Joint Computation Offloading and Resource Allocation under Task-Overflowed Situations in Mobile Edge Computing

Huijun Tang, Huaming Wu, *Member, IEEE*, Yubin Zhao, *Member, IEEE*, and Ruidong Li, *Senior Member, IEEE*

***Abstract*—With the rapid development of Artificial Intelligence (AI) and Internet of Things (IoT), we have to perform increasingly more resource-hungry and compute-intensive applications on IoT devices, where the available computing resources are insufficient. With the assistance of Mobile Edge Computing (MEC), offloading partial complex tasks from mobile devices to edge servers can achieve faster response time and lower energy consumption. However, it still suffers from finding the optimal offloading decision when the total amount of computations overflows the available computing resources in MEC systems. In this paper, we establish a multi-user and multi-task MEC model and design an offloading indicator, through which we analyze what the current environment belongs to. In the cases where the computational resources of devices are sufficient or partially sufficient, we utilize the relationship between the offloading indicator and the cost incurred by the tasks that are executed in the current workflow to find the optimal offloading decision. In the cases where the computation on local and edge are both insufficient, we propose a novel Offloading Algorithm based on K-means clustering and Genetic algorithm for solving Multiple knapsack problem (OAKGM), aiming not only to jointly optimize the time and energy incurred by the tasks that are executed in the current workflow, but also to penalize the overflowed computations so that the task pressure in the next workflow can be greatly reduced. In addition, a simplified Offloading Algorithm based on Multiple Knapsack Problem (OAMKP) is proposed to further cope with the environments with a large number of users or tasks. Experimental results demonstrate the effectiveness and superiority of the proposed algorithms when compared with several benchmark offloading algorithms, which can better exploit the computing capacities of IoT devices and the edge server, greatly avoid resource occupation in edge nodes and make sustainable MEC possible.**

***Index Terms*—Mobile Edge Computing, Internet of Things, Task Offloading, Resource Allocation, Multiple Knapsack Problem**

I. Introduction

DRIVEN by Artificial Intelligence (AI) in Internet-of-Things (IoT) systems, more and more mobile applications with large-scale Deep Neural Networks (DNNs), e.g., video analytics and augmented reality, are being deployed on resource-constrained mobile devices. Thus, the essential demand for computing capacity and low latency has exploded. Unfortunately, the resources of mobile devices are generally constrained [1], [2], e.g., insufficient storage, slow computing speed and low battery capacity, which fail to support the complex computation of DNN training and DNN inference [3].

Currently, Mobile Edge Computing (MEC) has been widely applied to video analytics, augmented reality, mobile big data, internet of vehicles and other fields [4]. With the assistance of MEC, one promising method is to offload computation-intensive tasks from IoT devices to edge servers, enabling resource-constrained devices to perform more complicated tasks with less time delay and lower energy consumption [5]–[8]. Generally, an edge server can serve several users, and each user can generate multiple tasks in a workflow [9], [10]. Al-Shuwaili *et al.* [11] applied edge computing to the augmented reality tasks, which involves a large amount of computation, e.g., the amount of computation required per task for computing a 1024×768 image can reach 2,640 cycles. Unlike cloud servers with abundant computing resources, edge servers generally suffer from limited computational resources. Although the computing power of edge servers is much greater than that of mobile devices, they are relatively low compared to cloud servers.

Intuitively, executing a huge number of tasks at the server tends to cause high response time. As a result, there exists an upper bound on concurrent task execution due to a large number of offloading requests and the limited computing power of edge servers. On the one hand, if all the IoT devices offload their tasks to an edge server for processing, the edge server will become overloaded and bring in additional delays during task execution. On the other hand, once the number of mobile users or computational tasks of each device increases significantly, the total amount of computation of intensive tasks are prone to overflow the overall computing capacity in the MEC environment.

In order to deal with the above challenges, we need to determine the priority of tasks so that the number of tasks that cannot be executed in the current workflow should be as few as possible. It is imperative to develop an offloading-decision strategy that not only considers the delay and energy consumption when executing tasks locally or at the edge, or transferring data between the device and the edge, but also considers the computational amount of tasks that cannot be executed in the current workflow when tasks are intensive. When the tasks of mobile users arrive in a workflow, there are three options for each task: the first one is to be executed locally on the mobile device, the second one is to be offloaded

H. Tang and H. Wu are with Center for Applied Mathematics, Tianjin University, Tianjin 300072, China. E-mail: {tanghuijune, whming}@tju.edu.cn

Y. Zhao is with the School of Microelectronics Science and Technology, Sun Yat-Sen University, Zhuhai, 519082, China. Email: zhaoyb23@mail.sysu.edu.cn

R. Li is with the Institute of Science and Engineering, Kanazawa University, Kanazawa 920-1192, Japan (e-mail: liruidong@ieee.org).

(Corresponding author: Huaming Wu)

TABLE I: Comparison of Selected Related Studies.

Ref.	Number of Tasks	Number of Users	Edge Server	Objectives	Constraints
Meng *et al.* [14]	Single	Multiple	Single	Energy	Total Computational Resource & Latency Requirement
Wang *et al.* [12]	Single	Multiple	Multiple	Energy	Latency Requirement
Kao *et al.* [15]	Multiple	Multiple	-	Energy	Latency Requirement
Wang *et al.* [16]	Single	Multiple	Single	Time+Monetary cost	Total Computational Resource & Total Bandwidth Resource
Fang *et al.* [17]	Single	Multiple	Multiple	Energy+Time	Latency Requirement
Ours	**Multiple**	**Multiple**	**Single**	**Energy+Time+w_n**	**Total Computational Resource**

w_n is the total amount of computation of tasks that cannot be executed in the current workflow

to the server at the edge and the third one is to be delayed to the next workflow. How to jointly make the optimal offloading decision and resource allocation depends on two factors: one is the cost, which is usually defined by the time delay and the energy consumption incurred from the computing process or the transmitting process; the other is the currently available computing resources of the system. This can be regarded as a multi-knapsack problem, which is an NP-hard problem that requires a large amount of computation and causes additional delays for offloading decision-making. This means that it is difficult to find a polynomial-time complexity algorithm to solve such a problem. Previous studies [12], [13] have tried to solve the multiple knapsack problem through heuristic algorithms, but only optimized the energy consumption when computing tasks or transmitting data, ignoring the total amount of computation of tasks that cannot be computed in the current workflow, which places a heavy burden, especially when excessive tasks need to take up plenty of resources in the next workflow.

In this paper, we mainly focus on situations where tasks with a large number of calculations such as augmented reality, will occasionally overflow the computation of the local devices or the edge server. In particular, we address a situation in the MEC when the amount of computations to be carried out exceeds the resource capacity of the network at the edge. We treat the task-overflowed issue in MEC Systems as the multiple knapsack problem and propose two novel offloading algorithms for solving this problem. As far as we know, this is the first work that not only optimizes the cost incurred in the current workflow, but also penalizes the computational burden left to the next workflow about computation offloading and resource allocation. The aim of this paper is to jointly minimize the time and energy incurred by the tasks, as well as to penalize the overflowed computation and exploit the resources as many as possible.

The major contributions of this paper can be summarized as follows:

- *Model of Overloaded Task Offloading Problem in MEC Systems:*
  - The task-overflowed situation that was rarely considered in previous studies is modeled as a multiple knapsacks problem whose NP-hardness is proved in the paper. We optimize offloading decisions in task-overflowed situations from the perspective of the overflowed computation, which is more refined than the perspective of the number of tasks which usually measures the queue.
  - We design a novel offloading indicator, which can directly indicate the offloading strategy when the system has sufficient computing resources. When the current computing resources are insufficient, we formulate a joint computation offloading and resource allocation strategy, which not only optimizes the combined delay and energy consumption generated from executing or transferring process, but also optimizes the total amount of calculations of the overloaded tasks in the current workflow.
- *Two Knapsack Problem-based Task Offloading Algorithms:* For small-scale MEC systems, we propose the OAKGM algorithm that combines K-means clustering and genetic algorithm to solve the multiple knapsack problem, while for large-scale MEC systems [18], we propose the OAMKP algorithm that can handle numerous users or tasks.
- *Performance Evaluation:* We conduct extensive simulation experiments under various system parameters. Simulation results demonstrate that when compared with the Genetic Algorithm (GA) and the Particle Swarm Optimization (PSO) algorithm, the proposed OAKGM and OAMKP algorithms can make better use of the computing capacities of IoT devices and edge nodes, alleviate the task pressure to the next workflow, significantly reduce the average cost, as well as avoid resource occupation in edge nodes.

The rest of the paper is organized as follows: Section II reviews the related work. Section III formulates the system model of task offloading in MEC with workload overloaded. The proposed OAKGM and OAMPK algorithms are described in detail in Section IV. Section V evaluates its performance by comparing it with state-of-the-art offloading methods. Finally, Section VI concludes this paper and points out some possible future work.

II. Related Work

It is known that offloading tasks from resource-constrained IoT devices to nearby edge servers will cause additional energy consumption or time delays due to data transmission between the local and the edge [19]. In recent years, a large amount of research (ranging from heuristic algorithms [20]–[22], genetic algorithms [23], [24], game theory [25], blockchain theory [26], [27], to deep reinforcement learning-based algorithms [28]–[30]) has been devoted to seeking the optimal offloading strategy in MEC systems, with the aim of improving system efficiency when the computing and communication resources are limited.

Several offloading-decision algorithms have been proposed in [31]–[33], with the purpose of minimizing energy consumption while ignoring the time delay. Prior research [34]–[36] has focused on joint optimization of offloading decisions and resource allocation for delay-sensitive or compute-intensive tasks. However, few studies have considered overloaded or overflowed tasks that cannot be computed in the current workflow on account of the limitation of computing resources of mobile devices and edge servers. Most recent studies have discussed the optimal offloading decision in the situation where the total computing capacity of the MEC system is sufficient for computing the current tasks [12], [14]–[17], [32], which cannot satisfy the future or even current demand of the MEC system. However, when taking the task-overflowed cases into account, it will cause the already complex offloading decision and resource allocation problem even more complicated, thereby posing a huge challenge. On the contrary, in this paper, we define a new weighted cost, which includes the time delay and energy consumption during the computing and transmitting process, as well as the amount of computation of overflowed tasks in the current workflow.

In general, the joint offloading decision and resource allocation problem can be treated as an integer programming problem [12], [37]. Xue *et al.* [37] regarded the computation offloading problem as an NP-hard problem and solved it through an iterative heuristic task-intensive assignment algorithm. Song *et al.* [38] proposed a dynamic programming algorithm to manage tasks at the edge of the network. Guo *et al.* [39] treated the offloading computing problem as a Mixed Integer Programming (MIP) problem and solved it by the Gurobi optimizer. Liu *et al.* [40] formulated the offloading decision problem in IoT environments as a MIP problem to minimize the energy consumption. Chen *et al.* [41] took the relay-assisted problem into consideration and formulated the model as a non-differentiable and non-convex optimization problem. However, all the above-mentioned works ignore the amount of computation of tasks that cannot be executed in the current workflow, which may bring huge computational pressure to the next workflow.

Some studies model the optimization process as a Lyapunov optimization problem [42] [43]. Ouyang *et al.* [42] proposed different situations depending on whether the mobility characteristic is known. They propose a greedy algorithm for short-term optimization and a Lyapunov-based algorithm for long-term optimization. It is necessary to study the short-term optimization because the characteristic of mobile devices and edge servers sometimes change in long term. In this paper, we consider the optimization in one workflow so that the change of the devices or the environment in different workflows will not influence the result of optimization.

Different from previous works, we design a novel task offloading indicator in the local-edge collaborative computing model. By utilizing this indicator, we consider different situations depended on the computational resources and the offloading indicator. Then, we propose different methods for different situations. When the computational resources on local and edge are both sufficient, optimal offloading decisions are made dependent on whether the offloading indicator is greater than 1. When the computational resources on local and edge are partially sufficient, we make offloading decisions by utilizing the relationship between the cost and the offloading indicator. When the computational resources on local and edge are both insufficient, finding an optimal offloading decision is a multiple knapsack problem that is NP-hard to solve. We convert the NP-hard problem into an easy-to-solve transformation problem, which consists of several simple integer programming problems, so that the original problem can be solved in polynomial time. We then propose two algorithms on the basis of the offloading indicator: one is called OAKGM, which combines K-means clustering and a genetic algorithm to search for different thresholds of offloading indicators for a variety of mobile devices; the other is OAMKP, which goes through all possible threshold values of offloading indicators for all mobile devices and finds the best threshold for the whole MEC system.

III. System model

In order to minimize the average system cost (weighted sum of delay and energy consumption) from IoT devices to the edge and alleviate the computational pressure of the next workflow, we consider an MEC system with a large amount of computational overflow.

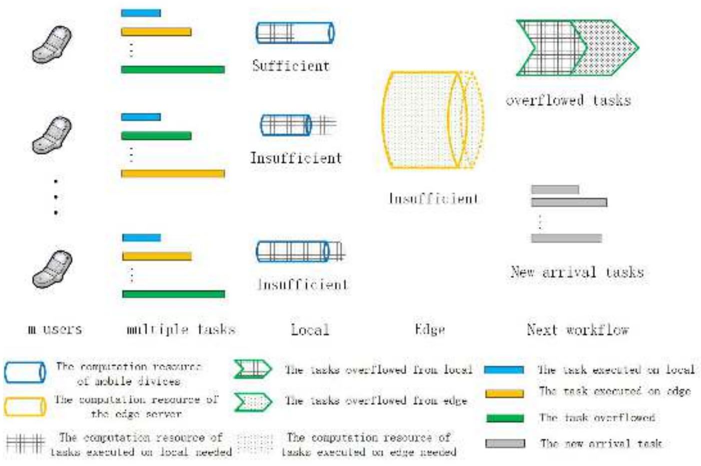

Sufficient
Insufficient
Insufficient
Insufficient
overflowed tasks
New arrival tasks
m users
multiple tasks
Local
Edge
Next workflow
The computation resource of mobile divices
The computation resource of the edge server
The computation resource of tasks executed on local needed
The tasks overflowed from local
The tasks overflowed from edge
The computation resource of tasks executed on edge needed
The task executed on local
The task executed on edge
The task overflowed
The new arrival task

Fig. 1: An illustration of the MEC system with multiple mobile devices and an edge server.

As depicted in Fig. 1, the IoT-edge computing environment consists of m IoT devices and an edge server, and is equipped with some computing nodes with different computing capacities. Mobile users can offload their tasks to the edge server. Without loss of generality, we consider a set of mobile users $\mathcal{M} = \{1, 2, \cdots, m\}$. We assume that tasks arrive in a time-slotted form, and the length of slot is t. In a workflow, each user has L arrival tasks, and the j task of the i^{th} user is denoted as $task_{ij}$. The data size of these tasks are expressed as a set $\mathcal{D} = \{D_{ij} | i = 1, 2, \cdots, m \text{ and } j = 1, 2, \cdots, L\}$. The major notations used in this paper are defined in Table II.

A. Local Computing Model

When a task is processed locally on the IoT device, the time delay and energy consumption can separately be represented as follows:

- The time delay when performing $task_{ij}$ locally can be calculated as:

$$T_{ij}^{l} = \frac{w_{ij}}{f_i}, \tag{1}$$

  where $f_i(cycle/s)$ is the computation speed of the i^{th} mobile device and w_{ij} is the amount of computation for $task_{ij}$.
- The energy consumption when performing $task_{ij}$ locally can be calculated as:

$$E_{ij}^{l} = P_i^{A} \cdot T_{ij}^{l} = \frac{w_{ij}P_i^{A}}{f_i}, \tag{2}$$

  where P_i^{A} is the active power.

Thus, the overall cost of $task_{ij}$ when it is processed in local can be calculated as:

$$Q_{ij}^{l} = \alpha T_{ij}^{l} + E_{ij}^{l} = \frac{w_{ij}}{f_i}(\alpha + P_i^{A}), \tag{3}$$

where $\alpha > 0$ is a weighting parameter used to measure the importance of the time delay relative to the energy consumption.

B. Edge Computing Model

The computing resources of local devices are usually limited, so we can offload complex tasks to edge servers. In the offloading process, the cost is composed of two parts: one part is incurred from the transmitting process, the other part is incurred from the computing process.

- The time delay during data transmission for $task_{ij}$ can be calculated as:

$$T_{ij}^{T} = \frac{D_{ij}}{R_i}, \tag{4}$$

  where R_i is the data rate of device i.
- The energy consumption during data transmission for $task_{ij}$ can be expressed as:

$$E_{ij}^{T} = P_i^{T} \cdot T_{ij}^{T} = \frac{D_{ij}}{R_i}P_i^{T}, \tag{5}$$

  where P_i^{T} is the transmission power of the i^{th} device.
- The time delay when computing $task_{ij}$ can be expressed as:

$$T_{ij}^{e} = \frac{w_{ij}}{f_e}, \tag{6}$$

  where $f_e(cycle/s)$ is the computing speed of the edge server, $w_{ij} = \rho_{ij}D_{ij}$ is the amount of computation of $task_{ij}$, and ρ_{ij} is the computational complexity of $task_{ij}$.
- The energy consumption when computing $task_{ij}$ can be calculated as:

$$E_{ij}^{e} = P_i^{I} \cdot T_{ij}^{l} = \frac{w_{ij}}{f_e}P_i^{I}, \tag{7}$$

  where P_i^{I} is the idle power.

TABLE II: Notations and their definitions.

Notations	Definitions
R	The data rate
m	The number of users of the mobile devices
L	The number of tasks of the mobile devices
$task_{ij}$	The j^{th} task of the i^{th} user
P^{A}	The active power
P^{I}	The idle power
P^{T}	The transmission power
D	The data size
w	The amount of computation
ρ	The computational complexity
τ	The offloading indicator
Θ	The threshold vector of the offloading indicator
θ	The threshold value of the offloading indicator
T_{ij}^{l}	The time delay when $task_{ij}$ is executed in the local
T_{ij}^{e}	The time delay when $task_{ij}$ is executed at the edge
E_{ij}^{l}	The energy consumption when $task_{ij}$ is executed in the local
E_{ij}^{e}	The energy consumption when $task_{ij}$ is executed at the edge
T_{ij}^{T}	The time delay when $task_{ij}$ is transmitted to the edge
E_{ij}^{T}	The energy consumption when $task_{ij}$ is transmitted to the edge
Q^{l}	The cost in the local
Q^{e}	The cost at the edge
V^{l}	The values of items in the improved local knapsack
V^{e}	The values of items in the improved edge knapsack
κ_{ij}	The offloading decision for $task_{ij}$
α	The weighting factor of the time delay
β	The weighting factor of the amount of overflowed computations
I	The offloading policy for all tasks
I^{next}	The indicator matrix of tasks to be executed in next workflow
I^{tr}	The indicator matrix of tasks executed elsewhere not guided by τ
I^{l}	The indicator matrix of tasks considered to be executed in the local
I^{e}	The indicator matrix of tasks considered to be executed at the edge
κ^{l}	The indicator matrix describes the task executed in the local
κ^{e}	The indicator matrix describes the task executed at the edge

Thus, the overall cost of $task_{ij}$ when it is processed at the edge can be calculated as:

$$\begin{aligned} Q_{ij}^{e} &= \alpha(T_{ij}^{T} + T_{ij}^{e}) + (E_{ij}^{T} + E_{ij}^{e}) \\ &= \frac{D_{ij}}{R_i}(\alpha + P_i^{T}) + \frac{w_{ij}}{f_e}(\alpha + P_i^{I}). \end{aligned} \tag{8}$$

C. Local-Edge Collaborate Computing Model

The optimal computing cost for executing $task_{ij}$ is the minimum of the cost executing in local and edge, which can be expressed as:

$$Q_{ij} = \min\{Q_{ij}^{l}, Q_{ij}^{e}\}, \tag{9}$$

where $task_{ij}$ is preferred to be offloaded to the edge server than computing in local when $Q_{ij}^{l} > Q_{ij}^{e}$, which satisfying the following condition:

$$\frac{\rho_{ij}}{f_i}(\alpha + P_i^{A}) > \frac{1}{R_i}(\alpha + P_i^{T}) + \frac{\rho_{ij}}{f_e}(\alpha + P_i^{I}), \tag{10}$$

where ρ_{ij} is the computational complexity of $task_{ij}$ and $\rho_{ij} = w_{ij}/D_{ij}$. To simplify the analysis, we then define an offloading indicator according to Eqs. (3), (8) and (9), as follows:

$$\tau_{ij} = \frac{1}{f_i} \frac{\rho_{ij} \cdot (\alpha + P_i^A)}{\frac{1}{R_i}(\alpha + P_i^T) + \frac{\rho_{ij}}{f_e}(\alpha + P_i^I)}. \tag{11}$$

Therefore, when $\tau_{ij} > 1$, it is preferred for $task_{ij}$ to be offloaded to edge, while when $\tau_{ij} < 1$, it is preferred for $task_{ij}$ to be executed on local devices. The following part consider joint computation offloading rather than a task offloading. When the computing resources in the local and edge are both sufficient, the cost function can be represented as follows:

$$\begin{aligned} C_n &= \sum_{i=1}^{m}\sum_{j=1}^{L}\left[(1-\kappa_{ij})Q_{ij}^l + \kappa_{ij}Q_{ij}^e\right] \\ &= \sum_{i=1}^{m}\sum_{j=1}^{L} Q_{ij}^e \cdot \left[(1-\kappa_{ij})\tau_{ij} + \kappa_{ij}\right], \end{aligned} \tag{12}$$

where $\kappa_{ij} = 0$ means that $task_{ij}$ is processed locally, and $\kappa_{ij} = 1$ means that $task_{ij}$ is processed at the edge. C_n is the sum of the cost of $task_{ij}$. Minimizing C_n is equal to Eq. (9). Thus, the optimal joint offloading decision for $task_{ij}$ when the computing resources in the local and edge are both sufficient can be denoted by:

$$\kappa_{ij} = \begin{cases} 0, & \tau_{ij} \leqslant 1, \\ 1, & \tau_{ij} > 1, \end{cases} \tag{13}$$

where $i = 1, 2, \cdots, m$, and $j = 1, 2, \cdots, L$

IV. The Overloaded Task Offloading Algorithm

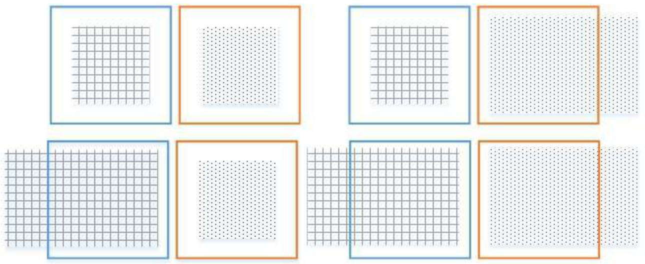
In this section, we propose two novel offloading decision algorithms, which aim to minimize the average system cost (weighted sum of delay and energy consumption) from IoT devices to the edge, and reduce the computational pressure caused by executing overloaded tasks in the next workflow.

A. Problem Formulation

On the one hand, when the computing resources of the local devices and the edge server are sufficient, we decide whether to offload according to the offloading indicator. On the other hand, however, when the computing resources are insufficient, i.e., $\sum_{j=1}^{L}(1-\kappa_{ij})w_{ij} > f_i t$ or $\sum_{i=1}^{m}\sum_{j=1}^{L}\kappa_{ij}w_{ij} > f_e t$, we need to determine which tasks should be performed locally, which ones should be performed at the edge, and which ones should be performed in the next workflow.

As depicted in Fig. 2, there are four cases in the aspect of computation resources of the MEC system:

- The total computation resource of the local device is **sufficient** for all tasks whose $\tau \leqslant 1$ in the device and at the same time the computation resource of the edge server is **sufficient** for all tasks whose $\tau > 1$.
- The total computation resource of the local device is **insufficient** for all tasks whose $\tau \leqslant 1$ in the device and the computation resource of the edge server is **sufficient** for all tasks whose $\tau > 1$.



🟦 The computation resources of a mobile device
🟧 The computation resources of the edge server
▦ The computation resources needed when computing the tasks whose $\tau < 1$
▨ The computation resources needed when computing the tasks whose $\tau > 1$

Fig. 2: Different cases in terms of computation resources in MEC systems.

- The total computation resource of the local device is **sufficient** for all tasks whose $\tau \leqslant 1$ in the device and at the same time the computation resource of the edge server is **insufficient** for all tasks whose $\tau > 1$.
- The total computation resource of the local device is **insufficient** for all tasks whose $\tau \leqslant 1$ in the device and the computation resource of the edge server is **insufficient** for all tasks whose $\tau > 1$.

Regardless of whether $task_{ij}$ is executed in the current workflow or the next workflow, the minimum cost of executing $task_{ij}$ is C_n^*, which is treated as a cost that must be paid. Here, the offloading decision is made according to Eq. (13). However, due to the insufficient computing ability of mobile devices and edge servers, not all tasks can be offloaded with minimum energy and time costs, which is exactly what we need to optimize. The extra cost incurred when tasks are not executed following the instructions of τ in the current workflow, can be formulated as:

$$|Q^e - Q^l| = \left|1 - \frac{1}{\tau}\right| \frac{w(\alpha + P^A)}{f_l} I^{tr}, \tag{14}$$

where I^{tr} is the indicator matrix of tasks executed elsewhere not guided by τ.

Furthermore, the amount of computation of tasks that are considered to be executed in the next workflow will be count into the cost function to keep the total amount of computation of tasks that are not computed in the current workflow as small as possible, which means the total amount of computation for the tasks that go to the next workflow will be punished in our cost function. Thus, our aim is to minimize the extra cost, including a penalty term of overflowed computation, which can be calculated by:

$$Cost_{extra} = \left|1 - \frac{1}{\tau}\right| CwI^{tr} + \beta w I^{next}, \tag{15}$$

where I^{next} illustrates the tasks that are considered to be executed in the next workflow, β is a penalty parameter and

$C = \frac{\alpha + P^A}{f_l}$.

$$(\boldsymbol{\mathcal{P}_1}) \min_{I^{tr}, I^{next}} : \sum_{i=1}^{m}\sum_{j=1}^{L} Cost_{extra}(I^{tr}, I^{next}), \tag{16}$$

$$\text{s.t.} : \sum_{j=1}^{L} w(I^l_{\tau<1} + I^l_{\tau>1}) \le f_i t, \tag{17}$$

$$\sum_{i=1}^{m}\sum_{j=1}^{L} w(I^e_{\tau>1} + I^e_{\tau<1}) \le f_e t, \tag{18}$$

$$I^l_{\tau<1} + I^e_{\tau>1} + I^{tr} + I^{next} = \mathbf{1}_{m\times L}, \tag{19}$$

$$I^{tr}, I^l_{\tau<1}, I^l_{\tau>1}, I^e_{\tau<1}, I^e_{\tau>1}, I^{next} \subseteq \{0,1\}^{mL} \tag{20}$$

where $I^l_{\tau<1}$ indicates the tasks whose $\tau < 1$ and meanwhile which are computed in local, $I^e_{\tau>1}$ indicates the tasks whose $\tau > 1$ and meanwhile which are computed at the edge, $I^{tr} = I^l_{\tau>1} + I^e_{\tau<1}$ indicates the tasks which are not computed following the instructions of τ in the current workflow, and I^{next} indicates the tasks which go to the next workflow. Eq. (17) indicates the total computation of tasks executed locally is less than the computing ability of the mobile device during a time slot. Eq. (18) indicates that the total computation of tasks offloaded to the edge server is less than the computing ability of the edge server during a time slot.

The objective function is to minimize the total extra cost, which includes two parts: one is the extra cost resulting from the concessions due to the limitation of the computing ability, and the other one is the penalty term for the overflowed computation.

Lemma 1. *$\boldsymbol{\mathcal{P}_1}$ is a multiple knapsacks problem and NP-hard.*

Proof. First, we consider an instance of $\boldsymbol{\mathcal{P}_1}$ whose $\tau > 1$ for all tasks. Then, we can get $I^e_{\tau<1}, I^l_{\tau<1} = 0$, $I^l = I^l_{\tau>1}$, $I^{tr} = I^l$, and $I^{next} = \mathbf{1}_{m\times L} - I^e - I^e$. Then $Cost_{extra} = (|1 - \frac{1}{\tau}|C - \beta)wI^l - \beta wI^e + \beta w$. Solving the instance is equal to solving the following problem:

$$(\boldsymbol{\mathcal{P}_2}) \min_{I^l, I^e} : \sum_{i=1}^{m}\sum_{j=1}^{L} \left[\left(\left|1 - \frac{1}{\tau}\right|C - \beta\right) wI^l - \beta wI^e\right], \tag{21}$$

$$\text{s.t.} : \sum_{j=1}^{L} wI^l \le f_i t, \tag{22}$$

$$\sum_{i=1}^{m}\sum_{j=1}^{L} wI^e \le f_e t, \tag{23}$$

$$I^l + I^e \le \mathbf{1}_{m\times L}, \tag{24}$$

$$I^l, I^e \subseteq \{0,1\}^{mL}. \tag{25}$$

The multiple knapsack problem $\boldsymbol{\mathcal{P}_3}$ described in [44] is known as a NP-hard problem.

$$(\boldsymbol{\mathcal{P}_3}) \max_{x} : \sum_{i=1}^{m}\sum_{j=1}^{n} v_i x_{ij}, \tag{26}$$

$$\text{s.t.} : \sum_{j=1}^{n} w_j x_{ij} \le c_i, \tag{27}$$

$$\sum_{i=1}^{m} x_{ij} \le \mathbf{1}_n, \tag{28}$$

$$x_{ij} \in \{0,1\}^n. \tag{29}$$

$\boldsymbol{\mathcal{P}_3}$ can be polynomial-time reducible to $\boldsymbol{\mathcal{P}_2}$, which is an instance of $\boldsymbol{\mathcal{P}_1}$. The reduction is as follows:

- Let $I^l = [I^1, I^2, \cdots, I^m]$. Reshape I^e, w as vectors I^e_{flat}, w_{flat} whose lengths are $m \times L$. x_i is corresponding to I^i, where $i = 1, \cdots, m$ and $x_{m+1} = I^e_{flat}$. Let $v_i = -(|1 - \frac{1}{\tau}|C - \beta)w$, where $i = 1, \cdots, m$ and $v_{m+1} = \beta w^T_{flat}$. Thus, Eq. (26) can be reduced to
$$\min_{I^l, I^e_{flat}} : \sum_{i=1}^{m}\sum_{j=1}^{L}\left[\left(\left|1 - \frac{1}{\tau}\right|C - \beta\right)wI^l - \frac{\beta}{mL} w^T_{flat} I^e_{flat}\right],$$
which is equal to Eq. (22).
- Let $c_i = f_i t$, $i = 1, \cdots, m$, $c_{m+1} = f_e t$, Then Eq. (27) can be reduced to $\sum_{j=1}^{L} w_j I^i \le f_i t$, $w^T_{flat} I^e_{flat} \le f_e t$ which are equal to Eqs. (23) and (24).
- $\sum_{i=1}^{m} x_{ij} \le \mathbf{1}_n$ and $x_{ij} \in \{0,1\}^n$ can be reduced to $I^l + I^e \le \mathbf{1}_{m\times L}$ and $I^l, I^e \subseteq \{0,1\}^{mL}$, respectively.

There are $m + 1$ containers whose capacity is the computing power of devices. I^i and I^e indicate tasks are packed into a local knapsack and an edge knapsack, respectively. Therefore, we can conclude that $\boldsymbol{\mathcal{P}_1}$ is a multiple knapsack problem and it is NP-hard. □

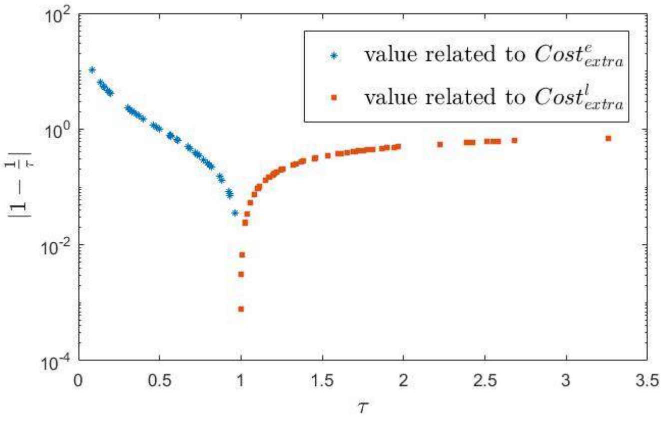


Fig. 3: The extra cost generated in the local and at the edge.

The extra costs generated in the local and at the edge are defined as $Cost^l_{extra}$ and $Cost^e_{extra}$, respectively.

$$Cost^l_{extra} = \left|1 - \frac{1}{\tau}\right| CwI^l_{\tau>1} + \beta wI^{next}_{\tau<1}, \tag{30}$$

$$Cost^e_{extra} = \left|1 - \frac{1}{\tau}\right| CwI^e_{\tau<1} + \beta wI^{next}_{\tau>1}. \tag{31}$$

From Eqs. (30) and (31), we find both $Cost_{extra}^{l}$ and $Cost_{extra}^{e}$ are positively associated with $|1-\frac{1}{\tau}|$. As depicted in Fig. 3, when the value of τ is closer to 1, the value of $|1-\frac{1}{\tau}|$ is smaller, which means it is preferred for $Cost_{extra}^{l}$ and $Cost_{extra}^{e}$ to execute tasks whose τ is closer to 1. In reality, the conclusion is understandable: $\tau_{ij}=1$ means executing on local is equal to executing on edge for $task_{ij}$. When the computing ability is insufficient, we need to choose which tasks are executed in the current workflow to minimize the extra cost and the rest of tasks will be executed in the next workflow. As mentioned above, the cost of the current workflow consists of two parts: one part is the lowest cost we need to paid whether tasks executed on local or on edge; the other part is the extra cost of tasks which are indicated by I^{tr}. The former cost is fixed, and the latter cost is what we need to optimize. In order to choose tasks from I^{tr} to minimizing the extra cost of the current workflow, we execute tasks in the current workflow whose τ is closer to 1 by using the limited remaining computing ability.

Lemma 2. *A mobile device user i has L tasks, which are sorted by $|1-\frac{1}{\tau}|$ from smallest to largest. I_i^{tr} is the indicator vector of user i which illustrates tasks executed elsewhere, without being guided by whether τ is greater than 1 or not. When the amount of computation of tasks are equal and I_i^{tr} indicate the task whose τ is closer to 1 as 1, $Cost_{extra}$ is smaller.*

Proof. First, let's assume I_i^{tr} and $I_i^{tr'}$ as follows:

$$I_i^{tr}=[\underbrace{1,1,\cdots,\underset{k-1}{1}}_{1},\underset{k}{0},\underbrace{0,\cdots,0}_{0}], \qquad k\geq 2,$$
$$I_i^{tr'}=[\underbrace{1,1,\cdots,\underset{k-1}{0}}_{1},\underset{k}{1},\underbrace{0,\cdots,0}_{0}], \qquad k\geq 2.$$

Then, $Cost_{extra}$ with the indicator I_i^{tr} can be calculated as:

$$\begin{aligned}Cost_{extra}(I_i^{tr}) &= \left|1-\frac{1}{\tau}\right|CwI^{tr}+\beta wI^{next}\\ &=\sum_{j=1}^{k-1}\left|1-\frac{1}{\tau_j}\right|Cw+\beta wI^{next}\\ &\leq\sum_{j=1}^{k-2}\left|1-\frac{1}{\tau_j}\right|Cw+\left|1-\frac{1}{\tau_k}\right|Cw+\beta wI^{next}\\ &=Cost_{extra}(I_i^{tr'})\end{aligned}$$

□

$\mathcal{P}_1$ is a 0-1 NP-hard problem [45], whose computational complexity will significantly arise due to the increase of binary variables. We convert $\mathcal{P}_1$ into several simple knapsack problems based on the relationship between $Cost_{extra}$ and τ, which are prone to be solved. On the one hand, we translate the threshold of τ from 1 to a new vector Θ which is more suitable to the current environment, and we decide whether to offload the task or not by the threshold and τ. On the other hand, the tasks which are assigned on the local or on the edge will further be allocated computational resources, and the tasks which are not be allocated computational resources will be computed in the next workflow. The preliminary offloading policies $I^l(\Theta)$ and $I^e(\Theta)$ of the current threshold Θ in the local and at the edge are computed as follows, respectively.

$$I_{ij}^{l}(\Theta)=\begin{cases}1, & \tau_{ij}\leq\Theta(i),\\ 0, & otherwise.\end{cases} \tag{32}$$

$$I_{ij}^{e}(\Theta)=\begin{cases}1, & \tau_{ij}>\Theta(i),\\ 0, & otherwise.\end{cases} \tag{33}$$

Furthermore, two simple knapsack problem are solved in local and edge to get the final offloading policy $I_{ij}^{*}(\theta)$ of the current threshold Θ, which is decided by $\kappa^l(\Theta)$ and $\kappa^e(\Theta)$ as follows:

$$I_{ij}^{*}(\Theta)=\begin{cases}0, & \kappa^l(\Theta)=1,\\ 1, & \kappa^e(\Theta)=1,\\ 2, & \kappa^l(\Theta)=0\ \&\ \kappa^e(\Theta)=0.\end{cases} \tag{34}$$

where $I_{ij}^{*}(\Theta)=0$ means computing $task_{ij}$ in local under the current Θ, $I_{ij}^{*}(\Theta)=1$ means computing $task_{ij}$ at the edge under the current Θ, and $I_{ij}^{*}(\Theta)=2$ means computing $task_{ij}$ in the next workflow under the current Θ. $\kappa^l(\Theta)$ and $\kappa^e(\Theta)$ are the results of two simple knapsack problems. We describe two simple knapsack problems in the following parts.

Therefore, the total cost of the system under the current θ can be calculated as:

$$cost(\Theta)=\sum_{i=1}^{m}\sum_{j=1}^{L}Cost(I_{ij}^{*}(\Theta)), \tag{35}$$

where

$$Cost(I_{ij}^{*}(\Theta))=\begin{cases}Q_{ij}^{l}, & I_{ij}^{*}(\Theta)=0,\\ Q_{ij}^{e}, & I_{ij}^{*}(\Theta)=1,\\ \beta w_{ij}, & I_{ij}^{*}(\Theta)=2.\end{cases} \tag{36}$$

B. Find the Optimal Θ^*

In MEC systems, the active power P_i^A, the idle power P_i^I, the transmission power P_i^T and the computational speed f_i are generally different for varying mobile devices, which have a great impact on the distribution of τ. For instance, as shown in Fig. 4, the values of τ of tasks in the $4th$ device are significantly different from those in the $5th$ device. So we define a threshold vector $\Theta_{m\times 1}$, where different mobile devices may have different threshold values. In the case of task overflow, the number of tasks or the amount of computation may be very large, so it is very urgent to relieve the computational pressure in the next workflow. In order to tackle this challenge, we propose two algorithms based on the multiple knapsack problem to offload tasks: the OAKGM algorithm is designed to search for an optimal threshold vector, while the OAMKP algorithm is designed to search for an optimal threshold value. Because the large difference between the values of τ of different tasks may cause a lot of unnecessary calculation and the values of τ of each device can be roughly divided into 2 3 clusters in Fig. 4, we utilize K-means clustering to cluster $\tau_i=\{\tau_{i1},\tau_{i2},\cdots,\tau_{iL}\}$ and get the K-table as shown in Fig. 6, where the values of τ_i are divided into three clusters.

When the computational resources of each device are sufficient, the threshold is one. However, when the computational

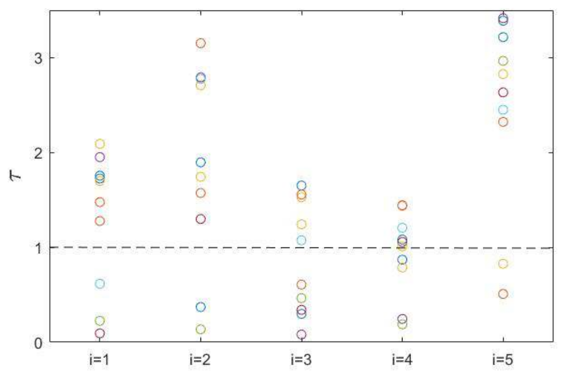
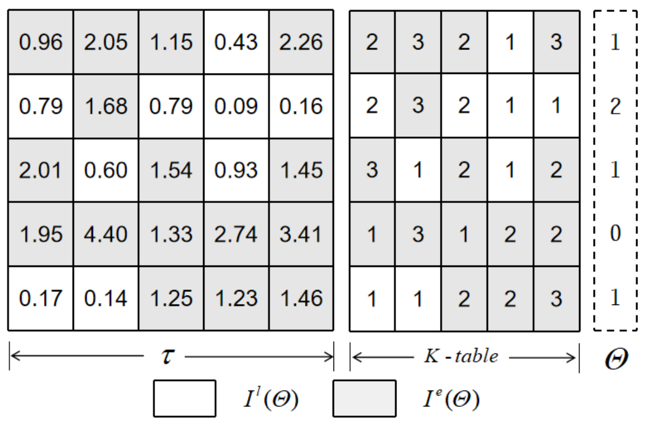


Fig. 4: The distribution of τ_i $(i = 1, 2, \cdots, 5)$.



0.96	2.05	1.15	0.43	2.26
0.79	1.68	0.79	0.09	0.16
2.01	0.60	1.54	0.93	1.45
1.95	4.40	1.33	2.74	3.41
0.17	0.14	1.25	1.23	1.46

2	3	2	1	3
2	3	2	1	1
3	1	2	1	2
1	3	1	2	2
1	1	2	2	3

1
2
1
0
1

Fig. 6: The illustration of the K-table

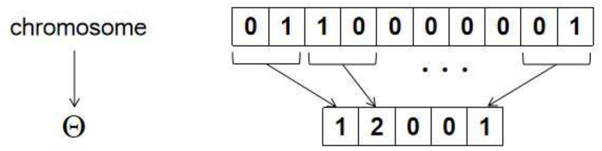


Fig. 5: The encoding of Θ in the OAKGM algorithm when the number of users is five.

resources are insufficient, we have to make a trade-off between the cost and the limitation of computational resources. We propose the OAKGM algorithm, the purpose of which is to search for the most suitable Θ to minimize the system cost and alleviate the computational pressure in the next workflow. After computing K-table by utilizing K-means clustering, we obtain the optimal threshold vector Θ^* through a genetic algorithm [46], [47]. The algorithmic process of the genetic algorithm is described as follows:

- **Encoding Chromosomes**: As depicted in Fig. 5, we encode every chromosome as a binary vector whose length is $m \times 2$, which corresponds to a threshold vector Θ.
- **Selection**: The fitness function of the genetic algorithm is equal to $-Cost(I^*_{ij}(\Theta))$, because the aim of our algorithm is to find an optimal Θ that minimizes the $Cost$ of the overflowed MEC system.
- **Crossover and Mutation**: To generate more high-fitness chromosomes, we use single-point crossover and swap mutation [47].

The proposed OAKGM algorithm is as illustrated in Fig. 6, the numbers of K-table correspond to the clusters of K-means clustering. $I^l(\Theta)$ and $I^e(\Theta)$ are the indicator matrices that represent the tasks considered to be executed in the local and at the edge, respectively, which can be calculated as follows:

$$I^l_{ij}(\Theta) = \begin{cases} 1, & K-table_{ij} \le \Theta(i), \\ 0, & \text{otherwise}. \end{cases} \tag{37}$$

$$I^e_{ij}(\Theta) = \begin{cases} 1, & K-table_{ij} > \Theta(i), \\ 0, & \text{otherwise}. \end{cases} \tag{38}$$

C. Optimization in the Local

In the case when the computing resources in the local are insufficient to compute all tasks whose $I^l_{ij}(\Theta)$ are equal to 1, which means $\sum_{j=1}^{l^l_i(\Theta)} w^l_{ij}(\Theta) > f_i t$, where $l^l_i(\Theta)$ is the number of tasks for the i^{th} device that satisfies $I^l_{ij}(\Theta) = 1$ and $w^l_i(\Theta)$ is the amount of computation corresponding to these tasks.

In this part, we need to determine which tasks should compute in the local and which tasks should compute in the next workflow. So it can be regarded as a knapsack problem: the capacity of the knapsack is $f_i t$; the item weight of the knapsack is w_{ij}, and the item value of the knapsack is $V^l_{ij}(\Theta)$, where $V^l_{ij}(\Theta)$ is the difference between the cost in the case when $task_{ij}$ is computed in the next workflow and the cost in the case when $task_{ij}$ computed in the local. The greater the difference is, the more cost it takes for the task to enter the next workflow, that is, the better it is computed in the local.

$$\begin{aligned} V^l_{ij}(\Theta) &= \beta w^l_{ij}(\Theta) - Q^l_{ij}(\Theta) \\ &= w_{ij}(\Theta)\Big(\beta - \frac{\alpha + P_i}{f_i}\Big), \end{aligned} \tag{39}$$

where the unit value of items is $v^l_i = \beta - \frac{\alpha+P_i}{f_i}$. Therefore, when Θ is given, the optimization problem can be described as follows:

$$(\boldsymbol{\mathcal{P}_4}) \quad \max_{\kappa^l_i(\Theta)} : \sum_{j}^{l^l_i(\Theta)} v^l_{ij} w^l_{ij}(\Theta) \kappa^l_{ij}(\Theta), \tag{40}$$

$$\text{s.t.} : \sum_{j}^{l^l_i(\Theta)} w^l_{ij}(\Theta) \kappa^l_{ij}(\Theta) \le f_i t, \tag{41}$$

$$\kappa^l_i(\Theta) \in \{0, 1\}^{l^l_i(\Theta)}, \tag{42}$$

where $\kappa^{l*}_i(\Theta)$ is the best solution under the current Θ, which makes the total cost of tasks whose $I^l_{ij}(\Theta) = 0$ are equal to 1 the lowest in the current Θ.

D. Optimization at the Edge

When the computing resources at the edge are insufficient to compute all tasks whose $I^e_{ij}(\Theta)$ are equal to 1, which means $\sum_{k=1}^{l^e(\Theta)} w^e_k(\Theta) > f_e t$, where $l^e(\Theta)$ is the number of tasks satisfying $I_{ij}^e(\Theta) = 1$, and $w^e(\Theta)$ is the computation corresponding to these tasks.

In this part, we need to determine which tasks should compute at the edge and which ones should compute in the next workflow. It can also be regarded as a knapsack problem: the capacity of the knapsack is $f_e t$, the item weight of the knapsack is $w_k(\Theta)$, and the item values of the knapsack is $V_k^e(\Theta)$, where $V_k^e(\Theta)$ is the difference between the cost if $task_{ij}$ is computed in the next workflow and the cost if $task_k$ is computed at the edge. The greater the difference is, the more cost it takes for the tasks to go to the next workflow, that is, the better it is executed at the edge.

$$\begin{aligned} V_k^e(\Theta) &= \beta w_k^e(\Theta) - Q_k^e(\Theta) \\ &= w_k^e(\Theta)\Big[\beta - \frac{\alpha + P_i}{f_i \tau_k^e(\Theta)}\Big], \end{aligned} \tag{43}$$

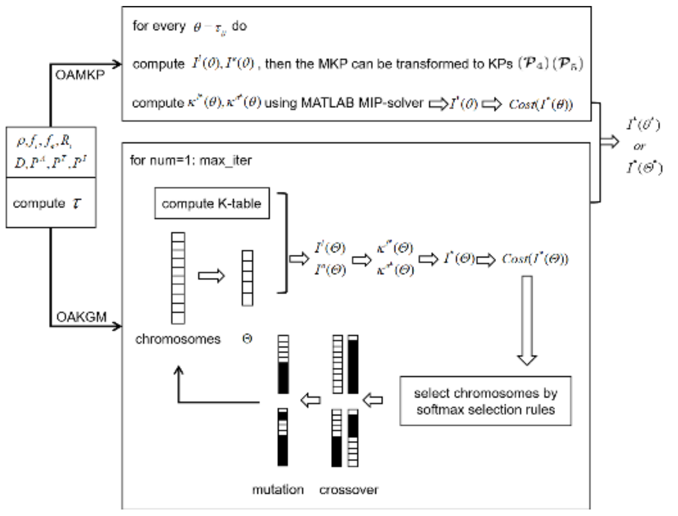
where $k = 1, 2, \cdots, l^e(\Theta)$ and $\tau_k^e(\Theta)$ is the offloading indicator corresponding to $w_k^e(\Theta)$. The unit value of items of the knapsack is $v_k^e(\Theta) = \beta - \frac{\alpha+P_k}{f_k \tau_k^e(\Theta)}$. Thus, when Θ is given, the problem is described as follows:

$$(\boldsymbol{\mathcal{P}_5}) \quad \max_{\kappa^e(\Theta)} : \quad \sum_k^{l^e(\Theta)} v_k^e(\Theta) w_k^e(\Theta) \kappa_k^e(\Theta), \tag{44}$$

$$\text{s.t.} : \quad \sum_k^{l^e(\Theta)} w_k^e(\Theta) \kappa_k^e(\Theta) \le f_e t, \tag{45}$$

$$\kappa^e(\Theta) \in \{0, 1\}^{l^e(\Theta)}, \tag{46}$$

where $\kappa^{e*}(\Theta)$ is the best solution that makes the total cost of tasks whose $I_{ij}^l(\Theta)$ are equal to 1. For convenience, we transform $\kappa^{e*}(\Theta)$ to a matrix, whose dimension is $m \times L$.



for every $\theta - \tau_{ij}$ do
compute $I^l(\theta), I^e(\theta)$, then the MKP can be transformed to KPs $(\mathcal{P}_4)$ $(\mathcal{P}_5)$
compute $\kappa^{l*}(\theta), \kappa^{e*}(\theta)$ using MATLAB MIP-solver $\Rightarrow I^*(\theta) \Rightarrow Cost(I^*(\theta))$
OAMKP
ρ, f_i, f_e, R_i
D, P^A, P^T, P^I
compute τ
OAKGM
for num=1: max_iter
compute K-table
chromosomes
Θ
$I^l(\Theta)$ $I^e(\Theta)$ $\Rightarrow$ $\kappa^{l*}(\Theta)$ $\kappa^{e*}(\Theta)$ $\Rightarrow I^*(\Theta) \Rightarrow Cost(I^*(\Theta))$
select chromosomes by softmax selection rules
mutation
crossover
$I^*(\theta^*)$ or $I^*(\Theta^*)$

Fig. 7: The illustration of algorithms

E. Two Offloading Algorithms based on Knapsack Problem

In the case when the current threshold is Θ, we gather $\kappa^{e*}(\Theta)$ and $\kappa^{l*}(\Theta)$ together so that we can obtain $I^*(\Theta)$ according to Eq. (34).

Algorithm 1 shows the process of obtaining the optimal solution through K-means clustering, genetic algorithm and

Algorithm 1 Offloading Algorithm based on K-means clustering and Genetic algorithm for solving the Multiple knapsack problem (OAKGM)

```
Input: $\alpha$
Output: $I^*(\Theta^*)$
  for workflows do
      Initialize env=[$\rho$, $f_e$, $f_i$, $R_i$, $D$, $P^A$, $P^T$, $P^I$]
      Compute $\tau$ by Eq. (11)
      Compute K-table by K-means clustering
      Initialize chromosomes
      for $num = 1 : max\_iter$ do
          Compute $\Theta$ by chromosomes as Fig. 5
          Compute $I^l(\Theta)$ and $I^e(\Theta)$ using $\Theta$ and K-table by
          Eq. (37) and Eq. (38), respectively
          Compute $\kappa^{l*}(\Theta)$ by solving ($\boldsymbol{\mathcal{P}_4}$)
          Compute $\kappa^{e*}(\Theta)$ by solving ($\boldsymbol{\mathcal{P}_5}$)
          Compute $I^*(\Theta)$ by Eq. (34)
          Compute $-Cost(I^*(\Theta))$ as fitness
          Do a softmax selection by fitness
          Do single-point crossovers
          Randomly select bits and flip them and generate new
          chromosomes
      end for
  end for
  return $Cost^*$ and $I^*(\Theta^*)$
```

Algorithm 2 Offloading Algorithm based on Multiple Knapsack Problem (OAMKP)

```
Input: $\alpha$
Output: $I^*(\theta^*)$
  for workflows do
      Initialize env=[$\rho$, $f_e$, $f_i$, $R_i$, $D$, $P^A$, $P^T$, $P^I$]
      Initialize $cost$, $s = 1$
      Compute $\tau$ by Eq. (11)
      Reshape and sort $\tau$ as $\tau'$
      for $s = 1 : m \times L$ do
          $\theta = \tau'(s)$
          Compute $\kappa^{l*}(\theta)$ by solving ($\boldsymbol{\mathcal{P}_4}$)
          Compute $\kappa^{e*}(\theta)$ by solving ($\boldsymbol{\mathcal{P}_5}$)
          Compute $I^*(\theta)$
          Compute the $Cost(I^*(\theta))$ of $I^*(\theta)$
          if $Cost(I^*(\theta)) < Cost^*$ then
              $Cost^* = Cost(I^*(\theta))$
              $s^* = s$
          end if
      end for
      $\theta^* = \theta(s^*)$
      Compute $I^*(\theta^*)$
  end for
  return $Cost^*$ and $I^*(\theta^*)$
```

knapsack problem. First the values of τ_i is classified into K clusters. Then we utilize genetic algorithm to find the optimal Θ rather than make a traversal on K^M cases whose time complexity grows exponentially with M, because genetic algorithm have advantages over traversal when computing scale grows exponentially. We find that the value of τ when the majority of tasks that are offloaded to the edge is greater than the value of τ when they are executed locally.

As the number of users or tasks becomes larger, we propose a simplified OAMKP algorithm to speed up computing, as illustrated in **Algorithm 2**, where the simplified threshold θ is a constant. Since the offloading indicator τ is discrete, we iterate over all values in τ as the possible values of θ to get an approximate optimal solution of the overloaded MEC system. Referring to [48], the time complexity of the OAMKP algorithm is $O\big(m \times L \times [\sum_i^m l_i^l(\theta)^2 + l^e(\theta)^2]\big)$.

V. Performance Evaluation

In this section, we conduct simulations to verify the effectiveness of our proposed algorithms under various parameter settings. Specifically, we compare the proposed method with several benchmark offloading schemes in the MEC environment.

A. Simulation Settings

In a workflow, tasks arrival and their data sizes are expressed as $\mathcal{D} = \{D_{ij} | i = 1, 2, \cdots, m$ and $j = 1, 2, \cdots, L\}$. The number of mobile devices in local is set as 10, the numbers of tasks in small scale environment are set as 5, 6, 7, 8 which corresponds to 50, 60, 70, 80 nodes, respectively. In big scale environment the numbers of tasks are set as 10, 15 and 20, which corresponds to 100, 150, 200 nodes, respectively. The number of clusters of τ_i is 3. In order to make sure that most of $\beta w_{ij} \in [0, 10]$, we set $\beta \in [1/w_{\max}, 20/w_{\max}]$, where $w_{\max} = D_{\max} \cdot \rho_{\max}$. Similar to [49], the parameter settings for our experiment are described in Table III.

In real-world scenarios, tasks arrive randomly and the number of tasks of each devices are not same in a time slot. Thus, the data sizes of the quarter tasks are randomly set to 0 in experiments so that the number of tasks varies between devices. To improve the reliability, we conduct experiments in fifty different environments and each of them is generated randomly according to the parameter settings, and we run fifty simulations under each environment. The computing ability of each devices f_i, the active power P_i^A, the idle power P_i^I, and the transmission power P_i^T are all different in different environments. So the simulations consider not only the changes of the numbers of tasks but also the changes of devices. Furthermore, the resource occupancy rate is defined as follows:

$$\textit{resource occupancy rate} = \frac{\sum_{i=1}^m \sum_{j=1}^L w_{ij} \cdot I'_{ij}}{\sum_{i=1}^m f_i t + f_e t}, \qquad (47)$$

where $I'_{ij} = \begin{cases} 0, & I^*_{ij} = 2, \\ 1, & \text{otherwise.} \end{cases}$ is the indicator matrix of tasks computed in the current workflow.

TABLE III: Evaluation Parameters

Parameters	Value
R	4 Gbps
P^A	$[0.1, 1]$ W
P^I	$[0.001, 0.002]$ W
P^T	$[0.01, 0.1]$ W
D	$[500, 1000]$ Kbit
ρ	$[10, 500]$
f_e	4000 MHz
f_i	$[500, 1200]$ MHz
α	1
β	4×10^{-8}

B. Benchmarks

We compare the proposed algorithms with five baseline offloading schemes, which are listed as follows:

- **Genetic Algorithm (GA)** [46]: In this method, we regard each offloading decision as a $m \times L$ dimensional chromosome. The number of chromosomes is 128, and the length of each chromosome is $m \times L$. The maximum number of generations is 200. Through a fitness function related to the cost, we select the superior chromosome and eliminate the inferior chromosomes through the processes of crossover, mutation, and selection. Finally, we can obtain the offloading decision with the highest fitness value.
- **Particle Swarm Optimization (PSO)** [50]: In this method, we regard each offloading decision as a map. The number of particles is 128. The velocity extremum that limits the maximum change of a particle in an iteration is 0.2. The two learning factors are both set to 2. The inertia weight is 0.8. When a particle arrives at a certain location on the map, the map will change its offloading options at this location and get a fitness value. Finally, we can obtain the offloading decision with the highest fitness value.
- **Random Offloading Policy (ROP)**: In this policy, we make offloading decisions randomly.
- **Only-Local Policy (OLP)**: There are two options for tasks under this policy: one is to compute tasks locally, and the other is to compute tasks in the next workflow.
- **Only-Edge Policy (OEP)**: There are two options for tasks under this policy: one is to compute tasks at the edge, and the other is to compute tasks in the next workflow.

C. Performance Comparison

Fig. 8 shows the average cost under different offloading methods. When compared with GA, PSO, ROP, OLP and OEP algorithms, the OAKGM algorithm is significantly superior in terms of average cost. For instance, compared with GA, the OAMKP method achieves 4.19%, 9.09%, 8.89% and 7.31% improvements when the numbers of tasks are 5, 6, 7 and 8, respectively. Compared with PSO, it achieves 8.18%, 7.13%, 5.50% and 4.87% improvements when the numbers of tasks are 5, 6, 7 and 8, respectively.

Fig. 9 shows that when compared with GA, PSO, ROP, OLP and OEP algorithms, the OAMKP algorithm is significantly

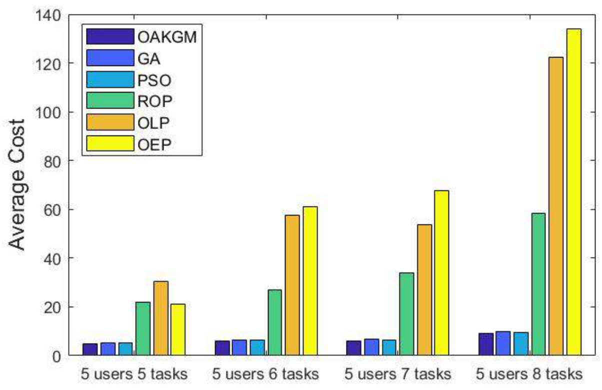


Fig. 8: Average costs of different algorithms under environments with varying numbers of users and tasks.

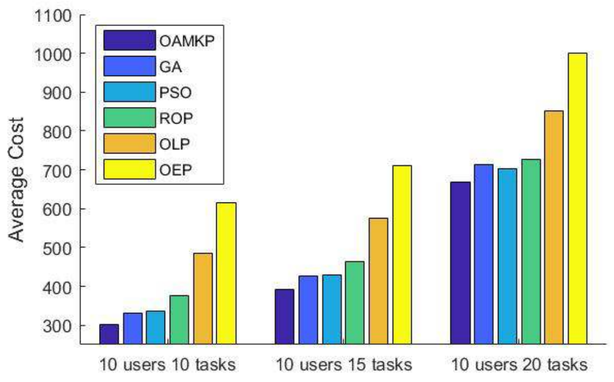


Fig. 9: Average costs of different algorithms under environments with varying numbers of users and tasks

superior in terms of average cost. For instance, compared with GA, the OAMKP method achieves 8.66%, 8.07% and 6.35% improvements when the numbers of tasks are 10, 15 and 20, respectively. Compared with PSO, it achieves 10.12%, 8.38% and 4.71% improvements when the numbers of tasks are 10, 15 and 20, respectively. The total amount of computation of the tasks when $m = 10$ and $L = 20$ is much larger than that when $m = 10$ and $L = 10$, which means that more computations can be overflowed, leading to a decrease in the improvement percentage as the number of tasks L increases.

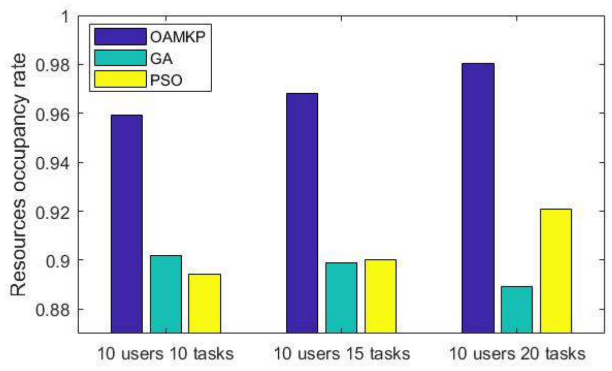


Fig. 10: The resource occupancy rates of different algorithms under environments with varying numbers of users and tasks

Due to the limitations of ROP, OLP and OEP schemes in terms of the resource occupancy rate, we will ignore them in the remaining experiments. Thus, we mainly compare the proposed OAMKP algorithm with GA and PSO algorithms.

Fig. 10 shows the average resource occupancy rates of the OAMKP, GA and PSO algorithms. It can be seen that our algorithm has a larger resource occupancy rate than the GA and PSO algorithms, which means that when our offloading-decision algorithm is deployed in the MEC system, the amount of computation to enter the next workflow will be greatly reduced. When the number of tasks increases, the average resource occupancy rates of our algorithm and the PSO algorithm increase, while the resource occupancy rate of the GA algorithm decreases. Obviously, when mobile devices and tasks become intensive, our algorithm greatly outperforms the GA and PSO algorithms.

Fig. 11 shows the average time cost under various environments with different numbers of users and tasks. Combined with Fig. 10, we know that our algorithm can execute more tasks in a similar or shorter time when compared with the GA and PSO algorithms.

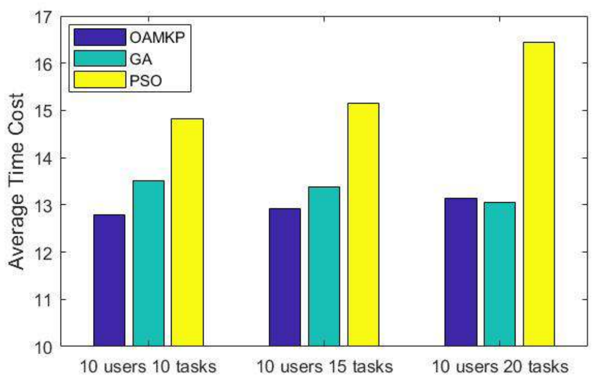


Fig. 11: Average time costs of different algorithms under environments with varying numbers of users and tasks

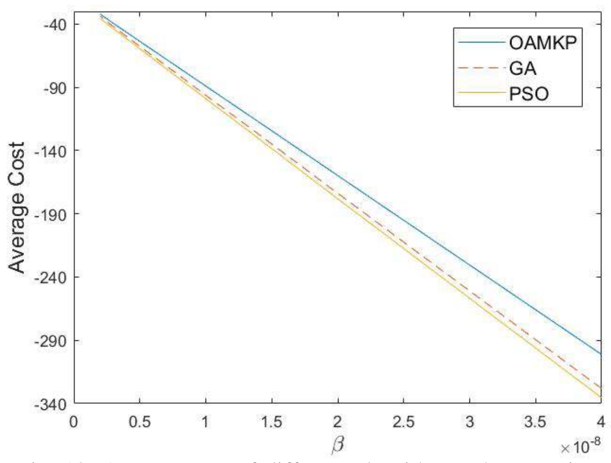


Fig. 12: Average cost of different algorithms when varying β

Figs. 12 and 13 show the influence of different β on the results of different algorithms. Under such β, most of the overflowed cost βw_{ij} is in $[0, 10]$, where w_{ij} is the amount of computation of the task, which is going to the next workflow.

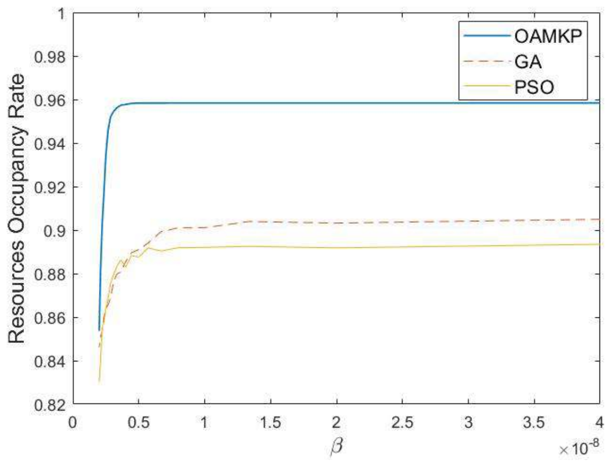


Fig. 13: Average resource occupancy rates of different algorithms when varying β

We can see that even though β changes, the result of the GA algorithm is similar to that of the PSO algorithm. The proposed algorithm always achieves better than the GA and PSO algorithms under different β, and the resource occupancy rate of our algorithm can reach 95.84%, which is much higher than that of the GA and PSO algorithms.

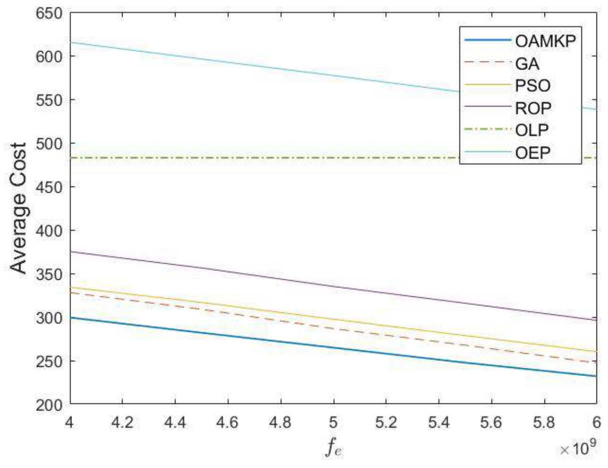


Fig. 14: Average costs of different algorithms when varying f_e

Fig. 14 shows the impact of different f_e on the average cost. When f_e increases, the average cost of OLP will not change because the tasks are only executed locally. The average costs of other algorithms decrease as the available total computing resources of each workflow is increasing. For environments with different f_e, the offloading decision of our algorithm is the best choice.

Fig. 15 shows the influence of different f_e on the resource occupancy rate. When f_e increases, the average resource occupancy rate of our algorithm decreases, while the rates of the GA and PSO algorithms increase slightly because of the randomness of these two algorithms, which makes them perform well in the environment where tasks are slightly more sparse. However, the best resource occupancy rate is still far from the OAMKP algorithm under different f_e.

Considering the queue effect, we compare it with two

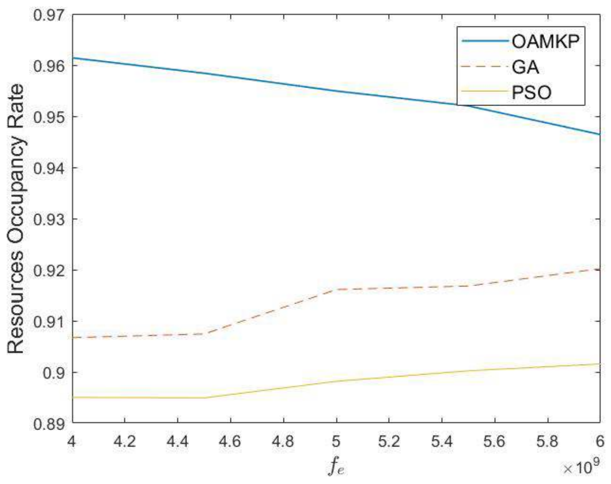


Fig. 15: Average resource occupancy rate of different algorithms when varying f_e

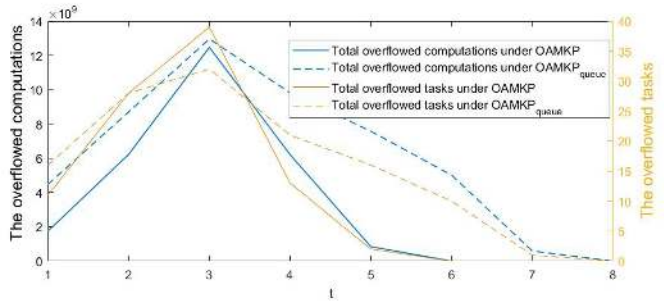


Fig. 16: Total overflowed computations and tasks under OAMKP and $OAMKP_{queue}$

methods: one is OAMKP, the other is OAMKP$_{queue}$, which replaces the penalty term for overflowed computations with the penalty term for the queue. We utilize a product of the penalty coefficient γ and the number of unexecuted tasks in the current workflow as the queueing penalty term. After simple scaling, we set the penalty coefficient to 5, which corresponds to $\beta = 4 \times 10^{-8}$. As shown in Fig. 16, the solid lines and the dotted lines correspond to OAMKP and OAMKP$_{queue}$ respectively, while the blue lines and yellow lines correspond to the total overflowed computations and total overflowed tasks respectively. As Fig. 16 shows, OAMKP ends tasks earlier than OAMKP$_{queue}$, which means that optimizing overflowed computations is more suited to solve the task-overflowed situation than optimizing the queue because the latter is a coarse-grain optimization.

VI. Conclusion and Future Work

In this paper, we consider the task-overflowed situation when the total amount of computations of the tasks to be carried out exceeds the total computing capacity of the MEC system. We design a novel task offloading indicator in local-edge collaborative computing environments, the purpose of which is not only to minimize the system cost, but also to alleviate the computational pressure of the next workflow. On the basis of the indicator, we propose two offloading algorithms based on the multiple knapsack problem, i.e., OAKGM and OAMKP. It is found that the former is more suitable for small-scale MEC systems composed of several users and tasks, while the latter is more suitable for larger-scale MEC systems with numerous users or multiple tasks. Experimental results demonstrate that the proposed algorithms can achieve better performance than existing GA and PSO algorithms. It can make better use of the computing capacities of IoT devices and edge servers, greatly avoid resource occupation at the edge nodes, and effectively reduce the computational pressure of the next workflow.

For future work, by utilizing energy harvesting technologies [51], we will comprehensively consider the characteristics of abundant computing resources in cloud computing and low transmission delay in green and sustainable MEC systems. In addition, we will attempt to tackle the task-overflowed issues in serverless edge computing frameworks [52], [53] with a focus on joint computation offloading and resource allocation.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant No. 62071327 and 61801325.

References

[1] K. Nakamura, P. Manzoni, M. Zennaro, J.-C. Cano, C. T. Calafate, and J. M. Cecilia, "FUDGE," in *Proceedings of the 1st Workshop on Experiences with the Design and Implementation of Frugal Smart Objects*. ACM, Sep. 2020.

[2] D. C. Nguyen, P. N. Pathirana, M. Ding, and A. Seneviratne, "Privacy-preserved task offloading in mobile blockchain with deep reinforcement learning," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2536–2549, dec 2020.

[3] M. Xue, H. Wu, R. Li, M. Xu, and P. Jiao, "EosDNN: An efficient offloading scheme for DNN inference acceleration in local-edge-cloud collaborative environments," *IEEE Transactions on Green Communications and Networking*, pp. 1–1, 2021.

[4] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

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

[6] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. IEEE, Apr. 2018.

[7] H. Wu, Y. Sun, and K. Wolter, "Energy-efficient decision making for mobile cloud offloading," *IEEE Transactions on Cloud Computing*, vol. 8, no. 2, pp. 570–584, Apr. 2020.

[8] S. Malik, H. Akram, S. S. Gill, H. Pervaiz, and H. Malik, "EFFORT: Energy efficient framework for offload communication in mobile cloud computing," *Software: Practice and Experience*, 2020.

[9] L. Huang, X. Feng, L. Zhang, L. Qian, and Y. Wu, "Multi-server multi-user multi-task computation offloading for mobile edge computing networks," *Sensors*, vol. 19, no. 6, p. 1446, Mar. 2019.

[10] I. A. Elgendy, W.-Z. Zhang, Y. Zeng, H. He, Y.-C. Tian, and Y. Yang, "Efficient and secure multi-user multi-task computation offloading for mobile-edge computing in mobile IoT networks," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2410–2422, dec 2020.

[11] A. Al-Shuwaili and O. Simeone, "Energy-efficient resource allocation for mobile edge computing-based augmented reality applications," *IEEE Wireless Communications Letters*, vol. 6, no. 3, pp. 398–401, Jun. 2017.

[12] J. Wang, Y. Yue, R. Wang, M. Yu, J. Yu, H. Liu, X. Ying, and R. Yu, "Energy-efficient admission of delay-sensitive tasks for multi-mobile edge computing servers," in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, Dec. 2019.

[13] Y. Geng, Y. Yang, and G. Cao, "Energy-efficient computation offloading for multicore-based mobile devices," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. IEEE, Apr. 2018.

[14] Y. Meng and J. Dai, "Energy-efficient joint computation offloading and resource allocation in multi-user mec systems," *Journal of Physics: Conference Series*, vol. 1693, no. 1, p. 012042 (12pp), 2020.

[15] Y. Kao, B. Krishnamachari, M. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Transactions on Mobile Computing*, vol. 16, no. 11, pp. 3056–3069, 2017.

[16] K. Wang, Z. Hu, Q. Ai, Y. Zhong, J. Yu, P. Zhou, L. Chen, and H. Shin, "Joint offloading and charge cost minimization in mobile edge computing," *IEEE Open Journal of the Communications Society*, vol. 1, pp. 205–216, 2020.

[17] J. Fang, J. Shi, S. Lu, M. Zhang, and Z. Ye, "An efficient computation offloading strategy with mobile edge computing for iot," *Micromachines*, vol. 12, no. 2, 2021.

[18] H. Kim, W.-K. Hong, J. Yoo, and S. eun Yoo, "Experimental research testbeds for large-scale WSNs: A survey from the architectural perspective," *International Journal of Distributed Sensor Networks*, vol. 11, no. 3, p. 630210, mar 2015.

[19] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[20] M.-H. Chen, B. Liang, and M. Dong, "Joint offloading and resource allocation for computation and communication in mobile cloud with computing access point," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. IEEE, May 2017.

[21] M. Avgeris, D. Dechouniotis, N. Athanasopoulos, and S. Papavassiliou, "Adaptive resource allocation for computation offloading," *ACM Transactions on Internet Technology*, vol. 19, no. 2, pp. 1–20, Apr. 2019.

[22] Y. Deng, Z. Chen, X. Yao, S. Hassan, and A. M. A. Ibrahim, "Parallel offloading in green and sustainable mobile edge computing for delay-constrained IoT system," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 12, pp. 12 202–12 214, Dec. 2019.

[23] J. Bi, H. Yuan, S. Duanmu, M. Zhou, and A. Abusorrah, "Energy-optimized partial computation offloading in mobile-edge computing with genetic simulated-annealing-based particle swarm optimization," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3774–3785, Mar. 2021.

[24] M. Goudarzi, H. Wu, M. Palaniswami, and R. Buyya, "An application placement technique for concurrent iot applications in edge and fog computing environments," *IEEE Transactions on Mobile Computing*, vol. 20, no. 4, pp. 1298–1311, 2021.

[25] S. Josilo and G. Dan, "Computation offloading scheduling for periodic tasks in mobile edge computing," *IEEE/ACM Transactions on Networking*, vol. 28, no. 2, pp. 667–680, Apr. 2020.

[26] H. Wu, K. Wolter, P. Jiao, Y. Deng, Y. Zhao, and M. Xu, "EEDTO: An energy-efficient dynamic task offloading algorithm for blockchain-enabled IoT-edge-cloud orchestrated computing," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2163–2176, Feb. 2021.

[27] A. Lakhan, M. Ahmad, M. Bilal, A. Jolfaei, and R. M. Mehmood, "Mobility aware blockchain enabled offloading and scheduling in vehicular fog cloud computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4212–4223, jul 2021.

[28] M. Tang and V. W. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020.

[29] G. Qu, H. Wu, R. Li, and P. Jiao, "DMRO: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3448–3459, 2021.

[30] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Transactions on Mobile Computing*, vol. 19, no. 11, pp. 2581–2593, Nov. 2020.

[31] L. Pu, X. Chen, J. Xu, and X. Fu, "D2d fogging: An energy-efficient and incentive-aware task offloading framework via network-assisted d2d collaboration," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3887–3901, Dec. 2016.

[32] K. Wang, K. Yang, and C. S. Magurawalage, "Joint energy minimization and resource allocation in c-RAN with mobile cloud," *IEEE Transactions on Cloud Computing*, vol. 6, no. 3, pp. 760–770, Jul. 2018.

[33] S. Li, Y. Tao, X. Qin, L. Liu, Z. Zhang, and P. Zhang, "Energy-aware mobile edge computation offloading for IoT over heterogenous networks," *IEEE Access*, vol. 7, pp. 13 092–13 105, 2019.

[34] Q. Wang, S. Guo, J. Liu, and Y. Yang, “Energy-efficient computation offloading and resource allocation for delay-sensitive mobile edge computing,” *Sustainable Computing: Informatics and Systems*, vol. 21, pp. 154–164, Mar. 2019.
[35] X. Lyu, H. Tian, W. Ni, Y. Zhang, P. Zhang, and R. P. Liu, “Energy-efficient admission of delay-sensitive tasks for mobile edge computing,” *IEEE Transactions on Communications*, vol. 66, no. 6, pp. 2603–2616, Jun. 2018.
[36] H. Wu, Z. Zhang, C. Guan, K. Wolter, and M. Xu, “Collaborate edge and cloud computing with distributed deep learning for smart city internet of things,” *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8099–8110, Sep. 2020.
[37] Y. Xue, X. Wu, and J. Yue, “An offloading algorithm of dense-tasks for mobile edge computing,” in *Proceedings of the International Conference on Wireless Communication and Sensor Networks*. ACM, May 2020.
[38] Y. Song, S. S. Yau, R. Yu, X. Zhang, and G. Xue, “An approach to QoS-based task distribution in edge computing networks for IoT applications,” in *2017 IEEE International Conference on Edge Computing (EDGE)*. IEEE, Jun. 2017.
[39] K. Guo, M. Yang, Y. Zhang, and Y. Ji, “An efficient dynamic offloading approach based on optimization technique for mobile edge computing,” in *2018 6th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*. IEEE, Mar. 2018.
[40] F. Liu, Z. Huang, and L. Wang, “Energy-efficient collaborative task computation offloading in cloud-assisted edge computing for iot sensors,” *Sensors*, vol. 19, no. 5, p. 1105, Mar 2019.
[41] X. Chen, Y. Cai, Q. Shi, M. Zhao, B. Champagne, and L. Hanzo, “Efficient resource allocation for relay-assisted computation offloading in mobile-edge computing,” *IEEE Internet of Things Journal*, vol. 7, no. 3, pp. 2452–2468, 2020.
[42] T. Ouyang, Z. Zhou, and X. Chen, “Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing,” *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2333–2345, 2018.
[43] Y. Ding, K. Li, C. Liu, Z. Tang, and K. Li, “Short- and long-term cost and performance optimization for mobile user equipments,” *Journal of Parallel and Distributed Computing*, vol. 150, pp. 69–84, apr 2021.
[44] M. Assi and R. A. Haraty, “A survey of the knapsack problem,” in *2018 International Arab Conference on Information Technology (ACIT)*, 2019.
[45] X. Wang, J. Wang, X. Wang, and X. Chen, “Energy and delay tradeoff for application offloading in mobile cloud computing,” *IEEE Systems Journal*, vol. 11, no. 2, pp. 858–867, 2017.
[46] I. Rojas, J. Gonzalez, H. Pomares, J. J. Merelo, P. A. Castillo, and G. Romero, “Statistical analysis of the main parameters involved in the design of a genetic algorithm,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 32, no. 1, pp. 31–37, 2002.
[47] S. Samanta, A. Choudhury, N. Dey, A. Ashour, and V. Balas, “Chapter 9 - quantum-inspired evolutionary algorithm for scaling factor optimization during manifold medical information embedding,” in *Quantum Inspired Computational Intelligence*, S. Bhattacharyya, U. Maulik, and P. Dutta, Eds. Boston: Morgan Kaufmann, 2017, pp. 285–326.
[48] Papadimitriou and H. Christos, “On the complexity of integer programming,” *Journal of the ACM (JACM)*, vol. 28, no. 4, pp. 765–768, 1981.
[49] X. Xu, Q. Liu, Y. Luo, K. Peng, X. Zhang, S. Meng, and L. Qi, “A computation offloading method over big data for IoT-enabled cloud-edge computing,” *Future Generation Computer Systems*, vol. 95, pp. 522–533, Jun. 2019.
[50] Y. Shi and R. C. Eberhart, “Parameter selection in particle swarm optimization,” in *Evolutionary Programming VII*, V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 591–600.
[51] F. Zhao, Y. Chen, Y. Zhang, Z. Liu, and X. Chen, “Dynamic offloading and resource scheduling for mobile edge computing with energy harvesting devices,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 2154–2165, 2021.
[52] M. Golec, R. Ozturac, Z. Pooranian, S. S. Gill, and R. Buyya, “iFaaSBus: A security and privacy based lightweight framework for serverless computing using iot and machine learning,” *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2021.
[53] M. S. Aslanpour, A. N. Toosi, C. Cicconetti, B. Javadi, P. Sbarski, D. Taibi, M. Assuncao, S. S. Gill, R. Gaire, and S. Dustdar, “Serverless edge computing: Vision and challenges,” in *2021 Australasian Computer Science Week Multiconference*. ACM, feb 2021.

Huijun Tang received the BSc degree from Jinan University, China in 2016 and the M.S. degree from Tianjin University, China in 2018. She is currently pursuing the PhD degree at the Center for Applied Mathematics, Tianjin University, China. Her research interests include internet of things, mobile edge computing and deep learning.

Huaming Wu received the B.E. and M.S. degrees from Harbin Institute of Technology, China in 2009 and 2011, respectively, both in electrical engineering. He received the Ph.D. degree with the highest honor in computer science at Freie Universität Berlin (FU Berlin), Germany in 2015. He is currently an Associate Professor in the Center for Applied Mathematics, Tianjin University, China. His research interests include wireless networks, mobile edge computing, internet of things and deep learning.

Yubin Zhao received his B.S. and M.S. in 2007 and 2010 respectively from Beijing University of Posts and Telecommunications (BUPT), Beijing, China. He received his Ph.D degree in computer science in 2014 from Freie Universität Berlin (FU Berlin), Berlin, Germany. He is currently an Associate Professor in Center for Cloud Computing, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China, since 2014. He serves as the guest editor and reviewer for several journals. He also received outstanding research award in CICCAT 2019. His current research interest includes wireless power transfer, indoor localization and target tracking.

Ruidong Li is an associate professor at Kanazawa University, Japan. Before joining this university, he was a senior researcher at the National Institute of Information and Communications Technology (NICT), Japan. He received the M.Sc. degree and Ph.D. degree in computer science from the University of Tsukuba in 2005 and 2008, respectively. He serves as the secretary of IEEE ComSoc Internet Technical Committee (ITC), and are the founders and chairs of IEEE SIG on Big Data Intelligent Networking and IEEE SIG on Intelligent Internet Edge. He is the associate editor of IEEE Internet of Things Journal, and also served as the guest editors for a set of prestigious magazines, transactions, and journals, such as IEEE communications magazine, IEEE network, IEEE TNSE. He also served as chairs for several conferences and workshops, such as the general co-chair for IEEE MSN 2021, AIVR2019, IEEE INFOCOM 2019/2020/2021 ICCN workshop, TPC co-chair for IWQoS 2021, IEEE MSN 2020, BRAINS 2020, IEEE ICDCS 2019/2020 NMIC workshop, and ICCSSE 2019. His research interests include future networks, big data, intelligent Internet edge, Internet of things, network security, information-centric network, artificial intelligence, quantum Internet, cyber-physical system, and wireless networks. He is a senior member of IEEE and a member of IEICE.