to the remote server.

4) Dependency Constraints: Before formulating the dependency-aware offloading problem, we present some necessary definitions, quality of service (QoS), and user budget constraints. Note that for a particular task of a mobile user, it cannot be executed unless all its precedent tasks have already been processed. We name this constraint as the *precedence constraint* followed by [26] and [27]. Let $TR^l_{m,n}$ be the time when sub-task n of mobile device m is ready to be processed, let $TF^l_{m,k}$, $TF^e_{m,k}$ and $TF^c_{m,k}$, $k \in pre(n)$ be the task finishing time of the kth predecessor of sub-task n on local device, edge server and the cloud server respectively. Then we have

$$TR^l_{m,n} = \max_{k \in pre(n)} \max\{TF^l_{m,k}, TF^e_{m,k}, TF^c_{m,k}\}, \tag{12}$$

where the receiving delay is neglected following [24], (12) can be rewritten as

$$\begin{aligned} TR^l_{m,n} \geq & (1 - x^c_{m,k,q}) \left[\left(1 - x^e_{m,k,p}\right) TF^l_{m,k} + x^e_{m,k,p} TF^e_{m,k}\right] \\ & + x^c_{m,k,q} TF^c_{m,k}, \qquad k \in pre(n), \end{aligned} \tag{13}$$

where $pre(n)$ denotes the predecessors of task n, $k \in pre(n)$ in (13) means that the local computing of task n can be executed only after task k has been executed. Therefore, the local task completion time of mobile device m, denoted by $TF^l_{m,n}$ is given as follows,

$$TF^l_{m,n} = \tau^l_{m,n} + TR^l_{m,n}. \tag{14}$$

Similarly, let $TR^e_{m,n}$, $TR^c_{m,n}$ be the time when sub-task n of mobile device m is ready to be processed on the edge server and the corresponding time at the remote cloud server, respectively, given as follows,

$$TR^e_{m,n} = \max\{TF^t_{m,n}, \max_{k \in pre(n)} TF^e_{m,k}, \max_{k \in pre(n)} TF^c_{m,k}\}, \tag{15}$$

and

$$TR^c_{m,n} = \max\left\{TF^t_{m,n} + \tau^r_{m,n}, \max_{k \in pre(n)} TF^c_{m,k}, TF^r_{m,n}\right\}. \tag{16}$$

In (15), $TF^t_{m,n}$ that is the transmission finishing time from mobile device m to the corresponding edge server is given by

$$TF^t_{m,n} = \tau^t_{m,n} + \max_{k \in pre(n)} TF^l_{m,k}, \tag{17}$$

which means the task transmission finishing time equals to the current sub-task's transmission time plus the local task finishing time of all the predecessor sub-tasks of the current sub-task. Because we want to obtain the accumulated time on the task transmission finishing time from the start of the application's execution.

The term $TF^e_{m,n}$ that is the task finishing time at the edge server is given by

$$TF^e_{m,n} = \tau^e_{m,n} + TR^e_{m,n}, \tag{18}$$

and $TF^c_{m,k}$ that is the task finishing time at remote cloud server is given by

$$TF^c_{m,n} = \tau^c_{m,n,q} + TR^c_{m,n}. \tag{19}$$

In (16), $TF^r_{m,n}$ denotes the completion time of transmission between edge and remote cloud servers, which can be defined similar to (17), given by

$$TF^r_{m,n} = \tau^r_{m,n} + \max_{k \in pre(n)} TF^e_{m,k}. \tag{20}$$

In (19), $TR^c_{m,n}$ is the time when sub-task n is ready for processing at the remote cloud server.

D. Utility Constraints and Costs

Next, we present the utility constraints of both edge servers and remote cloud servers as well as cost constraints of mobile devices. The utility of edge server can be derived as

$$U^e_p = \sum_{m=1}^{M} \sum_{n=1}^{N} \left(P^e_p d_{m,n} x^e_{m,n,p} - E^s_{m,n} x^c_{m,n,q} - E^e_{m,n} x^e_{m,n,p} \right), \tag{21}$$

$\forall p \in \mathcal{K}_e$, where P^e_p is the charging price by the pth edge server to cover the transmission or execution cost per unit data in the network. It can be observed from (21) that the transmission cost $E^s_{m,n}$ and execution cost $E^e_{m,n}$ cannot coexist for a particular task $n \in N$. Similarly, the utility of the remote cloud server can be expressed as

$$U^c_q = \sum_{m=1}^{M} \sum_{n=1}^{N} \left(P^c_q d_{m,n} x^c_{m,n,q} - E^c_{m,n} x^c_{m,n,q} \right), \forall q \in \mathcal{K}_c, \tag{22}$$

where P^c_q is the charging price of the qth remote cloud server. It should be noted that, to motivate computation offloading, the utility of both edge server and the remote cloud server should not be negative, therefore we have

$$U^e_p \geq 0, U^c_q \geq 0, \forall p \in \mathcal{K}_e, \forall q \in \mathcal{K}_c. \tag{23}$$

For the mobile device m processing sub-task n, its cost $C_{m,n}$ that consists of local execution cost, the payment for the corresponding edge server and the payment for the remote cloud server, can be expressed as

$$\begin{aligned} C_{m,n} = \big\{ \big(1 - x^c_{m,n,q}\big) \big[E^l_{m,n} \big(1 - x^e_{m,n,p}\big) + \\ (P^e_p d_{m,n} + E^t_{m,n}) x^e_{m,n,p} \big] + [P^c_q d_{m,n} + E^t_{m,n}] x^c_{m,n,q} \big\}, \\ \forall p \in \mathcal{K}_e, \forall q \in \mathcal{K}_c, \end{aligned} \tag{24}$$

where for a particular task $n \in N$ of mobile device m, $x^c_{m,n,q}$, $x^e_{m,n,p}$ cannot be one at the same time.

Finally, we derive the running time expression of the whole application for the mobile device. Denote the total application response time for mobile device m by T_m. Then T_m that is the time when all the tasks in an application are finished, is given by

$$T_m = TF_{m,N}. \tag{25}$$

We observe from (25) that the total application delay is the time when the final sub-task N of mobile device m has finished executing on the mobile device.

V. Problem Formulation

In this section, we study the offloading problem under two different assumptions on the edge processing, i.e., single-edge processing and multi-edge processing. While the first assumption only allows cooperation between one edge server and the cloud, the second one also allows collaborations among edge servers for further performance improvement.

A. Single-edge Processing

Definition 1: Single-Edge Processing (SEP) problem considers the case where sub-tasks of an application from one mobile user can be processed by the nearby edge server, or by the remote cloud. But there is no cooperation between edge servers.

To solve the SEP problem, considering mobile user budgets and the cost of servers, we try to design an effective computation offloading strategy with edge-cloud cooperation. The aim is to minimize the total application completion delay, which reflects the execution efficiency of the system. Similar to [28], since servers are usually operated by companies, to motivate them to participate in computation offloading, the utilities should not be negative. We do not impose similar utility constraints for the mobile devices on the basis that mobile phones are carried by users, they can charge batteries of their phones in economic prices. The costs of mobile devices can be represented by the maximum user budgets they are willing to pay. The detail problem is described in OPT-1.

$$\begin{aligned} \textbf{OPT-1} \quad & \min \sum_{m=1}^{M} T_m \\ s.t. \quad & C1:(13), C2:(15), C3:(16), \\ C4: \quad & U^e_p \geq 0, U^c_q \geq 0, \\ C5: \quad & x^c_{m,n,q} + x^e_{m,n,p} \leq 1, \\ C6: \quad & x^c_{m,n,q} \in \{0,1\}, x^e_{m,n,p} \in \{0,1\}, \\ C7: \quad & \sum_{n=1}^{N} C_{m,n} \leq \epsilon_m, \end{aligned} \tag{26}$$

where $\forall m \in \mathcal{M}$, $\forall n \in \mathcal{N}$, $\forall p \in \mathcal{K}_e$, $\forall q \in \mathcal{K}_c$ and ϵ_m is the monetary budget for mobile device m. Constraint $C1$ is the local task dependency constraint ensuring that sub-task n can start execution only after all its predecessor tasks have been finished. Constraints $C2$ and $C3$ are edge and remote cloud task dependency constraints, respectively; they imply that sub-task n can be executed on the edge and remote cloud server only after the task has been completely offloaded to the edge and remote cloud, respectively. Constraint $C4$ is the utility constraint for edge servers and the remote cloud server. Constraint $C5$ ensures that for a sub-task $n \in \mathcal{N}$, it can only be executed on one of the three places, *i.e.* the local mobile device, the edge server or the remote cloud server. The binary constraints are presented in $C6$. Constraint $C7$ is the budget constraint for mobile device m.

B. Multi-edge Processing

We then consider another scenario, in which sub-tasks of a single application are allowed to be processed by multiple edge servers located at multiple small cell base stations. This investigation will help to reveal the potential benefits of the edge-edge cooperation.

Definition 2: Multi-Edge Processing (MEP) problem is for the case where sub-tasks of an application from one mobile user can be processed by multiple edge servers and the remote cloud.

Given a DAG-structured application graph $G(V,E)$, M mobile devices, one remote cloud and p edge servers. Set $\mathcal{K}=\{\mathcal{K}_l,\mathcal{K}_e,\mathcal{K}_c\}$ represents different devices in the network, where K_l ($\mathcal{K}_l \subseteq \mathcal{K}$) stands for the mobile device subset, $\mathcal{K}_c$ is the subset of the remote cloud and $\mathcal{K}_e$ is the subset of edge servers. For each mobile device $m \in \mathcal{M}$, there is a fixed financial budget ϵ_m and for each application generated by the mobile device, there are n_i sub-tasks since $v_i \in V$. The MEP problem is essentially to find a schedule $\mathfrak{S}: v_i \rightarrow k$, $\forall v_i \in V$, $\exists k \in \mathcal{K}$, where k is the element in set $\mathcal{K}$, such that the total application finishing time is minimized with the given cost budgets. Let $E^{I_{st}}_{m,n}$ be the energy consumption for inter transmission between edge servers s and t. Thus, we have

$$E^{I_{st}}_{m,n} = \{\frac{d_{m,n}}{\omega} + l(s,t)\} \cdot P_0, \forall m \in \mathcal{M}, \forall n \in \mathcal{N}, \quad (27)$$

where $l(s,t)$ is the extra delay due to the multi-hop transmission between edge servers s and t. Therefore, the cost function of equation (24) can be rewritten as:

$$\begin{aligned} C'_{m,n} = \big\{ \big(1-x^c_{m,n,q}\big) [\, E^l_{m,n}\big(1-x^e_{m,n,p}\big) + \\ (P^e_p d_{m,n} + E^t_{m,n}) x^e_{m,n,p} + E^{I_{st}}_{m,n} x^I_{st}\,] + \\ (P^c_q d_{m,n} + E^t_{m,n}) x^c_{m,n,q} \big\}, \forall p \in \mathcal{K}_e, \forall q \in \mathcal{K}_c, \end{aligned} \quad (28)$$

where $x^I_{st} \in \{0,1\}$ is a binary variable. When $x^I_{st}=1$, it means that the sub-task from edge server s is executed at the tth edge server and $x^I_{st}=0$ otherwise. As a consequence, the utility of the edge server can be rewritten as

$$\begin{aligned} {U'}^e_p = \sum_{m=1}^{M}\sum_{n=1}^{N} \big(P^e_p d_{m,n} x^e_{m,n,p} - E^s_{m,n} x^c_{m,n,q} \\ - E^e_{m,n} x^e_{m,n,p} - E^{I_{st}}_{m,n} x^I_{st}\big). \end{aligned} \quad (29)$$

Then the MEP problem can be formulated as:

$$\textbf{OPT-2} \quad \min_{\exists \mathfrak{S}} \left(\sum_{m=1}^{M} T_m \right)$$

$$\begin{aligned} s.t. \quad & C8: \quad \sum_{n=1}^{n_i} C'_{m,n} \le \epsilon_m, \\ & C9: \quad {U'}^e_p \ge 0, U^c_q \ge 0, \\ & C10: \quad x^c_{m,n,q} + x^e_{m,n,p} + x^I_{st} \le 1, \\ & C11: \quad x^c_{m,n,q}, x^e_{m,n,p}, x^I_{st} \in \{0,1\}, \\ & C12: \quad w_{m,n} x^e_{m,n,p} + \sum_{u \in \mathcal{M}, v \in \mathcal{N}} w_{u,v} x^I_{sp} x^e_{u,v,s} \le W_p, \\ & \qquad s \ne p, s \in \mathcal{K}_e, u \ne m, v \ne n, and \quad C1, C2, C3. \end{aligned} \quad (30)$$

Different from **OPT-1**, for **OPT-2**, the total cost also includes the computation cost of the neighboring edge server if sub-tasks are offloaded to it. Meanwhile, we assume that edge servers possess the limited computation capacity. Therefore, constraints $C8$, $C9$, $C10$ and $C11$ are modifications from $C7$, $C4$, $C5$ and $C6$, respectively in **OPT-1**. Constraint $C12$ is the computation capability constraint and W_p is the maximum workload that the pth edge server can deal with. It should be noted that the pth edge server cannot process sub-tasks with a total size greater than its allowed workload W_p.

VI. Algorithms for Dependency-aware Offloading

Algorithm 1 Single edge greedy algorithm (SEG) for mobile device m

Input: $\mathcal{N}$: a sequence obtained by the depth first traversal of N sub-tasks on mobile device m;
$pre(n)$: the predecessors of sub-task n;
ϵ_m: the sum budget for mobile device m;
P^e_p: the charging price by the pth edge server;
P^c_q: the charging price of the qth remote cloud server.
Output: Offloading policy $\mathbb{X}$

begin
Initialize: $w_{m,n}, d_{m,n}, U^e_p \leftarrow 0$;
for n =1 to N **do**
  Compute $R_{m,n,p}, \tau^l_{m,n}, E^l_{m,n}$ by (1)-(3), respectively;
  if $pre(n) == \varnothing$ **then**
    $TR^l_{m,n} \leftarrow 0, TR^e_{m,n} \leftarrow 0, TR^c_{m,n} \leftarrow 0$
  else
    Compute $TF^t_{m,n}, TF^r_{m,n}$ by (17), (20), respectively;
    Compute $TR^l_{m,n}, TR^e_{m,n}, TR^c_{m,n}$ by (13), (15), (16), respectively;
  end if
  Compute $TF^l_{m,n}, TF^e_{m,n}, TF^c_{m,n}$ by (14), (18) and (19), respectively;
  if $TF^l_{m,n} < TF^e_{m,n}$ & $TF^l_{m,n} < TF^c_{m,n}$ **then**
    $x^e_{m,n,p} \leftarrow 0, x^c_{m,n,q} \leftarrow 0$;
  else
    if $\frac{P^c_q d_{m,n}}{E^c_{m,n}} \ge 1$ **then**
      $x^e_{m,n,p} \leftarrow 0, x^c_{m,n,q} \leftarrow 1$;
    else
      $x^e_{m,n,p} \leftarrow 1, x^c_{m,n,q} \leftarrow 0$;
    end if
  end if
  Compute $E^e_{m,n}, E^c_{m,n}$ by (5), (6), respectively;
  Compute $C_{m,n}$ by (24);
end for
while $\sum_{n=1}^{N} C_{m,n} > \epsilon_m$ **do**
  if $\sum_{n=1}^{N} x^c_{m,n,q} \ne 0$ **then**
    $y \leftarrow \arg\min_n \{E^c_{m,n}\}$;
    $x^e_{m,y,p} \leftarrow 1, x^c_{m,y,q} \leftarrow 0$;
  else
    $z \leftarrow \arg\min_n \{E^e_{m,n}\}$;
    $x^e_{m,z,p} \leftarrow 0, x^c_{m,z,q} \leftarrow 0$;
  end if
end while
while $U^e_p < 0$ **do**
  if $\sum_{n=1}^{N} x^c_{m,n,q} \ne 0 \& \{\frac{E^s_{m,n}}{E^e_{m,n}} > 1\} \ne \varnothing$ **then**
    $u \leftarrow \arg\max_n \{\frac{E^s_{m,n}}{E^e_{m,n}}\}$;
    $x^e_{m,u,p} = 1, x^c_{m,u,q} = 0$;
  else
    $v \leftarrow \arg\min_n \{\frac{P^e_p d_{m,n}}{E^e_{m,n}}\}$;
    $x^e_{m,v,p} \leftarrow 0, x^c_{m,v,q} \leftarrow 0$;
  end if
  Compute U^e_p by (21);
end while
end

A. Analysis and Algorithms for OPT-1

In the computation offloading decision procedure, the edge server will determine which sub-task should be executed at the mobile device, which sub-task should be offloaded to the edge server and which should further be transmitted to the remote cloud server by the edge server. The objective is to determine a strategy to minimize execution delay with the given budget constraints. Meanwhile, the task dependency constraints should be preserved. Thus, problem **OPT-1** can be reduced to a special 0-1 Knapsack problem, where the ϵ_m is the total volume of the package, so that **OPT-1** is also NP-hard. Note that if $TF^e_{m,n} < TF^l_{m,n}$ and $TF^e_{m,n} < TF^c_{m,n}$, then the sub-task will be offloaded to the edge server. If $TF^c_{m,n} < TF^l_{m,n}$ and $TF^c_{m,n} < TF^e_{m,n}$, the remote cloud will be chosen.

1) Greedy Implementation for SEP sub-problem: Based on the above analysis, first of all, we design a Single Edge Greedy (SEG) offloading algorithm to minimize the task completion time. To acquire the minimum finishing time of all sub-tasks in the application on mobile device m, the minimum completion time of sub-task n is selected from $TF^l_{m,n}$, $TF^e_{m,n}$, and $TF^c_{m,n}$. To meet the utility constraint of the remote cloud server, $P^c_q d_{m,n}$ should be larger than $E^c_{m,n}$ according to (6). By determining the offloading policy for each sub-task consecutively, we obtain the initial solution for mobile device m. This sub-procedure is shown between lines 3 and 32.

Then, we iteratively adjust the initial offloading policy to satisfy the sum cost budget for mobile device m. Considering the fact that in real-world scenarios, the price of remote cloud service denoted by *i.e.*, P^c_q is much more expensive than the service price of edge server denoted by P^{e2}_p. Therefore, the cost of a task on the remote cloud is higher than that on the edge server, then the sub-tasks on the cloud server can be directed to the edge servers in order to save the costs of mobile users, thus users can pay less money for the offloading services. If there have been some tasks deployed on the cloud, *i.e.*, $\sum_{n=1}^{N} x^c_{m,n,q} \neq 0$, we can move them to the edge node one by one according to the ascending order of $E^c_{m,n}$. Similarly, when there is no task deployed on the cloud, we can move tasks from the edge server to the mobile device one by one in ascending order of $E^e_{m,n}$.

Finally, we adjust the offloading policy to satisfy the utility constraint of the edge server. In order to improve the utility of the corresponding edge server, we remove tasks from cloud server to the edge in descending order of $\frac{E^s_{m,n}}{E^e_{m,n}}$, when $E^s_{m,n}$ is greater than $E^e_{m,n}$. If no task is deployed on cloud server, we remove tasks from edge servers to mobile devices in ascending order of $\frac{P^e_p d_{m,n}}{E^e_{m,n}}$. It is worth mentioning that, with the above operations, the total cost budget for mobile device m still holds, because of the decreased prices from the remote cloud to the edge server and from the edge server to the mobile device. The detail of the SEG algorithm is described in Algorithm 1. In the following, we specify the task graphs considered in this paper, and then prove the approximation ratio of the proposed greedy algorithm.

[2]For example, mobile users have to pay by month or year renting the remote cloud computing service from service providers while pay less amount of money for the service of an idle edge server within its local community.

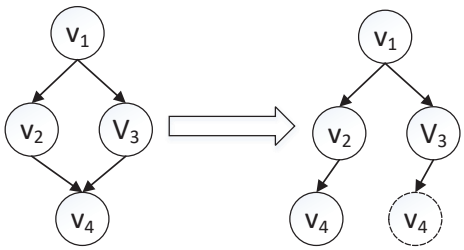


Fig. 2. Example of converting a directed acyclic graph to a tree shaped graph.

Definition 3: Tree shaped graph: A tree shaped graph of an application in this paper is a directed graph, in which every node in the tree has only one parent node except for the root, which is the task starting node (or the entry) in an application task graph. Each node in the graph is linked to the root. If the current node has more than two parent nodes, then we can replicate the current node and place each of them to the parent node. As shown in Fig. 2, node v_1 represents the starting sub-task and node v_4 is the task termination node (or the exit). By replicating node v_4, who has two parent nodes, we can obtain a tree shaped graph.

Therefore, based on Definition 3, we have Theorem 1. Theorem 1 utilizes the above defined tree shaped graph to derive the approximation ratio. The details are as follows.

Theorem 1: Given any general acyclic task graph with one start node and one end node, it can be transformed into a tree shaped graph with task dependency [29]. Let ϵ be an arbitrarily small positive constant and h be the longest branch of the tree shaped tasks. By setting $\delta = \frac{\epsilon T_{\max}}{h} > 0$, for a particular application, the greedy algorithm on computation offloading takes time of $O(N^2 + N\frac{h^2\delta}{\epsilon})$ and achieves the $(1+\epsilon)$ approximation ratio.

Proof: Let $T_{\max}$ be the maximum task finishing time of a whole application consisting of multiple sub-tasks. Given a sub-task i and its workload $\bar{\omega}_i$, if it is executed on the jth device (where $j = 0$ means the local execution, $j = 1$ means the remote cloud execution and $j \in [2, P+1]$ is for the edge execution), the execution delay can be $\Gamma_{ij} = \xi\frac{\bar{\omega}_i}{\bar{f}_j} + \eta$ following the expressions of (14), (18) and (19). The term $\bar{f}_j$ is CPU frequency of device j, ξ and η are two constant numbers. The largest delay is determined by the slowest CPU frequency $\bar{f}^{\min}_j$ and sub-task with the most heavy workload $\bar{\omega}^{\max}_i$ as $T_{\max} = \xi\frac{\bar{\omega}^{\max}_i}{\bar{f}^{\min}_j} + \eta$. Therefore, the upper bound of the maximum task finishing delay can be expressed as $hT^{\max}$. The adjusting steps take no more than $O(N)$ time to meet utility constraints of the edge server and the remote cloud. Since there are N rounds of iterations, the total time complexity of the greedy algorithm is $O(N^2 + NhT_{\max})$. If we set $\delta = \frac{\epsilon T_{\max}}{h}$, we then have $T_{\max} = \frac{h\delta}{\epsilon}$. Thus, the total time complexity can be rewritten as $O(N^2 + N\frac{h^2\delta}{\epsilon})$, where ϵ is an arbitrarily small positive constant.

Assume the optimal task finishing time is $L_{opt}(x^*)$, where the corresponding optimal strategy is denoted by x^*. Let $\hat{x}$ be

the minimum task finishing time obtained by executing Algorithm 1. We then have $L(\hat{x}) \leq L_{opt}(x^*)$. For a tree-shaped task graph with height h, at most N times of adjustments happen because of the sub-procedures from lines $24 \sim 32$ and lines $33 \sim 42$ of Algorithm 1. Following the equations defined in (14), (18) and (19), for those sequential adjustments defined in Algorithm 1, each delay is bounded by δ. Since the height of the tree is h, the total sequential adjustments is bounded by $h\delta$. Hence, we have

$$L(\hat{x}) \leq L_{opt}(x^*) \leq L_{opt}(x^*) + h\delta. \tag{31}$$

Let $T_{\min} = \xi \frac{\bar{\omega}_i^{\max}}{\bar{f}_j^{\max}} + \eta$, which means the minimum task finishing time. It is defined as the ratio between the most intensive workload executed on the most efficient server. Similarly, we have $T_{\max} = \xi \frac{\bar{\omega}_i^{max}}{\bar{f}_j^{\min}} + \eta$. The result of optimal strategy $L_{opt}(x^*)$ is at least equal to $T_{\min}$, then we have $L_{opt}(x^*) \geq T_{\min}$. Following (31), we have

$$\begin{aligned} L(\hat{x}) &\leq L_{opt}(x^*) + h\delta = L_{opt}(x^*) + \epsilon T_{\max} \\ &= L_{opt}(x^*) + \epsilon \left(\xi \frac{\frac{T_{\min}-\eta}{\xi} \bar{f}_j^{\max}}{\bar{f}_j^{\min}} + \eta \right) \\ &\leq L_{opt}(x^*) + \epsilon \left(\xi \frac{\frac{L_{opt}(x^*)-\eta}{\xi} \bar{f}_j^{\max}}{\bar{f}_j^{\min}} + \eta \right) \\ &\leq \left(1 + \epsilon \frac{\bar{f}_j^{\max}}{\bar{f}_j^{\min}} \right) L_{opt}(x^*) \\ &\overset{\epsilon'=\xi'\epsilon}{=} (1+\epsilon') L_{opt}(x^*). \end{aligned} \tag{32}$$

That is because in the multi-access mobile edge computing architecture, $\frac{\bar{f}_j^{\max}}{\bar{f}_j^{\min}}$ is bounded by a positive constant ξ'. Let $\epsilon' = \xi'\epsilon$ and we obtain $L(\hat{x}) \leq (1+\epsilon') L_{opt}(x^*)$, which concludes the proof. ■

B. Analysis and Algorithm of OPT-2

Different from the SEP problem, in MEP problem, sub-tasks of one user may be executed by multiple edge servers. It implies that **OPT-2** is more challenging to be solved. We first analyze the hardness of **OPT-2** via the reduction of the original problem to the multiple-choice Knapsack problem. Because of the unique budget constraints, we then prove that there is no polynomial time approximation algorithm for the MEP sub-problem by contradiction. Finally, we propose an efficient greedy algorithm.

1) Hardness of OPT-2: We first give the definition of Multidimensional Multiple-choice Knapsack Problem (MMKP) before proving that the MEP problem is NP-hard.

Definition 4: Multidimensional Multiple-choice Knapsack Problem (MMKP). Given B classes of resources denoted by $\{K_1, K_2, \cdots, K_B\}$. There are M groups, each with n_i items, where $i \in \{1, 2, \cdots, M\}$. Each item j, $\forall j \in \{1, 2, \cdots, n_i\}$ has a particular value u_{ij} and requires B distinct resources represented by vector $(r_{ij}^1, r_{ij}^2, \cdot, r_{ij}^B)$. The objective of MMKP is to maximize the total value of the collected items subject to the given resource constraints. It can be formulated as:

$$\begin{aligned} \text{Maximize}: \quad & \sum_{i=1}^{M} \sum_{j=1}^{n_i} u_{ij} x_{ij} \\ s.t. \quad & \sum_{i=1}^{M} \sum_{j=1}^{n_i} r_{ij}^k x_{ij} \leq C^k, \forall k \in \{1, 2, \cdots, B\}, \\ & \sum_{j=1}^{n_i} x_{ij} = 1, \forall i \in \{1, 2, \cdots, M\}, \\ & x_{ij} \in \{0, 1\}, \forall j \in \{1, 2, \cdots, n_i\}, \end{aligned} \tag{33}$$

where the first constraint is the resource constraint of the kth class resource. The second constraint in (33) means that only one item can be chosen from one group and the third constraint is the binary constraint.

We then prove the hardness of the MEP problem.

Theorem 2: The MEP problem is NP-hard.

Proof: Given any offloading strategy $\mathfrak{S}$, we verify whether it is a feasible solution by calculating utility and budget constraints in polynomial time. Consequently, MEP $\in$ NP. We then prove the NP-hardness of MEP by showing that the MMKP problem which is an NP-hard problem is a special case of the MEP problem.

Consider for a particular case when MEP has pipeline work flows. The $p+2$ kinds of devices in set $\mathcal{K}$ of MEP problem (See definition in subsection V-B) correspond to B classes of resources in MMKP. Sub-tasks in one application of a mobile device of MEP problem correspond to n_i items in each group of MMKP. Noting that mobile devices in MEP problem have limited cost budgets ϵ_m, each mobile device can only afford the limited number of computation resources from the computation devices (local mobile devices, edge servers and the remote cloud server) and the edge server can only serve the limited number of workloads from mobile devices as shown in constraint $C12$. Hence, it is equivalent to the case when there are limited resources in MMKP, as shown in the first constraint of (33). Since one item or one sub-task is chosen per class or application, both the MMKP and MEP problems are the same under this setting. The utility u_{ij} represents delay in MEP. Maximizing total utilities is equivalent to minimizing total delay. Since the special case is NP-hard, the original MEP problem is also NP-hard. ■

Theorem 3: MEP does not have a polynomial time approximation algorithm of any constant approximation ratio unless $P = NP$.

Proof: We show that the MEP sub-problem has no polynomial time approximation scheme following [30]. Given ϵ, which is a positive constant, there is an approximation algorithm $\mathfrak{L}$ to an instance x_{MEP} of the MEP problem with an approximation ratio less than $1+\epsilon$, that is $\frac{\mathfrak{L}(x_{MEP})}{OPT(x_{MEP})} = \frac{\mathfrak{L}(\hat{x})}{L_{opt}(x^*)} \leq 1+\epsilon$. For an instance x_{MMKP} of MMKP consisting of B classes, M groups with n_i ($n_i > 1, i = 1, 2, \cdots, N$) items in the ith group. For a particular class K, $K \in \{1, 2, \cdots, B\}$, the service capacity is set as C, i.e., the maximum available workload and resources for handling tasks. Given the MMKP instance x_{MMKP}, we can always construct

an instance of x_{MEP} by creating a network topology of B classes corresponding to B kinds of servers for mobile device m. For each sub-task n on mobile device m, we set the processing power of each server to be identical as P_0. For each sub-task n from mobile device m, the execution time is $\frac{w_{m,n}}{f_{m,n}}$, we set $f_{m,n}^{\max} = \max\{f_q^c, f_p^e, f_m^l\}$. We denote the minimum transmission delay between two distinct network entities as $T'^{\min}_{m,n} = \min\{TF^e_{m,n}, TF^c_{m,n}, E^{I_{st}}_{m,n}/P_0\}$. We first construct a constant factor $k = \frac{C}{Nw^{\max}_{m,n}}$. For each one of servers except for the mobile devices, without loss of generality, one of the running cost per second can be expressed as $C_S = k \cdot w_{m,n}/\{\frac{w_{m,n}}{f^{\max}_{m,n}} + T'^{\min}_{m,n}\}$. Since $\mathfrak{L}$ is an approximation algorithm of x_{MEP}, the total execution time $T_{\mathfrak{L}} \leq (1+\epsilon)T_{OPT}$, where T_{OPT} denotes the optimal total execution delay. When we have a schedule $\mathfrak{S}$, we can calculate total delay and costs as $T_{total} = \sum_{m=1}^{M}\sum_{n=1}^{N}\{\frac{w_{m,n}}{f_{m,n}} + T'^{\min}_{m,n}\} \geq \sum_{m=1}^{M}\sum_{n=1}^{N}\{\frac{w_{m,n}}{f^{\max}_{m,n}} + T'^{\min}_{m,n}\}$, $C_{total} = \sum_{m=1}^{M}\sum_{n=1}^{N} C_s\{\frac{w_{m,n}}{f_{m,n}} + T'_{m,n}\} \leq \sum_{m=1}^{M}\sum_{n=1}^{N} C_s\{\frac{w^{\max}_{m,n}}{f_{m,n}} + T'^{\min}_{m,n}\} \leq C$. Let $\sum_{m=1}^{M}\sum_{n=1}^{N}\{\frac{w_{m,n}}{f^{\max}_{m,n}} + T'^{\min}_{m,n}\}$ be $T_{\mathfrak{L}}$. When T_{total} achieves $T_{\mathfrak{L}}$, it is the optimal solution of x_{MEP} for this case, which corresponds to the optimal solution of x_{MMKP} where for class K, the item with $f^{\max}_{m,n}$ in each class is selected if the knapsack capacity is no less than C. The approximation algorithm $\mathfrak{L}$ of x_{MEP} with $\epsilon = \epsilon_{\mathfrak{L}}$ only returns a solution that corresponds to the optimal solution of x_{MMKP}, because for any other solutions would result in $T_{total} > (1+\epsilon)T_{\mathfrak{L}}$. Therefore, we only have to consider the following cases:

- Case i): If $\frac{w_{m,n}}{Nw^{\max}_{m,n}} \leq \frac{\frac{w_{m,n}}{f^{\max}_{m,n}} + T'^{\max}_{m,n}}{\frac{w_{m,n}}{f_{m,n}} + T'_{m,n}}$, the $\mathfrak{L}$ scheme can always achieve the constraint C.
- Case ii): If $\frac{w_{m,n}}{Nw^{\max}_{m,n}} > \frac{\frac{w_{m,n}}{f^{\max}_{m,n}} + T'^{\min}_{m,n}}{\frac{w_{m,n}}{f_{m,n}} + T'_{m,n}}$, the decision cannot achieve the constraint C.

In the above cases, if x_{MMKP} has a solution, we can solve x_{MMKP} in polynomial time by applying $\mathfrak{L}$ to x_{MEP}, which is not possible unless $P = NP$. Therefore, there is no polynomial time approximation algorithm for MEP, which concludes this proof. ■

2) Algorithm for MEP sub-problem: Based on the above analysis, to solve the MEP sub-problem, we propose a Greedy algorithm for Collaborative edge Computing (GCC) as shown in Algorithm 2. The GCC algorithm is similar to the greedy algorithm as shown in Algorithm 1 while it mainly differs from Algorithm 1 from lines $15 \sim 21$ as shown in Algorithm 2. For processing each sub-task of each users, if the service price of neighboring edge servers is less than the original edge server and the computation capability constraint $C12$ is met, then the sub-task is transmitted from the original edge server to the corresponding neighboring edge server so as to achieve the minimal delay among all edge servers.

Algorithm 2 GCC Algorithm for MEP problem

Input: $\mathcal{N}$: a sequence obtained by the depth first traversal of N sub-tasks on mobile device m;
ϵ_m: the sum budget for mobile device m.
Output: Offloading policy $\mathbb{X}$

1: **begin**
2: Initialize: $w_{m,n}$, $d_{m,n}$, $U_p^e \leftarrow 0$, FIFO queues at edge servers;
3: **for** $n = 1$ to N **do**
4: Calculate $R_{m,n,p}$, $\tau^l_{m,n}$, $E^l_{m,n}$ by (1)-(3), respectively for each mobile user m at the nearest access point;
5: **if** $pre(n) = \varnothing$ **then**
6: $TR^l_{m,n} \leftarrow 0, TR^e_{m,n} \leftarrow 0, TR^c_{m,n} \leftarrow 0$
7: **else**
8: Compute $TF^t_{m,n}, TF^r_{m,n}$ by (17), (20) respectively;
9: Compute $TR^l_{m,n}, TR^e_{m,n}, TR^c_{m,n}$ by (13), (15), (16), respectively;
10: **end if**
11: Compute $TF^l_{m,n}, TF^e_{m,n}, TF^c_{m,n}$ by (14), (18) and (19), respectively;
12: **if** $\frac{P^c_q d_{m,n}}{E^c_{m,n}} \geq 1 \& TF^c_{m,n} < TF^e_{m,n}$ **then**
13: $x^e_{m,n,p} \leftarrow 0$, $x^c_{m,n,q} \leftarrow 1$;
14: **else**
15: **for** $m = 1$ to M **do**
16: $x^e_{m,n,p} \leftarrow 1$, $x^c_{m,n,q} \leftarrow 0$;
17: **if** $P^e_t < P^e_p$ where $p \neq t$ and for the tth edge server, the workload constraint $C12$ of W_t in (30) is met **then**
18: Set the current workload $w_{m,n}$ on the pth edge server to the tth edge server if the incurred delay is minimal among all edge servers excluding the pth edge server
19: $x^e_{m,n,p} \leftarrow 0$, $x^e_{m,n,t} \leftarrow 1$;
20: **end if**
21: **end for**
22: **end if**
23: Compute $E^e_{m,n}, E^c_{m,n}$ by (5), (6), respectively;
24: Compute $C'_{m,n}$ by (28);
25: **end for**
26: **while** $\sum_{n=1}^{N} C'_{m,n} > \epsilon_m$ **do**
27: **if** $\sum_{n=1}^{N} x^c_{m,n,q} \neq 0$ & $C12$ holds **then**
28: $y \leftarrow \underset{n}{arg\min}\{E^c_{m,n}\}$;
29: $x^e_{m,y,p} \leftarrow 1, x^c_{m,y,q} \leftarrow 0$;
30: **else**
31: Look for the edge server with the minimum cost to offload;
32: **if** Cannot find == true **then**
33: $z \leftarrow \underset{n}{arg\min}\{E^e_{m,n}\}$;
34: $x^e_{m,z,p} \leftarrow 0, x^c_{m,z,q} \leftarrow 0$;
35: **end if**
36: **end if**
37: **end while**
38: **while** $U'^e_p < 0$ **do**
39: **if** $\sum_{n=1}^{N} x^c_{m,n,q} \neq 0 \& \{\frac{E^s_{m,n}}{E^e_{m,n}} > 1\} \neq \varnothing$ & $(C12)$ holds **then**
40: $u \leftarrow \underset{n}{arg\max}\{\frac{E^s_{m,n}}{E^e_{m,n}}\}$;
41: $x^e_{m,u,p} = 1, x^c_{m,u,q} = 0$;
42: **else**
43: $v \leftarrow \underset{n}{arg\min}\{\frac{P^e_p d_{m,n}}{E^e_{m,n}}\}$;
44: $x^e_{m,v,p} \leftarrow 0, x^c_{m,v,q} \leftarrow 0$;
45: **end if**

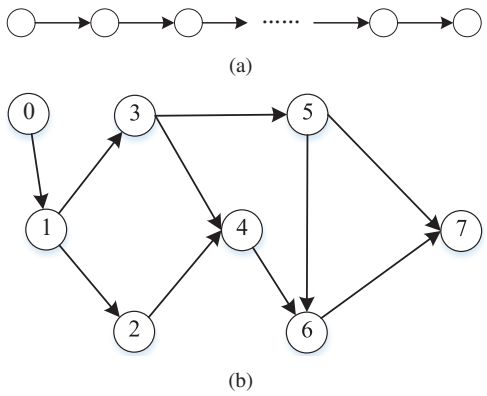


Fig. 3. (a) Sequential task graph used for the simulation of SEP; (b) General task graph used for simulation.

46: Compute U'^{e}_{p} by (29);
47: **end while**
48: **end**

Theorem 4: Let h be the longest branch of the task tree. The time complexity of Algorithm GCC is of $O(N^2PM + NhT_{\max})$, where $T_{\max}$ is the maximum computation delay.

Proof: Since Algorithm 2 is similar to Algorithm 1, we focus on the analysis on lines 15 $\sim$ 21 in Algorithm 2. There are P edge servers in the network. For each application, calculating the incurred delay is $O(N)$ at most, thus the time complexity of lines 17 $\sim$ 18 is $O(PN)$. The inner 'for' circulation (See line 15 of Algorithm 2) takes $O(M)$ rounds for M mobile users. As a result, the total complexity of lines 15 $\sim$ 21 is $O(PNM)$, which is additional cost compared to Algorithm 1. Therefore, the first 'for' circulation takes $O(N(PNM)) = O(N^2PM)$. Similarly to the proof of Theorem 1, the remained time complexity is $O(NhT_{\max})$. The obtained total time complexity is $O(N^2PM + NhT_{\max})$, where $T_{\max}$ is the maximum computation delay as defined in the proof to Theorem 1. ■

VII. Performance Evaluation

In this section, we firstly present the performance evaluation of greedy algorithm to the SEP sub-problem, then evaluate the solution to the MEP sub-problem. We also investigate the sensitivity of parameters.

A. Evaluation for the SEP Problem

1) Settings for SEP: We design experimental studies to evaluate the proposed algorithms. The results are averaged for $1,000$ time executions. We implement all the algorithms on the CloudSim simulator [31], which is a discrete event simulator [32]. By default, we let the noise variance between the edge and mobile device be $\sigma^2 = 1$ [4] and the channel bandwidth be $W = 5$MHz. The transmission rate between the edge server and remote cloud is set as $\omega = 10$Mbps. The CPU frequency of the edge server is set as $f_p^e = 3.6 \times 10^9$Hz [33]. The CPU frequency of the remote cloud server is $f_q^c = 3.6 \times 10^{10}$Hz [34] and the CPU frequency of the mobile device is $f_m^l = 1.0 \times 10^9$Hz [35]. The amount of power for transmission between edge and cloud is $P_0 = 0.1$W [36]. The charging price of edge server is $P_p^e = 0.001$ and the charging price of the remote cloud is $P_q^c = 0.004$. Key system parameters are set as $\alpha_e = 0.5$, $\beta_e = 0.4$, $\alpha_c = 0.6$, $\beta_c = 0.6$ [25].

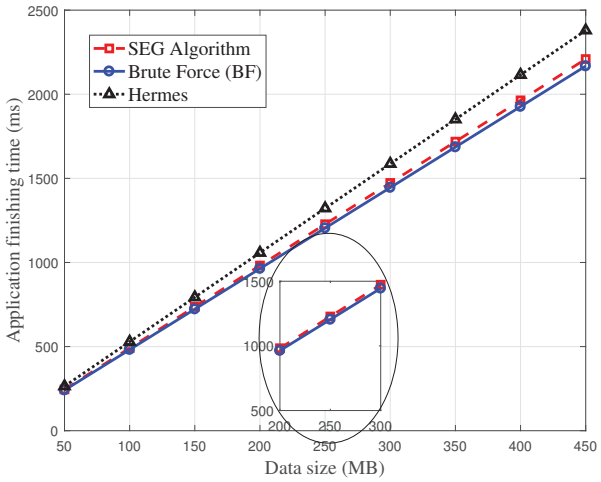


Fig. 4. The application finishing time with the increase of data size for sequential topology

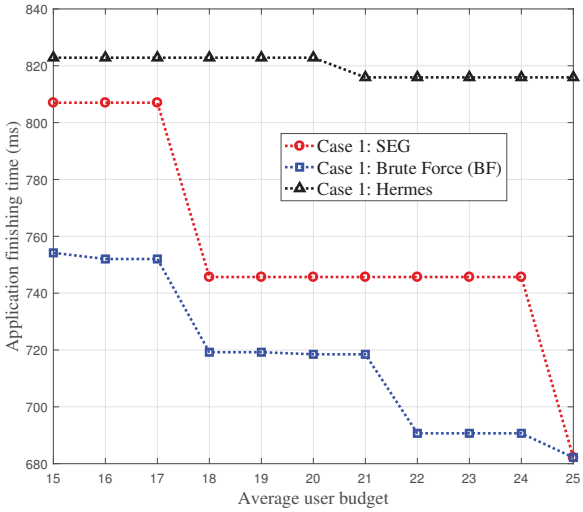


Fig. 5. Application finishing time with average user budget for general task graph when there is no special relationship between workload and data size.

First, we conduct a preliminary investigation for the performance of the proposed greedy algorithm with different workloads and data sizes with a sequential task dependency graph [37] shown in Fig. 3(a). By default, in an application, there are 8 sub-tasks with average data sizes [8.80, 3.70, 9.26, 3.24, 7.41, 6.48, 6.02, 5.09] MB [38] to be processed. The workload is proportional to the data size [39] by multiply the data size vector by 1.2. To vary the data sizes, we multiply the default data size vector by each of the numbers in the vector index, where index $= [1, 2, \cdots, 9]$. For multiple executions, we also vary both the workload and data size

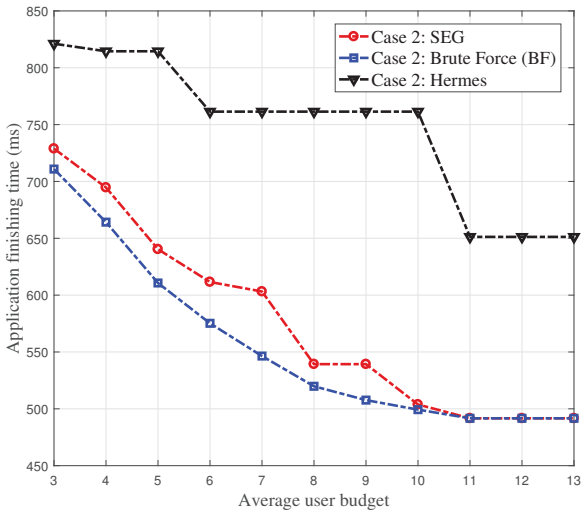


Fig. 6. Application finishing time with average user budget for general task graph when workload is in proportion to data size.

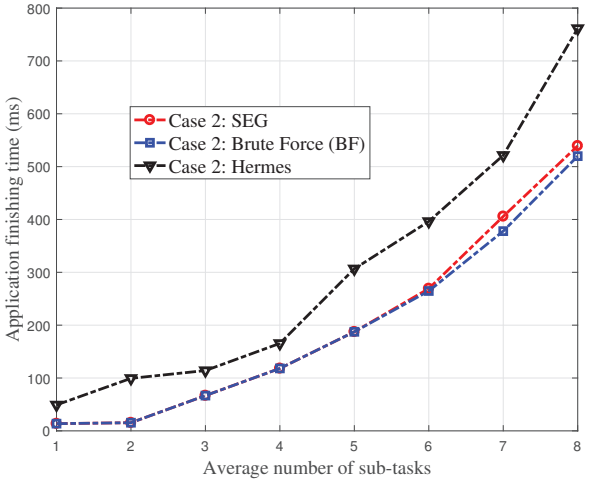


Fig. 8. Application finishing time with average number of sub-tasks in an application for case 2.

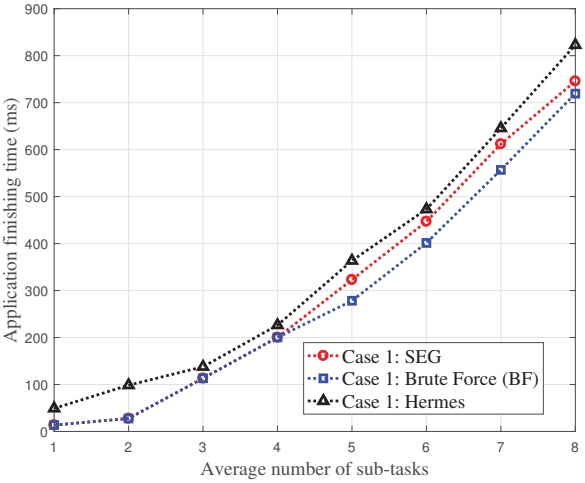


Fig. 7. Application finishing time with average number of sub-tasks in an application for case 1.

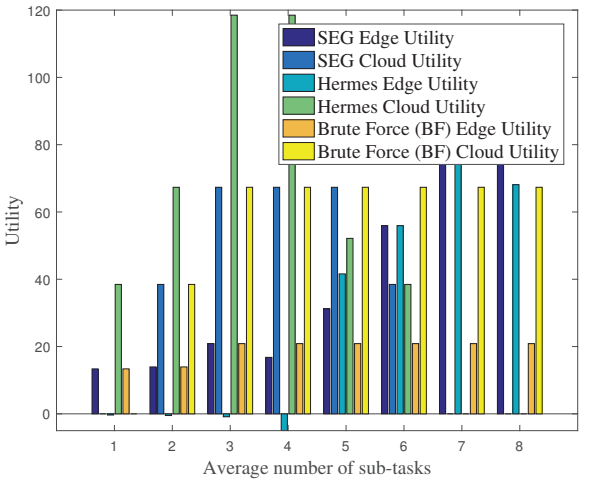


Fig. 9. Average utilities of edge/cloud servers with average number of sub-tasks for case 1.

vectors for each simulation instance by randomly adding a small positive or negative constant within $[-10, 10]$. For each user m, the budget is set as ϵ_m=$\sum$(data sizes of sub-task in an application)$\times 0.1$.

Then, to better compare the performance of algorithms, we also use a general task graph [40] for simulation shown in Fig. 3(b). By default, there are 8 sub-tasks in one application. Node 0 represents the starting sub-task while node 7 is the ending sub-task. For each internal sub-task, there are at most 2 incoming edges, 2 out-coming edges. Similar to [41], for each application, one internal node in the task graph is connected to its neighbor node with a probability of 0.3. The workload of each sub-task ranges in $[0, 50]$ with mean value of 33.1. For example, the workload vector of one application is $[49.10, 1.15, 35.26, 38.35, 34.19, 35.26, 41.10, 30.17]$ MB and the data size is [35.19, 14.81, 37.04, 12.96, 29.63, 25.93, 24.07, 20.37] MB. For comparing purpose, we compare proposed algorithms with Hermes mentioned in [14], which is an algorithm based on dynamic programming where there is no remote cloud. Considering the fact that in the original Hermes algorithm, there is no cooperation between edge server and the cloud server, we have modified the Hermes algorithm according to the following operations. We regard the remote cloud server as a special edge server. Then the original Hermes algorithm possesses the ability to handle both edge and the cloud servers. To be detail, we change the sub-index j's range of the variable $C[i, j, t]$ mentioned in Hermes from $1 \leq j \leq M$ to $1 \leq j \leq M + 1$. Where M represents the total number of edge servers and the additional $+1$ represents the cloud server and we adopt the same parameters to SEG algorithm. The brute force (BF) algorithm acts as the benchmark to derive the optimal solutions. In the following paragraphs, we use the terms modified Hermes and Hermes

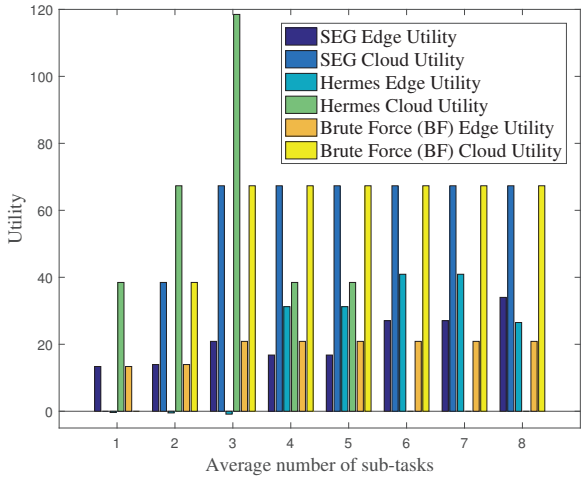


Fig. 10. Average utilities of edge/cloud servers with average number of sub-tasks for case 2.

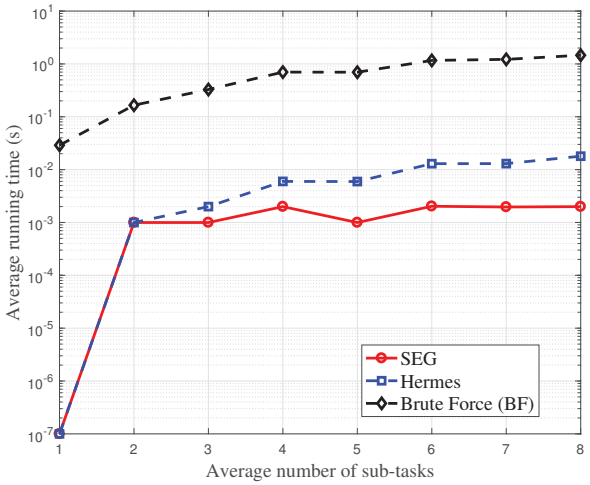


Fig. 12. Average running time with average number of sub-tasks for case 2.

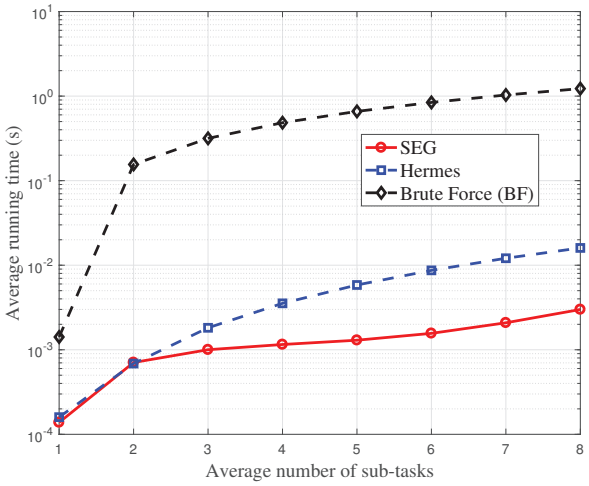


Fig. 11. Average running time with average number of sub-tasks for case 1.

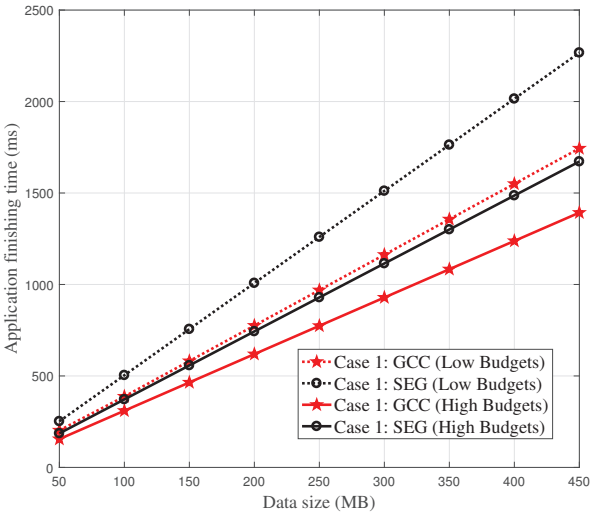


Fig. 13. Application finishing time with average data size for case 1 of MEP problem.

interchangeably.

2) Sequential task graph: As shown in Fig. 4, the application finishing time of all the algorithms shows linear growth when data size increases. On average, the SEG algorithm is more than 2% slower than the BF method while the modified Hermes algorithm is more than 10% slower than the BF method on the application finishing time. Therefore, the SEG algorithm is close to the optimal results and has achieved 7% performance gain over Hermes.

3) General task dependency graph: Fig. 5 and Fig. 6 show the relationships between average application finishing time and average user budget. There are two cases. **Case 1**: The default setting, where there is no special relationship between workload and data size. **Case 2**: The data size is in proportional to the workload and $\mathsf{data\ size} = \mathsf{workload} \times 0.1$. We can see that for all the cases and all the algorithms, the application finishing time decreases with the increase of average user budget. When the average user budget is constant, the SEG algorithm outperforms Hermes by about 8% on average for case 1 and the maximum performance gain is over 16.4%. For case 2, the SEG algorithm is about 23% better than Hermes on the application finishing time averagely with given budgets. That's because the SEG algorithm tends to select more sub-tasks to be executed on the remote cloud, which can reduce the execution time.

Fig. 7 and Fig. 8 illustrate the application finishing time with average number of sub-tasks in an application. It is observed that the application finishing time of both cases grow exponentially with the increasing of average number of sub-tasks. This is because when there are more sub-tasks in an application, following the computation model, it will take more time for the corresponding edge servers and remote cloud to process the sub-tasks, which results in the growth of both application finishing time. On average, the SEG algorithm is

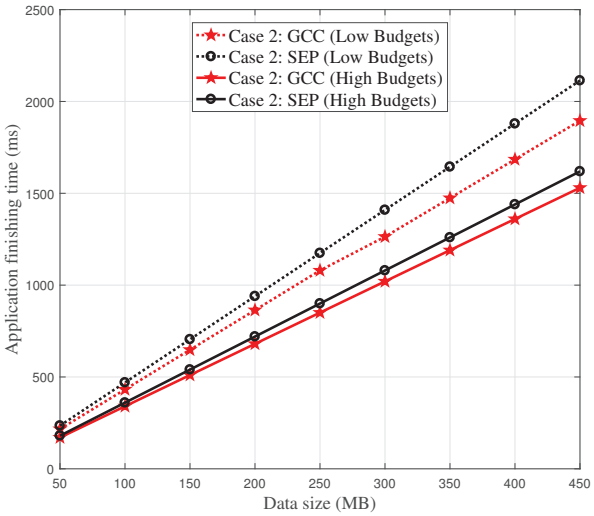


Fig. 14. Application finishing time with average data size for case 2 of MEP problem.

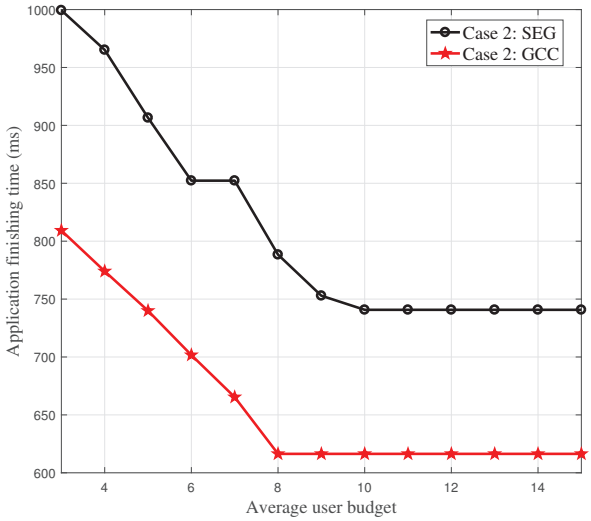


Fig. 16. Application finishing time with average user budget for case 2 of MEP problem.

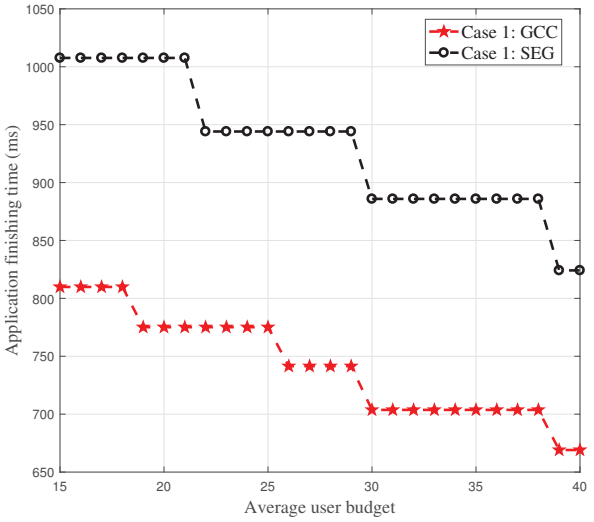


Fig. 15. Application finishing time with average user budget for case 1 of MEP problem.

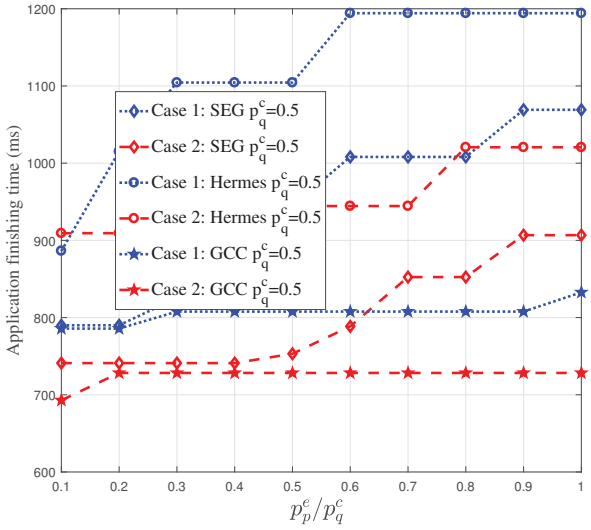


Fig. 17. Application finishing time with p_p^e/p_q^c.

25.6% more efficient than Hermes on the application finishing time given average number of sub-tasks for case 1. For case 2, the SEG algorithm saves 43.6% time over Hermes. Of all the cases, the proposed SEG algorithm is close to the BF method on the application finishing time.

Fig. 9 and Fig. 10 depict the average utilities of edge and cloud servers for the two cases. Clearly, we can observe that the proposed SEG algorithm and the BF benchmark algorithm can guarantee non-negative utilities of both edge and the cloud servers. In other words, edge and cloud servers are individual rational [18] in the proposed algorithms. In addition, SEG algorithm and BF algorithm can achieve similar average utilities at the server side. As shown in Figs. 11 and 12, the running time of all three algorithms grows with the average number of sub-tasks for both cases. Meanwhile, both SEG and the modified Hermes algorithms outperform the BF benchmark algorithm. On average, SEG is about 55.2% and 57.3% more efficient than the modified Hermes algorithm for both case 1 and case 2, which comply with the theoretical analysis on time complexity of the two algorithms.

B. *Evaluation for the MEP Problem*

1) Settings for MEP: To evaluate MEP, we set the task graph of mobile users as the directed acyclic graph as shown in Fig. 3(b). By default, we assume each sub-task has only two descendant sub-tasks for the benefit of presentation whereas our scheme is adapt to the case when there are more than two descendant sub-tasks for one sub-task.

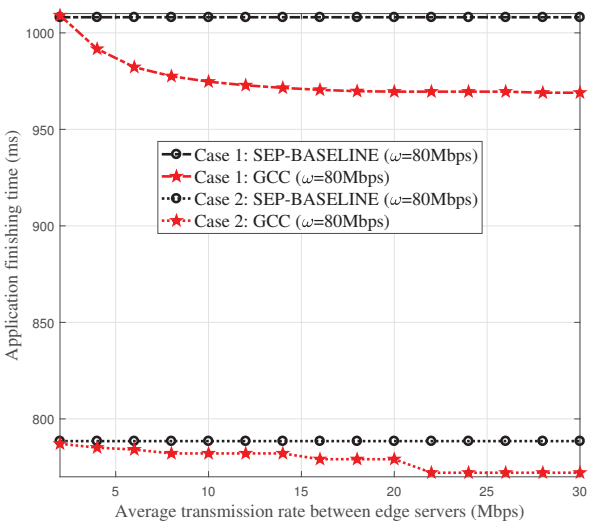


Fig. 18. Application finishing time with average transmission rate between edge servers.

The maximum depth (the longest hops) of the application task graph is set as 5. Moreover, we also compare GCC with the greedy algorithm SEG mentioned in Algorithm 1, where there are no collaborations between edge servers. In this simulation, we have four edge servers and one remote cloud center. For edge server p, its maximum workload W_p is set $W_p = 1000$, the charging price rages in $[0.001, 0.003]$. The additional delay $l(s,t)$ due to the inter transmission between edge servers is set as 0.1ms. In each small cell, there are 20 to 40 mobile devices and each of them has one splittable application to be executed. Similar to the workload settings mentioned in subsection VII-A1, there are also two cases. For case 1, there are no special relations between workloads and data sizes. The default workload vector is [12.28, 0.29, 8.82, 9.59, 8.55, 8.82, 10.28, 7.54] MB and the default data size vector is [8.79, 3.70, 9.26, 3.24, 7.41, 6.48, 6.02, 5.09] MB. To vary the workloads and data sizes, we multiply each one of the vectors by positive variables range from 1 to 9. Moreover, considering the budget influence, we further divide case 1 into two sub-cases, *i.e.*, the low budgets sub-case and the high budgets sub-case. For low budgets sub-case, the budget for each user m is set as ϵ_m=$\sum$(data size of sub-tasks in an application)$\times 0.1$ while for high budgets sub-case, ϵ_m=$\sum$(data size of sub-tasks in an application)$\times 0.5$. For high budget sub-case, the budgets are 5 times more than the low budgets sub-case. For case 2, the default workload is linear to the data size, that is, workload=data size $\times 1.2$ and the workload vector is [10.56, 4.44, 11.11, 3.89, 8.89, 7.78, 7.22, 6.11]MB. Just the same as case 1, there also exists two sub-cases considering budgets. For the benefit of demonstration, we assume each edge server can collaborate with its two adjacent neighbor edge servers with adequate resources since cooperation with edge servers too many hops away will result in longer internal transmission delays between edge servers, thereby degrading the performance of design goals.

2) Performance Evaluation for MEP: In Fig. 13 and Fig. 14, we compare application finishing time with the average data size for two cases. For case 1, the application finishing time has shown a linear growth with the increasing number of average data sizes. The proposed GCC algorithm performs the best in terms of the average application finishing time whenever budgets are low or high. In the low budgets sub-case, the proposed GCC algorithm is 22.7% more efficient than the SEG algorithm on the application finishing time. This is because when budgets are low, GCC will employ more edge servers than SEG, which will improve the efficiency. In the high budgets sub-case, GCC reduces the average application finishing time of SEG by 16.7%. The reason lies in the fact that, when there are abundant budgets, SEG will put more works on the remote cloud which may cause longer transmission delays than GCC. For case 2, GCC achieves almost the same performance on the average application finishing time as case 1 shown in Fig. 13. With collaboration between edge servers, the average application finishing time can be further reduced by GCC for at least 5.5% when comparing it with SEG algorithm. The details are shown in Fig. 14.

Fig. 15 and Fig. 16 demonstrate the changes of application finishing time with the variation of average user budget by re-defining case 1 and case 2 as follows. Case 1: the default workload vector is [49.11, 1.15, 35.26, 38.35, 34.19, 35.27, 41.11, 30.17]MB and the default data size vector is [35.19, 14.81, 37.04, 12.96, 29.63, 25.93, 24.07, 20.37]MB. We then vary the two vectors for each simulation instance by randomly adding a small positive or negative constant within $[-10, 10]$, and the mean value of data size for each application is 33.08 and the mean data size is 25.0. Case 2: workload vector=[49.11, 1.15, 35.26, 38.35, 34.19, 35.27, 41.11, 30.17]MB. The data size vector is obtained via workload $\times 0.1$. On average, GCC is superior to SEG on the application finishing time when average user budget changes for both cases. For both cases, the application finishing time decreases with the increase of average user budget. That is because when there are enough budgets, both the two proposed algorithms will offload more tasks to the remote cloud. For GCC, it will recruit more edge servers, thus the application finishing time of SEG can be further reduced. For case 1, GCC can reduce the average application finishing time by over 20.5% when comparing with SEG. For case 2, GCC saves more than 18.3% application completion time than SEG.

C. Sensitivity of Parameters

This subsection investigates the sensitivity of parameters, to show their influences on the optimization target, *i.e.*, the total application finishing time. As shown in Fig. 17, we fix the unit service price of the remote cloud server as $p_q^c = 0.5$ and adopt the settings for case 1 and case 2 mentioned in section VII-B2. It is observed that for both algorithms, with the growing ratio between unit price of the remote cloud and unit price of the edge server $\frac{p_p^c}{p_q^c}$, the application finishing time increases and converges to a constant. That's because when the unit price of the edge server become larger, more sub-tasks will be offloaded to the remote cloud if budget is

TABLE II
DISTANCE FROM ACCESS POINT, UP-LINK AND DOWN-LINK RATES AT EACH WAYPOINT OF PATHS

	Path V_1				Path V_2				Path V_3				Path V_4			
Distance (m)	7.5	7.2	7.5	4.5	4	3.3	4.8	5.5	7.59	3.5	3.5	5.5	5	5.7	3.8	2
Uplink rate (Mbps)	5.18	5.14	4.86	5.79	5.22	5.13	5.08	4.74	2.4	0.97	1.43	0.27	1.75	4.8	5.09	4.98
Downlink rate (Mbps)	6.43	5.3	5.79	7.59	15.6	15.34	11.84	9.84	0.07	0.09	0.51	0.03	0.09	4.47	7.59	7.2

sufficient at that time or otherwise, those sub-tasks will be executed on the local device. It can be directly drawn from Fig. 17 that, the application finishing time is less sensitive for the proposed GCC algorithm over the other two algorithms, whereas it is sensitive for Hermes. When the price ratio is constant, GCC can save more application completion time than Hermes by 23.9% for case 1 and 24.4% for case 2 averagely. Moreover, GCC is more efficient than the proposed SEG algorithm with 13.3% and 10.4% time reduction for case 1 and case 2 respectively.

Next, we investigate the influence of the transmission rate between edge servers. We set the transmission rate between the edge server and the remote cloud as $\omega = 80$Mbps. Then we vary the transmission rate between edge servers. As shown in Fig. 18, for all cases, the application finishing time for GCC is smaller than that of the SEG algorithm and decreases with the growth of transmission rate between edge servers. It can be concluded that, increase the transmission rate between edge servers can guarantee the high performance on the application finishing time.

D. Real-world Experiment

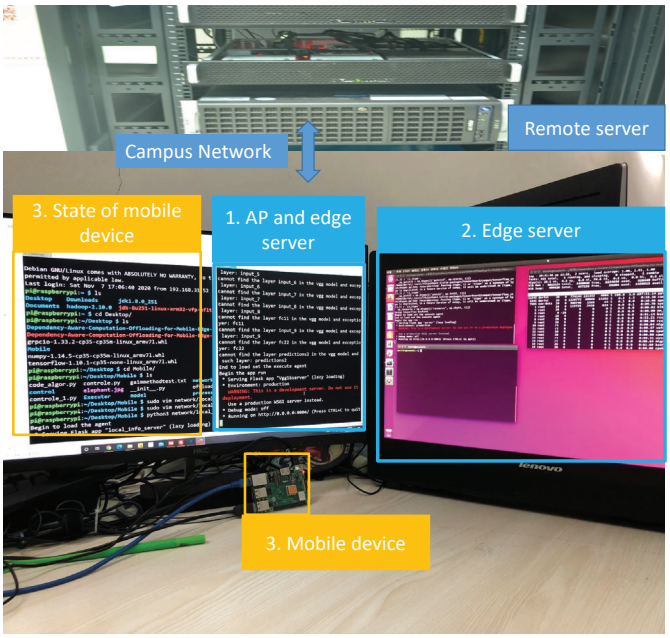


Fig. 19. Experimental test-bed setting

In the real-world experiment, we design and implement a radio frequency environment using heterogeneous WiFi access points with four edge servers at different corners of the laboratory. One of the edge servers with its access point and the remote server are depicted in Fig. 19. As shown in Fig. 20, the wireless routers are TP-LINK routers that can not only cover the whole lab, but also go through walls to reach the corridor. Four typical moving paths with different speeds $V_1 \sim V_4$ are set to obtain different channel gains and SINRs at different waypoints of the moving paths. Therefore, the fixed channel condition parameters can be measured at each of the waypoints. We let Android smartphone users move along each of the four paths and stop for a while at each waypoints, there are 20 mobile users in total.

For each route, the channel conditions are measured and listed in Table II. We average the finishing time obtained from each waypoint of a path. At each way point, there are 20 deep learning applications, whose task graph is shown as Fig. 21. Using ensemble learning technique, the application tries to classify the input images combining 16-layer and 19-layer VGG models [42]. We consider two different user budgets' setting, adequate budgets and limited budgets.

For Fig. 22, users have high budgets and are set as 100. It is observed that both SEG and GCC outperform the existing algorithm Hermes for all different paths, on the application finishing time. For example, for path V_1, the average application finishing time for GCC, SEG and Hermes are 9.58s, 11.01s, and 18.61s. On average, the proposed SEG algorithm can save 44.6% execution time than Hermes. Further, GCC can averagely enhance the performance of SEG by more than 12.7%. For the case when user budgets are limited, we vary user budgets from 20 to 60 with step size of 20 using path V_1. The results are depicted in Fig. 23. It is clear that the application finishing time decreases with the growth of user budgets for both SEG and GCC algorithms. That is because when budgets are adequate, more sub-tasks will be offloaded to the remote cloud server for execution, which reduces the total application finishing time. For example, when user budgets increase from 20 to 40, the application finishing time for GCC drops from 11.8s to 8.6s. However, for Hermes, the application finishing time shows no explicit relations with respect to user budgets, since budgets are not included in the Hermes algorithm. On average, the SEG algorithm is 42.1% more efficient than Hermes on the application finishing time. Moreover, GCC can further improve the performance of SEG by over 10% averagely on the average application finishing time.

VIII. CONCLUSIONS

This paper has proposed novel computation offloading schemes with device, edge and remote cloud collaboration. We have formulated two NP-hard application completion time minimization problems. We have firstly considered the typical case in a small cell where tasks from mobile devices can only

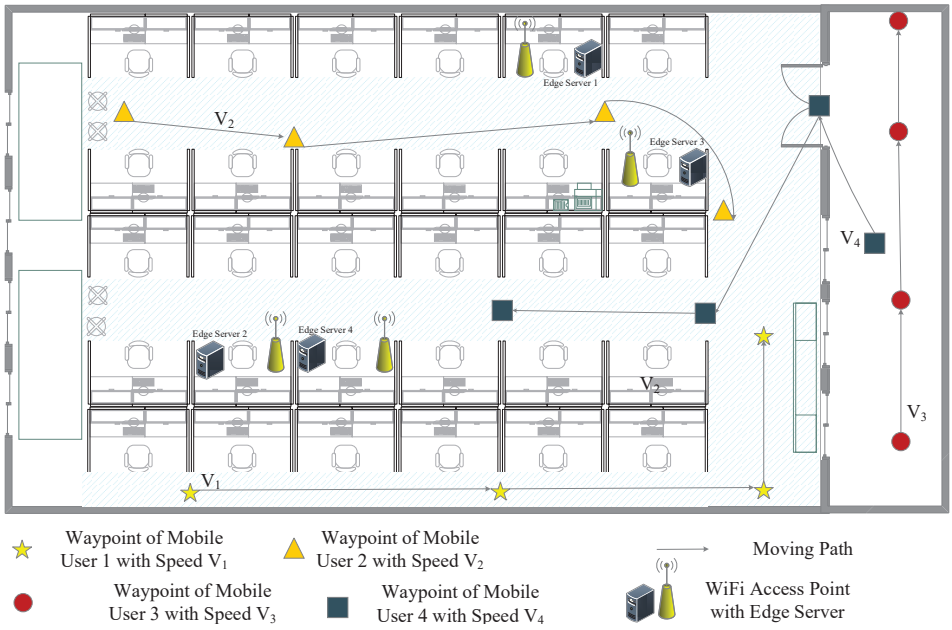


Fig. 20. Mobile trajectory of users.

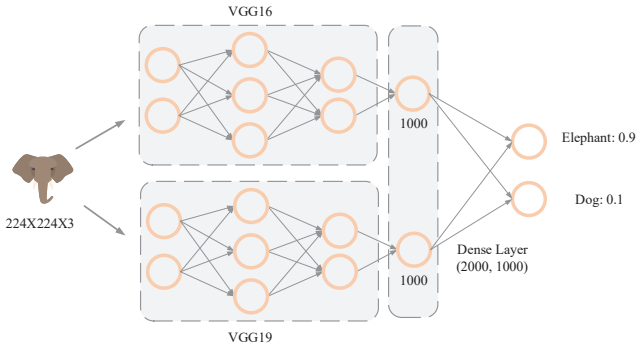


Fig. 21. Application used to label the input pictures.

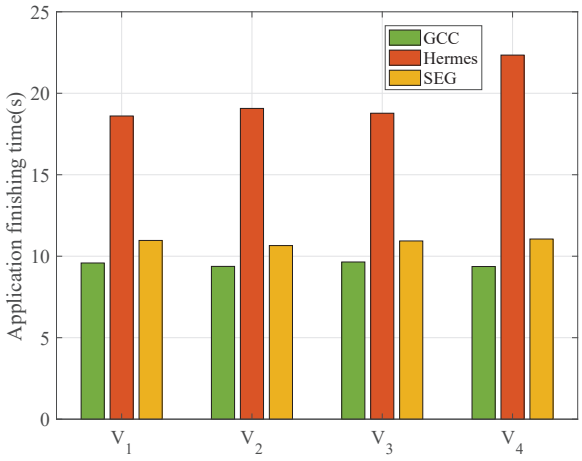


Fig. 22. Application finishing time for different moving paths.

be offloaded to edge servers at the small cell base station in their local vicinity or to the remote cloud. We then extend our model to the general case where edge nodes at different small cell base stations can cooperate with each other. For the first sub-problem, *i.e.* the SEP problem, we have proposed a greedy algorithm to obtain the sub-optimal offloading strategy on the minimal application finishing time. The sub-optimal offloading strategy has an approximation ratio of $1+\epsilon$. For the second sub-problem where edge nodes collaborate with each other, we have designed a greedy based collaborative edge computing algorithm to obtain the minimum application finishing time. We have conducted extensive simulations to verify the performance of our approaches. Simulation results have demonstrated the effectiveness of proposed algorithms. Real-world experiments comply with simulation results.

In the future, we plan to investigate the trade-off between user budgets and application completion time. We also plan to use coalitional game theory or auction theory to devise distributed algorithms.

ACKNOWLEDGMENT

This work was motivated when the first author was a visiting scholar at the department of electronic and computer engineering, the Hong Kong University of Science and Technology. This work was supported in part by the National Natural Science Foundation of China under Grant Nos. 61702115 and 62072118. It was supported by the Hong Kong Research Grants Council under Grant No. 16209418. It was also supported in part by Macao Science and Technology Development Fund under Grant No. 0026/2018/A1.

REFERENCES

[1] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—a key technology towards 5g," *ETSI White Paper*, vol. 11, no. 11, pp. 1–16, 2015.

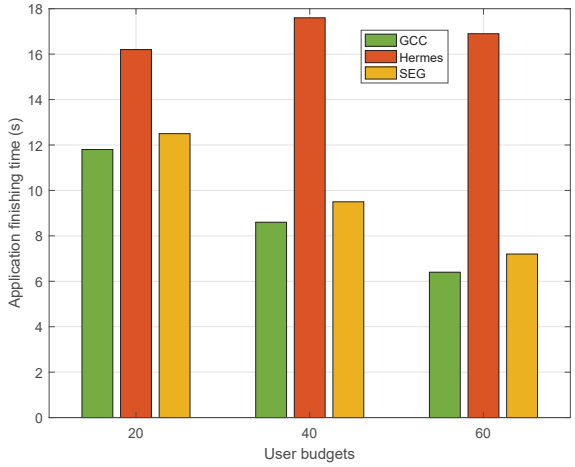


Fig. 23. Application finishing time with different user budgets for path V_1.

[2] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[3] J. Zhang and K. B. Letaief, "Mobile edge intelligence and computing for the internet of vehicles," *Proceedings of the IEEE*, vol. 108, no. 2, pp. 246–261, 2019.

[4] L. Chen, J. Wu, H.-N. Dai, and X. Huang, "Brains: Joint bandwidth-relay allocation in multi-homing cooperative d2d networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 6, pp. 5387–5398, 2018.

[5] W. Sun, J. Liu, and Y. Yue, "Ai-enhanced offloading in edge computing: When machine learning meets industrial iot," *IEEE Network*, vol. 33, no. 5, pp. 68–74, 2019.

[6] Z. Hao, S. Yi, and Q. Li, "Nomad: An efficient consensus approach for latency-sensitive edge-cloud applications," in *IEEE Conference on Computer Communications (INFOCOM)*, 2019, pp. 1–9.

[7] X. Wang, X. Chen, and W. Wu, "Towards truthful auction mechanisms for task assignment in mobile device clouds," in *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2017, pp. 1–9.

[8] C. You, K. Huang, H. Chae, and B. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 16, no. 3, pp. 1397–1411, 2017.

[9] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5994–6009, 2017.

[10] W. Zou, Y. Hu, J. Xuan, and H. Jiang, "Towards training set reduction for bug triage," in *Computer Software and Applications Conference*, 2011, pp. 576–581.

[11] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *Proceedings of 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 25–35.

[12] J. Guo, B. Song, S. Chen, F. R. Yu, X. Du, and M. Guizani, "Context-aware object detection for vehicular networks based on edge-cloud cooperation," *IEEE Internet of Things Journal*, pp. 1–9, 2019.

[13] X. Long, J. Wu, and L. Chen, "Energy-efficient offloading in mobile edge computing with edge-cloud collaboration," in *International conference on algorithms and architectures for parallel processing*, 2018, pp. 460–475.

[14] Y. H. Kao, B. Krishnamachari, M. R. Ra, and B. Fan, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," in *IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 1894–1902.

[15] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[16] B. P. Rimal and M. Maier, "Workflow scheduling in multi-tenant cloud computing environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 290–304, 2017.

[17] W. Zhang and Y. Wen, "Energy-efficient task execution for application as a general topology in mobile cloud computing," *IEEE Transactions on cloud Computing*, vol. 6, no. 3, pp. 708–719, 2018.

[18] L. Chen, J. Wu, X. X. Zhang, and G. Zhou, "Tarco: Two-stage auction for d2d relay aided computation resource allocation in hetnet," *IEEE Transactions on Services Computing*, pp. 1–14, 2017, doi:10.1109/TSC.2018.2792024.

[19] J. Yan, S. Bi, Y. J. Zhang, and M. Tao, "Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 235–250, 2020.

[20] R. Deng, R. Lu, C. Lai, and T. H. Luan, "Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing," in *Communications (ICC), IEEE International Conference on*. IEEE, 2015, pp. 3909–3914.

[21] M. H. Chen, B. Liang, and M. Dong, "Joint offloading and resource allocation for computation and communication in mobile cloud with computing access point," in *IEEE Conference on Computer Communications (INFOCOM)*, 2017, pp. 1–9.

[22] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "Hybrid method for minimizing service delay in edge cloud computing through vm migration and transmission power control," *IEEE Transactions on Computers*, vol. 66, no. 5, pp. 810–819, 2017.

[23] L. Chen, L. Huang, H. Xu, H. Deng, and Z. Sun, "Optimal channel assignment schemes in underlay crns with multi-pu and multi-su transmission pairs," in *Wireless Algorithms, Systems, and Applications (WASA). Lecture Notes in Computer Science*, vol. 9204. Springer International Publishing, 2015, pp. 29–39.

[24] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 2016, pp. 1–9.

[25] L. Rao, X. Liu, M. D. Ilic, and J. Liu, "Distributed coordination of internet data centers under multiregional electricity markets," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 269–282, 2012.

[26] W. Chen, C. T. Lea, and K. Li, "Dynamic resource allocation in ad-hoc mobile cloud computing," in *IEEE Wireless Communications and Networking Conference (WCNC)*, 2017, pp. 1–6.

[27] S. E. Mahmoodi, R. Uma, and K. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Transactions on Cloud Computing*, vol. 7, no. 2, pp. 301–313, 2016.

[28] Q. Wang, S. Guo, J. Liu, C. Pan, and L. Yang, "Profit maximization incentive mechanism for resource providers in mobile edge computing," *IEEE Transactions on Services Computing*, pp. 1–1, 2019.

[29] J. Qin, W. X. Zheng, and H. Gao, "Coordination of multiple agents with double-integrator dynamics under generalized interaction topologies," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 1, pp. 44–57, 2012.

[30] X. Lin and C. Q. Wu, "On scientific workflow scheduling in clouds under budget constraint," in *IEEE the 42nd International Conference on Parallel Processing (ICPP)*, vol. 46, no. 1, 2013, pp. 90–99.

[31] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[32] X. Li, M. A. Salehi, M. Bayoumi, N.-F. Tzeng, and R. Buyya, "Cost-efficient and robust on-demand video transcoding using heterogeneous cloud services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 3, pp. 556–571, 2018.

[33] E. S. Chung and J. C. Hoe, "High-level design and validation of the bluesparc multithreaded processor," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 10, pp. 1459–1470, 2010.

[34] X. Lyu, H. Tian, W. Ni, Y. Zhang, P. Zhang, and R. P. Liu, "Energy-efficient admission of delay-sensitive tasks for mobile edge computing," *IEEE Transactions on Communications*, pp. 1–14, 2018, doi:10.1109/TCOMM.2018.2799937.

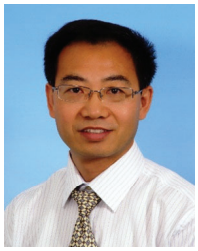
[35] C.-B. Park, B.-S. Park, H.-J. Uhm, H. Choi, and H.-S. Kim, "Ieee 802.15.4 based service configuration mechanism for smartphone," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 3, pp. 2004–2010, 2010.

[36] P.-J. Liu, Y.-K. Lo, H.-J. Chiu, and Y.-J. E. Chen, "Dual-current pump module for transient improvement of step-down dc–dc converters," *IEEE Transactions on Power Electronics*, vol. 24, no. 4, pp. 985–990, 2009.

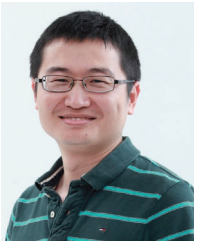
[37] W. Zhang, Y. Wen, and D. O. Wu, "Energy-efficient scheduling policy for collaborative execution in mobile cloud computing," in *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2013, pp. 190–194.

[38] Y. Yu, J. Zhang, and K. B. Letaief, "Joint subcarrier and cpu time allocation for mobile edge computing," in *Global Communications Conference (GLOBECOM)*. IEEE, 2016, pp. 1–6.
[39] Y. Gu, Z. Chang, M. Pan, L. Song, and Z. Han, "Joint radio and computational resource allocation in iot fog computing," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 8, pp. 7475–7484, 2018.
[40] W. Jigang and T. Srikanthan, "Algorithmic aspects of area-efficient hardware/software partitioning," *The Journal of Supercomputing*, vol. 38, no. 3, pp. 223–235, 2006.
[41] L. Chen, J. Wu, G. Zhou, and L. Ma, "Quick: Qos-guaranteed efficient cloudlet placement in wireless metropolitan area networks," *The Journal of Supercomputing*, pp. 1–23, 2018, doi:10.1007/s11227-018-2412-8.
[42] J. Zheng, X. Cao, B. Zhang, X. Zhen, and X. Su, "Deep ensemble machine for video classification," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 2, pp. 553–565, 2018.

Long Chen received his Bachelor degree of Engineering in Computer Science from Anhui University, Hefei, Anhui, China in 2011. After that, he was admitted to the University of Science and Technology of China (USTC). In 2013, he became a PhD candidate in the School of Computer Science and Technology, USTC. He received doctoral degree in 2016. He was in Guangdong University of Technology (GDUT) as a postdoctoral fellow within the excellent postdoctoral research fellow program under the supervision of Prof. Jigang Wu from 2016 to 2019. Now he is a faculty member at GDUT. His main research interests are network economics, mobile computing, vehicular ad-hoc networks, parallel and distributed computing.



Jigang Wu (M'10-) received B.Sc. degree from Lanzhou University, China in 1983, and doctoral degree from University of Science and Technology of China in 2000. He was with the Center for High Performance Embedded Systems, Nanyang Technological University, Singapore, from 2000 to 2010, as a research fellow. He was Tianjin distinguished professor and Dean of School of Computer Science and Software, Tianjin Polytechnic University, China, from 2010 to 2015. Now he is distinguished professor of Guangdong University of Technology. He has published more than 300 papers in IEEE TC, TPDS, TVLSI, TNNLS, TSMC, TVT and international conferences. His research interests include mobile computing, machine intelligence, high performance computing. He is a member of the IEEE. He serves in China Computer Federation as technical committee member in the branch committees, High Performance Computing, Theoretical Computer Science, and Fault Tolerant Computing.



Jun Zhang (S'06-M'10-SM'15) received the B.Eng. degree in Electronic Engineering from the University of Science and Technology of China in 2004, the M.Phil. degree in Information Engineering from the Chinese University of Hong Kong in 2006, and the Ph.D. degree in Electrical and Computer Engineering from the University of Texas at Austin in 2009. He is an Assistant Professor in the Department of Electronic and Information Engineering at the Hong Kong Polytechnic University. His research interests include wireless communications and networking, mobile edge computing and edge learning, distributed learning and optimization, and big data analytics. Dr. Zhang co-authored the book Fundamentals of LTE (Prentice-Hall, 2010). He is a co-recipient of the 2019 IEEE Communications Society & Information Theory Society Joint Paper Award, the 2016 Marconi Prize Paper Award in Wireless Communications, and the 2014 Best Paper Award for the EURASIP Journal on Advances in Signal Processing. He also received the 2016 IEEE ComSoc Asia-Pacific Best Young Researcher Award. He is an Editor of IEEE Transactions on Wireless Communications, IEEE Transactions on Communications, and Journal of Communications and Information Networks.

Hong-Ning Dai (M'06-SM'16) is an associate professor in Faculty of Information Technology at Macau University of Science and Technology. He obtained the Ph.D. degree in Computer Science and Engineering from Department of Computer Science and Engineering at the Chinese University of Hong Kong. He worked in Department of Information Engineering at the Chinese University of Hong Kong and the Hong Kong Applied Science and Technology Research Institute after his Ph.D. study. His research interests include Internet of Things, Big Data Analytics and Blockchains. He is serving as an Associate Editor for IEEE Access and Connection Science and an Editor of Ad Hoc Networks (Elsevier).

Xin Long is a master student in the School of Computer Science and Technology, Guangdong University of Technology. He will receive his master degree in 2022. His main research interests are data mining, computer networks and machine learning.

Mianyang Yao is a master student in the school of computer science and technology, Guangdong University of Technology. He will receive his master degree in 2023. His main research interests are computer networks and machine learning.