





# Mobility-Aware Multi-User Offloading Optimization for Mobile Edge Computing

Wenhan Zhan , Chunbo Luo , *Member, IEEE*, Geyong Min , Chao Wang , *Member, IEEE*, Qingxin Zhu, and Hancong Duan

**Abstract**—Mobile Edge Computing (MEC) is a new computing paradigm with great potential to enhance the performance of user equipment (UE) by offloading resource-hungry computation tasks to lightweight and ubiquitously deployed MEC servers. In this paper, we investigate the problem of offloading decision and resource allocation among multiple users served by one base station to achieve the optimal system-wide user utility, which is defined as a trade-off between task latency and energy consumption. Mobility in the process of task offloading is considered in the optimization. We prove that the problem is NP-hard and propose a heuristic mobility-aware offloading algorithm (HMAOA) to obtain the approximate optimal offloading scheme. The original global optimization problem is converted into multiple local optimization problems. Each local optimization problem is then decomposed into two subproblems: a convex computation allocation subproblem and a non-linear integer programming (NLIP) offloading decision subproblem. The convex subproblem is solved with a numerical method to obtain the optimal computation allocation among multiple offloading users, and a partial order based heuristic approach is designed for the NLIP subproblem to determine the approximate optimal offloading decision. The proposed HMAOA is with polynomial complexity. Extensive simulation experiments and comprehensive comparison with six baseline algorithms demonstrate its excellent performance.

**Index Terms**—Mobile edge computing, task offloading, resource allocation, mobility-aware offloading.

## I. INTRODUCTION

WITH the rapid development of Internet of things, user equipment (UE), e.g., mobile phone, wearable device, and vehicle terminal, becomes prevalent and smarter, which

This work was supported in part by the National Natural Science Foundation of China under Grants 61871096 and 61972075, in part by the National Key R&D Program of China under Grant 2018YFB2101300, in part by the EU H2020 Research and Innovation Programme under the Marie Skłodowska-Curie under Grant 752979, and in part by the China Scholarship Council.

boosts the proliferation of novel mobile applications, such as augmented reality, natural language processing, and face recognition. Most of these applications are computation-intensive, latency-sensitive, and bandwidth-demanding. However, the resources in UEs are usually limited. It is difficult to support these emerging applications with merely onboard systems. Mobile cloud computing (MCC) was proposed to solve this problem [1]. By offloading resource-intensive tasks to the powerful remote cloud, MCC brings several benefits: prolonging battery life, supporting sophisticated computations, and providing potentially unlimited storage.

Nevertheless, the long transmission distance between UE and cloud may cause high latency and jitter, which significantly affects the performance and usability of MCC [2]. Some bandwidth-demanding applications can also cause network congestion, which is harmful to the core network. Considering the weakness of MCC, mobile edge computing (MEC) was introduced in recent years [3]–[6]. In MEC, lightweight MEC servers are ubiquitously deployed in the vicinity of mobile users. Offloading can be done within the radio access networks, bringing lower network latency and less jitter. At the same time, the bandwidth to the core network can be saved, reducing the risk of network congestion.

In MEC, the improvement of application performance largely depends on effective and efficient offloading decisions [6], [7]. Since extra overhead is induced, such as the energy consumption for data transmission and the time cost for task remote execution, it is unwise to always offload all tasks. Besides, MEC servers distributed throughout the network edge are with limited capability, which may lead to fierce resource contentions among users. Considering different environments and user requirements (e.g., user movement and task latency requirement), obtaining the optimal offloading scheme is usually a challenging problem. So far, research on task offloading has attracted significant attention, including the acceleration of computing [8]–[10], the optimization of energy consumption [11]–[13], and the trade-off between these two [14]–[16].

When making offloading decisions, most existing studies assume a quasi-static scenario, where all UEs remain stationary during the offloading procedure that ranges from hundreds of milliseconds to several seconds [16]–[18]. However, in some real-world use cases, the mobility of UE cannot be neglected, especially for on-vehicle applications [19]: UEs can move with high speed in the cellular networks. Due to UE's movement, the wireless channel between UE and base station (BS) changes

dynamically, and offloading processes may even fail if UEs move outside BS's coverage region. When considering user movement, obtaining the optimal offloading solution becomes more challenging [20]. The mobility is also widely researched in MEC, and many methods are proposed to support user mobility from different perspectives, such as service migration [21], [22], path selection [23], and fine-grained control (e.g., transmission power control [24] and resource allocation [25]), etc.

In this paper, we investigate the problem of offloading decision and resource allocation among multiple moving UEs served by a BS. Because of user mobility, sufficient resources must be allocated to ensure the success of task offloading. However, the computation and wireless resources in BS's MEC server are limited, leading to severe contentions with the increase in the number of UEs. Under the constraints of resource limitation and UE's mobility, as well as task latency requirements, the BS needs to find the optimal offloading scheme to maximize the overall system-wide user utility, which is defined as a trade-off between task latency and energy consumption. This optimization problem is proved to be NP-hard and difficult to solve.

To address this challenge, a heuristic mobility-aware offloading algorithm (HMAOA) with polynomial complexity is proposed. It first converts the original global optimization problem into multiple local optimization problems by fixing the maximum number of UEs allowed for offloading in each local optimization problem. Then, each local optimization problem is decomposed into two subproblems: a convex subproblem for allocating computation resources among multiple offloaded tasks and a non-linear integer programming (NLIP) subproblem to decide which UEs to offload. The convex computation allocation subproblem is solved numerically, and a partial order based heuristic approach is designed for the NLIP offloading decision subproblem. The solution to the global optimization problem is obtained by solving all the local optimization problems. Extensive simulations are performed by comparing with six baseline algorithms, and the experiment results demonstrate the superior performance of HMAOA. The main contributions of this paper are summarized as follows:

- We investigate the offloading decision and resource allocation problem among multiple users, while users' mobility in the process of task offloading is taken into account. Under the constraints of resource limitation, user mobility, and task latency requirements, this optimization problem is formulated as a mixed-integer nonlinear programming (MINLP) problem.
- The proposed HMAOA has polynomial complexity. It transforms the original global optimization problem into multiple local optimization problems and then decomposes each local optimization problem into two subproblems: a convex computation allocation subproblem and an NLIP offloading decision subproblem. Because of the NP-hardness of the NLIP offloading decision subproblem, a partial order based heuristic approach is designed to obtain the approximate optimal offloading decision.
- Extensive simulation experiments are conducted by comparing with six baseline algorithms, including the exhaustive search method and fine-tuned genetic algorithm. The experiment results show that HMAOA can achieve more

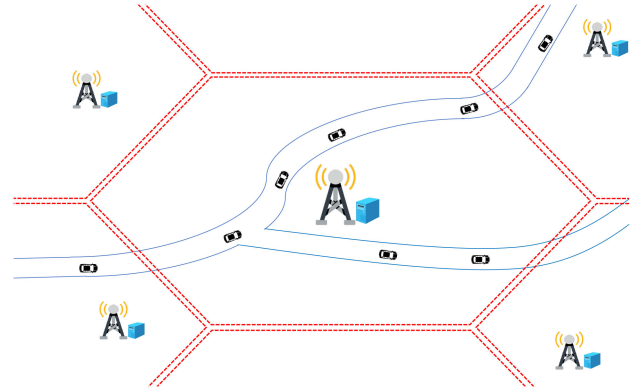


Fig. 1. System architecture of MEC task offloading in a vehicular scenario.

than 99.5% of the optimal system utility on average, confirming its excellent performance.

The rest of this paper is organized as follows. The system model and problem formulation are presented in Section II. In Section III, HMAOA is proposed to solve this problem through a series of transformations and decompositions. We discuss how to adapt HMAOA to other scenarios in Section IV. Simulation experiments are conducted in Section V. Section VI reviews the related works, and, finally, Section VII concludes this paper.

## II. SYSTEM MODEL

In this section, we first introduce the system architecture. Then, the detailed mathematical models of local and offloading execution and the overall designing objective are elaborated.

### A. System Architecture

In order to better demonstrate the impact of mobility, this paper considers a typical vehicular scenario, as shown in Fig. 1. A BS equipped with an MEC server acts as a proximate cloud to enhance the capability of UEs in its coverage region. UEs in the coverage region can move at high speed and, when necessary, request the BS for task offloading to reduce the time cost (i.e., latency) and energy consumption of task execution. After receiving the requests from UEs, the BS needs to make the offloading decision. Since the resources in the BS's MEC server are limited, contentions among UEs become fierce as the number of offloaded UEs increases. Hence, the offloading decision (which UEs can offload their tasks) and the resource allocation (how to allocate the computation and wireless resources for each task) should be jointly considered.

Suppose that  $N$  UEs in the coverage region request the BS for task offloading. Let  $\mathbf{N} = \{1, 2, \dots, N\}$  denote the UE set, and the task of UE  $n$  ( $n \in \mathbf{N}$ ) is defined as

$$OT_n \triangleq (d_n, c_n, \hat{d}_n),$$

where  $d_n$  specifies the size of the input data (in bits),  $c_n$  is the total number of CPU cycles required to accomplish the task, and  $\hat{d}_n$  is the maximum latency allowed for executing the task (in seconds), respectively. The information of  $d_n$  and  $c_n$  can be obtained by applying program profiler (e.g., as introduced in [15], [18], [26]). Since the size of the computation output is

generally much smaller than the input, we omit it in the definition as [15], [16], [18].

All the tasks are atomic, which means that a task can be executed either locally in the UE or remotely in the BS's MEC server (by offloading), but cannot be separated. Let  $a_n \in \{0, 1\}$  denote the offloading decision on  $OT_n$  made by the BS. If  $OT_n$  is assigned for local execution (i.e.,  $a_n = 0$ ),  $c_n$  CPU cycles will be directly processed on the local CPU of UE  $n$ . In contrast, if it is scheduled for offloading (i.e.,  $a_n = 1$ ),  $d_n$  bits of input data should first be transmitted to the BS via the wireless channel. Afterwards, the MEC server in the BS executes  $OT_n$  with the allocated computation resource (in CPU frequency) and then send the output back.

The mobility of UE directly affects the wireless channel between UE and BS. Trajectory prediction is one of the most widely studied topics in the field of intelligent transportation. Even in complicated situations, such as lane changing and turning (other than simply following the road), integrated prediction methods can achieve high predictive accuracy [27]. In this paper, it is thus assumed that UEs' trajectories within a short-term period,  $t^P$ , can be estimated accurately. Let the time when making the offloading decision be 0. The trajectory of UE  $n$  is defined as

$$\mathcal{T}_n(t) \triangleq (x_n(t), y_n(t)), t \in [0, t^P],$$

where  $t$  is a future time from the moment of the offloading decision, i.e., 0, and  $(x_n(t), y_n(t))$  is the coordinate of UE  $n$  at  $t$  taking the BS's location as the origin. After  $t^P$ , the mobility of UEs cannot be anticipated.

The coverage region of the BS can be presented as a polygon with holes. All the points in this polygon constitute a continuous point set, denoted as  $\mathbf{R}$ . When a UE is located in this region, i.e.,  $(x_n(t), y_n(t)) \in \mathbf{R}$ , the channel fading coefficient between the UE and the BS is *statistically* determined by the signal propagation environment. If the UE moves outside the coverage region, i.e.,  $(x_n(t), y_n(t)) \notin \mathbf{R}$ , the data transmission between the UE and the BS cannot be guaranteed (because of the distance, noise, and interference from other base stations, etc.). The UE can be served by a new BS, as well as the MEC server of the new BS, after a handover. Nevertheless, the application data of its ongoing offloading task need to traverse between the original BS and the new one via the core network, whose latency cannot be guaranteed. Worse still, the UE may even be out of service, which leads to offloading failure. Therefore, if a task is scheduled for offloading, sufficient resources must be allocated to ensure that the offloading process is completed before the UE moves outside the BS's coverage region. Otherwise, scheduling the task for local execution is assumed to be a better choice.

### B. Local Execution

Let  $f_n^l$  denote the CPU frequency of UE  $n$ , which represents the number of CPU cycles the UE can execute in each second. Then, the time cost for executing  $OT_n$  locally is given as

$$t_n^l = \frac{c_n}{f_n^l}.$$

In this paper, it is assumed that the local execution of a task can be completed within the task latency requirement, i.e.,  $t_n^l \leq d_n$ . We will discuss how to adapt our method to the situation without this assumption in Section IV.

The CPU power consumption of UE, denoted as  $p_n^l$ , is widely modeled to be a super-linear function of its CPU frequency [11], [17], [18], i.e.,

$$p_n^l = \xi \cdot (f_n^l)^\gamma.$$

Hence, the energy consumption for executing  $OT_n$  is

$$e_n^l = p_n^l \cdot t_n^l = \xi \cdot (f_n^l)^{\gamma-1} c_n.$$

### C. Offloading Execution

The offloading process of any task, e.g.,  $OT_n$ , consists of three phases: 1) transmission of input data from UE to MEC, 2) executing task remotely at MEC server, and 3) feeding computation result from MEC server back to UE. As previously mentioned, we ignored the overhead of the last phase (i.e., computation output reception) [15], [16], [18].

1) *Input Data Transmission*: Let  $B$  denote the system bandwidth, which is orthogonally shared by the UEs in the coverage region, through, e.g., single-carrier frequency division multiple access (SC-FDMA). In this paper, it is assumed that the system bandwidth is equally allocated to the UEs admitted to offload their tasks (i.e.,  $a_n = 1$ ). Then, the channel bandwidth for each UE to transmit its input data is subject to the number of offloaded UEs, which is given as

$$w = \frac{B}{\sum_{n \in \mathbf{N}} a_n}. \quad (1)$$

According to [28], the data transmission rate from UE  $n$  to the BS can be obtained by

$$r_n(w, t) = w \log_2 \left( 1 + \frac{p_n g_n(t)}{w \mathcal{N}} \right), \quad (2)$$

where  $\mathcal{N}$  represents the power spectral density of the white Gaussian noise,  $p_n$  is the transmission power of UE  $n$ , and  $g_n(t)$  denotes the channel gain from UE  $n$  to the BS at time point  $t$ , respectively.

The wireless channel from UE  $n$  to the BS is modeled as a Rayleigh fading channel with free space propagation path loss. The channel gain  $g_n(t)$  can be calculated by

$$g_n(t) = \frac{|h_n|^2}{\mathcal{L}_0 \cdot l_n^\alpha(t)}, \quad (3)$$

where  $h_n$  is a stochastic variable following the complex Gaussian distribution  $\mathcal{CN}(0, 1)$ , representing the small-scale fading.  $\mathcal{L}_0$  and  $\alpha$  are the path-loss constant and the path-loss exponent, respectively.  $l_n(t)$  is the distance between UE  $n$  and the BS, which can be easily calculated by  $l_n(t) = \sqrt{x_n^2(t) + y_n^2(t)}$ .

The value of  $r_n(w, t)$  can be obtained by integrating Eq. (3) into Eq. (2). Since it varies with UE's location, the time cost for task transmission is directly affected by user mobility. Considering that  $r_n(w, t)$  is a stochastic variable (because of  $h_n$ ), we calculate its expectation, i.e.,  $\mathbb{E}_{h_n}\{r_n(w, t)\}$ . According to [29], the expectation can be approximated by Taylor series

expansion as

$$\mathbb{E}_{h_n} \{r_n(w, t)\} \approx \frac{w}{\ln 2} e^{\frac{1}{\text{SNR}_n}} \cdot \left[ T_K(t_K) - T_K\left(\frac{1}{\text{SNR}_n}\right) \right], \quad (4)$$

where  $K$  is the Taylor order parameter to control the accuracy of this approximation, and  $\text{SNR}_n$  is the received signal to noise ratio from UE  $n$  to the BS, denoted as  $\text{SNR}_n = \frac{p_n}{w\mathcal{L}_0^k(t)}$ . The function  $T_K(x)$  is defined as

$$T_K(x) \triangleq \ln x + \sum_{n=1}^K \frac{(-1)^n x^n}{n \cdot n!},$$

and  $t_K$  is the solution to  $\sum_{n=0}^K (-t_K)^n/n! = 0$ .

When  $t \in [0, t^P]$ ,  $\mathbb{E}_{h_n} \{r_n(w, t)\}$  is continuous and positive. Hence, a cumulative area function can be constructed to show the relation between the size of the input data, i.e.,  $d_n$ , and its transmission time, i.e.,  $t_n^{\text{tx}}$ , as

$$d_n = \int_0^{t_n^{\text{tx}}} \mathbb{E}_{h_n} \{r_n(w, t)\} dt. \quad (5)$$

Since  $t_n^{\text{tx}}$  is a function of  $w$ , we denote it as  $t_n^{\text{tx}}(w)$ .

With  $t_n^{\text{tx}}(w)$ , the energy consumption for input data transmission can be calculated by

$$e_n^{\text{tx}}(w) = p_n t_n^{\text{tx}}(w). \quad (6)$$

2) *Task Remote Execution*: After  $OT_n$  is transmitted to the BS, the BS should allocate a certain amount of CPU frequency for executing it. Let  $f_n$  denote the CPU frequency allocated to  $OT_n$ . The time cost for executing  $OT_n$  remotely is

$$t_n^{\text{ex}}(f_n) = \frac{c_n}{f_n}. \quad (7)$$

As mentioned before, all the tasks to be offloaded need to share the limited computation resource in the BS. Let  $F_0$  denote the total available CPU frequency of the BS. We have

$$\sum_{n \in \mathbf{N}} a_n f_n \leq F_0. \quad (8)$$

3) *Offloading Execution Model*: The total time cost for offloading  $OT_n$ , denoted as  $t_n^o$ , is composed of two parts: the data transmission time and the remote execution time, i.e.,

$$t_n^o(w, f_n) = t_n^{\text{tx}}(w) + t_n^{\text{ex}}(f_n). \quad (9)$$

To UE  $n$ , the energy consumption for offloading  $OT_n$  is only associated with the data transmission, which is given by

$$e_n^o(w) = e_n^{\text{tx}}(w). \quad (10)$$

To make the task offloading in this mobility scenario successful, two additional constraints should be satisfied. First, as previously mentioned, if  $OT_n$  is scheduled for offloading (i.e.,  $a_n = 1$ ), the offloading process must be completed before UE  $n$  moves outside the BS's coverage region (i.e.,  $\mathbf{R}$ ), i.e.,

$$\mathcal{T}_n(t) \in \mathbf{R}, \quad \forall t \in [0, t_n^o(w, f_n)], a_n = 1. \quad (11)$$

In other words, all UEs scheduled for offloading (i.e.,  $a_n = 1$ ) must be in the coverage region of the BS (i.e.,  $\mathcal{T}_n(t) \in \mathbf{R}$ ) during the entire offloading process (i.e.,  $\forall t \in [0, t_n^o(w, f_n)]$ ).

Second, since the trajectory of UE cannot be predicted after  $t^P$ , the data transmission after that cannot be anticipated. Therefore, the time consumption for task offloading is constrained to be less than  $t^P$ , i.e.,

$$t_n^o(w, f_n) \leq t^P, \quad \forall n \in \mathbf{N}, a_n = 1. \quad (12)$$

As mentioned before, the trajectory prediction of UE within a short-term period can achieve high accuracy (as [27]). In this paper,  $t^P$  is assumed to be in the range of a few seconds. Considering that the completion time of task offloading ranges from hundreds of milliseconds to several seconds (as [16]–[18]), this additional constraint is reasonable.

#### D. Problem Formulation

In mobile system, the user experience for executing a task, e.g.,  $OT_n$ , is determined mainly by its execution latency, i.e.,  $t_n$ , and its energy consumption, i.e.,  $e_n$ . Specifically,  $t_n$  and  $e_n$  can be obtained by

$$t_n = a_n t_n^o(w, f_n) + (1 - a_n) t_n^l, \quad (13)$$

$$e_n = a_n e_n^o(w) + (1 - a_n) e_n^l. \quad (14)$$

We design a quality-of-experience (QoE)-based utility function as [15], [16] to measure the user utility of executing  $OT_n$ , which is defined as a trade-off between the time and energy consumption of the task compared with local execution, i.e.,

$$u_n(a_n, w, f_n) \triangleq \beta_n^T \left( \frac{t_n^l - t_n}{t_n^l} \right) + \beta_n^E \left( \frac{e_n^l - e_n}{e_n^l} \right), \quad (15)$$

where  $\beta_n^T \in [0, 1]$  and  $\beta_n^E \in [0, 1]$  are set by the user, indicating the user's preference on time and energy consumption when executing  $OT_n$ . We set  $\beta_n^T + \beta_n^E = 1$ . If the task is urgent, the user can increase the time consumption preference. Otherwise, if the UE is running in a low battery mode, the energy consumption preference can be raised.

It is worth noting that the user utility on  $OT_n$ , i.e.,  $u_n(a_n, w, f_n)$ , indicates the improvement in user experience over local execution. The improvement is measured by  $(t_n^l - t_n)/t_n^l$  and  $(e_n^l - e_n)/e_n^l$ , respectively. When  $OT_n$  is executed locally, the user utility  $u_n$  equals 0, i.e.,  $u_n(0, w, f_n) = 0$ , which brings no improvement. If the remote execution of  $OT_n$  brings lower time and energy consumption compared with local execution, the user utility  $u_n$  can be positive. However, offloading too many tasks leads to longer task latency because of the heavy resource contention. In this case, the user utility  $u_n$  can be negative.

We define the system utility as the weighted sum of all user utility, denoted as  $\sum_{n \in \mathbf{N}} \phi_n u_n(a_n, w, f_n)$ , where  $\phi_n \in [0, 1]$  indicates the preference of the service provider (SP) on UE  $n$ , which can be determined based on user's payment [15].



Our target is to find the optimal offloading decision and resource allocation to maximize the system-wide user utility:

$$\mathbf{GP}: \max_{\mathbf{a}, \mathbf{f}} \sum_{n \in \mathbf{N}} \phi_n u_n(a_n, w, f_n)$$

$$\text{s.t.: } a_n \in \{0, 1\}, \quad (16)$$

$$f_n > 0, \quad \forall n \in \mathbf{N}, a_n = 1, \quad (17)$$

$$t_n \leq d_n, \quad \forall n \in \mathbf{N}, \quad (18)$$

where  $\mathbf{a} = \{a_1, a_2, \dots, a_N\}$  and  $\mathbf{f} = \{f_1, f_2, \dots, f_N\}$  are the vectors of variables to be optimized. Constraint (16) states that a task can be either locally executed or offloaded. All offloaded tasks must be allocated a certain amount of computation resource, which is ensured by constraint (17). Constraint (18) guarantees that all the tasks in  $\mathbf{N}$  should be completed on time (subject to the maximum latency, i.e.,  $d_n$ ). Considering the non-linearity of the optimization function, i.e.,  $\sum_{n \in \mathbf{N}} \phi_n u_n(a_n, w, f_n)$ , and value ranges of the variables to be optimized, problem **GP** is an MINLP problem [30].

### III. SOLUTION

#### A. Problem Transformation

Constraint (18) can be rewritten as two constraints based on the offloading decision  $\mathbf{a}$ , i.e.,

$$t_n^l \leq d_n, \quad \forall n \in \mathbf{N}, a_n = 0, \quad (19)$$

$$t_n^o(w, f_n) \leq d_n, \quad \forall n \in \mathbf{N}, a_n = 1. \quad (20)$$

As mentioned before, a task can be completed by local execution within the task latency requirement. Hence, constraint (19) can be ignored in the subsequent optimization.

Constraint (11) indicates that the offloading process must be completed before UE  $n$  moves outside the BS's coverage region (i.e.,  $\mathbf{R}$ ). Hence, it can be transferred from a spatial constraint to a temporal constraint. The key is to determine the time when UE  $n$  first leaves the BS's coverage region, denoted as  $t_n^R$ . With the trajectory of UE  $n$  (i.e.,  $\mathcal{T}_n(t)$ ) and the coverage region of the BS (i.e.,  $\mathbf{R}$ ),  $t_n^R$  can be easily obtained (which will be introduced latter in this section). Then, we rewrite constraint (11) as

$$t_n^o(w, f_n) \leq t_n^R, \quad \forall n \in \mathbf{N}, a_n = 1. \quad (21)$$

Because of (9), constraints (12), (20) and (21) can be rewritten, respectively, as

$$t_n^{\text{ex}}(f_n) \leq t^P - t_n^{\text{tx}}(w), \quad \forall n \in \mathbf{N}, a_n = 1, \quad (22)$$

$$t_n^{\text{ex}}(f_n) \leq d_n - t_n^{\text{tx}}(w), \quad \forall n \in \mathbf{N}, a_n = 1, \quad (23)$$

$$t_n^{\text{ex}}(f_n) \leq t_n^R - t_n^{\text{tx}}(w), \quad \forall n \in \mathbf{N}, a_n = 1. \quad (24)$$

Let  $\mathbf{O}$  denote the set of UEs scheduled for offloading, i.e.,  $\mathbf{O} = \{n | a_n = 1, n \in \mathbf{N}\}$ . Since executing a task locally brings

0 utility, our optimization objective can be rewritten as

$$\mathbf{GP2}: \max_{\mathbf{O}, \mathbf{f}} \sum_{n \in \mathbf{O}} \phi_n u_n(1, w, f_n)$$

$$\text{s.t.: } w = B/|\mathbf{O}|, \quad (25)$$

$$\mathbf{O} \subseteq \mathbf{N}, \quad (26)$$

$$f_n > 0, \quad \forall n \in \mathbf{O}, \quad (27)$$

$$\sum_{n \in \mathbf{O}} f_n \leq F_0, \quad (28)$$

$$t_n^{\text{ex}}(f_n) \leq d_n - t_n^{\text{tx}}(w), \quad \forall n \in \mathbf{O}, \quad (29)$$

$$t_n^{\text{ex}}(f_n) \leq t_n^R - t_n^{\text{tx}}(w), \quad \forall n \in \mathbf{O}, \quad (30)$$

$$t_n^{\text{ex}}(f_n) \leq t^P - t_n^{\text{tx}}(w), \quad \forall n \in \mathbf{O}. \quad (31)$$

The number of offloaded UEs, i.e.,  $|\mathbf{O}|$ , varies from 1 to  $N$ , and the bandwidth allocated to each UE, i.e.,  $w$ , changes accordingly as (25). The discrete variation of  $w$  leads to different objective function (i.e.,  $\sum_{n \in \mathbf{O}} \phi_n u_n(1, w, f_n)$ ) and unstable constraint (i.e., (29)–(31)). To deal with this problem, we convert the original global optimization problem **GP2** into  $N$  local optimization problems with fixed channel bandwidth. The local optimization problem is defined as

$$\mathbf{LP}: \max_{\mathbf{O}, \mathbf{f}} \sum_{n \in \mathbf{O}} \phi_n u_n(1, \hat{w}, f_n)$$

$$\text{s.t.: } |\mathbf{O}| \leq O^{\max}, \quad (26)–(31), \quad (32)$$

where  $\hat{w} = B/O^{\max}$ , and  $O^{\max}$  is fixed in each local optimization problem, but varies from 1 to  $N$  across different local optimization problems. It indicates the maximum number of UEs allowed for offloading in the corresponding local optimization problem. Note that the optimal solution of a local optimization problem may contain less than  $O^{\max}$  offloaded tasks. If so, allocating more bandwidth to each offloading task can bring higher system utility. Hence, the optimal solution of this local optimization problem is not the global optimal solution. When all of the  $N$  local optimization problems (i.e., **LP**) with different  $O^{\max}$  are solved, the global optimal solution, i.e., the solution of **GP2**, can be obtained by simply comparing their system utility. Therefore, we focus on solving **LP** in the rest of this paper.

In **LP**, with the fixed channel bandwidth for each UE, i.e.,  $\hat{w}$ , the time cost for input data transmission, i.e.,  $t_n^{\text{tx}}(\hat{w})$ , can be calculated according to (5). Since the trajectory  $\mathcal{T}_n(t)$  can be arbitrary, it is difficult to derive the analytical solution of  $t_n^{\text{tx}}(\hat{w})$ , as well as  $t_n^R$  (mentioned before). Therefore, a numerical adjusted trapezoidal method is adopted to obtain both the approximate values of  $t_n^{\text{tx}}(\hat{w})$  and  $t_n^R$ , as shown in Algorithm 1. The trajectory of UE  $n$  is divided into multiple tiny segments based on the time interval, *step*. In each segment, the data transmission rate, i.e.,  $\mathbb{E}_{h_n}\{r_n(\hat{w}, \text{timePast})\}$ , is assumed to be fixed. In this way, we can obtain the amount of data transmitted in each *step*, i.e.,  $\Delta$ . Then, the data transmission time, i.e.,  $t_n^{\text{tx}}(\hat{w})$ , is the accumulated time interval when the input data transmission finished (i.e.,  $\text{dataSend} \geq d_n$ ). In each *step*, we also check if

---

**Algorithm 1: Time Approximation (TA).**


---

```

1: procedure TA( $n, \hat{w}, step$ )
2:    $t_n^R \leftarrow t^P, t_n^{tx}(\hat{w}) \leftarrow +\infty$ 
3:    $dataSend \leftarrow 0, timePast \leftarrow 0$ 
4:    $sendFinish \leftarrow \text{false}$ 
5:   while  $timePast < t^P$  do
6:      $timePast \leftarrow timePast + step$ 
7:     if not  $sendFinish$  then
8:        $\Delta \leftarrow step \cdot \mathbb{E}_{h_n}\{r_n(\hat{w}, timePast)\}$ 
9:        $dataSend \leftarrow dataSend + \Delta$ 
10:      if  $dataSend \geq d_n$  then
11:         $t_n^{tx}(\hat{w}) \leftarrow timePast$ 
12:         $sendFinish \leftarrow \text{true}$ 
13:      end if
14:    end if
15:    if  $\mathcal{T}_n(timePast) \notin \mathbf{R}$  then  $\triangleright$  move
    outside cell
16:       $t_n^R \leftarrow timePast$ 
17:      break
18:    end if
19:  end while
20:  return  $t_n^R, t_n^{tx}(\hat{w})$ 
21: end procedure

```

---

UE  $n$  is still within the BS's coverage region (through, e.g., ray casting method [31]) and record the time when it first leaves this region, i.e.,  $t_n^R$ .

As previously mentioned, the trajectory of UE cannot be predicted after  $t^P$ . Therefore, in Algorithm 1, if UE  $n$  never moves outside the BS's coverage region within the period of  $t^P$ , we set the value of  $t_n^R$  as the longest time interval that can be predicted, i.e.,  $t^P$ . In addition, there are two situations that make the data transmission time, i.e.,  $t_n^{tx}(\hat{w})$ , unable to obtain: 1) UE  $n$  leaves the coverage region before the completion of input data transmission; 2) The input data transmission cannot be finished before  $t^P$ . If either of these two situations occurs, it indicates that  $OT_n$  is not suitable for offloading under the current channel bandwidth. Therefore, we set the value of  $t_n^{tx}(\hat{w})$  to infinity, which causes  $OT_n$  to be assigned for local execution because of constraints (29)–(31). The accuracy of the time approximation in Algorithm 1 is subject to the length of  $step$ , and  $\lceil t^P/step \rceil$  iterations are needed in the worst case.

With the values of  $t_n^R$  and  $t_n^{tx}(\hat{w})$ , constraints (29)–(31) can be combined into one, i.e.,

$$t_n^{ex}(f_n) \leq t_n^{outlmt}, \quad \forall n \in \mathbf{O}, \quad (33)$$

where  $t_n^{outlmt}$  is the minimum value in  $d_n - t_n^{tx}(\hat{w})$ ,  $t_n^R - t_n^{tx}(\hat{w})$ , and  $t^P - t_n^{tx}(\hat{w})$ . Since  $t_n^{ex}(f_n)$  is positive, a UE with  $t_n^{outlmt} \leq 0$  should not be offloaded. Let  $\mathbf{N}_1 = \{n | t_n^{outlmt} > 0, n \in \mathbf{N}\}$ . The search space of  $\mathbf{O}$  can be truncated from  $\mathbf{N}$  to  $\mathbf{N}_1$ , i.e.,  $\mathbf{O} \subseteq \mathbf{N}_1$ . According to (7), the constraint can be written as

$$f_n \geq \frac{c_n}{t_n^{outlmt}}, \quad \forall n \in \mathbf{O}. \quad (34)$$

For UE  $n \in \mathbf{N}_1$ , only when  $\phi_n u_n(1, \hat{w}, f_n) \geq 0$ , it is reasonable to offload  $OT_n$ . Otherwise, executing the task locally is a better choice, which increases its utility as well as the system's. Let  $\phi_n u_n(1, \hat{w}, f_n) \geq 0$ , we can derive:

$$t_n^{ex}(f_n) \leq t_n^{inlmt}, \quad \forall n \in \mathbf{O}, \quad (35)$$

where  $t_n^{inlmt} \triangleq \beta_n^E t_n^l (e_n^l - e_n^o) / (\beta_n^T e_n^l) + t_n^l - t_n^{tx}(\hat{w})$ . Similarly, only when  $t_n^{inlmt} > 0$ , it can bring positive utility to offload  $OT_n$ . Let  $\mathbf{N}_2 = \{n | t_n^{inlmt} > 0, n \in \mathbf{N}_1\}$ . The search space of  $\mathbf{O}$  can be further truncated to  $\mathbf{N}_2$ , i.e.,  $\mathbf{O} \subseteq \mathbf{N}_2$ . We have

$$f_n \geq \frac{c_n}{t_n^{inlmt}}, \quad \forall n \in \mathbf{O}. \quad (36)$$

Constraints (27), (34), and (36) can be combined as

$$f_n \geq k_n, \quad \forall n \in \mathbf{O}, \quad (37)$$

where  $k_n$  is the maximum value in  $0$ ,  $c_n/t_n^{inlmt}$ , and  $c_n/t_n^{outlmt}$ . It represents the minimum CPU frequency required to meet once  $OT_n$  is offloaded. When  $k_n > F_0$ , we can only execute such task locally because of the shortage of computation resource. Hence, the search space of  $\mathbf{O}$  is further truncated, i.e.,  $\mathbf{O} \subseteq \mathbf{N}_3$ , where  $\mathbf{N}_3 = \{n | k_n \leq F_0, n \in \mathbf{N}_2\}$ . We have

$$k_n \leq f_n \leq F_0, \quad \forall n \in \mathbf{O}. \quad (38)$$

According to (6), (7), (9), (10), and (13)–(15), the objective function can be rewritten as

$$\sum_{n \in \mathbf{O}} \phi_n u_n(1, \hat{w}, f_n) = \varrho_n - \frac{\varsigma_n}{f_n},$$

where  $\varrho_n$  and  $\varsigma_n$  are two positive constants. Their values can be obtained by  $\varrho_n \triangleq \phi_n \beta_n^T (t_n^l - t_n^{tx}(\hat{w})) / t_n^l + \phi_n \beta_n^E (e_n^l - e_n^o) / e_n^l$  and  $\varsigma_n \triangleq \phi_n \beta_n^T f_n^l$ , respectively. Now, the local optimization problem **LP** is transformed into

$$\begin{aligned}
\mathbf{LP2}: \quad & \max_{\mathbf{O}, \mathbf{f}} \sum_{n \in \mathbf{O}} \varrho_n - \frac{\varsigma_n}{f_n} \\
& \text{s.t.}: \quad \mathbf{O} \subseteq \mathbf{N}_3, \\
& (28), (32), (38), \quad (39)
\end{aligned}$$

*Theorem 1:* Problem **LP2**, as well as **GP**, is NP-hard.

*Proof:* See Appendix A. ■

### B. Problem Decomposition

Problem **LP2** can be decomposed into two subproblems: one for offloading decision and one for computation allocation.

1) *Computation Allocation:* Once the offloading decision set, i.e.,  $\mathbf{O}$ , is given, problem **LP2** reduces to a convex problem to allocate the CPU frequency in the BS's MEC server, i.e.,

$$\mathbf{SP1}: \text{Func}(\mathbf{O}) = \max_{\mathbf{f}} \sum_{n \in \mathbf{O}} \varrho_n - \frac{\varsigma_n}{f_n}$$

s.t.: (28), (38).

Let  $h(\mathbf{f}) = \sum_{n \in \mathbf{O}} \varrho_n - \varsigma_n / f_n$ . We have  $\frac{\partial^2 h(\mathbf{f})}{\partial f_i \partial f_j} = 0$ , ( $i \neq j$ ) and  $\frac{\partial^2 h(\mathbf{f})}{\partial f_i^2} = -2\varsigma_i / f_i^3 < 0$ , which means that the Hessian matrix of  $h(\mathbf{f})$  is negative definite. Hence, **SP1** is a convex optimization, which can be solved with the Lagrangian duality and

---

**Algorithm 2: Frequency Allocation Algorithm ( $Func$ ).**

---

```
1: procedure  $Func(\mathbf{O}, \mathbf{k}, piece)$ 
    $\triangleright \mathbf{k} = \{k_1, k_2, \dots, k_n\}$ 
2:    $totalFrequency \leftarrow \sum_{n \in \mathbf{O}} k_n$ 
3:   if  $totalFrequency > F_0$  then
4:     return  $\mathbf{f} \leftarrow \mathbf{0}, util \leftarrow -\infty$ 
5:   end if
6:    $\mathbf{f} \leftarrow \mathbf{k}$ 
7:    $\mathbf{g} \leftarrow -\mathbf{y}/\mathbf{f}^2 \quad \triangleright$  obtain gradients
    $\mathbf{g} = \{g_1, g_2, \dots, g_n\}$ 
8:   while  $totalFrequency < F_0$  do
9:      $i \leftarrow \arg \min_{i \in \mathbf{O}} \{g_i\} \quad \triangleright$  find the lowest
       gradient
10:     $f_i \leftarrow f_i + piece$ 
11:     $g_i \leftarrow -y_i/f_i^2$ 
12:     $totalFrequency \leftarrow totalFrequency + piece$ 
13:  end while
14:   $util \leftarrow \sum_{n \in \mathbf{O}} \varrho_n - \varsigma_n/f_n$ 
15:  return  $\mathbf{f}, util$ 
16: end procedure
```

---

Karush-Kuhn-Tucker conditions [32]. However, this needs to solve a multi-variable system of non-linear equations, which brings high complexity. We construct Algorithm 2 numerically, which divides  $F_0$  into many tiny atomic pieces and assigns them one by one to each UE. The key is to find the UE with the fastest change on each assignment, which is the one with the lowest gradient. When each piece is small enough, the final allocation can adequately approach the optimal result.

The complexity of Algorithm 2 is subject to the total available frequency, i.e.,  $F_0$ , the length of each tiny piece, i.e.,  $piece$ , the minimum frequency of UEs, i.e.,  $\mathbf{k}$ , and the number of offloaded tasks, i.e.,  $|\mathbf{O}|$ . In the worst case, the complexity of this algorithm is  $O(F_0/step \times O^{\max})$ .

2) *Offloading Decision*: With Algorithm 2, the numerical solution of **SP1** can be obtained. Then, we need to search for the optimal subset  $\mathbf{O}$  from  $\mathbf{N}_3$  as the offloading decision, i.e.,

$$\begin{aligned} \mathbf{SP2}: \quad & \max_{\mathbf{O}} Func(\mathbf{O}) \\ & \text{s.t.: (32), (39).} \end{aligned}$$

Due to the non-linearity of  $Func(\cdot)$ , **SP2** is an NLP problem, which is still NP-hard [33]. Hence, we design a partial order based heuristic approach to solve this subproblem with polynomial complexity.

Let  $Util_n(f) \triangleq \varrho_n - \varsigma_n/f$ . For each  $OT_n \in \mathbf{N}_3$ , if  $f_n \geq k_n$ , we have  $Util_n(f_n) > 0$ ; meanwhile, if  $f_n \in [k_n, F_0]$ , the derivative of  $Util_n(f_n)$  is always positive, i.e.,  $Util'_n(f_n) = \varsigma_n/f_n^2 > 0$ , indicating that  $Util_n(f_n)$  is monotonically increasing on  $[k_n, F_0]$ . Hence,  $Util_n(k_n)$  is the lowest utility  $OT_n$  can contribute to the system utility once offloaded. Then,  $Util_n(f_n)$  increases as  $f_n$  grows, until it consumes all the CPU frequency in the MEC server, i.e.,  $Util_n(F_0)$ . Here, the minimum frequency requirement  $k_n$ , the lowest utility  $Util_n(k_n)$  and the highest

utility  $Util_n(F_0)$  of  $OT_n$  are the key features for constructing our heuristic approach.

*Definition 1*: Define the binary relation *priority equal*:  $\forall n, m \in \mathbf{N}_3$ ,  $OT_n$  is priority equal to  $OT_m$  ( $OT_n \simeq OT_m$ ) if and only if they are with the same minimum frequency requirement and the same lowest and highest utility, i.e.,

$$\begin{aligned} OT_n \simeq OT_m & \iff k_n = k_m \\ & \wedge Util_n(k_n) = Util_m(k_m) \\ & \wedge Util_n(F_0) = Util_m(F_0). \end{aligned}$$

From Definition 1, we can deduce that  $OT_n \simeq OT_m \Rightarrow Util_n(f) = Util_m(f), \forall f \in [k_n, F_0]$ . Therefore, for two priority equivalent tasks, when making the offloading decision, there is no difference in choosing either of them, which means that they have the same priorities.

*Definition 2*: Define the binary relation *priority higher*:  $\forall n, m \in \mathbf{N}_3$ ,  $OT_n$  is priority higher than  $OT_m$  ( $OT_n \succ OT_m$ ) if and only if their minimum frequency requirements satisfy  $k_n < k_m$ , and their utility at  $k_m$  and  $F_0$  satisfy  $Util_n(k_m) > Util_m(k_m)$  and  $Util_n(F_0) > Util_m(F_0)$ , respectively, i.e.,

$$\begin{aligned} OT_n \succ OT_m & \iff k_n < k_m \\ & \wedge Util_n(k_m) > Util_m(k_m) \\ & \wedge Util_n(F_0) > Util_m(F_0). \end{aligned}$$

*Lemma 1*:  $\forall n, m \in \mathbf{N}_3$ ,  $OT_n \succ OT_m \Rightarrow Util_n(f) > Util_m(f), \forall f \in [k_m, F_0]$ .

*Proof*: See Appendix B. ■

With this lemma, if  $OT_n \succ OT_m$ ,  $OT_n$  contributes more in system utility than  $OT_m$  when  $f \in [k_m, F_0]$ ; and  $OT_n$  is the only choice when  $f \in [k_n, k_m]$ . Hence, when  $OT_n$  is priority higher than  $OT_m$ , choosing  $OT_n$  is always better than  $OT_m$ , which means that  $OT_n$  has a higher priority than  $OT_m$ .

*Lemma 2*: With the binary relation of priority higher, the UE set,  $\mathbf{N}_3$ , is a strict partially ordered set (poset).

*Proof*: See Appendix C. ■

Let  $(\mathbf{N}_3, \succ)$  denote the poset. Based on  $(\mathbf{N}_3, \succ)$ , we can generate a Hasse diagram (HD) [34], which is a transitive reduced directed acyclic graph (DAG). In an HD, when  $OT_i \succ OT_j$ , there is one and only one path from  $OT_i$  to  $OT_j$ .

Poset  $(\mathbf{N}_3, \succ)$  becomes a totally ordered set when all the tasks in it are comparable with each other. In this case, the generated HD reduces to a chain, which means that it can be sorted into a sequence under the relation of priority higher.

*Definition 3*: For a sequence  $Seq = (i_1, i_2, \dots, i_n)$ , the  $k$ th head-subsequence of  $Seq$  is the subsequence of  $Seq$  constructed by the first  $k$  items in  $Seq$  in their original order. We denote it as  $head(Seq, k) \triangleq (i_1, i_2, \dots, i_k)$ .

*Lemma 3*: If  $(\mathbf{N}_3, \succ)$  is a totally ordered set, i.e., a chain, and let  $Seq_{\mathbf{N}_3}$  denote the sequence from its highest priority task to its lowest one, then the optimal offloading decision set is one of  $Seq_{\mathbf{N}_3}$ 's head-subsequences.

*Proof*: See Appendix D. ■

Because of the non-linearity of  $Func(\cdot)$ , it is not straightforward to find which head-subsequence of  $Seq_{\mathbf{N}_3}$  is the optimal

---

**Algorithm 3:** Offloading Decision on Chain (ODC).

---

```
1: procedure ODC( $Seq_{\mathbf{N}_3}, O^{\max}$ )
    $\triangleright Seq_{\mathbf{N}_3} = (OT'_1, OT'_2, \dots, OT'_{|\mathbf{N}_3|})$ 
2:  $taskContent \leftarrow \min(|Seq_{\mathbf{N}_3}|, O^{\max})$ 
3:  $\mathbf{O} \leftarrow \emptyset, \mathbf{f} \leftarrow \mathbf{0}, util \leftarrow 0$ 
4: for  $i \leftarrow 1, taskContent$  do
5:   if  $\sum_{x=1}^i k_{OT'_x} > F_0$  then  $\triangleright$  no enough frequency
6:     break
7:   end if
8:    $[\hat{\mathbf{f}}, \hat{util}] \leftarrow Func(head(Seq_{\mathbf{N}_3}, i))$   $\triangleright$ 
     Algorithm 2
9:   if  $util > \hat{util}$  then
10:     $\mathbf{O} \leftarrow head(Seq_{\mathbf{N}_3}, i), \mathbf{f} \leftarrow \hat{\mathbf{f}}, util \leftarrow \hat{util}$ 
11:   end if
12: end for
13: return  $\mathbf{O}, \mathbf{f}, util$ 
14: end procedure
```

---

offloading decision. We only need to find the one with maximum system utility from the  $|\mathbf{N}_3|$  head-subsequences of  $\mathbf{N}_3$ . Algorithm 3 describes this procedure. In the worst case, it needs to call Algorithm 2  $O^{\max}$  times.

Generally, not every task in  $(\mathbf{N}_3, \succ)$  is comparable to each other. In this case, we consider constructing a linear extension of  $(\mathbf{N}_3, \succ)$ , which is a totally ordered set that contains all the relations in the original poset [34]. Let  $(\mathbf{N}_3, \succ^*)$  denote the linear extension. We have  $\forall OT_i, OT_j \in (\mathbf{N}_3, \succ), OT_i \succ OT_j \Rightarrow OT_i \succ^* OT_j, OT_i, OT_j \in (\mathbf{N}_3, \succ^*)$ .

*Lemma 4:* The optimal offloading decision set of **SP2** is a head-subsequence in one of  $(\mathbf{N}_3, \succ)$ 's sorted linear extensions.

*Proof:* See Appendix E.  $\blacksquare$

According to the order-extension principle, every partial order can be extended to at least one total order by topological sorting algorithms [34]. With this principle, an intuitive method is to enumerate all the linear extensions of  $(\mathbf{N}_3, \succ)$ . However, this method has huge complexity (to count all the linear extensions of a finite partial order is #P-complete [35]). In the worst case, if all the tasks in  $(\mathbf{N}_3, \succ)$  are incomparable with each other, there will be  $|\mathbf{N}_3|!$  linear extensions.

Based on the above analysis, in this paper, we heuristically construct a task comparison method, which turns  $(\mathbf{N}_3, \succ)$  into a linear extension and then obtains the approximate optimal offloading decision by Algorithm 3.

The relation between two tasks on their contributions to the system utility changes with the allocated frequency.  $\forall OT_n, OT_m \in (\mathbf{N}_3, \succ)$ , there exist four different situations.<sup>1</sup>

- 1) When  $k_n \leq k_m \wedge Util_n(k_m) \geq Util_m(k_m) \wedge Util_n(F_0) \geq Util_m(F_0)$ , we have  $OT_n \succ OT_m$  or  $OT_n \simeq OT_m$ . In this case, we define  $OT_n \succ^* OT_m$ .
- 2) When  $k_n > k_m \wedge Util_n(k_n) \geq Util_m(k_n) \wedge Util_n(F_0) \geq Util_m(F_0)$ , we find that  $k_n$  is a *change point*. It means that if  $f \in [k_n, F_0]$ ,  $OT_n$  contributes more

to the system utility, i.e.,  $Util_n(f) \geq Util_m(f)$ , and if  $f \in [k_m, k_n)$ ,  $OT_m$  contributes more (only  $OT_m$  is available in this case).

- 3) When  $k_n \leq k_m \wedge Util_n(k_m) \geq Util_m(k_m) \wedge Util_n(F_0) < Util_m(F_0)$ , we derive the change point  $f_{chg}$ , which satisfies the equation  $Util_n(f_{chg}) = Util_m(f_{chg})$ . If  $f \in [k_n, f_{chg})$ ,  $Util_n(f) \geq Util_m(f)$ , and if  $f \in [f_{chg}, F_0]$ ,  $Util_m(f) \geq Util_n(f)$ .
- 4) When  $k_n \leq k_m \wedge Util_n(k_m) < Util_m(k_m) \wedge Util_n(F_0) \geq Util_m(F_0)$ , there are two change points  $f_{ch1}$  and  $f_{ch2}$ . Let  $f_{ch1} = k_m$ , and  $f_{ch2}$  is the solution of  $Util_n(f) = Util_m(f)$ . If  $f \in [k_n, f_{ch1}) \cup [f_{ch2}, F_0)$ ,  $Util_n(f) \geq Util_m(f)$ , and if  $f \in [f_{ch1}, f_{ch2})$ ,  $Util_m(f) \geq Util_n(f)$ .

We find that, in situation (1), the relation between two tasks on their contributions to the system utility is independent to the allocated frequency  $f$ . We only need to retain the relations in the original set  $(\mathbf{N}_3, \succ)$  to ensure the accuracy of topological sorting. However, in situations (2), (3), and (4), there exist one or two change points. When  $f$  changes, the relation of the two tasks changes too. More conditions should be considered in order to compare them.

When a change point of two tasks, i.e.,  $f_{chg}$ , is small, e.g., near 0, the relation on  $(0, f_{chg}]$  can hardly take effect because almost every offloaded task should be allocated a certain amount of frequency. Similarly, when a change point is large, e.g., near  $F_0$ , the relation on  $(f_{chg}, F_0]$  has little impact since a task can hardly consume all the resources. Hence, we define two parameters  $\Psi_l, \Psi_h \in (0, 1]$  to control two frequency thresholds,  $bound_l(\Psi_l)$  and  $bound_h(\Psi_h)$ , and only consider the task relations in the range of  $(bound_l(\Psi_l), bound_h(\Psi_h))$ .  $bound_l$  and  $bound_h$  are defined as

$$bound_l = (1 - \Psi_l)F_0/O^{\max},$$

$$bound_h = (1 + (O^{\max} - 1)\Psi_h)F_0/O^{\max}.$$

The values of  $\Psi_l$  and  $\Psi_h$  are set empirically. When they become larger, the ignored change points become less.

When there is at least one change point between  $bound_l$  and  $bound_h$ , the relation between two tasks becomes subtle. In this case, we empirically determine their relation by comparing the area under their utility functions, i.e.,  $Util(\cdot)$ . The area can well synthesize  $k_n, Util_n(k_n)$ , and  $Util_n(F_0)$ , simultaneously. The evaluation in Section V will show its excellent property. The area of  $OT_n$  when  $f \in [bound_l, bound_h]$  can be calculated by definite integral:

$$\begin{aligned} Area(OT_n) &= \int_{f_1}^{f_2} Util_n(f)df \\ &= \varrho_n(f_2 - f_1) - \varsigma_n(\log(f_2) - \log(f_1)), \end{aligned}$$

where  $f_1 = \max(bound_l, k_n)$  and  $f_2 = bound_h$ .

We summarize the task comparison method in Table I. Now, any two tasks in  $(\mathbf{N}_3, \succ)$  are comparable with each other by the heuristic method. Sorting algorithms can be adopted to build the sequence, and one of its head-sequences is the approximate optimal offloading decision.

<sup>1</sup> $OT_n$  and  $OT_m$  can be any tasks in  $(\mathbf{N}_3, \succ)$ . If we exchange their positions in each situation, the result is also correct. There is no need to list them repeatedly.



TABLE I  
THE COMPARISON TABLE OF TWO TASKS

Situation	Change Point	Larger Task
(1)	No	$OT_n$
(1)	No	$OT_n$
(2)	$f_{chg} \leq bound_l$	$OT_n$
(2)	$f_{chg} \geq bound_h$	$OT_m$
(2)	$bound_l < f_{chg} < bound_h$	Task with larger area
(3)	$f_{chg} \leq bound_l$	$OT_m$
(3)	$f_{chg} \geq bound_h$	$OT_n$
(3)	$bound_l < f_{chg} < bound_h$	Task with larger area
(4)	$f_{ch1}, f_{ch2} \leq bound_l$	$OT_n$
(4)	$f_{ch1}, f_{ch2} \geq bound_h$	$OT_m$
(4)	$f_{ch1} < bound_l, f_{ch2} \geq bound_h$	$OT_m$
(4)	Other conditions	Task with larger area

---

**Algorithm 4:** Heuristic Mobility-Aware Offloading Algorithm.

---

```

1: procedure HMAOA( $\mathbf{N}$ )
2:   for  $O^{\max} \leftarrow 1, N$  do
3:     for all  $n \in \mathbf{N}$  do
4:       Compute  $t^R$  and  $t_n^{\text{tx}}(\hat{w})$  by Algorithm 1.
5:       if  $OT_n \in \mathbf{N}_3$  then
6:         Compute  $Area(OT_n)$ .
7:       end if
8:     end for
9:     Sort  $\mathbf{N}_3$  into  $Seq_{\mathbf{N}_3}$  in accordance with
       Table I.
10:    Obtain optimal  $\mathbf{O}$ ,  $\mathbf{f}$ , and utility by
       Algorithm 3.
11:    Record  $\mathbf{O}$ ,  $\mathbf{f}$ , and utility under  $O^{\max}$ .
12:  end for
13:  Find the  $\mathbf{O}$  and  $\mathbf{f}$  with maximum utility in all
       records.
14:  Send the offloading decision to UEs.
15: end procedure

```

---

### C. Heuristic Mobility-Aware Offloading Algorithm

The proposed HMAOA is summarized in Algorithm 4. After receiving the offloading requests from  $N$  UEs, HMAOA enters  $N$  iterations, each of which has different  $O^{\max}$  (from 1 to  $N$ ). In each iteration,  $\hat{w}$  is fixed, i.e.,  $\hat{w} = B/O^{\max}$ . For each offloading request, Algorithm 1 is performed to obtain  $t^R$  and  $t_n^{\text{tx}}(\hat{w})$ . Then, the search space is truncated to  $\mathbf{N}_3$ . HMAOA computes the area under the utility function of all tasks (i.e.,  $Util_n(\cdot)$ ) as introduced in Section III-B2, which can boost the task comparison afterwards. Based on the proposed heuristic task comparison method, sorting algorithms can be adopted to turn  $\mathbf{N}_3$  into an ordered sequence. After that, the approximate optimal offloading decision in this iteration is obtained by Algorithm 3 and recorded. Finally, all the recorded offloading decisions are compared to find the one with the highest utility as the offloading decision.

Three critical operations determine the complexity of HMAOA: Algorithm 1, with the complexity of  $O(t^P/step)$ , is invoked  $N^2$  times; the sorting algorithm, usually with the complexity of  $O(N \times \log(N))$ , is called  $N$  times; Algorithm 3,

which invokes Algorithm 2  $N$  times, is called  $N$  times. In the worst case, the complexity of Algorithm 2 is  $O(F_0/step \times N)$ . Therefore, the complexity of HMAOA is  $O(N^3 \times (F_0/step))$ .

## IV. DISCUSSION

This section discusses how to adapt the proposed HMAOA to other scenarios.

### A. Latency Requirement

As previously mentioned, in this paper, we assume that all tasks can be completed by local execution within their task latency requirements, i.e.,  $t_n^l \leq d_n$ . This assumption is reasonable because MEC services are not always available, and a UE must complete its task by itself if no MEC server exists (e.g., out of service). Nevertheless, in some special use cases (e.g., a very urgent task), the latency requirement of a task may be less than its local execution time, i.e.,  $d_n < t_n^l$ . This means that the task should always be offloaded, and sufficient resources should be allocated. Let  $\hat{\mathbf{O}}$  denote the set of all these tasks, i.e.,  $\hat{\mathbf{O}} = \{n | d_n < t_n^l, n \in \mathbf{N}\}$ . Since all tasks in  $\hat{\mathbf{O}}$  should be offloaded, we have  $\hat{\mathbf{O}} \subseteq \mathbf{O}$ . The original global optimization problem changes into

$$\begin{aligned}
 \text{GP3: } \max_{\mathbf{O}, \mathbf{f}} \quad & \sum_{n \in \mathbf{O}} \phi_n u_n(1, w, f_n) \\
 \text{s.t.: } \quad & \hat{\mathbf{O}} \subseteq \mathbf{O} \subseteq \mathbf{N}, \\
 & (25), (27)–(31). \tag{40}
 \end{aligned}$$

The corresponding local optimization problem changes into

$$\begin{aligned}
 \text{LP3: } \max_{\mathbf{O}, \mathbf{f}} \quad & \sum_{n \in \mathbf{O}} \varrho_n - \frac{s_n}{f_n} \\
 \text{s.t.: } \quad & \hat{\mathbf{O}} \subseteq \mathbf{O} \subseteq \mathbf{N}_4, \\
 & (28), (32), (38), \tag{41}
 \end{aligned}$$

where  $\mathbf{N}_4 = \{n | c_n/t_n^{\text{outlimt}} \leq F_0, n \in \mathbf{N}_1\}$ . In LP3, a new search space of  $\mathbf{O}$ , i.e.,  $\mathbf{N}_4$ , is derived because constraint (36) does not apply to this scenario: A task in  $\hat{\mathbf{O}}$  should always be offloaded even it brings negative utility.

By pre-determining some elements of the offloading set  $\mathbf{O}$  before making the offloading decision, HMAOA can be well adapted to the new scenario. Specifically, in each local optimization we search from the *difference set* between  $\mathbf{N}_4$  and  $\hat{\mathbf{O}}$ , i.e.,  $\mathbf{N}_4 - \hat{\mathbf{O}}$ , for the optimal head-subsequence, which, combined with  $\hat{\mathbf{O}}$ , can produce the maximum system utility. Note that some local optimization problems (i.e., LP3) of GP3 may be infeasible because  $\hat{\mathbf{O}} \not\subseteq \mathbf{N}_4$ ,  $|\hat{\mathbf{O}}| > O^{\max}$ , or  $\sum_{n \in \hat{\mathbf{O}}} f_n > F_0$ . This does not affect the global optimization. We just need to optimize the feasible ones and, among them, find the optimal solution.

However, there exists the situation where all the derived local optimization problems are infeasible. It indicates that the global optimization problem itself is infeasible because of resource limitation. How to deal with this situation is beyond the scope of this paper, which is an interesting topic to investigate in our future work.

## B. Resource Multiplexing

Since users' requests arrive dynamically, HMAOA can work in an online mode to achieve resource multiplexing in the time dimension: The BS collects the released communication and computation resources in real time and allocates current available resources to the requesting UEs. In this mode, our original offloading optimization problem can be rewritten as

$$\mathbf{GP4}: \max_{\mathbf{O}, \mathbf{f}} \sum_{n \in \mathbf{O}} \phi_n u_n(1, w, f_n)$$

$$\text{s.t. } w = \hat{B}/|\mathbf{O}|, \quad (42)$$

$$\mathbf{O} \subseteq \hat{\mathbf{N}}, \quad (43)$$

$$\sum_{n \in \mathbf{O}} f_n \leq \hat{F}_0, \quad (27), (29)-(31). \quad (44)$$

where  $\hat{\mathbf{N}}$  is the set of current requesting UEs, and  $\hat{B}$  and  $\hat{F}_0$  are current available wireless bandwidth and CPU frequency in the MEC server, respectively. Since **GP4** is equivalent to **GP2**, HMAOA can solve this problem without any modification.

## C. Multiple Coverage Regions

In this paper, the coverage region of one BS is considered to be the basic spatial unit of offloading decision and resource allocation. By dividing the physical space into multiple spatial units based on BS's coverage region and only focusing on one of them, the offloading problem can be simplified. At the same time, applying the solution for one coverage region (i.e., HMAOA) to others is straightforward.

The overall system performance can be further improved if BSs can cooperate. This can ensure the success of ongoing offloaded tasks even when UEs traverse among BS's coverage regions. To further optimize the offloading scheme in this situation is an interesting topic to investigate at the next stage, in which the handover cost, delay, the available resources of both the original and the target BS, and the topology and status of the edge and core network should all be taken into account.

## V. EVALUATION RESULTS

In this section, simulation experiments are adopted to evaluate the proposed HMAOA. All the experiments are run on Matlab R2018b using a server with Intel Xeon CPU E5-2650 @ 2.2 GHz processor. Table II summarizes the main simulation parameters and their default values.

In our simulation, the coverage region of the BS is considered to be a circle with a radius of  $R = 100$  meters. The system bandwidth of the BS is set as  $B = 20$  MHz, and the total available CPU frequency is set as  $F_0 = 20$  GHz. All the wireless communication parameters, including the path-loss, the white Gaussian noise density, and the UE's transmission power, are set in accordance with the 3GPP specification [36].

Each UE has its motion parameters, including location, speed, and direction. In the simulations, for simplicity, we consider all

TABLE II  
MAIN SIMULATION PARAMETERS

Parameters	Value
Radius of the coverage region $R$	100 m
Available CPU frequency on BS $F_0$	20 GHz
System bandwidth $B$	20 MHz
Path-loss from UE to BS	$128.1 + 37.5 \log_{10}(l(\text{km}))$ dBm
White Gaussian noise density $\mathcal{N}$	-174 dBm/Hz
UE's transmission power $p_n$	23 dBm
Speed of each UE	[10, 60] m/s
Direction of each UE	$[\pi/4, \pi]$
UE's local CPU frequency $f_n^l$	[0.5, 1.5] GHz
User's time preference $\beta_n^T$	[0.25, 0.75]
SP's user preference $\phi_n$	[0.25, 0.75]
Input data size of each task $d_n$	(0, 3] MB
Total CPU cycles of each task $c_n$	(0, 3] GCycles
Latency requirement of each task $d_n$	[1, 5] seconds
Trajectory prediction time interval $t^P$	4 seconds
Power consumption coefficient $\xi$	$10^{-11}$
Power consumption coefficient $\gamma$	2
Taylor order parameter $K$	5

UEs moving straightly with a constant speed. In each simulation run,  $N$  moving UEs are uniformly distributed in the BS's coverage region, and their speed and direction are chosen from the uniform distributions as shown in Table II.

For the sake of generality, all the parameters regarding UE, i.e., the UE's local CPU frequency  $f_n^l$ , the user's time consumption preference  $\beta_n^T$ , and the user preference of SP  $\phi_n$ , are set as random variables following the uniform distributions as shown in Table II. The same consideration holds for the parameters regarding task, i.e., the input data size  $d_n$ , the number of CPU cycles  $c_n$ , and the latency requirement  $d_n$ . To ensure that the local execution of a task is completed within its latency requirement,  $d_n$  is set to  $c_n/f_n^l$  if its randomly generated value is less than  $c_n/f_n^l$ .

We set the time interval, in which UEs' trajectories can be predicted, i.e.,  $t^P$ , to 4 seconds. The power consumption coefficients, i.e.,  $\xi$  and  $\gamma$ , and the Taylor order parameter, i.e.,  $K$ , are set in accordance with [17] and [29], respectively.

The proposed HMAOA is compared with the following baseline algorithms.

- **Exhaustive Search Method (ESM)**: In this method, all the  $2^N$  combinations of offloading decisions are evaluated to find out the optimal solution with the highest utility. Because of the enormous computational complexity, it takes a huge amount of time to run this method.
- **Genetic Algorithm (GA)**: As a meta-heuristic algorithm, GA is a practical solution for this kind of combinatorial optimizations. The GA implementation in Matlab global optimization toolbox is adopted. The offloading decision is encoded into the chromosome of each individual. As the algorithm progresses, individuals that contribute higher system utility gain more opportunities for reproduction. When GA terminates, the most adaptable individual is selected as the final offloading decision. The settings of GA are summarized in Table III.
- **Stationary Exhaustive Search Method (SESM)**: This method makes offloading decisions without considering UEs' mobility. It assumes that all UEs are stationary, and

TABLE III  
SETTINGS OF THE GENETIC BASELINE ALGORITHM

Settings	Values	Settings	Values
Population Size	100	Elite count	2
Maximum Generations	200	Creation function	Uniform
Maximum Stall Generations	80	Crossover Function	Scattered
Constraint Tolerance	$10^{-3}$	Crossover Fraction	0.8
Function Tolerance	$10^{-6}$	Mutation Function	Gaussian

adopts ESM to find the optimal offloading solution. Since UEs are actually moving, following SESM's offloading decision may cause task failure. The performance of SESM is the upper bound of all the methods that do not consider UEs' mobility.

- **Independent Decision Execution (IDE):** In this method, a UE assumes that the resources in the MEC server are dedicated to its offloaded task, and makes offloading decisions independently. This method takes the mobility of UE into account. When it is possible to gain a higher utility compared with local execution, the UE offloads its task. This is a classic approach that does not consider resource allocation [12], [16].
- **All Edge Execution (AEE):** All UEs offload their tasks, and they need to compete for the limited communication and computation resources.
- **All Local Execution (ALE):** In contrast with AEE, offloading is not allowed in this method, and all tasks are executed locally.

It is worth noting that, without considering user mobility or resource limitation, the offloading decisions made by SESM, IDE, and AEE may cause task failure and wasted resources. How to deal with the failed tasks in these methods is beyond the scope of this paper. In this experiment, we simply assume that a UE can obtain how many resources are allocated for it from the response of the MEC server. If the allocated resource is not enough for task offloading, the UE executes the task locally, which brings 0 user utility.

With different numbers of UEs ( $N = 5, 10, \dots, 30$ ), the simulation is run for 200 iterations. In each iteration, the randomly generated parameters, under which all the algorithms run, are the same. Then, the average outcome is adopted as the simulation result.

*System Utility With Varying Number of Users:* We first evaluate the average system utility with different numbers of users, and the evaluation results are shown in Fig. 2. Because ALE always produces 0 system utility, we omit it in the figures of this section unless necessary. There are two bound parameters in HMAOA, i.e.,  $\Psi_l$  and  $\Psi_h$ . The effect of their values is evaluated in later simulation, and at this time, they are set to  $\Psi_l = 0.8$  and  $\Psi_h = 0.3$ , respectively. We can see that the performance of HMAOA is very close to ESM. It always achieves more than 99.5% of the optimal system utility under different numbers of users. The performance of GA and HMAOA is close when  $N$  is small. As  $N$  increases, the performance gap between GA and HMAOA becomes larger. The average system utility of GA, when  $N = 30$ , is only about 75% of HMAOA. When  $N > 20$ , because of the huge search space, the utility of GA even decreases. Since mobility is not considered, SESM may offload

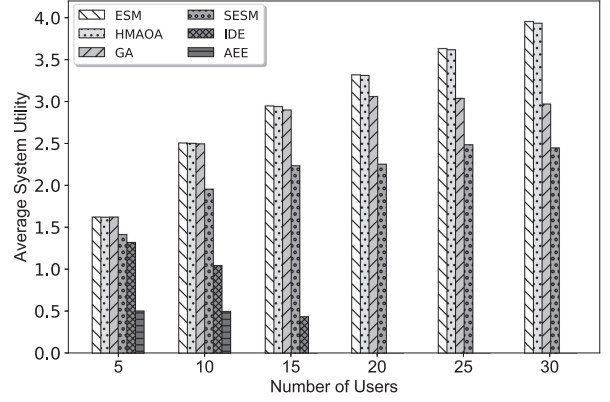


Fig. 2. Average system utility with different numbers of users.

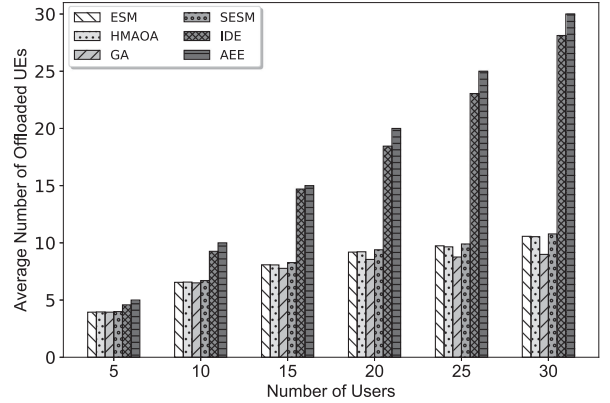


Fig. 3. Average number of offloaded UEs with different numbers of users.

some inappropriate tasks, which contributes 0 user utility (as mentioned before). Its performance is strictly lower than ESM, HMAOA, and GA. When  $N \geq 15$ , the average system utility of AEE becomes negative, and so does IDE when  $N \geq 20$ . Because of the fierce resource contentions, resources allocated to each UE are not enough to produce positive utility. We omit the negative utility segments in Fig. 2 since the performance of these methods is even worse than ALE.

*Number of Offloaded UEs With Varying Number of Users:* Then, we evaluate the average number of offloaded UEs with different numbers of users in the same setting as before, as shown in Fig. 3. Since ALE never offloads, we omit it in this figure. Because AEE always offloads a task, its offloaded UE number increases linearly with the increase of user number. In our simulations, IDE always assumes that a UE can monopolize all the resources in the MEC server. Because the resources on an MEC server are usually adequate for only one task, the performance of IDE is similar to the AEE. ESM, HMAOA, GA, and SESM perform similarly. Constrained by the limited resources, the numbers of their offloaded UEs grow very slowly and are stable at around 10.

*Algorithm Running Time With Varying Number of Users:* Fig. 4 shows the average running time of different algorithms with different numbers of users. Due to the extremely high complexity of ESM, SESM, and GA, we set the y-axis in Fig. 4 to a logarithmic scale. From this figure, we can see the exponential complexity of ESM and SESM. GA is always with

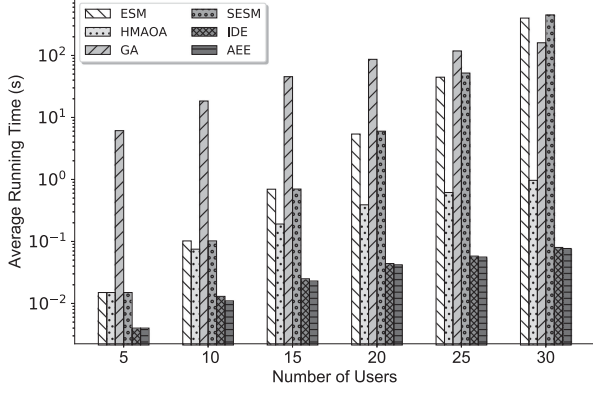


Fig. 4. Average running time of algorithms with different numbers of users.

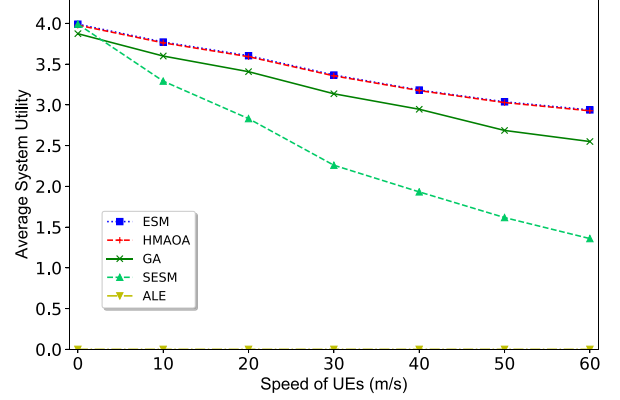


Fig. 6. Average system utility under different user speed when  $N = 20$ .

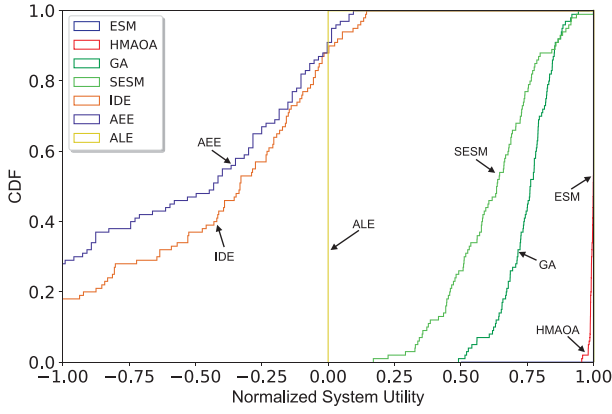


Fig. 5. Variation of the performance of different methods when  $N = 30$ .

high complexity. To find the optimal solution, a large population size and many generations are necessary. Since the algorithm logic of IDE and AEE are simple, their running times are negligible. The running time of HMAOA increases with  $N$ . Thanks to its polynomial complexity ( $O(N^3 \times (F_0/step))$ ), its running time is less than 1 second even when  $N = 30$ .

*Variation of System Utility:* The performance of each offloading method is analyzed across different simulation runs. In each simulation run, the system utility of each method is normalized, which is presented as a ratio of the system utility to the optimal one obtained by ESM. Obviously, this ratio cannot be greater than 1, and the method with a higher ratio performs better in each simulation run. We adopt the cumulative distribution function (CDF) to show the distribution of the ratios across all the simulation runs when  $N = 30$ , as shown in Fig. 5. Because the system utility of ALE is always equal to 0, its CDF curve shows a vertical line, i.e.,  $x = 0$ . When  $N = 30$ , because of the heavy resource contention, IDE and AEE nearly always produce negative system utility (only about 10% of the outcome is positive). Since AEE offloads all the tasks, its performance is the worst among all methods. In the worst case, SESM and GA achieve 17.0% and 49.1% of the optimal system utility, respectively. HMAOA shows its significant performance and excellent stability. It can achieve 95.6% of the optimal system utility even in the worst case. Its curve is very close to  $x = 1$ ,

i.e., the curve of ESM (the normalized system utility of ESM is always 1).

*Average System Utility With Varying User Speed:* Fig. 6 shows the relation between the average system utility and user speed under different algorithms. The number of UEs is fixed, i.e.,  $N = 20$ , and the speed of all users varies from 0 m/s (i.e., stationary) to 60 m/s. The average system utility of all the algorithms decreases as the speed of UEs increases because many tasks that can contribute higher system utility cannot be offloaded any more considering the mobility constraint. ESM always brings the highest system utility, and the performance of HMAOA is almost the same as it. The curve of GA is lower than HMAOA, as well as ESM. We omit the curves of IDE and AEE in Fig. 6 since their average system utility is always negative when  $N = 20$ , even worse than ALE (always equals to 0). The performance of SESM is the same as ESM when UEs are stationary. As the speed of a UE grows, the probability of it leaving the BS's coverage region before the completion of its offloaded task increases. Without considering this situation, SESM may offload some inappropriate tasks, leading to 0 user utility. Its performance, hence, drops rapidly with the increase of user speed comparing with other methods.

*Performance of HMAOA With Varying Bound Parameters:* The performance of HMAOA is affected by two bound parameters, i.e.,  $\Psi_l$  and  $\Psi_h$ . As introduced before, the value ranges of  $\Psi_l$  and  $\Psi_h$  are  $(0, 1]$ , and the less  $\Psi_l$  or  $\Psi_h$  is, the more change points is ignored. In this simulation, four pairs of bound parameters are chosen, from  $(\Psi_l = 1, \Psi_h = 1)$  to  $(\Psi_l = 0.5, \Psi_h = 0.3)$ . Note that, when both of them are set to be 1, the comparison of tasks degenerates into the comparison of areas under their utility functions. Fig. 7 shows the effects of the two bound parameters to the performance of HMAOA. When  $N \leq 15$ , the HMAOA with higher bound parameters performs slightly better. That is because when the number of users is small, the resource allocation varies greatly, and it is not reasonable to ignore too many change points. When the number of users becomes larger, e.g.,  $N \geq 20$ , the severe resource competition makes it unlikely that one UE is allocated with too many resources, and the resource allocation among UEs tends to be even. In this case, a lower bound parameter can achieve a better result. In practical



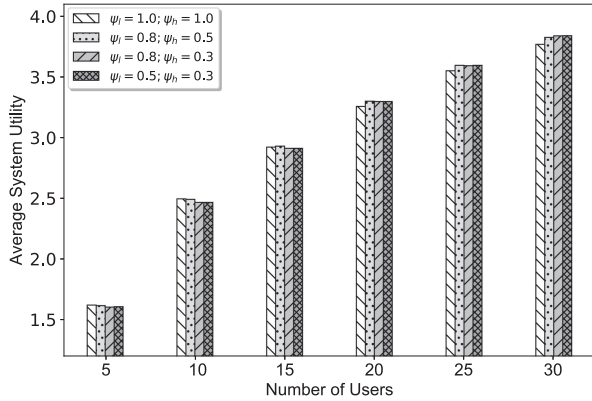


Fig. 7. The performance of HMAOA with varying bound parameters.

situations, we can adjust the bound parameters with the variation of user numbers to achieve higher performance.

## VI. RELATED WORKS

The first technique to enable MEC is cloudlet proposed in 2009, which deploys the cloud infrastructure to the network edge, bringing satisfactory user experience [2], [7]. To integrate the MEC concept into the mobile network architecture, the European Telecommunications Standards Institute (ETSI) makes a lot of contributions, and many standards regarding ETSI MEC are published [4].

As introduced before, offloading decision-making in MEC is a challenging problem and receives extensive attention. Many previous studies assume a quasi-static scenario when making offloading decisions. In [11], Lin *et al.* present a three-step algorithm to minimize energy dissipation of a directed acyclic graph-based application while meeting the application deadline. Mao *et al.* propose an offloading algorithm to minimize the task execution delay on a device with energy harvesting and frequency scaling function [37]. Reference [38] investigates the offloading problem of a full granularity application to minimize the energy consumption of the application.

When it comes to multi-user scenarios, how to allocate the limited resources at the network edge becomes a challenge. Reference [15] tries to maximize the system utility of multiple users by jointly optimizing their transmission power and the computation resource allocation in the BS. Chen *et al.* propose a game theory-based algorithm to make the offloading decision among multiple users in a wireless contention environment to minimize the system-wide overhead [17], [18]. The authors in [39] study the task offloading and resource allocation in a vehicular scenario, where server selection, task partition, and computation resource allocation are jointly considered.

Other than offloading decision and resource allocation, mobility is also an important research topic in MEC. There are many methods to deal with user mobility from different perspectives. Service migration is widely researched to ensure service continuity and availability by migrating a service entity closer to its user. For example, Sun *et al.* propose a novel service placement algorithm for a cloudlet network architecture to optimize the tradeoff between migration gain and migration cost [21]. Since

the cost of migrating the virtual disk of a service entity is high, the authors in [22] propose to place multiple virtual disk replicas of a service entity into suitable cloudlets and only migrate the service entity among these cloudlets to reduce the average service delay. Path selection tries to optimize the data delivery paths for task offloading. In [23], the authors propose an algorithm to select the optimal path for offloading data delivery between UE and MEC servers to minimize the transmission delay. Many studies use fine-grained control to ensure the success of a specific offloading task during user movement. Reference [24] offers a cloud-aware power control algorithm to adjust the transmission power of the BS to prevent the handover from happening before the offloading result received. In [25], we propose a GA-based computation allocation algorithm to ensure the success of task offloading before a handover occurs.

In this paper, we investigate offloading decision and resource allocation among multiple UEs to achieve the optimal system utility. Due to the mobility of UEs, sufficient resources must be allocated to ensure the completion of the offloading process within BS's coverage region.

## VII. CONCLUSION

As a key technology of Internet of things, MEC enables UEs to support resource-intensive applications with significantly lower latency and less energy consumption. In this paper, we focus on the challenging problem of offloading decision and resource allocation among multiple moving UEs in MEC. A heuristic mobility-aware offloading algorithm is proposed to maximize the system utility under the constraints of user mobility, resource limitation, and task latency. The proposed HMAOA is with polynomial complexity. Extensive simulation results demonstrate that it can achieve offloading performance very close to the optimal solution, but with high efficiency.

## APPENDIX PROOF OF THEOREM 1

Here, we first introduce 0-1 knapsack problem: there is a knapsack, whose maximum weight capacity is  $W_0$ , and a set of items  $\mathbf{I}$ . Each item in the set has a weight  $w_n$  and a value  $v_n$ . The objective is to maximize the sum of the values of the items packed in the knapsack so that the sum of the weights is less than or equal to the knapsack's capacity. It can be formulated as follows:

$$\begin{aligned} & \max_{\mathbf{O}} \sum_{n \in \mathbf{O}} v_n \\ & \text{s.t. } \mathbf{O} \subseteq \mathbf{I} \\ & \sum_{n \in \mathbf{O}} w_n \leq W_0, \end{aligned}$$

where  $\mathbf{O}$  is the item set decided for packing. The 0-1 knapsack problem is known as NP-hard [33].

Let  $\mathbf{I} = \mathbf{N}_3$ ,  $w_n = k_n$ ,  $v_n = \varrho_n - \varsigma_n/f_n$  and  $O^{\max} = |\mathbf{I}|$ , respectively. The 0-1 knapsack problem is reduced to a special case of **LP2**, in which constraint (38) is  $k_n \leq f_n \leq k_n$ . Therefore, if there is an algorithm to solve **LP2**, we can use it to solve the 0-1

knapsack problem. Hence, problem **LP2** is NP-hard. Since **GP** is polynomial-time reducible to **LP2**, the original problem **GP** is also NP-hard.

APPENDIX B  
PROOF OF LEMMA 1

$\forall n, m \in \mathbf{N}_3$ , if  $OT_n \succ OT_m$ , then  $k_n < k_m$ ,  $Util_n(k_m) > Util_m(k_m)$ , and  $Util_n(F_0) > Util_m(F_0)$ . We have

$$\begin{cases} (\varrho_n - \varrho_m)k_m > \varsigma_n - \varsigma_m \\ (\varrho_n - \varrho_m)F_0 > \varsigma_n - \varsigma_m. \end{cases}$$

There are three different cases to be considered:

- When  $\varrho_n > \varrho_m$ ,  $k_m \leq f \leq F_0 \Rightarrow (\varrho_n - \varrho_m)F_0 \geq (\varrho_n - \varrho_m)f \geq (\varrho_n - \varrho_m)k_m > \varsigma_n - \varsigma_m$ . Hence, when  $\varrho_n > \varrho_m$ ,  $Util_n(f) > Util_m(f)$ ,  $\forall f \in [k_m, F_0]$ ;
- When  $\varrho_n = \varrho_m$ ,  $(\varrho_n - \varrho_m)k_m > \varsigma_n - \varsigma_m \Rightarrow \varsigma_n - \varsigma_m < 0$ , and  $k_m \leq f \leq F_0 \Rightarrow (\varrho_n - \varrho_m)f = 0 > \varsigma_n - \varsigma_m$ . Hence, when  $\varrho_n = \varrho_m$ ,  $Util_n(f) > Util_m(f)$ ,  $\forall f \in [k_m, F_0]$ ;
- When  $\varrho_n < \varrho_m$ ,  $(\varrho_n - \varrho_m)F_0 > \varsigma_n - \varsigma_m \Rightarrow \varsigma_n - \varsigma_m < 0$ . Meanwhile,  $k_m \leq f \leq F_0 \Rightarrow (\varrho_n - \varrho_m)k_m \geq (\varrho_n - \varrho_m)f \geq (\varrho_n - \varrho_m)F_0 > \varsigma_n - \varsigma_m$ . Hence, when  $\varrho_n < \varrho_m$ ,  $Util_n(f) > Util_m(f)$ ,  $\forall f \in [k_m, F_0]$ .

To sum up, the lemma is thus proved.

APPENDIX C  
PROOF OF LEMMA 2

- 1)  $\forall n \in \mathbf{N}_3$ , we have  $OT_n \not\succeq OT_n$ .
- 2)  $\forall n, m \in \mathbf{N}_3$ ,  $OT_n \succ OT_m \Rightarrow OT_m \not\succeq OT_n$ .
- 3)  $\forall n, m, l \in \mathbf{N}_3$ , if  $OT_n \succ OT_m$ , then  $k_n < k_m$ ,  $Util_n(k_m) > Util_m(k_m)$ , and  $Util_n(F_0) > Util_m(F_0)$ . If  $OT_m \succ OT_l$ , then  $k_m < k_l$ ,  $Util_m(k_l) > Util_l(k_l)$ , and  $Util_m(F_0) > Util_l(F_0)$ . Hence,  $k_n < k_m < k_l$  and  $Util_n(F_0) > Util_m(F_0) > Util_l(F_0)$ . Furthermore, with Lemma 1,  $OT_n \succ OT_m \Rightarrow Util_n(f) > Util_m(f)$ ,  $\forall f \in [k_m, F_0]$ , and  $OT_m \succ OT_l \Rightarrow Util_m(f) > Util_l(f)$ ,  $\forall f \in [k_l, F_0]$ . Because of  $[k_m, F_0] \supset [k_l, F_0]$ , we can obtain  $\forall f \in [k_l, F_0]$ ,  $Util_n(f) > Util_m(f) > Util_l(f)$ .

The lemma is thus proved.

APPENDIX D  
PROOF OF LEMMA 3

We can use any sorting algorithm to build  $Seq_{\mathbf{N}_3}$ , denoted as  $Seq_{\mathbf{N}_3} = (OT'_1, OT'_2, \dots, OT'_{|\mathbf{N}_3|})$ , where  $OT'_1 \succ OT'_2 \succ \dots \succ OT'_{|\mathbf{N}_3|}$ . The proof is by contradiction. Assume that the optimal offloading decision set, which brings the highest system utility, is not one of  $Seq_{\mathbf{N}_3}$ 's head-subsequences. We can firstly sort the set, and rewrite it into a subsequence of  $Seq_{\mathbf{N}_3}$  as  $Seq^{\max} = (OT''_1, OT''_2, \dots, OT''_M)$ , where  $M$  denotes the cardinality of the offloading decision set. Algorithm 2 obtains the optimal system utility  $Func(Seq^{\max})$  and the optimal frequency allocation  $\mathbf{f}^{\max} = (f''_1, f''_2, \dots, f''_M)$ . Because  $Seq^{\max}$  is not one of  $Seq_{\mathbf{N}_3}$ 's head-subsequences, i.e.,  $Seq^{\max} \neq head(Seq, M)$ , there should be at least one item in  $head(Seq, M)$ , which is

not in  $Seq^{\max}$ , and we denote any one of them as  $OT'_x$ . We also can derive that the last item in  $Seq^{\max}$ ,  $OT''_M$ , is not in  $head(Seq, M)$ . Obviously,  $OT'_x \succ OT''_M$ , hence  $k_{OT'_x} < k_{OT''_M}$  and  $Util_{OT'_x}(f) > Util_{OT''_M}(f)$ ,  $\forall f \in [k_{OT''_M}, F_0]$ . Replacing  $OT''_M$  in  $Seq^{\max}$  with  $OT'_x$ , a higher system utility is obtained since  $Util_{OT'_x}(f''_M) > Util_{OT''_M}(f''_M)$ . However,  $Seq^{\max}$  is already the optimal offloading decision, which leads to a contradiction. Therefore, the optimal offloading decision set is one of  $Seq_{\mathbf{N}_3}$ 's head-subsequences.

APPENDIX E  
PROOF OF LEMMA 4

Assume that the optimal offloading decision set, which brings the highest system utility, is  $\mathbf{O}$ , and the rest tasks in  $\mathbf{N}_3$  form a set  $\overline{\mathbf{O}}$ . We have  $\overline{\mathbf{O}} \cup \mathbf{O} = \mathbf{N}_3$  and  $\overline{\mathbf{O}} \cap \mathbf{O} = \emptyset$ . This lemma will be proved in two steps.

- $\forall OT_i \in \overline{\mathbf{O}}$  and  $\forall OT_j \in \mathbf{O} \Rightarrow OT_i \not\succeq OT_j$ .

The proof is by contradiction. Assume that there exist  $OT_i \in \overline{\mathbf{O}}$  and  $OT_j \in \mathbf{O}$ , satisfying  $OT_i \succ OT_j$ . Then, we can replace  $OT_j$  in  $\mathbf{O}$  with  $OT_i$  to bring higher system utility. However,  $\mathbf{O}$  is the optimal offloading decision set, and the utility cannot be higher, which leads to a contradiction. Hence, the proposition is thus proved.

- $\mathbf{O}$  can be rewritten as a head-subsequence of one of sorted linear extension of  $(\mathbf{N}_3, \succ)$ .

This proposition is constructively provable. Since  $\mathbf{O} \subseteq \mathbf{N}_3$  and  $\overline{\mathbf{O}} \subset \mathbf{N}_3$ ,  $(\mathbf{O}, \succ)$  and  $(\overline{\mathbf{O}}, \succ)$  are posets. We can apply any topological sorting algorithm to linearly extend  $(\mathbf{O}, \succ)$  and  $(\overline{\mathbf{O}}, \succ)$ , and turn them into  $Seq_{\mathbf{O}}$  and  $Seq_{\overline{\mathbf{O}}}$ . Then, we combine the two sequences into  $Seq_{\mathbf{N}_3} = (Seq_{\mathbf{O}}, Seq_{\overline{\mathbf{O}}})$ . According to the first proposition ( $\forall OT_i \in \overline{\mathbf{O}}$  and  $\forall OT_j \in \mathbf{O} \Rightarrow OT_i \not\succeq OT_j$ ), we know  $Seq_{\mathbf{N}_3}$  is sorted. So we can build  $(\mathbf{N}_3, \succ^*)$  based on  $Seq_{\mathbf{N}_3}$ , which is a totally ordered set, and  $\forall OT_i, OT_j \in (\mathbf{N}_3, \succ)$ ,  $OT_i \succ OT_j \Rightarrow OT_i \succ^* OT_j$ , and  $OT_i, OT_j \in (\mathbf{N}_3, \succ^*)$ . That means  $Seq_{\mathbf{N}_3} = (Seq_{\mathbf{O}}, Seq_{\overline{\mathbf{O}}})$  is a sorted linear extension of  $(\mathbf{N}_3, \succ)$ , and  $Seq_{\mathbf{O}}$  is the head-subsequence.

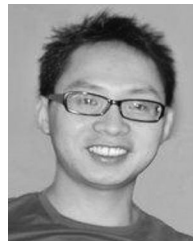
REFERENCES

- [1] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, "Heterogeneity in mobile cloud computing: Taxonomy and open challenges," *IEEE Commun. Surv. Tut.*, vol. 16, no. 1, pp. 369–392, Jan.–Mar. 2014.
- [2] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct.–Dec. 2009.
- [3] T. K. Rodrigues, K. Suto, H. Nishiyama, J. Liu, and N. Kato, "Machine learning meets computation and communication control in evolving edge and cloud: Challenges and future perspective," *IEEE Commun. Surv. Tut.*, to be published, doi: [10.1109/COMST.2019.2943405](https://doi.org/10.1109/COMST.2019.2943405).
- [4] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—A key technology towards 5G," White Paper, European Telecommunications Standards Institute, Sep. 2015.
- [5] Y.-Y. Shih, W.-H. Chung, A.-C. Pang, T.-C. Chiu, and H.-Y. Wei, "Enabling low-latency applications in fog-radio access networks," *IEEE Netw.*, vol. 31, no. 1, pp. 52–58, Jan./Feb. 2016.
- [6] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.
- [7] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surv. Tut.*, vol. 19, no. 3, pp. 1628–1656, Jul.–Sep. 2017.

- [8] 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 Trans. Comput.*, vol. 66, no. 5, pp. 810–819, May 2016.
- [9] T. G. Rodrigues, K. Suto, H. Nishiyama, N. Kato, and K. Temma, "Cloudlets activation scheme for scalable mobile edge computing with transmission power control and virtual machine migration," *IEEE Trans. Comput.*, vol. 67, no. 9, pp. 1287–1300, Sep. 2018.
- [10] A.-C. Pang, W.-H. Chung, T.-C. Chiu, and J. Zhang, "Latency-driven cooperative task computing in multi-user fog-radio access networks," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, Atlanta, USA, Jun. 2017, pp. 615–624.
- [11] X. Lin, Y. Wang, Q. Xie, and M. Pedram, "Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment," *IEEE Trans. Services Comput.*, vol. 8, no. 2, pp. 175–186, Mar./Apr. 2015.
- [12] W. Zhang, Y. Wen, and D. O. Wu, "Collaborative task execution in mobile cloud computing under a stochastic wireless channel," *IEEE Trans. Wireless Commun.*, vol. 14, no. 1, pp. 81–93, Jan. 2015.
- [13] 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 Trans. Commun.*, vol. 66, no. 6, pp. 2603–2616, Jun. 2018.
- [14] H. Guo, J. Liu, J. Zhang, W. Sun, and N. Kato, "Mobile-edge computation offloading for ultradense IoT networks," *IEEE Internet Things J.*, vol. 5, no. 6, pp. 4977–4988, Dec. 2018.
- [15] X. Lyu, H. Tian, C. Sengul, and P. Zhang, "Multiuser joint task offloading and resource optimization in proximate clouds," *IEEE Trans. Veh. Technol.*, vol. 66, no. 4, pp. 3435–3447, Apr. 2017.
- [16] H. Cao and J. Cai, "Distributed multiuser computation offloading for cloudlet-based mobile cloud computing: A game-theoretic machine learning approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 752–764, Jan. 2018.
- [17] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 974–983, Apr. 2015.
- [18] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.
- [19] J. Du, F. R. Yu, X. Chu, J. Feng, and G. Lu, "Computation offloading and resource allocation in vehicular networks based on dual-side cost minimization," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1079–1092, Feb. 2019.
- [20] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Comput.*, vol. 4, no. 2, pp. 26–35, Mar./Apr. 2017.
- [21] X. Sun and N. Ansari, "Primal: Profit maximization avatar placement for mobile edge computing," in *Proc. IEEE Int. Conf. Commun.*, Kuala Lumpur, Malaysia, May, 2016, pp. 1–6.
- [22] X. Sun and N. Ansari, "Adaptive avatar handoff in the cloudlet network," *IEEE Trans. Cloud Comput.*, vol. 7, no. 3, pp. 664–676, Jul.–Sep. 2019.
- [23] J. Plachy, Z. Becvar, and P. Mach, "Path selection enabling user mobility and efficient distribution of data for computation at the edge of mobile network," *Comput. Netw.*, vol. 108, pp. 357–370, 2016.
- [24] P. Mach and Z. Becvar, "Cloud-aware power control for real-time application offloading in mobile edge computing," *Trans. Emerg. Telecommun. Technol.*, vol. 27, no. 5, pp. 648–661, 2016.
- [25] W. Zhan, H. Duan, and Q. Zhu, "Multi-user offloading and resource allocation for vehicular multi-access edge computing," in *Proc. IEEE Int. Conf. Ubiquitous Comput. Commun.*, Shenyang, China, Oct. 2019, pp. 50–57.
- [26] A. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, "A survey of mobile cloud computing application models," *IEEE Commun. Surv. Tut.*, vol. 16, no. 1, pp. 393–413, Jan.–Mar. 2013.
- [27] G. Xie, H. Gao, L. Qian, B. Huang, K. Li, and J. Wang, "Vehicle trajectory prediction by integrating physics-and maneuver-based approaches using interactive multiple models," *IEEE Trans. Ind. Electron.*, vol. 65, no. 7, pp. 5999–6008, Jul. 2018.
- [28] A. Goldsmith, *Wireless Communications*. New York, NY, USA: Cambridge Univ. Press, 2005.
- [29] L. Wei, R. Q. Hu, Y. Qian, and G. Wu, "Energy efficiency and spectrum efficiency of multihop device-to-device communications underlying cellular networks," *IEEE Trans. Veh. Technol.*, vol. 65, no. 1, pp. 367–380, Jan. 2016.
- [30] Y. Pochet and L. A. Wolsey, *Production Planning by Mixed Integer Programming*. Berlin, Germany: Springer, 2006.
- [31] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*. New York, NY, USA: Springer, 2012.
- [32] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
- [33] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, Boston, MA, USA: Springer, 1972, pp. 85–103.
- [34] G. Schmidt, *Relational Mathematics*. New York, NY, USA: Cambridge Univ. Press, 2011.
- [35] G. Brightwell and P. Winkler, "Counting linear extensions," *Order*, vol. 8, no. 3, pp. 225–242, 1991.
- [36] "Further advancements for E-UTRA physical layer aspects," 3rd Generation Partnership Project, Sophia Antipolis, France, Tech. Rep. 3GPP TR 36.814 V9.0.0, Mar. 2010.
- [37] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.
- [38] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, Oct. 2016.
- [39] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint load balancing and offloading in vehicular edge computing and networks," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4377–4387, Jun. 2019.



**Wenhan Zhan** received the B.E. and M.Sc. degrees in 2010 and 2013, respectively, from the University of Electronic Science and Technology of China (UESTC), Chengdu, China, where he is currently working toward the Ph.D. degree. Since 2013, he has been an Experimentalist with UESTC. From 2018 to 2019, he was a Visiting Scholar with the Department of Computer Science, University of Exeter, U.K. His research interests mainly lie in distributed system, cloud computing, edge computing, and artificial intelligence.



**Chunbo Luo** (Member, IEEE) received the Ph.D. degree in high performance cooperative wireless networks from the University of Reading, Reading, U.K. His research interests focus on developing model-based and machine learning algorithms to solve networking and engineering problems, such as wireless networks, with a particular focus on unmanned aerial vehicles. His research has been supported by NSFC, Royal Society, EU H2020, and industries. He is a fellow of the Higher Education Academy.



**Geyong Min** received the B.Sc. degree in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 1995, and the Ph.D. degree in computing science from the University of Glasgow, Glasgow, U.K., in 2003. He is currently a Professor of high-performance computing and networking with the Department of Computer Science, College of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter, U.K. His research interests include future internet, computer networks, wireless communications, multimedia systems, information security, high-performance computing, ubiquitous computing, modeling, and performance engineering.



**Chao Wang** (Member, IEEE) received the B.E. degree from the University of Science and Technology of China, Hefei, China, in 2003, and the M.Sc. and Ph.D. degrees from the University of Edinburgh, Edinburgh, U.K., in 2005 and 2009, respectively. From 2009 to 2012, he was a Postdoctoral Research Associate with the KTH-Royal Institute of Technology, Stockholm, Sweden. Since 2013, he has been with Tongji University, Shanghai, China, where he is currently an Associate Professor. He is currently taking a Marie Skłodowska-Curie Individual Fellowship with

the University of Exeter, U.K. His research interests include information theory and signal processing for wireless communication networks, as well as data-driven research and applications for smart city and intelligent transportation systems.



**Hancong Duan** received the B.Sc. degree from Southwest Jiaotong University, Chengdu, China, in 1995, and the M.E. and Ph.D. degrees from the University of Electronic Science and Technology of China, Chengdu, China, in 2005 and 2007, respectively. He is currently a Professor of computer science with the University of Electronic Science and Technology of China. His research interests include large-scale P2P content delivery network, distributed storage, cloud computing, and artificial intelligence.



**Qingxin Zhu** received the Ph.D. degree in cybernetics from the University of Ottawa, Ottawa, ON, Canada, in 1993. From 1993 to 1996, he was a Postdoctoral Researcher with the Department of Electronic Engineering, University of Ottawa and the School of Computer Science, Carleton University, Canada. He was a Senior Researcher of Nortel and Omnimark in Canada from 1996 to 1997. Since 1998, he has been with the University of Electronic Science and Technology of China, Chengdu, China, as a Professor and the Director of Operations Research

Office. From 2002 to 2003, he was a Senior Visiting Scholar with the Computer Department, Concordia University, Montreal, QC, Canada. His research interests include bioinformatics, operational research, and optimization.