An OO-Based Approach of Computing Offloading and Resource Allocation for Large-Scale Mobile Edge Computing Systems

Yufu Tan, Sikandar Ali, Haotian Wang, and Jiwei Huang(✉)

Beijing Key Laboratory of Petroleum Data Mining, China University of Petroleum - Beijing, Beijing 102249, China
{2019211243,2018011122}@student.cup.edu.cn,
{sikandar,huangjw}@cup.edu.cn

Abstract. Mobile edge computing (MEC) is an emerging paradigm to meet the increasing real-time performance demands for Internet of Things and mobile applications. By offloading the computationally intensive workloads to edge servers, the quality of service (QoS) could be greatly improved. However, with the growing popularity of MEC, the MEC systems grow extremely large, and thus the QoS optimization suffers from search space explosion problem, making it impractical in real-life scenarios. To attack this challenge, this paper studies the joint optimization of task offloading and computational resource allocation for large-scale MEC systems. We formulate this problem as a cost minimization problem and illustrate the NP-hardness of this problem. In order to solve this problem, we divide the original problem into two sub-problems and introduce the theory of Ordinal Optimization (OO) to search for a near-optimal computing offloading and resource allocation policy within a significantly reduced search space. Finally, the efficacy of our approach is validated by simulation experiments.

Keywords: Mobile Edge Computing (MEC) · Computing offloading · Resource allocation · Ordinal optimization · Large-scale MEC systems

1 Introduction

With the rapid development of Internet of Things (IoT) and Mobile Internet, the amount of mobile data traffic and the number of mobile devices have surged. According to the forecast [1], by 2022, mobile will account for 20% of total IP traffic and the world's mobile data traffic will almost reach to one zettabyte. Moreover, there will be 12.3 billion mobile devices including M2M modules by 2022, which outnumber the estimated global population (8 billion) at that moment by 1.5 times. At the same time, more and more innovative applications

H. Gao and X. Wang (Eds.): CollaborateCom 2021, LNICST 407, pp. 65–83, 2021.
https://doi.org/10.1007/978-3-030-92638-0_5

are emerging, such as Augmented Reality (AR) [2], Natural Language Processing (NLP) [3], Virtual Reality (VR) [4] and self-driving [5]. These applications are all computing-intensive, time-sensitive, and energy-intensive. However, due to limited resources of the mobile devices (i.e., computational resources and battery capacity), these applications are difficult to achieve a satisfactory experience for users. Therefore, due to the contradiction between the huge demand for computational resources as well as energy consumption of these emerging applications and limited resources of mobile devices, the advance of future mobile platforms becomes a major challenge.

Mobile Cloud Computing (MCC) is a traditional architecture which is deemed to be able to address the above challenges [6]. Mobile devices are able to offload computationally intensive workloads to a central cloud for execution by MCC. Without sacrificing the mobility and convenience of mobile devices, the rich computing resources of the central cloud can be utilized to enhance the support of mobile devices for these emerging applications. However, MCC also introduces high latency as data has to be transferred to remote cloud servers for processing [7].

Mobile Edge Computing (MEC) is an innovative paradigm designed to offer an IT service environment as well as cloud computing capabilities at the edge of the mobile network that is geographically close to mobile users [8]. On the one hand, like MCC, MEC is able to provide rich computational resources for mobile devices. On the other hand, MEC can achieve less latency as well as network load and offer an improved user experience compared to MCC.

Since the era of cloud computing, task scheduling and resource management have always been a high-profile issue in the academic as well as industrial community [9]. Furthermore, as one of the most important issues in MEC, how to get an excellent task offloading policy which decides whether a task is executed locally or offloaded to the edge server has attracted attention in recent years. A majority of the existing researches focus on the optimization of latency or energy consumption.

In [10], the authors studied a single-user MEC system that allows parallel processing of computationally intensive tasks at the local device as well as at the edge server to reduce the average latency of each task. In order to find the optimal offloading policy, they presented a one-dimensional search policy based on the theory of Markov Decision Process. Numerical results shown that the proposed policy is significantly better than the greedy scheduling policy. For multi-user MEC system, Kan *et al.* [6] formulated the multi-user offloading decision as a cost minimization problem. By classifying and prioritizing mobile devices, they developed a heuristic algorithm considering both computation and radio resource allocation. Furthermore, they also considered the variety of latency requirements of tasks. Mao *et al.* [11] considered a multi-user MEC system where tasks arrive stochastically. They formulated a power consumption minimization problem with task buffer stability constraints. Based on the theory of Lyapunov optimization, they developed an online algorithm to decide the CPU-cycle frequencies for execution locally as well as the transmission power and bandwidth allocation

for execution remotely. Numerical results showed that the proposed algorithm is capable to keep the balance between the quality of computation experience and the energy consumption of mobile devices. In a dynamic MEC system, it is very challenging to find an excellent offloading policy. To attack this problem, the theory of Reinforcement Learning (RL) has been introduced [12]. Li *et al.* [13] presented an offloading method based on RL in a multi-user MEC system. In the beginning, they developed an offloading method based on Q-learning algorithm. But numerical results shown that the number of possible actions may surge as the users increases. In that case, the action-value Q would be too complicated to be computed and stored. Therefore, to solve this problem, they presented to use a Deep Q-Network (DQN) to estimate the Q-table. Simulation results shown that, under different system parameters, the proposed methods perform better than other two methods. In our previous work [14], we studied the problems of dynamic task scheduling and resource management in MEC system to maximize revenue of the edge service providers. While the discrete nature of our problem makes it is formulated as an integer programming (IP) problem, which is NP-hard. To tackle this problem, we researched the totally unimodular constraints of the problem, which help us convert the original problem into a linear programming (LP) problem. Then, experiments was conducted to verify this approach.

All of the above studies just focused on optimizing the execution latency or the energy consumption of mobile devices while neglecting the energy consumption of the edge servers. However, reducing the energy consumption of edge servers is critical because it not only reduces carbon dioxide emissions but also cuts down the costs for service providers. Furthermore, the Fifth Generation Mobile Communication Technology has become a research hotspot in the communications industry and academia in recent years. In order to achieving seamless coverage, it is necessary to densely deploy vast quantities of small cells. Consequently, the ultra-dense cellular network becomes a core feature of 5G cellular networks [15]. Therefore, it is essential to focus on multi-user computing offloading in the ultra-dense network. Chen *et al.* [16] propose a novel framework of task offloading for MEC in ultra-dense networks. They studied the problem of computational offloading in ultra-dense network to optimize the average latency and save the battery of mobile devices. They propose an efficient offloading policy, however, the final offloading policy is too dependent on the given initial task placement policy.

This paper proposes a method based on Ordinal Optimization (OO) to minimize latency and energy consumption of edge servers in large-scale MEC systems. The main contributions of the work are listed as follows.

– We propose large-scale MEC systems offloading problem, which aims to minimize the execution latency and energy consumption of edge servers in the ultra-dense network with multiple users by joint optimization of offload policy and computation resource allocation. Further, the problem is formulated as an NP-hard problem.

– In order to attack the above NP-hard problem, the original problem is divided into two sub-problem: i.e., the task placement problem and the resource allocation problem. Furthermore, we obtain the closed-form solution of the optimal resource allocation under an arbitrary given task placement policy.
– We introduce the theory of Ordinal Optimization intending to get a near-optimal task placement policy. Then, an efficient offloading algorithm is proposed and its efficiency is verified.

The rest of the paper is organized as follows. Section 2 describes the system model definition and the problem formulation. In Sect. 3, we develop a near-optimal offloading policy based on the theory of Ordinal Optimization. In Sect. 4, we compare the performance of our novel policy with three baseline policies. Finally, Sect. 5 concludes the paper.

2 System Model and Problem Formulation

In this section, we first introduce the large-scale MEC systems. Then, we formulate the problem and show that this problem is NP-hard. The scenario, we are considering is shown in Fig. 1. In the large-scale MEC systems, we assume that there exists n wireless base station, labeled as $\mathcal{B} = \{b_1, b_2, ..., b_n\}$. Each base station is equipped with an edge server, and thus we can also use $\mathcal{B}$ to denote the set of edge servers. We denote the set of mobile users as $\mathcal{U} = \{u_1, u_2, ...u_m\}$, and consider that each mobile user u_i has a computation task $T_i = (\omega_i, s_i)$ that can be processed locally or offloaded to one of the edge servers via the wireless channel. Similarly, we also use $\mathcal{U}$ to denote the set of mobile devices. Here, ω_i denotes the CPU cycles required to accomplish the task T_i, and s_i represents the data size of task T_i. Each server can provide services to multiple users within its signal coverage. Naturally, each user also can offload computation task to any edge server whose signal coverage includes the user's location. The system model denote $\mathcal{A}(u_i)$ as the set of edge servers that can provide services for the user u_i.

2.1 Communication Model

When the computation task of mobile user u_i is offloaded to edge server b_j, the uplink data rate can be expressed as follows:

$$r_{i,j} = B \log_2(1 + \frac{p_i^T g_{i,j}}{\sigma^2}) \tag{1}$$

here B is channel bandwidth, p_i^T is the transmission power of user u_i and σ^2 is noise power of the mobile device. Since the duration of data transmission is short, we assume that the users are not moving during the task offloading. So the channel gain between user u_i and base station b_j which is denoted as $g_{i,j}$

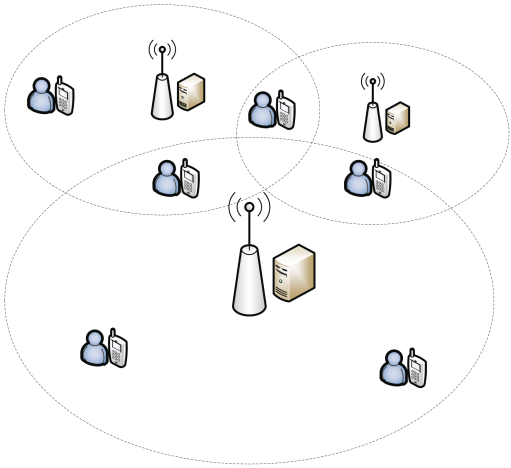


Fig. 1. Model of the large-scale MEC systems

can be regarded as a constant. Therefore, the transmission latency from mobile device u_i to base station b_j can be expressed as follows:

$$t_{i,j}^{T} = \frac{s_i}{r_{i,j}} \tag{2}$$

Furthermore, the transmission energy consumption of mobile device u_i can be also attained as follows:

$$\epsilon_{i,j}^{T} = p_i^{T} t_{i,j}^{T} \tag{3}$$

2.2 Computing Model

It is obvious that, task can be executed locally or remotely. Therefore, the two models are discussed separately.

Local Computing. For local computing, the computation capability of mobile device u_i is defined as f_i^L (cycles/second). Therefore, the local latency of task T_i is:

$$t_i^L = \frac{\omega_i}{f_i^L} \tag{4}$$

Moreover, the computation energy consumption for processing locally can be obtain as:

$$\epsilon_i^L = p_i^L t_i^L \tag{5}$$

where p_i^L is the computation power of mobile device u_i.

Table 1. Notation table

Symbol	Example
n	The number of base stations (edge servers)
m	The number of mobile users (tasks)
$\mathcal{B}$	The set of base stations (edge servers)
b_j	j^{th} base station (edge server)
$\mathcal{U}$	The set of mobile users (mobile devices)
u_i	i^{th} mobile user (mobile device)
T_i	i^{th} task
s_i	The data size of T_i
ω_i	The CPU cycles of T_i
$\mathcal{A}(u_i)$	The set of edge server that can provide services to mobile user u_i
$r_{i,j}$	The uplink data rate between u_i and b_j
σ^2	The noise power of the mobile device
B	The channel bandwidth
$g_{i,j}$	The channel gain between u_i and b_j
$t_{i,j}^T$	The transmission latency from u_i to b_j
t_i^L	The local computing latency of u_i
p_i^T	The transmission power of u_i
p_i^L	The computing power of u_i
$\epsilon_{i,j}^T$	The transmission energy consumption for transferring T_i to transfer to b_j
ϵ_i^L	The local computation energy consumption of T_i
$\mathbf{f}^E$	The set of edge server frequencies
f_j^E	The frequency of edge server b_j
f_i^L	The frequency of mobile device b_i
$\kappa_{j,i}$	The proportion of edge server b_j computation resources allocated to task T_i
p_j^E	The power of edge server b_j
p_j^{max}	The max power of edge server b_j
p_j^{idle}	The idle power of edge server b_j
u_j	The utilization of edge server b_j
$p_{i,j}^E$	The extra power consumed to process task T_i on the server b_j
$\epsilon_{i,j}^E$	The extra energy consumed to process task T_i on the server b_j
E_i^{max}	The total battery capacity of mobile device u_i
α_i	The remainder energy consumption relative to the total battery capacity E_i^{max}

Mobile Edge Computing. The computation resources of edge servers can be denoted as $\boldsymbol{f}^E = (f_1^E, f_2^E, \cdots, f_n^E)$, where f_j^E denotes the computational resource of jth edge server. The latency of task T_i processed on edge server b_j consists of transmission latency and computation latency. Thus, the edge latency can be obtained as follows:

$$t_{i,j}^E = \frac{\omega_i}{\kappa_{j,i} f_j^E} + t_{i,j}^T \tag{6}$$

where $\kappa_{j,i}$ is the proportion of computation resource of edge server b_j allocated to task T_i. It should be noted that the transmission delay of returning processed result from edge server to mobile device is ignored here. Since, the output data

is usually very small and the downlink rate is much higher than uplink rate, therefore, its impact on the overall computation overhead is negligible.

To calculate the power consumed by an edge server b_j, a simple utilization-based model that proved to be very accurate is used [17,18]:

$$p_j^E = (p_j^{max} - p_j^{idle})u_j + p_j^{idle} \tag{7}$$

where u_j denote CPU utilization of the edge server b_j, p_j^{idle} and p_j^{max} are the respective average power values when the edge server b_j is idle or fully utilized. It can be seen from the above equation that p_j^E consists of two parts: The former is the extra power consumed when processing tasks, which is related to the workload of the server b_j. While the latter is the power consumed regardless of whether the server is working, which is irrelevant to the workload of the server. Thus, for the sake of convenience, we only consider the former. Therefore, the extra power of edge server b_j consumed by task T_i is:

$$p_{i,j}^E = (p_j^{max} - p_j^{idle})\kappa_{j,i} \tag{8}$$

Consequently, the energy consumption of the task T_i processed on the edge server b_j is:

$$\epsilon_{i,j}^E = \frac{\omega_i}{\kappa_i^j f_j^E} \cdot P_{i,j}^E = \frac{\omega_i}{f_j^E}(p_j^{max} - p_j^{idle}) \tag{9}$$

(9) indicates that $\epsilon_{i,j}^E$ is irrelevant to $\kappa_{j,i}$. Table 1 shows the notation in this paper.

2.3 Problem Formulation

In this paper, a simple additive weighting (SAW) method to minimize the task latency and the extra energy consumption of edge servers has been incorporated. Therefore, we can formulate the problem as follows:

$$\begin{aligned}
&\min_{\boldsymbol{x},\boldsymbol{\kappa}} f(\boldsymbol{x},\boldsymbol{\kappa}) = \beta_1 \sum_{i=1}^{m}[x_{i,0}t_i^L + \sum_{j=1}^{n} x_{i,j}t_{i,j}^E] + \beta_2 \sum_{i=1}^{m}\sum_{j=1}^{n} x_{i,j}\epsilon_{i,j}^E \\
\text{s.t.}\quad & C1: x_{i,j} \in \{0,1\}, \forall i = 1,2,\cdots,m, \forall j = 0,1,2,...,n \\
& C2: \sum_{j=0}^{n} x_{i,j} = 1, \forall i = 1,2,\cdots,m \\
& C3: 0 \le \kappa_{j,i} \le 1, \forall i = 1,2,\cdots,m, \forall j = 1,2,...,n \\
& C4: 0 \le \sum_{i=1}^{n} x_{i,j}\kappa_{j,i} \le 1, \forall j = 1,2,\cdots,n \\
& C5: x_i\epsilon_i^L \le \alpha_i E_i^{max}, \forall i = 1,2,\cdots,m \\
& C6: (1-x_i)\epsilon_i^T \le \alpha_i E_i^{max}, \forall i = 1,2,\cdots,m \\
& C7: \sum_{j \in \mathcal{A}(u_i) \cup \{u_i\}} x_{i,j} = 1, \forall i = 1,2,\cdots,m
\end{aligned} \tag{10}$$

where $x_{i,0} = 1$ ($x_{i,0} = 0$) represents that task T_i will be computed on local mobile device (edge server), $x_{i,j} = 1$ ($j = 1, 2, \cdots, n$) represents task T_i will be computed on edge server b_j, otherwise, $x_{i,j} = 0$. $\kappa_{j,i}$ represents the computation resources allocated to the task T_i by the edge server b_j, β_1 and β_2 are weight factors, and $\beta_1 + \beta_2 = 1$. The constraint C1 and C2 indicate that tasks can only be computed on the local device or one of the edge servers. The constraint C3 and C4 state that the total computation resources allocated from an edge server must not exceed its computation capacity. The constraint C5 and C6 indicate that the energy consumption for computing and transmission task cannot exceed the remaining energy of the mobile device. The last constraint C7 indicates task can only be computed locally or offloaded to edge servers whose signal coverage includes the user's location. Obviously, (10) is a mixed-integer non-linear programming problem, which is NP-hard [19,20].

3 An Efficient Offloading Algorithm Based on Ordinal Optimization

In this section, we demonstrate an efficient offloading algorithm based on Ordinal Optimization to tackle the problem mentioned in (10). We solve the problem by dividing the original problem into two sub-problems as follows:

- **Computation resource allocation problem**: This problem aims to decide how much computation resource, an edge server allocate to each task offloaded to the edge server.
- **Task placement problem**: This problem aims to decide whether each task will be processed locally or remotely, and if offloaded, to which edge server it will be offloaded.

3.1 Computation Resource Allocation Problem

Theorem 1. *Given $\boldsymbol{x} = \boldsymbol{x}^0$, using $\mathcal{T}(b_j)$ to represent the set of tasks offloaded to the edge server b_j, then the optimal resources allocated by edge server b_j for the ith task is given by $\frac{\sqrt{\omega_i}}{\sum_{m=1}^{|\mathcal{T}(b_j)|}\sqrt{\omega_m}}$.*

Proof. Given $\boldsymbol{x} = \boldsymbol{x}^0$, the objective function of (10) can be expressed as follows:

$$\begin{aligned} f(\boldsymbol{\kappa}) = f(\boldsymbol{x}^0, \boldsymbol{\kappa}) &= \beta_1 \sum_{i=1}^{m} [x_{i,0}^0 \frac{\omega_i}{f_i^L} + \sum_{j=1}^{n} x_{i,j}^0 (\frac{\omega_i}{\kappa_{j,i} f_j^E} + \frac{s_i}{r_{i,j}})] \\ &+ \beta_2 \sum_{i=1}^{m} \sum_{j=1}^{n} x_{i,j}^0 \frac{\omega_i}{f_j^E} (p_j^{max} - p_j^{idle}) \\ &= \beta_1 \sum_{i=1}^{m} \sum_{j=1}^{n} x_{i,j}^0 \frac{\omega_i}{\kappa_{j,i} f_j^E} + C \end{aligned} \tag{11}$$

where $C = \beta_1 \sum_{i=1}^{m} x_{i,0}^0 \frac{\omega_i}{f_i^L} + \beta_1 \sum_{i=1}^{m} \sum_{j=1}^{n} x_{i,j}^0 \frac{s_i}{r_{i,j}} + \beta_2 \sum_{i=1}^{m} \sum_{j=1}^{n} x_{i,j}^0 \frac{\omega_i}{f_j^E} (p_j^{max} - p_j^{idle})$ is a constant. We denote the set of offloaded tasks is $\mathcal{T}$ and the set of edge servers which there are tasks place on is $\mathcal{E}$. For convenience, we assume $\mathcal{T} = \{T_1, T_2, \cdots, T_{|\mathcal{T}|}\}$ and $\mathcal{E} = \{b_1, b_2, \cdots, b_{|\mathcal{E}|}\}$. So (10) can be converted as follows:

$$\begin{aligned} \min_{\boldsymbol{\kappa}} \quad & f(\boldsymbol{\kappa}) = \beta_1 \sum_{i=1}^{|\mathcal{T}|} \frac{\omega_i}{\kappa_{j_i,i} f_{j_i}^E} + C \\ \text{s.t.} \quad & C1 : 0 < \kappa_{j_i,i} \leq 1, \forall i = 1, 2, \cdots, |\mathcal{T}|, b_{j_i} \in \mathcal{E} \\ & C2 : 0 < \sum_{i \in \mathcal{T}(b_j)} \kappa_{j,i} \leq 1, \forall j \in \mathcal{E} \end{aligned} \tag{12}$$

where b_{j_i} represents the edge server to which task T_i will be offloaded, $\mathcal{T}(b_j)$ is the set of tasks offloaded to the edge server b_j. We noticed that the resource allocation of one server does not effect the resource allocation of another server, therefore, the objective function of (12) can be expressed as a combination of n independent minimization problems. Thus, (12) can be transform as follows:

$$\begin{aligned} \min_{\boldsymbol{\kappa}} \quad & f(\boldsymbol{\kappa}) = \beta_1 \min_{\kappa_{1,\cdot}} \sum_{i \in \mathcal{T}(b_1)} \frac{\omega_i}{\kappa_{1,i} f_1^E} + \beta_1 \min_{\kappa_{2,\cdot}} \sum_{i \in \mathcal{T}(b_2)} \frac{\omega_i}{\kappa_{2,i} f_2^E} \\ & + \cdots + \beta_1 \min_{\kappa_{n,\cdot}} \sum_{i \in \mathcal{T}(b_n)} \frac{\omega_i}{\kappa_{n,i} f_n^E} + C \\ \text{s.t.} \quad & C1 : 0 < \kappa_{j,i} \leq 1, \forall j \in \mathcal{E}, i \in \mathcal{T}(b_j) \\ & C2 : 0 < \sum_{i \in \mathcal{T}(b_j)} \kappa_{j,i} \leq 1, \forall b_j \in \mathcal{E} \end{aligned} \tag{13}$$

where $|\mathcal{T}(b_1)| + |\mathcal{T}(b_2)| + \cdots + |\mathcal{T}(b_n)| = |\mathcal{T}|$. As for one minimization problem in objective function of (13), it is not difficult to see that $\sum_{i=1}^{l^j} \kappa_i^j = 1$ must be hold to minimize the objective function. Finally, the original problem is converted into $|\mathcal{E}|$ minimization problems, jth of which is as follows:

$$\begin{aligned} \min_{\boldsymbol{\kappa}_{j,\cdot}} \quad & f(\boldsymbol{\kappa}_{j,\cdot}) = \sum_{i \in \mathcal{T}(b_j)} \frac{a_i}{\kappa_{j,i}} \\ \text{s.t.} \quad & C1 : \kappa_{j,i} > 0, \forall i \in \mathcal{T}(b_j) \\ & C2 : \sum_{i \in \mathcal{T}(b_j)} \kappa_{j,i} = 1 \end{aligned} \tag{14}$$

where $a_i = \frac{\omega_i}{f_j^E}$. Resorting to Inequality of Arithmetic and Geometric Means, we can get the closed-form of the optimal resource allocation:

$$\begin{aligned}
\sum_{i\in\mathcal{T}(b_j)} \frac{a_i}{\kappa_{j,i}} &= \sum_{i\in\mathcal{T}(b_j)} \frac{a_i \sum\limits_{i\in\mathcal{T}(b_j)} \kappa_{j,i}}{\kappa_{j,i}} \\
&= \sum_{i\in\mathcal{T}(b_j)} a_i + \sum_{i\in\mathcal{T}(b_j)} \sum_{\substack{m\in\mathcal{T}(b_j)\\ m\neq i}} \frac{a_i\kappa_{j,m}}{\kappa_{j,i}} \\
&= \sum_{i\in\mathcal{T}(b_j)} a_i + \frac{1}{2} \sum_{i\in\mathcal{T}(b_j)} \sum_{\substack{m\in\mathcal{T}(b_j)\\ m\neq i}} \left(\frac{a_i\kappa_{j,m}}{\kappa_{j,i}} + \frac{a_m\kappa_{j,i}}{\kappa_{j,m}}\right) \\
&\geq \sum_{i\in\mathcal{T}(b_j)} a_i + \frac{1}{2} \sum_{i\in\mathcal{T}(b_j)} \sum_{\substack{m\in\mathcal{T}(b_j)\\ m\neq i}} \sqrt{2\frac{a_i\kappa_{j,m}}{\kappa_{j,i}} \cdot \frac{a_m\kappa_{j,i}}{\kappa_{j,m}}} \\
&= \sum_{i\in\mathcal{T}(b_j)} a_i + \frac{1}{2} \sum_{i\in\mathcal{T}(b_j)} \sum_{\substack{m\in\mathcal{T}(b_j)\\ m\neq i}} \sqrt{2a_ia_m}
\end{aligned} \tag{15}$$

if and only if $\kappa_{j,i} = \frac{\sqrt{a_i}}{\sum_{m\in\mathcal{T}(b_j)}\sqrt{a_m}}$, $\sum_{m\in\mathcal{T}(b_j)} \frac{a_i}{\kappa_i^j} = \sum_{i\in\mathcal{T}(b_j)} a_i + \frac{1}{2} \sum_{i\in\mathcal{T}(b_j)} \sum_{\substack{m\in\mathcal{T}(b_j)\\ m\neq i}} \sqrt{2a_ia_m}$. So, the optimal allocation of resources for ith task offloaded to the edge server b_j is:

$$\kappa_{j,i} = \frac{\sqrt{\omega_i/f_j^E}}{\sum\limits_{m\in\mathcal{T}(b_j)} \sqrt{\omega_m/f_j^E}} = \frac{\sqrt{\omega_i}}{\sum\limits_{m\in\mathcal{T}(b_j)} \sqrt{\omega_m}} \tag{16}$$

3.2 Task Placement Problem and An OO-Based Offloading Algorithm

We have got the optimal resource allocation of the edge servers under an arbitrary given task placement policy. Thus, the original problem is converted into getting the optimal task placement policy, which means we should decide whether each task is processed locally or remotely and to which edge server task should be placed on. In other words, if we obtain the optimal task placement policy, the optimal resource allocation under this policy must be the optimal solution to the original problem. Unfortunately, the decision space of the task placement problem might be $\Theta = \{b \mid b \in \mathcal{A}(u_1)\ or\ b = u_1\} \times \{b \mid b \in \mathcal{A}(u_2)\ or\ b = u_2\} \times \cdots \times \{b \mid b \in \mathcal{A}(u_m)\ or\ b = u_m\}$, which is extremely large. Hence, we consider introducing the concept of Ordinal Optimization (OO) to solve this problem.

Ordinal Optimization (OO) was first proposed by Ho *et al.* [21] to address the stochastic complex simulation-based optimization problems (SCP). Moreover, OO has also been proved to be suitable for solving deterministic complex problems (DCP) and gets some successful applications. "Soft optimization for hard problems" is the quintessence of OO, which means after softening the goal, the hard problem can be tackled within a tolerable time. It is highly impracticable to find the global optimal solution, so, we seek to find a near-optimal solution instead with high probability. The objective can be formed as:

$$Pr(|G \cap S| \geq k) \geq \alpha \tag{17}$$

Here, G denotes the good enough set (i.e., the top-$|G|$ solutions of the problem). S denotes the set of selected solutions (i.e., the estimated top-$|S|$ solutions selected by simulation experiments). $Pr(|G \cap S| \geq k)$ called alignment probability denotes the probability that the number of good enough solutions in S is no less than k, and k is called the alignment level. OO can pick out S to ensure that (17) holds under the $|G|$, k and α specified by the user. The Horse Race rule is a commonly used method for selecting set S in OO. The rule is as follows:

1. Select N samples uniformly and randomly from Θ.
2. Use a crude but computationally fast model to estimate the performances of these N samples.
3. Estimate the Ordered Performance Curve (OPC) class and the error level of the crude model.
4. Use the table in [22] to get $|S|$, the size of S.
5. Use the crude model to select the estimated top-$|S|$ samples to form S.
6. Evaluate each sample in S with a precise model.
7. Apply the best sample from the evaluation results in Step 6.

Therefore, we design an OO-based offloading algorithm by combining the Ordinal Optimization with Theorem 1. The specific offloading algorithm is shown in Algorithm 1, where Line 1 aims to replace Θ with Θ_N. According to [23], when $N \geq 1000$, we can treat Θ_N as a reasonable representative of Θ, Lines 2–3 estimate the performance of each sample in Θ_N roughly, Lines 4–6 aim to find the set S, Lines 7–8 evaluate the performance of each sample in S precisely. Finally, the sample with the best performance as the good enough offloading policy is found.

Algorithm 1: OO-based offloading algorithm

Input: Θ, k, g, α
Output: a good enough offloading policy

```
select N samples uniformly and randomly from Θ to form set Θ_N
foreach sample in Θ_N do
    use a crude but computationally fast model to estimate the performances of the sample
estimate the Ordered Performance Curve (OPC) class and the noise level of the crude model
use the table presented in [22] to calculate the size of selected set s
pick out the top-s samples to form set S
foreach sample in S do
    use a precise model to evaluate the performances of the sample
pick out the sample with best performance as the good enough offloading policy we found
```

4 Performance Evaluation

In this section, we discuss the performance of the OO-based offloading algorithm we proposed.

4.1 Simulation Setup

Consider a 700 m $\times$ 300 m^2 area in which 16 base station are evenly distributed, and each base station are equipped with an edge server. The overage radius of each base station is set as 100 m. The computation capability and full state power of each edge server are set as 5 GHz and 500 W, respectively. According to the survey result in [24], the idle power of the edge server accounts for 60% of the full state power. The mobile devices are randomly scattered in the area, and the computation capability of each mobile device is 1 GHz. The size of tasks is randomly chosen from [0.5, 1] MB, and the processing density of tasks is 500 cycles/bit, which means 500 CPU cycles are required to process each bit of the task. Besides, the channel gain $g_{i,j}$ is modeled by $(d_{i,j})^{-\alpha}$, where $\alpha = 4$ denotes the path loss factor and $d_{i,j}$ denotes the distance between user u_i and edge server b_j [25]. The other simulation parameters are given in Table 2.

Table 2. Simulation parameters

Parameter	Value
Number of users, m	{30, 40, 50, 60, 70, 80, 90}
Number of edge servers, n	16
Channel bandwidth, B	10 MHz
Transmission power of mobile device u_i, p_i^T	0.5 W
Computing power of mobile device u_i, p_i^L	0.5 W
Noise power, σ^2	10^{-13} W
Size of task T_i, s_i	[0.5, 1] MB
Frequency of mobile device u_i, f_i^L	1 GHz
Frequency of edge server b_j, f_j^E,	5 GHz
Remaining energy of the mobile device u_i, $\alpha_i E_i^{max}$	1000 mAh
Idle power of edge server b_j, p_j^{idle}	300 W
Max power of edge server b_j, p_j^{max}	500 W [26]

4.2 Determine Crude Model and OPC Class

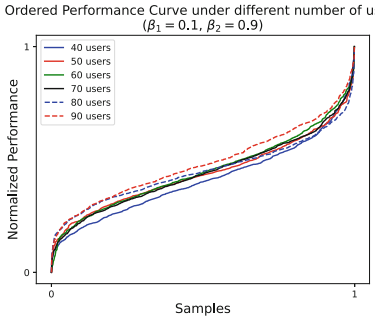
In the crude model, the transmission rate $r_{i,j}$ is replaced by the overall average transmission rate $r_{average}$. For each sample, we denote the number of offloaded tasks as $m^{offload}$, then set $\kappa_{j,i}(\forall i, j)$ to $\kappa = \min\{\frac{n}{m^{offload}}, 1\}$. So the crude model can be expressed as follows:

$$\begin{aligned}\hat{f}(\boldsymbol{x}, \boldsymbol{\kappa}) = \beta_1 \sum_{i=1}^{m}[x_{i,0}\frac{\omega_i}{f_i^L} + \sum_{j=1}^{n} x_{i,j}(\frac{\omega_i}{\kappa f_j^E} + \frac{s_i}{r_{average}})] \\ + \beta_2 \sum_{i=1}^{m}\sum_{j=1}^{n} x_{i,j}\frac{\omega_i}{\kappa f_j^E}(p_j^{max} - p_j^{idle})\end{aligned} \tag{18}$$

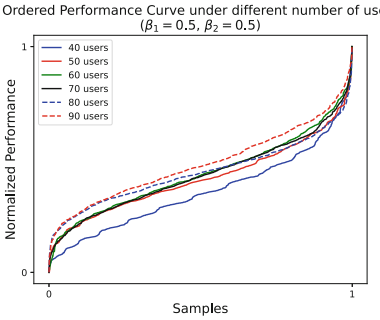
Then, we select 1000 samples uniformly from Θ to form set Θ_N, which is a reasonable representative of Θ. Next, with the $g = 50$, $\alpha = 0.95$ and $k = 5$, we estimate the OPC class and calculate s. The OPC is shown in Fig. 2. As we can see, no matter what the β_1, β_2, and the number of users are, the OPC class is Bell [23]. According to [22], when the error level is a large error, medium error, or small error, s is 40, 80, and 140 respectively.

Just like [13,27], we compare the OO-based offloading policy with other three offloading policies:

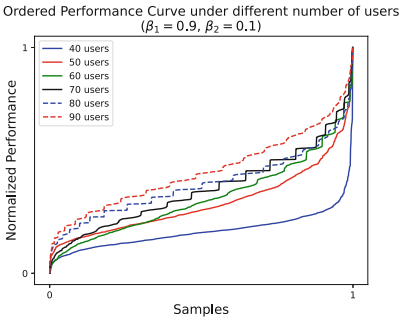
- **Local computing policy:** This policy involves no offloading. All computation tasks are executed locally on mobile devices.
- **Edge computing policy:** In this policy, all mobile device user offload their tasks to one edge server.
- **Random offloading policy:** In this policy, all computation tasks are randomly decided whether to offload.



(a) OPC when $\beta_1 = 0.1$ and $\beta_2 = 0.9$



(b) OPC when $\beta_1 = 0.5$ and $\beta_2 = 0.5$



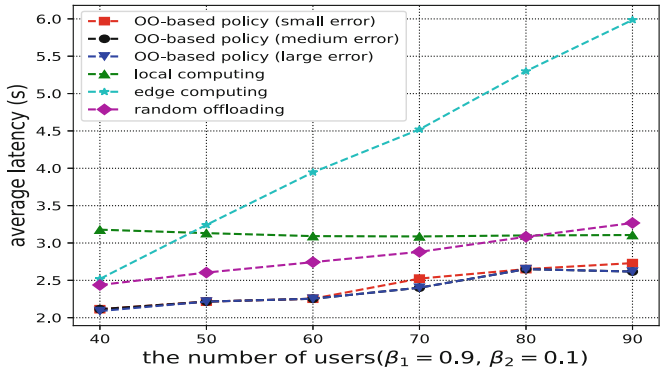
(c) OPC when $\beta_1 = 0.9$ and $\beta_2 = 0.1$

Fig. 2. Ordered performance curve

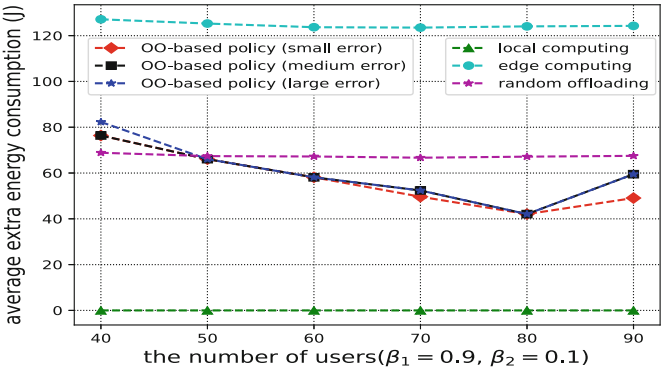
For the four offloading policies, we evaluate performance in terms of average latency and average extra energy consumption of all edge servers.

4.3 Comparison with Baseline Methods

The average task latency and average extra edge server energy consumption of OO-based policy with different error levels and the other three baseline offloading policies under different weights are shown in Figs. 3, 4 and 5 respectively. It can be observe that different β_1 and β_2 do not affect the performance of the three baseline policies (i.e., the performance curves of three baseline policies under different β_1 and β_2 are the same). In other words, the OO-based offloading policy proposed in this paper is more adaptable to different situations. For edge computing policy, as the number of users increases, the average latency increases rapidly, while the average extra energy consumption remains constant but actually the highest of all offloading policies. This is because all tasks consume as much energy as shown in (9) and the increase in the number of users leads to the reduction of computing resources allocated to each task. For local computing policy, since all task processed locally, the average extra consumption is always 0 and the latency of each task is local computation latency. This is the reason



(a) The average task latency.

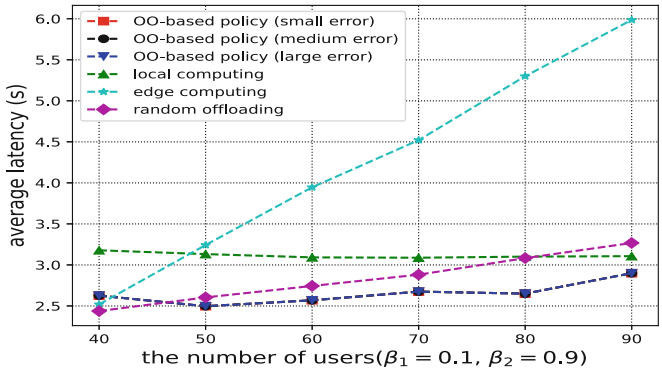


(b) The average extra energy consumption.

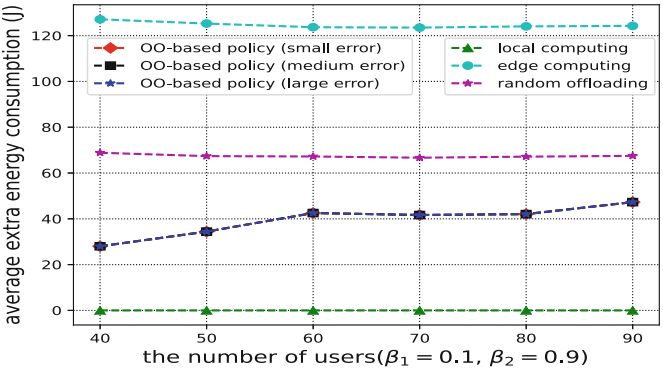
Fig. 3. The performance of different offloading policies when $\beta_1 = 0.9$ and $\beta_2 = 0.1$.

that there is little fluctuation in average task latency. As for the performance of the random offloading policy, as the average result of 1000 random trials, if the number of users increases, the average task latency tends to increase while the average extra energy consumption remaining stable. Furthermore, the performance of OO-based policy under different error levels are extremely close to each other, which means the error level of the crude model is small noise. So, we just need to evaluate the performance of the top-40 samples selected by the crude model to get a good-enough policy.

In Fig. 3, we compared the performance of the aforementioned policies when $\beta_1 = 0.9$ and $\beta_2 = 0.1$. In this case, we are more concerned about the average latency rather than the average extra edge server consumption. It can be observed that no matter how many user is, the OO-based policy always achieves significantly lower latency than other policies. In this case, although more attention is paid to average task latency, still, in most cases (i.e., $m \geq 50$), the average



(a) The average task latency

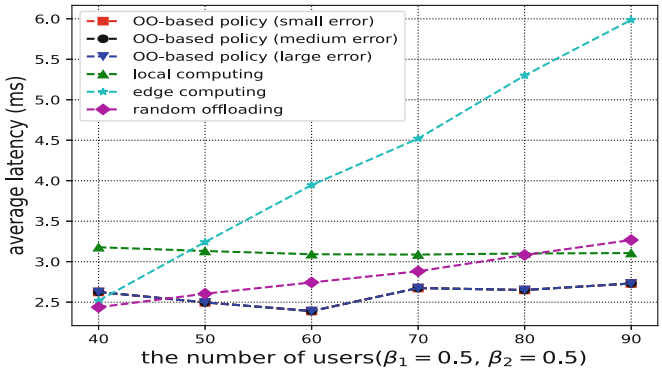


(b) The average extra energy consumption.

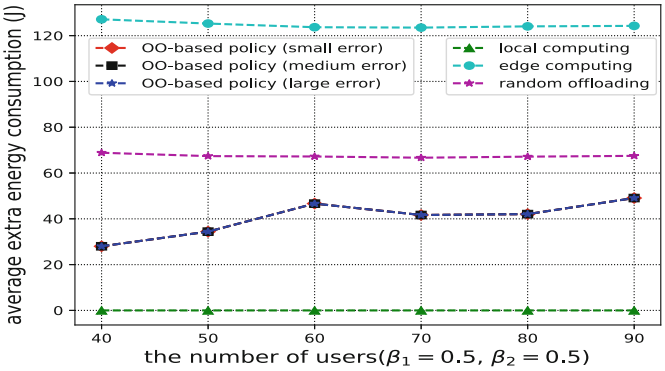
Fig. 4. The performance of different offloading policies when $\beta_1 = 0.1$ and $\beta_2 = 0.9$.

extra energy consumption of OO-based policy is better than random offloading policy and edge computing policy.

Figure 4 illustrates the performance of four offloading policies when $\beta_1 = 0.1$ and $\beta_2 = 0.9$. Contrary to the above situation, in this case, we are more concerned about extra energy consumption than task latency. As for energy consumption, OO-based policy is significantly better than edge computing policy and random offloading policy. Similar to the situation above, in most cases (i.e., $m \geq 50$), the average task latency of OO-based policy is the best.



(a) The average task latency.



(b) The average energy consumption of four offloading policies

Fig. 5. The performance of different offloading policies when $\beta_1 = 0.5$ and $\beta_2 = 0.5$.

Figure 5 illustrates the performance of four offloading policies when $\beta_1 = 0.5$ and $\beta_2 = 0.5$. In this case, we considered offloading task and extra energy consumption equally important. At this time, OO-based policy has achieved good results in reducing task latency and extra energy consumption. It can be observed that although the latency of OO-based policy is slightly higher than that of random offloading policy and edge computing policy, the extra energy consumption of OO-based policy is significantly better when the number of users is 40. Besides, when $m \geq 50$, both the average task latency and average extra energy consumption of OO-based are best except the extra energy consumption of local computing (i.e., 0).

5 Conclusion

In this paper, we first propose a problem of minimizing latency and server energy consumption in the large-scale MEC systems. Then, we illustrate this problem as NP-hard. To solve this problem, we consider dividing the original problem into two sub-problems: computation resource allocation problem and task placement problem. Next, we get the closed-form solution of the optimal resource allocation under an arbitrary given task placement policy. So, the original problem is converted into finding the optimal task placement policy. However, the decision space is extremely large. So, we introduce the concept of Ordinal Optimization to find the near-optimal task placement policy. Afterwards, we design an efficient offloading algorithm based on Ordinal Optimization. Finally, simulation results have shown that the OO-based offloading policy is not only more efficient as compared to the other three baseline policies, but also more adaptable to different situations. In the future work, real-world mobility data will be employed to validate our proposed offloading algorithm.

Acknowledgment. This work is supported by Beijing Nova Program (No. Z201100 006820082), National Natural Science Foundation of China (No. 61972414), National Key Research and Development Plan (No. 2016YFC0303700), Beijing Natural Science Foundation (No. 4202066), and the Fundamental Research Funds for Central Universities (Nos. 2462018YJRC040 and 2462020YJRC001).

References

1. Forecast, G.: Cisco visual networking index: global mobile data traffic forecast update, 2017–2022. Update **2017**, 2022 (2019)
2. Azuma, R.T.: A survey of augmented reality. Presence Teleoperators Virtual Environ. **6**(4), 355–385 (1997)
3. Ranjan, N., Mundada, K., Phaltane, K., Ahmad, S.: A survey on techniques in NLP. Int. J. Comput. Appl. **134**(8), 6–9 (2016)
4. Zhao, Q.: A survey on virtual reality. Sci. China Ser. F Inf. Sci. **52**(3), 348–400 (2009)
5. Badue, C., et al.: Self-driving cars: a survey. Expert Syst. Appli. **165**, 113816 (2020)
6. Kan, T.Y., Chiang, Y., Wei, H.Y.: Task offloading and resource allocation in mobile-edge computing system. In: 2018 27th Wireless and Optical Communication Conference (WOCC), pp. 1–4. IEEE (2018)
7. Mach, P., Becvar, Z.: Mobile edge computing: a survey on architecture and computation offloading. IEEE Commun. Surv. Tutorials **19**(3), 1628–1656 (2017)
8. Hu, Y.C., Patel, M., Sabella, D., Sprecher, N., Young, V.: Mobile edge computing-a key technology towards 5G. ETSI White Paper **11**(11), 1–16 (2015)
9. Huang, J., Zhang, C., Zhang, J.: A multi-queue approach of energy efficient task scheduling for sensor hubs. Chin. J. Electron. **29**(2), 242–247 (2020)
10. Liu, J., Mao, Y., Zhang, J., Letaief, K.B.: Delay-optimal computation task scheduling for mobile-edge computing systems. In: 2016 IEEE International Symposium on Information Theory (ISIT), pp. 1451–1455. IEEE (2016)

11. Mao, Y., Zhang, J., Song, S., Letaief, K.B.: Power-delay tradeoff in multi-user mobile-edge computing systems. In: 2016 IEEE Global Communications Conference (GLOBECOM), pp. 1–6. IEEE (2016)
12. Xu, Z., Wang, Y., Tang, J., Wang, J., Gursoy, M.C.: A deep reinforcement learning based framework for power-efficient resource allocation in cloud rans. In: 2017 IEEE International Conference on Communications (ICC), pp. 1–6. IEEE (2017)
13. Li, J., Gao, H., Lv, T., Lu, Y.: Deep reinforcement learning based computation offloading and resource allocation for mec. In: 2018 IEEE Wireless Communications and Networking Conference (WCNC), pp. 1–6. IEEE (2018)
14. Huang, J., Li, S., Chen, Y.: Revenue-optimal task scheduling and resource management for IoT batch jobs in mobile edge computing. Peer-to-Peer Netw. Appl. **13**(5), 1776–1787 (2020)
15. Ge, X., Tu, S., Mao, G., Wang, C.X., Han, T.: 5G ultra-dense cellular networks. IEEE Wirel. Commun. **23**(1), 72–79 (2016). https://doi.org/10.1109/MWC.2016.7422408
16. Chen, M., Hao, Y.: Task offloading for mobile edge computing in software defined ultra-dense network. IEEE J. Sel. Areas Commun. **36**(3), 587–597 (2018)
17. Gao, Y., Guan, H., Qi, Z., Wang, B., Liu, L.: Quality of service aware power management for virtualized data centers. J. Syst. Archit. **59**(4–5), 245–259 (2013)
18. Fan, X., Weber, W.D., Barroso, L.A.: Power provisioning for a warehouse-sized computer. ACM SIGARCH Comput. Archit. News **35**(2), 13–23 (2007)
19. Pochet, Y., Wolsey, L.A.: Production Planning by Mixed Integer Programming. Springer Science and Business Media, Heidelberg (2006)
20. Vu, T.T., Van Huynh, N., Hoang, D.T., Nguyen, D.N., Dutkiewicz, E.: Offloading energy efficiency with delay constraint for cooperative mobile edge computing networks. In: 2018 IEEE Global Communications Conference (GLOBECOM), pp. 1–6. IEEE (2018)
21. Ho, Y.C., Sreenivas, R., Vakili, P.: Ordinal optimization of DEDS. Discrete Event Dyn. Syst. **2**(1), 61–88 (1992)
22. Lau, T.E., Ho, Y.C.: Universal alignment probabilities and subset selection for ordinal optimization. J. Optim. Theory Appl. **93**(3), 455–489 (1997)
23. Ho, Y.C., Zhao, Q.C., Jia, Q.S.: Ordinal Optimization: Soft Optimization for Hard Problems. Springer Science & Business Media, Heidelberg (2008)
24. Dayarathna, M., Wen, Y., Fan, R.: Data center energy consumption modeling: a survey. IEEE Commun. Surv Tutorials **18**(1), 732–794 (2015)
25. Liu, Z., Yang, Y., Wang, K., Shao, Z., Zhang, J.: Post: parallel offloading of splittable tasks in heterogeneous fog networks. IEEE Internet Things J. **7**(4), 3170–3183 (2020)
26. Li, Y., Wang, S.: An energy-aware edge server placement algorithm in mobile edge computing. In: 2018 IEEE International Conference on Edge Computing (EDGE), pp. 66–73. IEEE (2018)
27. Zhang, Y., Niyato, D., Wang, P.: Offloading in mobile cloudlet systems with intermittent connectivity. IEEE Trans. Mob. Comput. **14**(12), 2516–2529 (2015)