Offloading Dependent Tasks in MEC-enabled IoT Systems: A Preference-based Hybrid Optimization Method

Kuanishbay Sadatdiynov
Shenzhen University
Laizhong Cui (cuilz@szu.edu.cn)
Shenzhen University
Joshua Zhexue Huang
Shenzhen University

Research Article

Keywords: Multi-access Edge Computing, computation offloading optimization, latency, energy consumption, and task failure

Posted Date: July 15th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1824328/v1

Offloading Dependent Tasks in MEC-enabled IoT Systems: A Preference-based Hybrid Optimization Method

Kuanishbay Sadatdiynov, Laizhong Cui and Joshua Zhexue Huang

College of Computer Science and Engineering, Shenzhen University, Shenzhen, 518061, Guangdong, China.

*Corresponding author(s). E-mail(s): cuilz@szu.edu.cn; Contributing authors: kuanishbay@szu.edu.cn; zx.huang@szu.edu.cn;

Abstract

The rapid development of IoT-based services has resulted in an exponential increase in the number of connected smart mobile devices (SMDs). Processing the massive data generated by the large number of SMDs is becoming a big problem for mobile devices, servers, and wireless communication channels. A Multi-access Edge Computing (MEC) paradigm partially mitigates this problem by deploying edge server nodes at the edge of wireless networks nearby SMDs, but the challenge still remains due to the limited computation capacity of MEC servers and the bandwidth of wireless channels. In addition, the dependency of tasks generated by applications on SMDs increases the complexity of the problem. In this paper, we propose a constrained multiobjective computation offloading optimization solution to resolve the problem of task dependency under limited resources. This solution improves the Quality of Service (QoS) through minimizing the latency, energy consumption, and rate of task failure caused by limited resources. We propose a two-staged hybrid computation offloading optimization method to solve the problem. In the first stage, the computation offloading decisions are made based on the preferences of tasks. Then, in the second stage, global optimal solutions are found using the modified Non-Dominated Sorting Genetic Algorithm (NSGA-III). The overall efficiency of the proposed method is increased owing to the preference-based algorithm reinforcing the NSGA-III algorithm by generating a better initial population. The results of extensive experiments show that the efficiency of the proposed method is significantly better than the existing methods.

Keywords: Multi-access Edge Computing, computation offloading optimization, latency, energy consumption, and task failure

1 Introduction

With the fast development of services on the Internet of Things (IoT) networks (e.g., smart cities, building & home automation, smart manufacturing, health care, automotive, and wearables), the number of smart mobile devices (SMDs) is increasing exponentially [1, 2]. A challenge is how to timely handle the massive amount of data generated by them. Multi-access Edge Computing (MEC) is a perspective paradigm in which edge server nodes are deployed at the edge of wireless networks to process the data of nearby SMDs. The MEC paradigm is aimed at ensuring short latency and data privacy because the data is processed within a wireless network area [3]. However, it also creates a local computation burden when multiple SMDs connect simultaneously on an MEC server node, because many applications running on SMDs generate computation-intensive and/or data-intensive tasks that require MEC nodes to complete, but the MEC nodes do not have sufficient computing capacity and storage [4]. This dilema can be mitigated by optimizing the computation offloading of SMDs to the edge server to improve the QoS.

Real-time applications (e.g., mobile games, augmented reality (AR), and autonomous vehicles) require instantly processed computation tasks. Therefore, the completion time (latency) of tasks is an important factor for them. Some authors addressed latency as a metric for computation offloading optimization [5–9]. Other applications (e.g., health care sensors, drones, wearables) require a prolonged battery lifetime. Hence, the energy consumption of SMDs is another objective for computation offloading optimization [10–13]. Some authors considered latency and energy consumption as joint metrics for multiobjective computation offloading optimization [14–18]. However, from the comprehensive review of these existing studies, we can find that task failure, caused by the limitation of resources, has not been considered as an optimization metric jointly with latency and energy consumption. This metric is important because the limitation of resources leads to task failure when overloading the resources. We should consider the rate of task failure when seeking the optimal solutions to minimize latency and energy consumption. This aspect is a rigorous challenge in the case of interdependent tasks that must maintain the order of execution.

Partial computation offloading is aimed at avoiding overloading an SMD by full local execution, or a wireless channel and a MEC server by fully offloading the tasks. In partial computation offloading, tasks are executed either locally or offloaded onto a MEC server. The last one consists of two processes: transmission of a task through a wireless channel and execution of the task on a MEC server. We can model the local computing, wireless transmission, and edge computing processes as a queueing system network using three queueing theory models. The models allow us to evaluate the latency, energy consumption, and probability of task failure caused by limitations of processors or wireless channels. We formulate it as a constrained multiobjective computation offloading optimization (CMCOO) problem to minimize the mentioned objectives. The optimal solution is found by considering the dependencies of the tasks. To achieve this, we propose a two-stage hybrid method. In the first stage, the tasks are classified into computation-intensive or data-intensive tasks. After that, we make offloading decisions in advance for some tasks based on preferences. In the second stage, a modified Non-Dominated Genetic Algorithm (NSGA-III) is used to find the best offloading decisions for the remaining tasks that don't meet the preferences.

Current solutions to computation offloading optimization do not consider the situations of task dependency and failure. To optimize the multiple objectives of latency and energy consumption, some authors solve the problems by giving weight values to each objective and converting it to a single-objective optimization problem [19–21]. These solutions require rerunning the optimization algorithm when the weights of the objectives are changed. Other solutions choose to optimize one objective by treating other objective functions as constraints [15, 22]. Some solutions use the Lagrange multiplier method to solve the CMCOO problem [23, 24]. However, when constraints are complex, this approach becomes difficult to use. The Non-dominated Sorting Algorithm was proposed because it is faster than other multiobjective optimization algorithms [14]. It gives Pareto optimal solutions in which a user can select the optimal solution according to real-time requirements. The performance of the NSGA-III algorithm depends on the randomly generated initial population. A good initial population can lead to faster convergence of the algorithm.

In this paper, we propose a method that categorizes the computation tasks of SMD applications and makes offloading decisions in two stages on the classified computation tasks: making offloading decisions on some classes of computation tasks based on preferences; and using the NSGA-III algorithm to optimize the offloading decisions on the remaining tasks left from the first stage. The initial population of the NSGA-III algorithm is generated considering the decisions of the first stage. Thus, the first stage helps to generate a better initial population. This approach can help improve the performance of the overall optimization process.

The main contributions of this paper are as follows:

1. The local computing, task transmission, and edge computing processes are modeled using queueing theory models that consider the limitations of processors and wireless channels.

2. We optimize task failure rate jointly with latency and energy consumption in a MEC-based IoT system with interdependent tasks. To our knowledge, this is the first work to take task failure and task dependency into consideration.
3. We propose a preferences-based hybrid computation offloading multiobjective optimization method to minimize the mentioned objectives.
4. We conduct extensive simulation experiments in various cases with the proposed and existing methods. The results of the experiments show that the proposed method can give better results compared with existing methods.

The rest of the paper is organized as follows. The literature on the computation offloading problem is reviewed in Section 2. We describe our system model and formulate the optimization problem in Section 3. The proposed method is described in Section 4. We present the experiment results and compare our findings to the baseline methods in Section 5. In Section 6, we state our conclusions and remark on possible directions for future research.

2 Related work

Offloading part of computation tasks (referred to as tasks hereafter) onto remote resources (cloud/edge computing) is widely used in practice. MEC is suitable for latency-sensitive applications because the edge server nodes are deployed at the edge of a wireless network. MEC has significant advantages in terms of latency and data privacy compared to cloud computing. Computation offloading in MEC, its limitations, and issues with them were studied in [3, 25]. Also, Gasmi et al. [26] pointed out several research problems in computation offloading in MEC, including issues related to computation offloading of dependent tasks.

Recent studies on methods and frameworks of optimal computation offloading and task scheduling to increase the performance of SMDs were comprehensively reviewed in [27]. Offloading dependent tasks were considered in [12, 14, 15, 28–31]. Here, we discuss some examples of applications that use the NSGA algorithms [32–34] to solve multiobjective computation offloading optimization problems.

Afrin et al. [14] proposed a computation offloading method using the NSGA-II algorithm to minimize the makespan, energy consumption, and monetary cost in an edge cloud-based multi-robot system. To solve the problem, they proposed a modified NSGA-II algorithm that pre-sorts the initial population based on the task size and processing speed of the resources. Then, to balance the values of all objectives in subsequent generations, it selects the chromosomes having the minimum distance solution from the Pareto-front to the origin.

Another example is the method provided by Cui et al. [16] to minimize the latency and energy consumption of SMDs. The authors explained that utilizing the multiobjective optimization algorithm allows for selecting the best solution among the Pareto optimal set and avoiding rerunning the algorithm when the condition changes. They used the $M/M/1$ type of queue for modeling computing processes on SMDs and a MEC server. The latency was considered as the sum of waiting time in the queue and execution time of a task. This system works well when tasks are executed on a computer with unlimited resources.

Xu et al. [15] proposed a computation offloading method for cases with multiple computing units, for instance, edge servers with several virtual machines (VMs). They modeled the task execution process on a cloudlet using a $M/M/c/\infty$ queue. A task can be offloaded to an idle one of the c VMs on the cloudlet. The objective of the work was to minimize the energy consumption under deadline constraints in a system that consists of a collaboration of local, edge, and cloud computing resources. They successfully used the NSGA-II algorithm to achieve the goal.

Xu et al. [35] proposed an NSGA-III algorithm-based method to minimize the completion time and energy consumption of IoT devices. They applied simple additive weighting (SAW) and multiple-criteria decision-making (MCDM) techniques to select an optimal schedule strategy. The tasks are executed either on the mobile device, or on the cloudlet, or on the cloud server. That is, the tasks are offloaded onto the cloud server if the cloudlet is busy. If all VMs are busy, then tasks are lost. However, they did not consider the number of lost tasks in their work.

Mao et al. in [36] proposed a Lyapunov optimization-based dynamic computation offloading algorithm. They considered latency and task failure as the performance metrics of the multiobjective optimization problem, then converted it into a single optimization. Their proposed algorithm was oriented to the independent task case.

The works mentioned above were aimed at interesting solutions to improve the performance of SMDs in various situations. Two gaps can be observed in modeling the task computation or transmission processes and in performance metrics. Using the $M/M/1$ or $M/M/c/infty$ queueing models, for example, is not appropriate for systems with limited resources. Using these models, we cannot evaluate the number of lost tasks due to the limited resources of the computing unit or wireless channel. Therefore, the latency calculated by those models may differ from the actual value. Also, the performance of the NSGA algorithm depends on the initial randomly generated populations.

In this work, we use the $M/M/c/K$ queueing to model the local computing, transmission, and edge computing processes. It allows us to evaluate the number of lost tasks in computing units and wireless channels with limited resources. Thus, we optimize the latency, energy consumption, and task failure as a constrained multiobjective optimization computation offloading problem. To solve the problem, we propose a two-staged preference-based hybrid method. In the first stage, offloading decisions are made based on preferences. In the second stage, the modified NSGA-III algorithm is adopted. The differences between this work and existing works are given in Table 1. The initial population of the NSGA-III algorithm is generated according to the

Table 1 Comparison of the proposed method to existing computation offloading methods

Ref.	**Contributions**	**Optimizes latency**	**Optimizes energy consumption**	**Optimizes task failure**	**Follows task dependency requirements**
[8]	Joint computation offloading, resource allocation, and migration optimization	✓	×	×	×
[9]	A heuristic task migration computation offloading scheme	✓	×	×	×
[12]	A combination of two algorithms, using the advantages of edge and cloud computing	×	✓	×	✓
[13]	Method based on decentralized Federated Learning	×	✓	×	×
[14]	A multiobjective evolutionary approach (NSGA-II)	✓	✓	×	✓
[15]	A multiobjective evolutionary approach (NSGA-II)	✓	✓	×	✓
[16]	A multiobjective evolutionary approach (NSGA-II)	✓	✓	×	×
[17]	Experiments on real-world and synthetic datasets	×	✓	×	×
[18]	An online distributed computation offloading algorithm	✓	✓	×	×
[21]	Green Energy and Latency Aware Task Assignment; experiments on a real-world dataset	✓	✓	×	×
[22]	A method for dynamic selection of edge cloud for offloading tasks	✓	✓	×	✓
[31]	A model-free approach based on reinforcement learning	✓	✓	×	✓
[35]	A multiobjective evolutionary approach (NSGA-III)	✓	✓	×	✓
[36]	A Lyapunov optimization dynamic computation offloading method	✓	×	✓	×
This work	A hybrid preference-based method employing the NSGA-III algorithm	✓	✓	✓	✓

result of the first stage. Thus, we can improve the overall performance of the optimization algorithm. The details are presented in the following sections.

3 Queueing Models and Problem Formulation

We consider a MEC system consisting of a MEC server, a set of SMDs, and several small cells, as shown in Fig. 1. A small cell eNodeB (SeNB) is connected wirelessly with SMDs in a small cell. The bandwidth of the wireless channel is assigned to SMDs equally within the small cell. The SeNBs are connected in a wired manner to Macro eNodeB (MeNB), where a MEC server is deployed. It is allowed to offload the tasks of several SMDs onto the MEC server simultaneously.

Let M SMDs be denoted by the set of $\mathcal{M} = \{U_1, U_2, ..., U_M\}$. On an SMD, an application generates N tasks, denoted by the set of $\mathcal{N} = \{\tau_{m,1}, \tau_{m,2}, ..., \tau_{m,N}\}$. We assume the rate of generated tasks λ (i.e., the number of tasks per time unit) follows the exponential distribution. The tasks are compound, i.e., each task can contain several dependent subtasks. The dependency of subtasks is given by the directed acyclic graph (DAG) $G = (V, E)$, where V denotes a subtask and E denotes the precedence constraint between subtasks i and k. Any subtasks might be executed either on SMD or the MEC server, except for the first and the last ones, because a task is generated at SMD and its final result is shown at SMD. Each subtask $\tau_{m,n,k}$ is defined by two parameters $(d_{m,n,k}, c_{m,n,k})$, where $d_{m,n,k}$ represents the data size (DS) of the subtask and $c_{m,n,k}$ represents the number of cycles required to complete

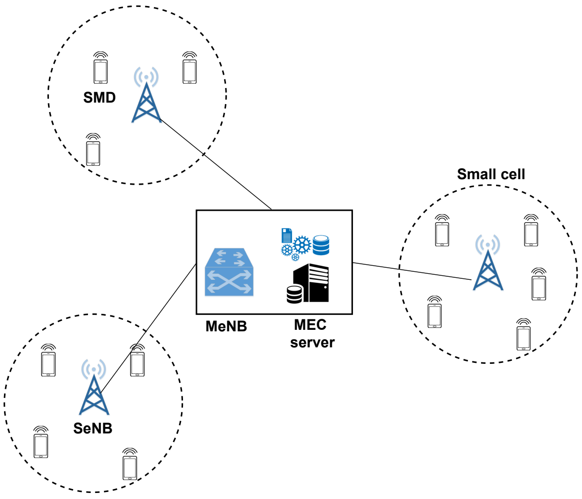


Fig. 1 Illustration of the model of a MEC system.

(NRCC). The values of DS and NRCC are given by the exponential distribution. An example of a compound task with interdependent subtasks is given in Fig. 2.

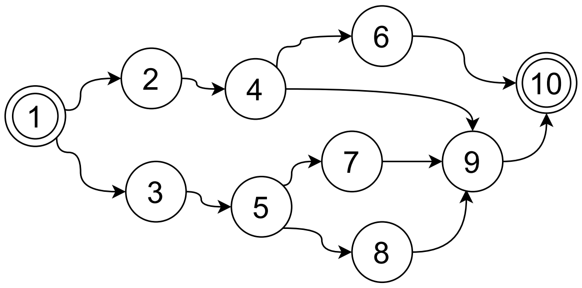


Fig. 2 An example of one task which consists of interdependent subtasks.

Since subtasks can be executed either locally or on a MEC server, the flow of subtasks is divided into two parts. Assume δ_m part of the flow is offloaded through a wireless channel and executed on the MEC server. The remaining γ_m part of flow is executed locally, where $\gamma_m = (1 - \delta_m)$. We model these processes by adopting models from queueing theory. In the queueing models, we consider the limitations of the computation capacities of SMDs, the MEC server, and the bandwidth of the wireless channel. The fraction of offloaded subtasks can be calculated as

$$\delta_m = \frac{\sum_{n=1}^{N_m} \sum_{k=1}^{10} x_{m,n,k}}{N_m} \tag{1}$$

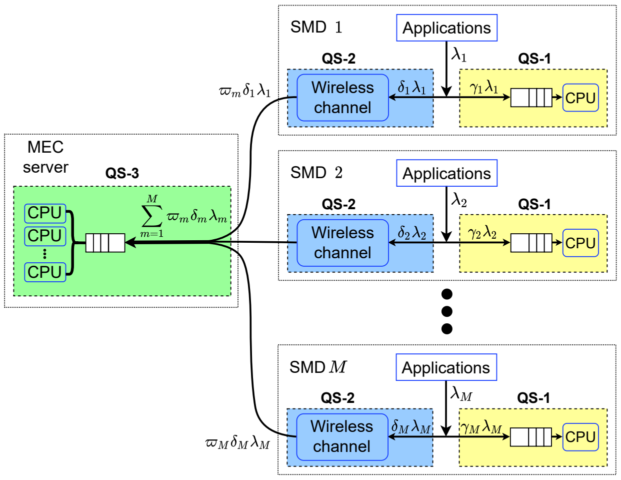


Fig. 3 Queueing system network.

where N_m is the number of tasks of SMD U_m. $x_{m,n,k} \in \{0,1\}$ represents an offloading decision, where $x_{m,n,k} = 0$ means the subtask is executed locally, and $x_{m,n,k} = 1$ means the subtask is offloaded onto the edge server.

Fig. 3 depicts the network model of the queueing system, which has three queueing system models denoted as QS-1, QS-2, and QS-3. QS-1 models the γ_m part of the subtask flow λ_m arriving at the CPU of an SMD to process. An SMD has a single computing unit and a limited buffer K_l to keep the subtasks in a queue. An arriving subtask is lost if the buffer of an SMD is full. Thus, the local subtask execution process is modeled using the $M/M/1/K_l$ type of queueing system with loss. In QS-2, a subtask transmission by wireless channel is modeled using the $M/M/1/1$. Because we assume the wireless channel does not have any buffer to keep the queue, a subtask is lost directly if the wireless channel bandwidth is busy with the transmission of the previous subtask. The probability of losing a task (task failure) in transmission is defined using π_m^{tr}. At the MEC server in QS-3, all flows are aggregated, i.e., $\sum_{m=1}^{M}\varpi_m\delta_m\lambda_m$, where $\varpi_m = 1 - \pi_m^{tr}$. A MEC server has several virtual machines (VMs) and a limited buffer. A task is assigned to and executed in a VM randomly. As a result, we model the edge computing process as $M/M/c/K$, where c is the number of VMs and K is the buffer size of the MEC server. The input flow is the sum of the output flows of SMDs following the same distribution. The values of K_l, c, and K are given.

3.1 Local Computing

An SMD has a single computation unit, and its capacity is limited. The CPU of the SMD only serves one subtask at a time, and $K_l - 1$ subtasks are held waiting for service in the SMD's buffer. The service time of the computation unit follows an exponential distribution. Therefore, the task execution process on the SMD is modeled by the $M/M/1/K_l$ type of the queueing system. The average service time of an SMD can be calculated as

$$\widetilde{b}_m = \frac{\sum_{n=1}^{N_m} \sum_{k=1}^{10} (1 - x_{m,n,k}) c_{m,n,k}}{\sum_{n=1}^{N_m} \sum_{k=1}^{10} (1 - x_{m,n,k}) f_m} \tag{2}$$

where f_m is the computation rate of the SMD, which is given by the number of CPU cycles.

The SMD utilization coefficient is calculated as $\rho_m = \gamma_m \lambda_m \widetilde{b}_m$, where γ_m represents the QS-1 input flow, defined as $\gamma_m = 1 - \delta_m$. The average waiting time for service of a subtask $\tau_{m,n,k}$ on an SMD can be calculated as [37]:

$$\widetilde{w}_m = \begin{cases} \frac{1}{\gamma_m \lambda_m} \left(\frac{\rho_m}{1-\rho_m} - \frac{\rho_m (K_l \rho_m^{K_l} + 1)}{1 - \rho_m^{K_l+1}} \right), & \rho_m \neq 1 \\ \frac{K_l (K_l - 1)}{2 \gamma_m \lambda_m (K_l + 1)}, & \rho_m = 1 \end{cases} \tag{3}$$

where K_l is the number of subtasks held in QS-1.

The local execution time of a subtask on an SMD is equal to the sum of the service time by the CPU and the average waiting time for service in a queue, and it is calculated as

$$T^l_{m,n,k} = (1 - x_{m,n,k}) \left(\frac{c_{m,n,k}}{f_m} + \widetilde{w}_m \right) \tag{4}$$

Considering the dependency of subtasks, we define the ready time of a subtask for local execution as follows:

$$RT^l_{m,n,k} = \max_{z \in \mathbf{pred}(k)} \max\{FT^l_{m,n,z}, FT^e_{m,n,z}\} \tag{5}$$

where $\mathbf{pred}(k)$ is the set of immediate predecessors of the subtask $\tau_{m,n,k}$.

The completion time of the subtask $\tau_{m,n,k}$ in the local execution can be calculated as

$$FT^l_{m,n,k} = T^l_{m,n,k} + RT^l_{m,n,k} \tag{6}$$

The probability of task failure caused by buffer limitation of an SMD can be calculated as [37]

$$\pi^l_m = \begin{cases} \frac{\rho_m^{K_l+1} (1 - \rho_m)}{1 - \rho_m^{K_l+1}}, & \rho_m \neq 1 \\ \frac{1}{K_l + 1}, & \rho_m = 1 \end{cases} \tag{7}$$

The amount of energy used by an SMD while a subtask is being executed locally is calculated as

$$E^l_{m,n,k} = (1 - x_{m,n,k}) \kappa c_{m,n,k} f_m^2 \tag{8}$$

where $\kappa = 10^{-26}$ is the switched capacitance coefficient, and it depends on chip architecture [11].

3.2 Communication Model

The subtask uploading transmission rate R_m from SMD U_m to access point (AP) is calculated according to the Shannon–Hartley theorem

$$R_m = \frac{B}{s}\log_2\left(1 + \frac{P_m G_m}{\sigma + I_m}\right) \tag{9}$$

where B denotes the bandwidth of a wireless channel, s is the number of channels, and P_m denotes the transmission power of SMD U_m. Furthermore, G_m represents the channel gain between the SMD U_m and the AP, σ represents the background noise power, and I_m is the interference parameter caused by SMDs in other cells on the same channel, and it is calculated as follows

$$I_m = \sum_{l=1, l\neq m}^{M} a_{l,m}\, p_l\, G_l \tag{10}$$

where $a_{l,m} \in \{0,1\}$ is a binary variable. If the channel used by SMD U_m and SMD U_l is the same, $a_{l,m} = 1$; otherwise, $a_{l,m} = 0$. The transmission power of SMD U_l is p_l, and the channel gain between SMD U_l and AP is G_l.

We assume the wireless channel does not have a buffer to store subtask data. The arriving subtasks are dropped when a wireless channel is overloaded. We model this process as an $M/M/1/1$ type queueing system, which means the system has a single computation unit and can operate only a single subtask at a time. When $K_l = 1$, the average waiting time for service is $\widetilde{w} = 0$ according to Eq. (3). Therefore, the transmission time of a task from SMD to AP is calculated as

$$T^{tr}_{m,n,k} = \frac{x_{m,n,k} d_{m,n,k}}{R_m} \tag{11}$$

The probability of task failure due to a wireless channel's bandwidth limitation can be computed as [37]

$$\pi^{tr}_m = \frac{\rho_R}{1+\rho_R} \tag{12}$$

where $\rho_R = \delta_m \lambda_m \widetilde{b}_{tr}$ is the utilization coefficient of a wireless channel. The average service time of a wireless channel is calculated as

$$\widetilde{b}_{tr} = \frac{\sum_{n=1}^{N_m}\sum_{k=1}^{10} x_{m,n,k} d_{m,n,k}}{\sum_{n=1}^{N_m}\sum_{k=1}^{10} x_{m,n,k} R_m} \tag{13}$$

The energy consumption of an SMD during transmission of a subtask is calculated as

$$E^{tr}_{m,n,k} = P_m T^{tr}_{m,n,k} \tag{14}$$

3.3 Edge Computing

In Edge Computing, subtask flows from all SMDs are aggregated. We assume the edge server has c virtual machines and a limited buffer. Other arriving subtasks are lost because the Edge server can only operate and hold K subtasks in the buffer. We model the edge computing process as an $M/M/c/K$ queueing system.

The average service time of the Edge server is calculated by the average workload from all SMDs over the computation rate of the Edge server as

$$\widetilde{b}_E = \frac{\sum_{m=1}^{M}\sum_{n=1}^{N}\sum_{k=1}^{10} x_{m,n,k}c_{m,n,k}}{\sum_{m=1}^{M}\sum_{n=1}^{N}\sum_{k=1}^{10} x_{m,n,k}F} \tag{15}$$

where F denotes the computation rate of the Edge server, which is given by the number of CPU cycles.

The utilization coefficient of an edge server is indicated by $r = \sum_{m=1}^{M}\delta_m\lambda_m\widetilde{b}_E$, where $\rho_E = r/c$. The probability that an edge server is in idle status is defined as [37]

$$p_0 = \begin{cases} \left[\frac{r^c}{c!}\left(\frac{1-\rho_E^{K-c+1}}{1-\rho_E}\right) + \sum_{n=0}^{c-1}\frac{r^n}{n!}\right]^{-1}, & \rho_m \neq 1 \\ \left[\frac{r^c}{c!}(K-c+1) + \sum_{n=0}^{c-1}\frac{r^n}{n!}\right]^{-1}, & \rho_m = 1 \end{cases} \tag{16}$$

The average waiting time of a subtask for service by an edge server is calculated as follows:

$$\widetilde{w}_E = \frac{p_0 r^c \rho_E\left[1-\rho_E^{K-c+1}-(1-\rho_E)(K-c+1)\rho_E^{K-c}\right]}{\sum_{m=1}^{M}\delta_m\lambda_m c!(1-\rho_E)^2} \tag{17}$$

If $\rho_E = 1$, L'Hôpital's rule is applied twice, as shown in [37].

Thus, the execution time of a subtask on the Edge server is equal to the sum of service time and waiting time in a queue, and it is calculated as

$$T^e_{m,n,k} = x_{m,n,k}\left(\frac{c_{m,n,k}}{F} + \widetilde{w}_E\right) \tag{18}$$

The ready time of the subtask $\tau_{m,n,k}$ on the computing edge is calculated as

$$RT^e_{m,n,k} = \max\left\{T^{tr}_{m,n,k}, \max_{z\in\mathbf{pred}(k)}\left\{FT^l_{m,k,z}, FT^e_{m,k,z}\right\}\right\} \tag{19}$$

The completion time of a subtask on the edge server is calculated as

$$FT^e_{m,n,k} = T^e_{m,n,k} + RT^e_{m,n,k} \tag{20}$$

The probability of task failure caused by overloading on the Edge server is calculated as [37]

$$\pi_e = \frac{r^K p_0}{c^{K-c} c!}, \tag{21}$$

We ignore the energy consumption of the Edge server because its power is provided by the power network. In Edge computing, we consider only the energy consumption by SMDs while offloading the subtasks onto the edge server.

3.4 Problem Formulation

The overall completion time of any subtask is equal to the sum of the completion times of all subtasks on local computing and edge computing, and it can be calculated as

$$T_{m,n,k} = FT^l_{m,n,k} + FT^e_{m,n,k} \tag{22}$$

Similarly, the overall energy consumption of any subtask can be calculated as

$$E_{m,n,k} = E^l_{m,n,k} + E^{tr}_{m,n,k} \tag{23}$$

The average task completion time of tasks for all SMDs is calculated as

$$T = \frac{1}{MN} \sum_{m=1}^{M} \sum_{n=1}^{N} T_{m,n,10} \tag{24}$$

where the completion time of the 10th subtask is the completion time of a task.

The average energy consumption of tasks across all SMDs is calculated as

$$E = \frac{1}{MN} \sum_{m=1}^{M} \sum_{n=1}^{N} \sum_{k=1}^{10} E_{m,n,k} \tag{25}$$

The value of the probability of task failure in the whole system is calculated by the sum of probabilities in all queueing systems as follows:

$$\Pi = \sum_{m=1}^{M} \pi^l_m + \pi^{tr}_m + \pi_e \tag{26}$$

To minimize the energy consumption of SMDs, the average task completion time, and the probability of task failure by finding the optimal computation offloading decisions, a constrained multiobjective optimization problem is formulated as follows:

$$\begin{aligned} \text{P1} &: \min_{x_{m,n,k}} \{T, E, \Pi\} \\ \text{s.t.} & \\ \text{C1} &: T_{m,n,10} \le \theta_{m,n} \end{aligned}$$

$$\begin{aligned} \mathrm{C2} &: x_{m,n,k} \in \{0,1\} \quad k \in [2;\ 9] \\ \mathrm{C3} &: x_{m,n,k} = 0, \quad k \in \{1,10\} \\ \mathrm{C4} &: T_{m,n,k} \geq T_{m,n,z}, \quad z \in \mathbf{pred}(k) \end{aligned} \tag{27}$$

Constraint $C1$ defines that the completion time of each task cannot exceed the deadline given by an application. Constraint $C2$ defines that offloading decision is binary. Constraint $C3$ defines that optimal offloading decisions are sought for subtasks from the second to the ninth in each task. The first and tenth subtasks are executed locally. Constraint $C4$ shows that the subtask k not be finished before its predecessor.

To solve **P1**, we propose a preference-based hybrid computation offloading method, which is described in detail in the next section.

4 A Hybrid Offloading Method

As a solution to the problem $P1$, we propose a two-staged hybrid computation optimization method. Assume there are M SMDs, each generating N_m tasks, and the total number of tasks is N, whereas each task contains 10 dependent subtasks according to the system model. It is required to build a computation offloading framework that finds the optimal offloading decision for each subtask and optimizes the objective functions under given constraints.

Let α be the proportion of subtasks that get the offloading decisions based on preferences. Assume that the initial values of the population size and the number of iterations, G_i and I_i, are given. The population size and the number of iterations are reduced to 10 during the process, which depends on the number of tasks. We assume these volumes of population and iterations are enough for the algorithm to converge when the selected number of tasks is at its minimum. To begin with, given a set of N_s tasks, a task is selected from each SMD. Thus, there are totally $N_s \times 10$ subtasks that wait for offloading decisions. In addition, the offloading decisions for some subtasks are made based on preferences. Furthermore, the modified NSGA-III algorithm is adopted to find the optimal decisions for the remaining subtasks. Finally, the selected task is removed from the task set, and the next task is selected to find the optimal offloading decisions for its subtasks. This process continues until all tasks receive offloading decisions. A general description of the proposed hybrid method is given in Fig. 4, and its pseudo-code is given in Algorithm 1. The following sections are dedicated to showing how the proposed method works in detail.

4.1 Making Offloading Decisions by Preferences

As mentioned in Section 3, a subtask has two parameters: $d_{m,n,k}$, the data size (DS) of the subtask, and $c_{m,n,k}$, the number of required CPU cycles (NRCC) to complete the subtask. Offloading a subtask with a big DS onto the MEC server through a limited wireless channel leads to high latency and more energy

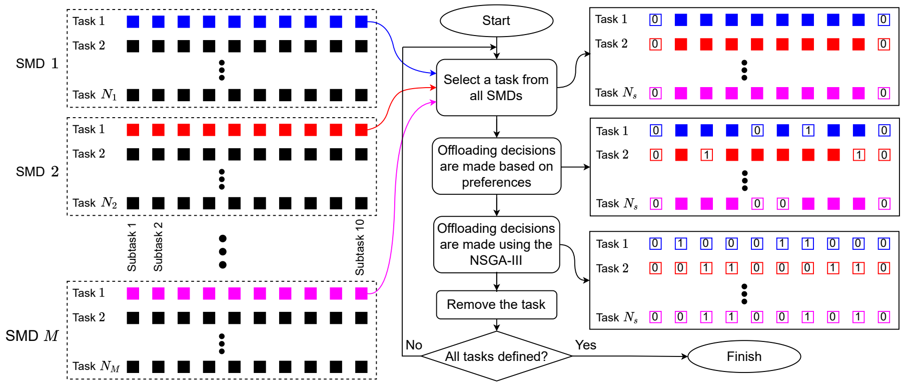


Fig. 4 A general description of the proposed solution.

Algorithm 1 A hybrid computation offloading algorithm

```
Require: $\mathcal{N}$, $\alpha$, $G_i$, $I_i$
Ensure: Offloading decisions
 1: while $\mathcal{N} \neq \emptyset$ do
 2:     Select a task from each SMD
 3:     $N_s \leftarrow$ number of selected tasks
 4:     $\xi = N_s/N$
 5:     $G = \lfloor \xi \cdot G_i + 10 \rfloor$
 6:     $I = \lfloor \xi \cdot I_i + 10 \rfloor$
 7:     Making offloading decisions based on preferences ($\alpha$)
 8:     Making offloading decisions using the NSGA-III ($N_s$, $G$, $I$ )
 9:     Remove tasks $N_s$
10: end while
```

consumption. In contrast, if a subtask is computation-intensive, then executing the subtask locally with limited computation capability is also not efficient. Taking the above considerations into account, we define two preferences for making immediate offloading decisions:

1. Preference 1: If a subtask is not computation-intensive but its DS is large, then it is executed locally.
2. Preference 2: If a subtask is computation-intensive and its DS is small, then it is offloaded onto the MEC server.

Note that the preferences are applicable for all subtasks of a task except for the first and last ones. According to the system model, the first and last subtasks are always executed on the SMD.

Thus, we make immediate offloading decisions based on preferences for some subtasks in a set. For the remaining subtasks that do not satisfy the preferences, the offloading decisions are made by using the modified NSGA-III algorithm in stage 2 (see Table 2).

Table 2 Decisions made using different methods on different conditions

$d_{m,n,k}$	$c_{m,n,k}$	Decision	$x_{m,n,k}$
large	big	*	*
large	small	execute locally	0
small	small	*	*
small	big	offload onto the MEC server	1

* - decisions made using the NSGA-II algorithm

The proportion α of subtasks that satisfy immediate offloading decisions is determined as follows. We intuitively assume that less than 50% of tasks can be offloaded based on preferences ($\alpha < 0.5$). The default value of α is found through trial and error. The immediate offloading decisions based on preferences are made as follows.

The default value of α is found through trial and error. The immediate offloading decisions based on preferences are are made as follows.

1. The weighted values of two parameters for each subtask are calculated by

$$d^w_{m,n,k} = \frac{d_{m,n,k}}{\sum_{k=2}^{9} d_{m,n,k}}, \tag{28}$$

$$c^w_{m,n,k} = \frac{c_{m,n,k}}{\sum_{k=2}^{9} c_{m,n,k}}. \tag{29}$$

where $n \in \{1, N\}$ is the index of a selected task from the SMD. The maximum values of the weighted DS $d^w_{m,max}$ and weighted NRCC $c^w_{m,max}$ are determined.

2. Thresholds are used to classify the tasks, and they are determined by α, $d^w_{m,max}$, and $c^w_{m,max}$ as

$$\begin{aligned} d_{t1} &= \alpha d^w_{m,max}, \\ d_{t2} &= (1-\alpha) d^w_{m,max}, \\ c_{t1} &= \alpha c^w_{m,max}, \\ c_{t2} &= (1-\alpha) c^w_{m,max}. \end{aligned} \tag{30}$$

where $0 < \alpha < 0.5$.

3. Identify the subtasks that can get immediate offloading decisions. According to the preferences, the subtasks with the largest DS and the lowest NRCC and the subtasks with the smallest DS and the highest NRCC can be found by the following rules:

$$O^{'}_{m,n,k} = (d^w_{m,n,k} \geq d_{t2}) \ \& \ (c^w_{m,n,k} < c_{t1}), \tag{31}$$

$$O^{''}_{m,n,k} = (d^w_{m,n,k} < d_{t1}) \ \& \ (c^w_{m,n,k} \geq c_{t2}). \tag{32}$$

4. Finally, immediate offloading decisions are made as: the subtasks which satisfy the condition in (31) are executed locally, and the subtasks which satisfy the condition in (32) are offloaded onto the MEC server as follows:

$$x_{m,n,k} = \begin{cases} 0, & \text{if } O^{'}_{m,n,k} = TRUE, \\ 1, & \text{if } O^{''}_{m,n,k} = TRUE. \end{cases} \tag{33}$$

The subtasks that do not meet both conditions remain undecided, and decisions are made in the next stage.

Fig. 5 illustrates how the subtasks get immediate offloading decisions based on preferences. The subtasks are shown in a two-dimensional space (blue points) by their weighted DS and NRCC values. The thresholds (red dotted lines) are drawn using the α value and the maximum values of the weighted DS and NRCC. The subtasks in the squares S_1 and S_2 are identified to assign the offloading decisions in advance based on preferences. Herewith, the subtasks in the square S_1 are executed locally, and the subtasks in the square S_2 are offloaded onto the MEC server. A process of finding the immediate offloading decisions based on preferences is given in Algorithm 2. The offloading decisions for the remaining subtasks are made by using the modified NSGA-III algorithm in the next stage.

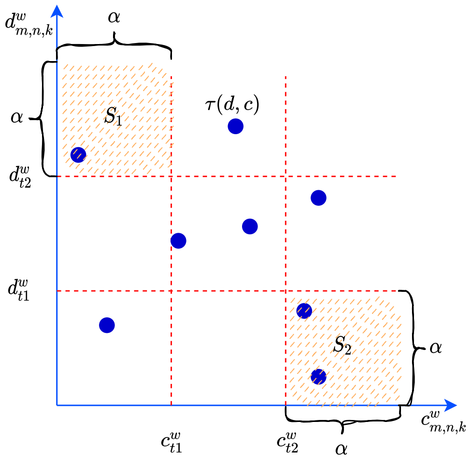


Fig. 5 Classification of subtasks based on preferences.

4.2 Find Optimal Offloading Decisions using the Modified NSGA-III Algorithm

The variant of the Non-dominated Sorting Genetic Algorithm, i.e., NSGA-II, has shown high performance among multiobjective optimization algorithms [14]. However, the crowding distance operation in the NSGA-II algorithm does not work well for many-objective problems [34]. The problem **P1** in (27) seeks optimal solutions to minimize the three objective functions under its complex constraints. That is why we adopted and modified the third version of the NSGA to solve the **P1**.

Algorithm 2 returns a set of offloading decisions that is incomplete. It means offloading decisions were defined based on preferences for the first and

Algorithm 2 Finding offloading decisions based on preferences

```
Require: $\mathcal{N}$, $\alpha$, $\mathcal{A} = \emptyset$
Ensure: Offloading decisions, $\mathcal{A}$
 1: while $\mathcal{N} \neq \emptyset$ do
 2:     Calculate weighted values of the parameters by Eqs. (28) and (29)
 3:     Calculate thresholds by Eq. (30)
 4:     Identify the tasks which get the offloading decisions based on preferences by Eqs. (31) and (32)
 5:     if $(d^w_{m,n,k} \geq d_{t2})$ & $(c^w_{m,n,k} < c_{t1}$ then
 6:         $x_{m,n,k} = 0$
 7:     else
 8:         $\mathcal{A} = \mathcal{A} \cup \{\tau_{m,n,k}\}$
 9:     end if
10:     if $(d^w_{m,n,k} < d_{t1})$ & $(c^w_{m,n,k} \geq c_{t2})$ then
11:         $x_{m,n,k} = 1$
12:     else
13:         $\mathcal{A} = \mathcal{A} \cup \{\tau_{m,n,k}\}$
14:     end if
15: end while
```

last subtasks and some subtasks between them. We need to find offloading decisions for the remaining subtasks to solve the **P1**. Usually, the performance of the NSGA-III algorithm depends on the randomly generated population. A good population serves to find an optimal solution faster. We use that property of the NSGA-III algorithm in our hybrid method, i.e., the modified NSGA-III algorithm, which we will explain step by step below.

1. Generate reference points: The reference directions are selected as the base model for the optimization. We use the known Das and Dennis [38] systematic approach to determine the set of reference points in each generation, as described in [33]. This approach defines reference points uniformly distributed on the entire normalized hyperplane. There is only one reference point to which all the individuals will be associated. In the reference points array, each row represents a reference line and each column is a variable. The total number of reference points W in a problem with Y objectives is calculated as

$$W = \binom{(Y + a - 1)}{a} \tag{34}$$

where a refers to the number of divisions considered along each objective axis.

2. Generate at random the initial populations: The diversity of the population should be maintained; otherwise, it leads to premature convergence. Conversely, the population size should not be kept very large as it can cause a genetic algorithm to slow down, while a smaller population is not enough for a good mating pool. The size of the population is given by G.

We generate a random initial population considering the result of the previous stage. The initial population contains M chromosomes. The number of chromosomes is equal to the number of tasks selected from M SMDs. Each chromosome represents a set of offloading decisions for a task that consists of 10 genes. Each gene is denoted by $\{0, 1\}$ that represents the offloading decision of a subtask. Note that the initial population is generated for the remaining subtasks, which do not satisfy the preferences in the previous stage. It means some genes on a chromosome are predefined.

3. Evaluate the solutions: We calculate the objective functions using Eqs. (24), (25), and (26). Constraints in (27) are defined as constraint violations (CVs) as follows:

$$CV^1_{m,n} = \begin{cases} 0, & \text{if } T_{m,n,10} \le \theta_{m,n} \\ T_{m,n,10} - \theta_{m,n} & \text{otherwise} \end{cases} \tag{35}$$

$$CV^2_{m,n,k} = \begin{cases} 0, & \text{if } T_{m,n,k} \ge T_{m,n,z} \\ T_{m,n,k} - T_{m,n,z} & \text{otherwise} \end{cases} \tag{36}$$

$$CV = \sum_{m=1}^{M} \sum_{n=1}^{N} (CV^1_{m,n} + \sum_{k=1}^{10} CV^2_{m,n,k}) \tag{37}$$

The chromosomes that satisfy constraints in (27) are feasible solutions. In the next step, the best chromosomes are picked out based on the values of the objective functions and the CV.

4. Sorting and selection: The selection phase is to select the fittest chromosomes and let them pass their genes to the next generation. First, the values of objective functions are normalized (5). Then, a fast nondominated sorting is performed according (6) *Definition 1*:

Definition 1: A solution u_1 dominates another solution u_2, if any one of the following conditions is true:

1. u_1 is a feasible solution, and u_2 is an infeasible solution.
2. u_1 and u_2 are feasible and $f(u_1) \preceq f(u_2)$
3. u_1 and u_2 are infeasible and $CV(u_1) < CV(u_2)$

Furthermore, solutions are associated with reference points (7), and the fast nondominated sorting is performed on an updated population (8). Two pairs of chromosomes (parents) are selected randomly according to their fitness scores and CV to produce new offspring using different recombination and mutation operators.

10. Crossover: Crossover is the most significant phase in a genetic algorithm. It creates offspring by combining pairs of parents in the current population during evolution. Two chromosomes are randomly selected for picking out from the population. The crossover operation is performed for a certain percentage (pC) of the population as follows:

$$\begin{cases} u_1' = \vartheta u_1 + (1-\vartheta)u_2 \\ u_2' = (1-\vartheta)u_1 + \vartheta u_2 \end{cases} \tag{38}$$

where u_1' and u_2' are both offspring, and $\vartheta \in \{0,1\}$ is the random integer variable.

11. Mutation: Certain new offspring are formed by the crossover operation. Then, the algorithm creates mutations by randomly changing the genes of individual parents. The mutation operation is applied with a probability of mutation of pM. A high probability of mutation will increase the diversity in the population and prevent premature convergence. The values of the crossover and mutation probabilities are given as the simulation parameters in the next section.

$$u_i' = \begin{cases} 1-u_i, & \text{if rand } < \text{pM} \\ u_i, & \text{otherwise} \end{cases} \tag{39}$$

where u_i' is the offspring of u_i, $i \in [1,10]$ is the gene of chromosome.

In the next step, the parent and offspring populations are combined (12). The combined population is evaluated (13), sorted and selected (14). According to the sorting result, a new parent population is created (15). This process continues until the stop condition is met (9). A stopping criterion is defined as the given number of iterations. After several iterations, the algorithm returns a Pareto optimal set. All steps of the working process are given in Algorithm 3.

5 Experiments

In this section, we present several experiments to evaluate the performance of the proposed hybrid method. The evaluation is made by comparing this method with existing methods based on algorithms such as Multi-Objective Particle Swarm Optimization (MOPSO), NSGA-II, and random offloading. Experiments were performed on a PC with an Intel Core i5 CPU (3.2 GHz and 16 Gb of RAM), and MATLAB 2021a was used for simulation.

5.1 Simulation Environment

Based on the system model, we created a simulation environment which consists of the models of a MEC server, SMDs, and wireless channels. We conducted the experiments in the environment with 5-10 small cells. The number of SMDs in each small cell was in the range of [20-45]. The number of tasks generated by an SMD was in the range of [50-300]. According to the statistics by Google, the arrival time of computation tasks follows an exponential distribution [39]. The data size (DS) of the subtask and the number of required CPU cycles to complete (NRCC) of the subtask were generated by the exponential distribution of an average value of 200 kbit, 5 MHz, respectively. Initial values of the size of the population (G) and a maximum number of iterations

Algorithm 3 Finding optimal offloading decisions using the NSGA-III algorithm

```
Require: $\mathcal{N}$, $\mathcal{A}$, $G_i$, $I_i$, $pC$, $pM$
Ensure: Offloading decisions
 1: Generate reference points
 2: Generate the initial population
 3: Determine objective functions and CV
 4: Sorting and Selection:
 5:         Normalize the fitness values of solutions
 6:         Fast nondominated sorting
 7:         Associate each solution with reference points
 8:         Fast nondominated sorting
 9: while stopping condition is met do
10:     Crossover
11:     Mutation
12:     Combine parent and offspring populations
13:     Calculate objective functions and CV
14:     Sorting and Selection
15:     Create a new parent population
16: end while
17: return Pareto optimal set
```

Table 3 Simulation parameters

Parameter	Value
Number of SeNBs	5-10
Number of SMDs, M	20-45
Number of tasks, N	50-300
Number of subtasks in each task, K	10
Average DS of a subtask, $d_{m,n,k}$	200 kbit
Average NRCC of a subtask, $c_{m,n,k}$	5 MHz
Computation capability of an SMD, f_m	100 MHz
Computation capability of the Edge node, F	1 GHz
Bandwidth of wireless channel, B	40 MHz
Average deadline to complete a task, $\theta_{m,n}$	2 ms
Initial population size (G_i)	100
Initial number of iteration (I_i)	100
Probability of crossover (pC)	0.8
Probability of mutation (pM)	0.3

(P) were given, and they were adjusted during the simulation according to the number of computation tasks. The values of all variables used in the simulation are listed in Table 3.

5.2 Comparisons on Pareto Optimal Solutions

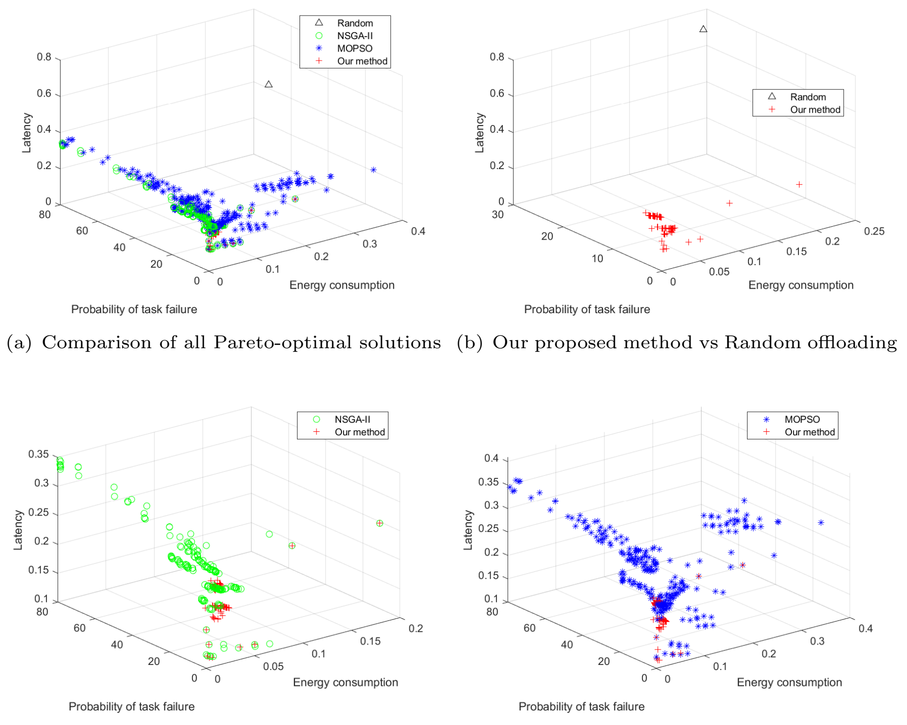
The computation offloading methods optimized with multiobjectives return several optimal solutions called Pareto Optimal set (Fig. 6). When the condition changes, the user selects the best one according to the real situation without rerunning the optimization algorithm. Fig. 6(a) shows the results of three methods produced by genetic algorithms and a random offloading method. The random offloading method produced only one solution shown as a triangle in the figures. The solutions of other three methods are shown with different symbols explained in the legend. We can see that, compared with other three methods, the results of the random offloading method are the worst in all three objectives: Latency, Task failure and Energy consumption. Among the other three methods, our proposed method produced the best results.

Fig. 6(a) shows the results of all methods together. We can see that the results from optimization methods are much better than the random offloading method. Fig. 6(b) is the comparison of the proposed method and the random offloading method. Fig. 6(c) shows the comparison results of the proposed method and the NSGA-II algorithm-based optimization method. Although both methods produced good results, we can see that the NSGA-II algorithm-based optimization method also produced solutions with high probability of task failure. Similar results can also be observed in Fig. 6(d) where the MOPSO algorithm-based optimization method produced high probability of task failure. These comparisons demonstrate that our method is able to reduce the task failure.

5.3 Comparison on Resource Utilization

The resources such as computation units of SMDs and MEC servers and the bandwidth of wireless channels are limited. Overloading the task workload with limited resources leads to long latency or even failure of the task. We test the proposed and existing methods in cases with overloading the task workload. For this purpose, we conducted experiments with varied workloads by increasing the number of tasks and SMDs. In each experiment, the value of the resource utilization coefficient was computed. The values of the resource utilization coefficient of an SMD, wireless channel, and the MEC server were calculated using $\rho_m = \gamma_m \lambda_m \widetilde{b}_m$, $\rho_R = \delta_m \lambda_m \widetilde{b}_{tr}$, and $r = \sum_{m=1}^{M} \delta_m \lambda_m \widetilde{b}_E$ respectively. The experiments were performed 100 times and the average and standard deviation of the utilization coefficient were calculated.

We evaluated the proposed method and other three methods in the settings of computations tasks from 50 to 300, where each task consists of 10 subtasks. The best results of the experiments are highlighted in bold in Table 4. The results show that the number of tasks does not affect resource utilization significantly. This is because the proposed computation offloading framework selects only one task from each SMD to find the optimal decision each time. The next task is selected after the decision on the previous one is made. The highest utilization of the resource appeared on the wireless channel. Since the



(a) Comparison of all Pareto-optimal solutions (b) Our proposed method vs Random offloading

(c) Our proposed method vs NSGA-II (d) Our proposed method vs MOPSO

Fig. 6 Comparison of Pareto-optimal solutions.

Table 4 Impact of number of tasks to resource utilization (%) in varied methods (mean ± std)

		Number of tasks					
		50	100	150	200	250	300
Average resource utilization of SMD (%)	Random	18.31 ± 2.05	$\mathbf{25.38 \pm 1.35}$	18.01 ± 0.60	28.45 ± 1.28	14.54 ± 0.92	$\mathbf{13.29 \pm 0.52}$
	NSGA-II	19.74 ± 0.08	34.83 ± 0.24	23.17 ± 0.22	55.92 ± 0.34	16.48 ± 0.06	23.81 ± 0.13
	MOPSO	21.06 ± 0.00	36.75 ± 0.00	25.90 ± 0.00	82.37 ± 0.00	16.55 ± 0.00	31.43 ± 0.00
	Our method	$\mathbf{17.80 \pm 0.52}$	27.04 ± 0.87	$\mathbf{17.98 \pm 0.32}$	$\mathbf{27.74 \pm 0.74}$	$\mathbf{13.65 \pm 0.21}$	16.20 ± 0.32
Average resource utilization of wireless channel (%)	Random	95.43 ± 46.26	142.80 ± 64.09	122.78 ± 26.19	90.13 ± 16.07	269.63 ± 52.65	144.04 ± 30.24
	NSGA-II	27.27 ± 3.29	33.29 ± 1.97	35.75 ± 1.68	$\mathbf{33.42 \pm 1.16}$	38.04 ± 1.17	41.64 ± 0.65
	MOPSO	29.91 ± 0.00	43.96 ± 0.00	40.19 ± 0.00	47.36 ± 2.64	39.87 ± 0.00	38.29 ± 1.07
	Our method	$\mathbf{24.74 \pm 2.12}$	$\mathbf{31.31 \pm 1.68}$	$\mathbf{32.53 \pm 1.62}$	37.57 ± 2.66	$\mathbf{36.84 \pm 1.35}$	$\mathbf{37.86 \pm 2.20}$
Average resource utilization of edge server (%)	Random	11.64 ± 1.05	22.80 ± 1.17	15.11 ± 0.69	19.45 ± 0.59	26.46 ± 0.92	23.72 ± 0.72
	NSGA-II	1.37 ± 0.02	2.33 ± 0.25	2.61 ± 0.07	2.68 ± 0.20	3.18 ± 0.02	3.13 ± 0.03
	MOPSO	$\mathbf{1.34 \pm 0.00}$	2.91 ± 0.00	1.93 ± 0.00	2.64 ± 0.83	3.67 ± 0.00	3.76 ± 0.19
	Our method	1.69 ± 0.14	$\mathbf{1.98 \pm 0.58}$	$\mathbf{2.55 \pm 0.14}$	$\mathbf{2.54 \pm 0.43}$	$\mathbf{2.84 \pm 0.11}$	$\mathbf{2.51 \pm 0.07}$

resource utilization is less than 85-90%, the probability of task lost is low. In the table, we can see that most best results were produced by our proposed method.

In the experiments, we also investigated the impact of increase of the number of SMDs on resource utilization by gradually putting more SMDs to the SeNBs. The nominal bandwidth of the wireless channel was divided equally by the number of connected SMDs. In addition, the increase of SMDs created

Table 5 Impact of number of SMDs to resource utilization (%) in varied methods (mean ± std)

		Number of SMDs					
		20	25	30	35	40	45
Average resource utilization of SMD (%)	Random	**15.21 ± 0.52**	**12.40 ± 0.46**	**14.25 ± 0.48**	**14.40 ± 0.44**	**11.94 ± 0.34**	**15.60 ± 0.40**
	NSGA-II	23.6 ± 0.11	20.19 ± 0.02	21.45 ± 0.05	21.82 ± 0.03	22.53 ± 0.03	20.70 ± 0.04
	MOPSO	22.17 ± 0.00	18.40 ± 0.00	20.22 ± 0.00	19.94 ± 0.00	22.88 ± 0.00	19.50 ± 0.45
	Our method	22.01 ± 0.32	17.42 ± 0.38	17.83 ± 0.39	18.43 ± 0.38	19.07 ± 0.65	17.04 ± 0.38
Average resource utilization of wireless channel (%)	Random	112.62 ± 15.0	110.34 ± 16.0	165.34 ± 21.4	172.40 ± 17.92	205.71 ± 26.37	331.39 ± 21.2
	NSGA-II	45.40 ± 2.48	71.31 ± 3.04	84.81 ± 4.13	91.27 ± 5.28	104.38 ± 6.14	128.18 ± 0.02
	MOPSO	61.97 ± 7.41	98.94 ± 0.00	117.43 ± 2.05	115.23 ± 0.00	127.56 ± 15.75	248.16 ± 19.83
	Our method	**42.14 ± 3.58**	**68.20 ± 6.41**	**82.57 ± 7.46**	**89.21 ± 8.23**	**98.24 ± 8.43**	**112.71 ± 13.22**
Average resource utilization of edge server (%)	Random	29.63 ± 1.46	28.47 ± 1.30	39.88 ± 1.75	43.54 ± 1.88	43 ± 1.55	64.24 ± 2.56
	NSGA-II	4.35 ± 0.05	6.17 ± 0.02	8.23 ± 1.03	**15.24 ± 1.02**	19.45 ± 1.83	27.26 ± 1.73
	MOPSO	4.67 ± 0.04	6.65 ± 0.10	**7.54 ± 0.91**	15.53 ± 1.29	24.58 ± 2.07	26.44 ± 2.25
	Our method	**3.85 ± 0.09**	**5.72 ± 0.57**	8.42 ± 0.84	16.80 ± 1.46	**18.87 ± 1.58**	**23.00 ± 1.61**

more computation workload for the MEC server. When the random computation method is used to make the offloading decisions, the local server resources were less utilized, but the wireless channel was overloaded, resulting in high probability of failure tasks. However, The multiobjective optimization methods can better balance the resource utilization by finding optimal solutions on offloaded tasks. Table 5 shows the experiment results of four methods on resource utilization. We can see that the proposed method produced good overall results. The increase in the number of SMDs does not affect the utilization of local computing resources. It affects only the utilization of wireless channels and the MEC server. These results demonstrate that increasing the number of tasks does not affect the utilization of resources significantly in the case when the framework selects one task at a time from each SMD.

5.4 Impact of Varied Numbers of SMDs on Performance Metrics

From the previous section, we see that the increase on the number of SMDs significantly affects resource utilization. To understand how the number of SMDs affects the performance of the optimization algorithms, we conducted experiments to find the relations between the elapsed time of the optimization algorithms and the number of SMDs in the SeNBs. The elapsed time was obtained from the internal function of MATLAB. The number of SMDs used in the experiments was from 20 to 45. For each given number of SMDs, we used the same input datasets for all methods. However, the datasets were randomly generated on the fly for experiments. Therefore, the values of objective functions depend on the generated datasets. The values of bandwidth and computing capacity of the MEC servers were fixed during all experiments.

Table 6 shows the experiment results. The same experiment in each setting was executed one hundred times. The average values and their standard deviations are given in column. We can see in the bottom line that our proposed method is 58% and 21% faster than NSGA-II and MOPSO. In addition, it shows better results in optimizing the three objectives. For instance, the latency of our method is reduced by 72% (Random), 18% (NSGA-II), and 30% (MOPSO) compared with other methods. The probability of task failure is

Table 6 Impact of the number of SMDs to the performance (mean ± std)

		Number of SMDs					
		20	25	30	35	40	45
Latency	Random	0.78 ± 0.06	1.41 ± 0.07	1.05 ± 0.04	1.27 ± 0.05	0.91 ± 0.03	2.08 ± 0.15
	NSGA-II	0.21 ± 0.07	0.42 ± 0.14	0.27 ± 0.10	1.14 ± 1.08	$\mathbf{0.13 \pm 0.06}$	0.35 ± 0.29
	MOPSO	0.24 ± 0.08	0.50 ± 0.20	0.30 ± 0.10	1.33 ± 1.21	0.16 ± 0.10	0.42 ± 0.45
	Our method	$\mathbf{0.20 \pm 0.04}$	$\mathbf{0.41 \pm 0.11}$	$\mathbf{0.22 \pm 0.03}$	$\mathbf{0.58 \pm 0.17}$	0.32 ± 0.14	$\mathbf{0.32 \pm 0.19}$
Energy Consumption	Random	0.24 ± 0.01	0.27 ± 0.01	0.23 ± 0.01	0.31 ± 0.01	0.29 ± 0.01	0.23 ± 0.01
	NSGA-II	0.02 ± 0.04	$\mathbf{0.01 \pm 0.02}$	$\mathbf{0.01 \pm 0.02}$	$\mathbf{0.08 \pm 0.12}$	$\mathbf{0.01 \pm 0.01}$	$\mathbf{0.02 \pm 0.04}$
	MOPSO	0.04 ± 0.06	0.04 ± 0.04	0.04 ± 0.05	0.13 ± 0.15	0.01 ± 0.02	0.04 ± 0.06
	Our method	$\mathbf{0.02 \pm 0.01}$	0.11 ± 0.05	0.03 ± 0.01	0.22 ± 0.08	0.09 ± 0.06	0.08 ± 0.06
Probability of task failure (%)	Random	1.04 ± 0.17	3.23 ± 1.39	9.07 ± 2.49	13.81 ± 3.52	18.93 ± 6.56	25.38 ± 4.08
	NSGA-II	0.09 ± 0.01	1.65 ± 0.57	2.89 ± 0.79	2.22 ± 0.46	3.12 ± 0.46	6.59 ± 2.72
	MOPSO	0.12 ± 0.08	4.69 ± 0.62	6.99 ± 1.10	5.19 ± 1.85	6.80 ± 1.26	19.04 ± 3.69
	Our method	$\mathbf{0.03 \pm 0.00}$	$\mathbf{0.25 \pm 0.06}$	$\mathbf{0.85 \pm 0.44}$	$\mathbf{0.98 \pm 0.67}$	$\mathbf{2.47 \pm 0.96}$	$\mathbf{6.18 \pm 2.31}$
Elapsed time	NSGA-II	437.45 ± 22.66	533.52 ± 35.50	833.30 ± 46.58	1160.06 ± 60.59	1130.91 ± 64.95	1332.13 ± 82.58
	MOPSO	147.49 ± 14.01	304.61 ± 28.41	418.81 ± 29.99	718.55 ± 51.46	511.60 ± 43.16	793.55 ± 61.07
	Our method	$\mathbf{140.88 \pm 8.42}$	$\mathbf{201.72 \pm 10.67}$	$\mathbf{409.39 \pm 20.31}$	$\mathbf{463.67 \pm 22.58}$	$\mathbf{453.28 \pm 20.73}$	$\mathbf{608.27 \pm 30.18}$

reduced by 84% (Random), 35% (NSGA-II), and 74% (MOPSO). Regarding the energy consumption, the proposed method showed positive result compared with the random offloading, and negative result compared with the NSGA-II and MOPSO algorithms.

6 Conclusion

The rapidly increasing number of IoT devices creates new opportunities for users to have more services, and it also causes a problem of processing massive data. Partial offloading of the computation tasks from IoT devices onto nearby MEC edge nodes partially solves the problem. The problem remains challenging with designing a fast and effective computation offloading framework in resource-limited systems. A combination of preferences based on logical rules and an optimization tool can help to solve the problem. This paper is aimed to propose a hybrid method for fast optimal offloading of dependent computation tasks in a resource-limited IoT-MEC environment. The proposed method achieved better performance compared with existing methods. Especially, the time to search for optimal offloading decisions is reduced. This work will be tested in real-world experiments, and a decentralized method of computation offloading will be investigated.

Declarations

Ethical Approval and Consent to participate

Yes.

Human and Animal Ethics

Not applicable.

Consent for publication

Yes.

Availability of supporting data

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Funding

This work has been partially supported by the National Key R&D Program of China under Grant No. 2018YFB1800805, the National Natural Science Foundation of China under Grant No. 61772345, 61902257, 61972261, the Pearl River Young Scholars funding of Shenzhen University, and the Shenzhen Science and Technology Program under Grant No. RCYX20200-714114645048, No. JCYJ20190808142207420, and No. GJHZ2-0190822095416463.

Authors' contributions

Kuanishbay Sadatdiynov: Conceptualization, Investigation, Software, Methodology, Writing - original draft. **Laizhong Cui**: Conceptualization, Project administration, Resources, Validation. **Joshua Zhexue Huang**: Supervision, Funding acquisition, Methodology, Writing - review & editing.

Acknowledgments

In our experiments, we used the NSGA-II [40], NSGA-III [41], and MOPSO [42] algorithms. Dr. Mostapha Kalami Heris is acknowledged for delivering the implementations.

Author information

Authors and Affiliations

College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, 518061, Guangdong, P.R. China

Kuanishbay Sadatdiynov, Laizhong Cui and Joshua Zhexue Huang

References

[1] Bharadwaj, H.K., Agarwal, A., Chamola, V., Lakkaniga, N.R., Hassija, V., Guizani, M., Sikdar, B.: A review on the role of Machine Learning in enabling IoT based healthcare applications. IEEE Access **9**, 38859–38890 (2021). https://doi.org/10.1109/ACCESS.2021.3059858

[2] Khanna, A., Kaur, S.: Internet of things (IoT), applications and challenges: A comprehensive review. Wirel. Pers. Commun. **114**(2), 1687–1762 (2020). https://doi.org/10.1007/s11277-020-07446-4

[3] Mach, P., Becvar, Z.: Mobile edge computing: A survey on architecture and computation offloading. IEEE Commun. Surv. Tutorials **19**(3), 1628–1656 (2017). https://doi.org/10.1109/COMST.2017.2682318

[4] Hu, J., Li, K., Liu, C., Li, K.: Game-based task offloading of multiple mobile devices with qos in mobile edge computing systems of limited computation capacity. ACM Trans. Embed. Comput. Syst. **19**(4) (2020). https://doi.org/10.1145/3398038

[5] Liu, M., Yu, F.R., Teng, Y., Leung, V.C.M., Song, M.: Distributed resource allocation in blockchain-based video streaming systems with mobile edge computing. IEEE Trans. Wireless Communications **18**(1), 695–708 (2019). https://doi.org/10.1109/TWC.2018.2885266

[6] Zhou, J., Zhang, X., Wang, W.: Joint resource allocation and user association for heterogeneous services in multi-access edge computing networks. IEEE Access **7**, 12272–12282 (2019). https://doi.org/10.1109/ACCESS.2019.2892466

[7] Yi, C., Cai, J., Su, Z.: A multi-user mobile computation offloading and transmission scheduling mechanism for delay-sensitive applications. IEEE Trans. Mob. Comput. **19**(1), 29–43 (2020). https://doi.org/10.1109/TMC.2019.2891736

[8] Zhang, L., Cao, B., Li, Y., Peng, M., Feng, G.: A multi-stage stochastic programming-based offloading policy for Fog enabled IoT-eHealth. IEEE J. Sel. Areas Commun. **39**(2), 411–425 (2021). https://doi.org/10.1109/JSAC.2020.3020659

[9] Qiao, B., Liu, C., Liu, J., Hu, Y., Li, K., Li, K.: Task migration computation offloading with low delay for mobile edge computing in vehicular networks. Concurrency and Computation: Practice and Experience **34**(1) (2021). https://doi.org/10.1002/cpe.6494

[10] Zhang, J., Xia, W., Yan, F., Shen, L.: Joint computation offloading and resource allocation optimization in heterogeneous networks with mobile edge computing. IEEE Access **6**, 19324–19337 (2018). https://doi.org/10.1109/ACCESS.2018.2819690

[11] Liu, P., Xu, G., Yang, K., Wang, K., Meng, X.: Jointly optimized energy-minimal resource allocation in cache-enhanced mobile edge computing systems. IEEE Access **7**, 3336–3347 (2019). https://doi.org/10.1109/ACCESS.2018.2889815

[12] Yang, L., Zhong, C., Yang, Q., Zou, W., Fathalla, A.: Task offloading for directed acyclic graph applications based on edge computing in industrial internet. Inf. Sci. **540**, 51–68 (2020). https://doi.org/10.1016/j.ins.2020.

06.001

[13] Shen, S., Han, Y., Wang, X., Wang, Y.: Computation offloading with multiple agents in edge-computing-supported IoT. TOSN **16**(1), 8–1827 (2020). https://doi.org/10.1145/3372025

[14] Afrin, M., Jin, J., Rahman, A., Tian, Y., Kulkarni, A.: Multi-objective resource allocation for edge cloud based robotic workflow in smart factory. Future Gener. Comput. Syst. **97**, 119–130 (2019). https://doi.org/10.1016/j.future.2019.02.062

[15] Xu, X., Fu, S., Yuan, Y., Luo, Y., Qi, L., Lin, W., Dou, W.: Multiobjective computation offloading for workflow management in cloudlet-based mobile cloud using NSGA-II. Comput. Intell. **35**(3), 476–495 (2019). https://doi.org/10.1111/coin.12197

[16] Cui, L., Xu, C., Yang, S., Huang, J.Z., Li, J., Wang, X., Ming, Z., Lu, N.: Joint optimization of energy consumption and latency in mobile edge computing for Internet of Things. IEEE Internet of Things Journal **6**(3), 4791–4803 (2019). https://doi.org/10.1109/JIOT.2018.2869226

[17] Alkhalaileh, M., Calheiros, R.N., Nguyen, Q.V., Javadi, B.: Data-intensive application scheduling on mobile edge cloud computing. J. Netw. Comput. Appl. **167**, 102735 (2020). https://doi.org/10.1016/j.jnca.2020.102735

[18] Xia, S., Yao, Z., Li, Y., Mao, S.: Online distributed offloading and computing resource management with energy harvesting for heterogeneous MEC-enabled IoT. IEEE Transactions on Wireless Communications **20**, 6743–6757 (2021). https://doi.org/10.1109/TWC.2021.3076201

[19] Cui, Y., Zhang, D., Zhang, T., Chen, L., Piao, M., Zhu, H.: Novel method of mobile edge computation offloading based on evolutionary game strategy for iot devices. AEU - International Journal of Electronics and Communications **118**, 153134 (2020). https://doi.org/10.1016/J.AEUE.2020.153134

[20] Tong, Z., Deng, X., Ye, F., Basodi, S., Xiao, X., Pan, Y.: Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment. Inf. Sci. **537**, 116–131 (2020). https://doi.org/10.1016/j.ins.2020.05.057

[21] Chalapathi, G.S.S., Chamola, V., Johal, W., Aryal, J., Buyya, R.: Energy and latency aware mobile task assignment for green cloudlets. Simulation Modelling Practice and Theory, (2022). https://doi.org/10.1016/j.simpat.2022.102531

[22] Chakraborty, S., Mazumdar, K.: Sustainable task offloading decision using

genetic algorithm in sensor mobile edge computing. Journal of King Saud University - Computer and Information Sciences **34**(4), 1552–1568 (2022). https://doi.org/10.1016/j.jksuci.2022.02.014

[23] Zhang, W., Wen, Y.: Energy-efficient task execution for application as a general topology in mobile cloud computing. IEEE Trans. Cloud Comput. **6**(3), 708–719 (2018). https://doi.org/10.1109/TCC.2015.2511727

[24] Li, Y., Xia, S., Zheng, M., Cao, B., Liu, Q.: Lyapunov optimization-based trade-off policy for mobile cloud offloading in heterogeneous wireless networks. IEEE Trans. Cloud Comput. **10**(1), 491–505 (2022). https://doi.org/10.1109/TCC.2019.2938504

[25] Jiang, C., Cheng, X., Gao, H., Zhou, X., Wan, J.: Toward computation offloading in edge computing: A survey. IEEE Access **7**, 131543–131558 (2019). https://doi.org/10.1109/ACCESS.2019.2938660

[26] Gasmi, K., Dilek, S., Tosun, S., Ozdemir, S.: A survey on computation offloading and service placement in fog computing-based IoT. J. Supercomput. **78**(2), 1983–2014 (2022). https://doi.org/10.1007/s11227-021-03941-y

[27] Sadatdiynov, K., Cui, L., Zhang, L., Huang, J.Z., Salloum, S., Mahmud, M.S.: A review of optimization methods for computation offloading in edge computing networks. Digital Communications and Networks, (2022). https://doi.org/10.1016/j.dcan.2022.03.003

[28] Guo, S., Liu, J., Yang, Y., Xiao, B., Li, Z.: Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing. IEEE Trans. Mob. Comput. **18**(2), 319–333 (2019). https://doi.org/10.1109/TMC.2018.2831230

[29] Yang, L., Cao, J., Cheng, H., Ji, Y.: Multi-user computation partitioning for latency sensitive mobile cloud applications. IEEE Trans. Computers **64**(8), 2253–2266 (2015). https://doi.org/10.1109/TC.2014.2366735

[30] Orhean, A.I., Pop, F., Raicu, I.: New scheduling approach using reinforcement learning for heterogeneous distributed systems. J. Parallel Distributed Comput. **117**, 292–302 (2018). https://doi.org/10.1016/j.jpdc.2017.05.001

[31] Pan, S., Zhang, Z., Zhang, Z., Zeng, D.: Dependency-aware computation offloading in mobile edge computing: A reinforcement learning approach. IEEE Access **7**, 134742–134753 (2019). https://doi.org/10.1109/ACCESS.2019.2942052

[32] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist

multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation **6**(2), 182–197 (2002). https://doi.org/10.1109/4235.996017

[33] Deb, K., Jain, H.: An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. IEEE Trans. Evol. Comput. **18**(4), 577–601 (2014). https://doi.org/10.1109/TEVC.2013.2281535

[34] Jain, H., Deb, K.: An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part II: handling constraints and extending to an adaptive approach. IEEE Trans. Evol. Comput. **18**(4), 602–622 (2014). https://doi.org/10.1109/TEVC.2013.2281534

[35] Xu, X., Liu, Q., Luo, Y., Peng, K., Zhang, X., Meng, S., Qi, L.: A computation offloading method over big data for IoT-enabled cloud-edge computing. Future Gener. Comput. Syst. **95**, 522–533 (2019). https://doi.org/10.1016/j.future.2018.12.055

[36] Mao, Y., Zhang, J., Letaief, K.B.: Dynamic computation offloading for mobile-edge computing with energy harvesting devices. IEEE Journal on Selected Areas in Communications **34**(12), 3590–3605 (2016). https://doi.org/10.1109/JSAC.2016.2611964

[37] Shortle, J.F., Thompson, J.M., Gross, D., Harris, C.M.: Fundamentals of Queueing Theory, 5th edn. Wiley Series in Probability and Statistics. Wiley, ??? (2018)

[38] Das, I., Dennis, J.E.: Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems. SIAM J. Optim. **8**(3), 631–657 (1998). https://doi.org/10.1137/S1052623496307510

[39] Zhao, T., Zhou, S., Guo, X., Niu, Z.: Tasks scheduling and resource allocation in heterogeneous cloud for delay-bounded mobile edge computing. In: IEEE International Conference on Communications, ICC 2017, Paris, France, May 21-25, 2017, pp. 1–7. IEEE, ??? (2017). https://doi.org/10.1109/ICC.2017.7996858

[40] Heris, M.K.: NSGA-II in MATLAB. Yarpiz (2015). https://yarpiz.com/56/ypea120-nsga2 Accessed 2022-07-05

[41] Heris, M.K.: NSGA-III: Non-dominated Sorting Genetic Algorithm, the Third Version — MATLAB Implementation. Yarpiz (2016). https://yarpiz.com/456/ypea126-nsga3 Accessed 2022-07-05

[42] Heris, M.K.: Multi-Objective PSO in MATLAB. Yarpiz (2015). https://yarpiz.com/59/ypea121-mopso Accessed 2022-07-05