Computation Offloading for Object-Oriented Applications in a UAV-based Edge-Cloud Environment

***Abstract*—The high mobility and maneuverability of unmanned aerial vehicles (UAVs) enable them to act as temporary base stations (BSs) in the extreme environments, expanding the computing capacities of terminals for intelligent applications, most of which are object-oriented ones. Computation offloading in a UAV-based edge-cloud environment, is a good way to improve the performance of these object-oriented intelligent applications, while the computation-intensive tasks are offloaded to the cloud, and the data-intensive ones are offloaded to the edge. Though computation offloading over the cloud, edges and terminals, has been broadly studied, existed works mostly establish scheduling algorithms on program high-level abstraction without consideration of challenges from program structures. This study focuses on task scheduling for offloading object-oriented applications, while considering the 'encapsulation' characteristic. We proposed a time-driven offloading strategy based on a particle swarm optimization algorithm employing the operators of genetic algorithm with floating encoding (PGFE). This strategy introduces the randomly two-point crossover and mutation operator of a genetic algorithm to effectively avoid converging on local optima. The simulation results show that our strategy can reduce the average execution time of object-oriented applications by 11.78%-48.02%, compared with other benchmark solutions in a UAV-based edge-cloud environment.**

***Index Terms*—Mobile edge computing, Computation offloading, Task Scheduling, Object-oriented applications, UAV.**

I. Introduction

With the development of software and hardware technology for the unmanned aerial vehicles (UAVs), UAVs have attracted tremendous attention in various fields, such as target detection [1], remote sensing [2], surveillance [3], service delivery [4], pollution detection [5], and agriculture [6], while the size and price of which are becoming smaller and cheaper, respectively. The high mobility and maneuverability of UAVs enable them not only to enter some dangerous places for a rescue mission or pollution detection instead of humans, but also to act as temporary base stations (BSs) in the extreme environments, expanding the computing capacities of terminals for intelligent applications, most of which are object-oriented ones [7].

Computation offloading in a UAV-based edge-cloud environment, consisting of the cloud, edge (*i.e.*, UAVs), and terminals, is a good way to improve the performance of these object-oriented intelligent applications [7], while the computation-intensive tasks are offloaded to the cloud, and the data-intensive ones are offloaded to the edge. Computation offloading over the cloud, edge, and terminals, has been broadly studied [8]. These studies mostly establish scheduling algorithms on program high-level abstraction [9], and much less attention is paid to challenges from program structures.

Our previous studies [10], [11] provide a mechanism that supports the offloading at the granularity of objects. However, it still faces challenges to do scheduling for offloading these object-oriented applications in a UAV-based edge-cloud environment. On the one hand, we should consider data dependencies between method invocations cased by 'encapsulation' characteristic of objects, which leads to object migration during offloading. On the other hand, considering the mobility of UAVs, the UAV-based edge-cloud environment changes from time to time, so the offloading strategy needs to be resulted in a short time.

This paper focuses on task scheduling for offloading object-oriented applications in a UAV-based edge-cloud environment. In particular, a novel computation offloading strategy for object-oriented applications is proposed, on the basis of a particle swarm optimization (PSO) algorithm employing the operators of a genetic algorithm (GA) with a floating encoding scheme (PGFE). The resulting strategy can obtain an optimal average execution time of object-oriented applications in a UAV-based edge-cloud environment with respect to commonly adopted benchmark datasets.

The main contributions of this study are composed of the following three parts:

- The computation offloading model for object-oriented applications in a UAV-based edge-cloud environment is built, which fully considers the object migration during the computation offloading.
- A novel floating encoding mechanism is proposed, which makes up for the shortcomings of the extra adjustment process with the traditional encoding mechanism of the PSO algorithm.
- A time-driven offloading strategy based on PGFE for object-oriented applications is proposed, which effectively reduces the average execution time of object-oriented applications caused by method executions and data transmission between them.

The rest of this paper is organized as follows: first of all, we summarize related works to this study in Section II. In Section III, we give a detailed introduction to the problem definition and analysis. Section IV introduces the algorithm used in this paper. In Section V, the simulation work and the results are discussed. Finally, in Section VI, the paper is summarized and the future research direction is given.

II. Related Work

The high mobility and maneuverability of UAVs enable them to act as edge servers in extreme environments, while many studies focus on computation offloading in a UAV-based edge computing environment [12]–[14]. Zhang *et al.* [12] studied an energy-efficient UAV enabled mobile edge computing (MEC) framework incorporating non-orthogonal multiple access (NOMA), which deployed multiple UAVs as edge servers to provide computing assistance to ground users, and adopted NOMA to reduce the energy consumption of computation offloading. Hu *et al.* [13] focused on the joint optimization problem of UAVs position, slot allocation, and computation task partition, aiming to minimize the energy consumption of all users on the basis of ensuring that all user tasks can be completed. This work proposes a global optimal solution, which is obtained by two-dimensional searching for UAV positions. In these studies, the offloaded tasks are all 'infinitesimally divisible', and the data dependencies in the applications are not considering. Callegar *et al.* [14] developed a framework that could make the optimal offloading decision for UAV applications based on the network, computation workload parameters, and current status. When the UAVs act as edge servers in the network, they provided edge services for smaller mobile devices in the network. However, the above researches usually fail to address the object-oriented applications with latency-sensitive requirements in a UAV-based edge computing environment.

In order to address the above issue, many studies focus on computation offloading in UAV-based edge-cloud environments, consisting of the cloud, edge (*i.e.*, UAVs), and terminals [15], [16]. Liu *et al.* [15] studied the computation offloading and routing optimization of UAVs under the UAV-Edge-Cloud computing architecture, and proposed a polynomial near-optimal approximation algorithm to solve the above problem. Aiming at the security of the physical layer, Bai *et al.* [16] proposed a computation offloading strategy for UAV-MEC system, and finally found the best solution for active and passive eavesdroppers, while the UAVs act as edge servers in the network. The above work pays much less attention to computation offloading with data dependencies in an application.

A few studies focus on task scheduling for offloading objected-oriented applications in edge-cloud environments [17]–[19]. Xian *et al.* [17] designed an adaptive timeout scheme for single-task offloading to improve energy-saving efficiency on mobile devices. By citing Lyapunov optimization, Awak *et al.* [18] studied the dynamic computing offloading strategy, with the goal of minimizing CPU and network energy consumption in the actual network environment. Sardellitti *et al.* [19] proposed an iterative algorithm for joint optimization of radio and computing resources for multi-cell mobile edge computing under the constraints of waiting time and power budget. However, the above works established scheduling algorithms on program high-level abstraction, without consideration of challenges from program structures.

III. Problem Definition and Analysis

This study aims to reduce the average execution time of object-oriented applications mainly caused by method executions and data transmission between them, while considering the 'encapsulation' characteristic of objects during offloading process. This subsection discusses the assumption made in this paper, such as, how the UAV-based edge-cloud environment is constructed, how the object-oriented applications are defined, and how the time-driven offloading strategy is generated.

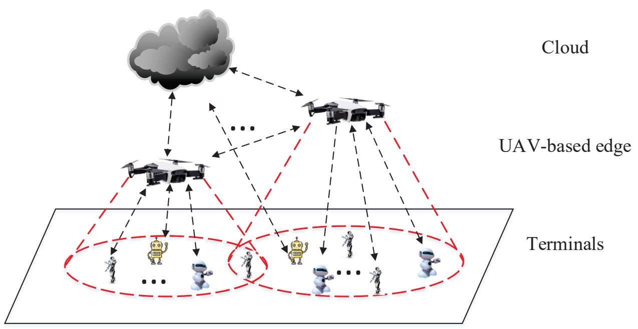


Fig. 1. An illustration of a UAV-based edge-cloud environment.

A. UAV-based Edge-Cloud Environment

The UAV-based edge-cloud environment $\boldsymbol{U} = \{\boldsymbol{U}_{\mathrm{c}}, \boldsymbol{U}_{\mathrm{e}}, \boldsymbol{U}_{\mathrm{t}}\}$ consists of the cloud, UAV-based edge, and terminals. Figure 1 gives an illustration of a UAV-based edge-cloud environment. The edge platform is composed of several UAVs, and each platform has various computing nodes with different computing capacity. In this paper, 'server' is used to represent the various computing nodes in different platforms. Therefore, the UAV-based edge-cloud environment has n servers $\boldsymbol{U} = \{s_1, s_2, ..., s_n\}$, and a server s_i is expressed as

$$s_i =< \boldsymbol{\kappa}_i, p_i, \gamma_i >, \tag{1}$$

$$\boldsymbol{\kappa}_i = (x_i, y_i, z_i), \tag{2}$$

where $\boldsymbol{\kappa}_i$ is the position of server s_i in the three-dimensional space, and (x_i, y_i, z_i) corresponds to the respective dimensions of $\boldsymbol{\kappa}_i$. p_i indicates the computing capacity of server s_i, usually represented by its CPU power. In this paper, the computing capacity of each server is assumed to be known in advance and not fluctuant during computation offloading. As we focus on computation offloading in this study, the storage capacity of each server is assumed to satisfy the storage needs for all object-oriented applications. $\gamma_i \in \{0, 1, 2\}$ indicates the type of server s_i. If $\gamma_i = 0$, s_i is the terminal usually with poor computing capacity. If $\gamma_i = 1$, s_i is the UAV (*i.e.*, belongs to edge) usually with normal computing capacity. If $\gamma_i = 2$, s_i belongs to the cloud usually with powerful computing capacity.

In a UAV-based edge-cloud environment, any two servers are connected and exchanging data through a specific link, due to that the edge platform is composed of several UAVs. We assume that the wireless channel in the computing environment is simulated as Rayleigh Fading [20]. Therefore, the transmission speed of wireless link $r_{i,j}$ between server s_i and server s_j can be calculated using Shannon's formula [21] as (3).

$$r_{i,j} = B \cdot \log_2 \left(1 + \frac{p_{i,j} \cdot g_{i,j}}{\sigma^2}\right), \forall s_i, s_j \in \boldsymbol{U}, \tag{3}$$

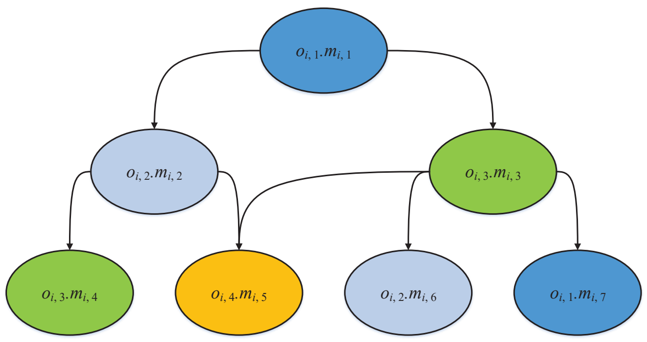


Fig. 2. A computation-intensive object-oriented application.

where B is the channel bandwidth, $p_{i,j}$ is the transmission power between server s_i and server s_j, σ^2 is the channel noise power, and $g_{i,j}$ is the wireless channel gain between server s_i and server s_j, whose details is described as (4).

$$g_{i,j} = (l_{i,j})^{-\alpha}\beta_{i,j} = \|\boldsymbol{\kappa}_i - \boldsymbol{\kappa}_j\|^{-\alpha}\beta_{i,j}, \forall s_i, s_j \in \boldsymbol{U}, \quad (4)$$

where $l_{i,j}$ is the spatial distance between server s_i and server s_j, α is the path attenuation index, and $\beta_{i,j}$ is the Rayleigh Fading factor between server s_i and server s_j.

B. Object-oriented Applications

Each terminal generates an object-oriented application irregularly. There are many types of object-oriented applications, $\boldsymbol{T} = \{\boldsymbol{T}_1, \boldsymbol{T}_2, ..., \boldsymbol{T}_g\}$, generated by all terminals at a certain moment in a UAV-based edge-cloud environment. A object-oriented application $\boldsymbol{T}_i \in \boldsymbol{T}$ is represented by a tuple $\boldsymbol{T}_i = (\boldsymbol{M}_i, \boldsymbol{E}_i, \boldsymbol{O}_i)$, which is composed of several computation methods belonging to different objects. $\boldsymbol{M}_i = \{m_{i,1}, m_{i,2}, ..., m_{i,q}\}$ is the set of methods contained in application $\boldsymbol{T}_i$. $\boldsymbol{E}_i = \left\{< e_i^{k,l}, d_i^{k,l} > | k, l \in \{1, 2, ..., q\}\right\}$ is the set of data dependencies between each pair of methods in application $\boldsymbol{T}_i$, where $e_i^{k,l} \in \{0, 1\}$ indicates whether there is data transfer from method $m_{i,k}$ to method $m_{i,l}$. If $e_i^{k,l} = 1$, there is a dataset transferred from $m_{i,k}$ to $m_{i,l}$, whose transmission amount is $d_i^{k,l}$. Otherwise, there is no data transfer from $m_{i,k}$ to $m_{i,l}$. $\boldsymbol{O}_i = \{o_{i,1}, o_{i,2}, ..., o_{i,w}\}$, $w \le q$, is the set of objects contained in application $\boldsymbol{T}_i$. A object, $o_{i,j} = \left(\boldsymbol{M}_i^j, d_{i,j}\right)$, usually corresponds to a specific set of methods $\boldsymbol{M}_i^j \subset \boldsymbol{M}_i$, where $d_{i,j}$ indicates the transmission amount of $o_{i,j}$ when the object migration is occurred between two different servers. Note that a method belongs to only one object, and a object may contains several methods. As in Figure 2, $\boldsymbol{T}_i$ has 4 objects $\{o_{i,1}, ..., o_{i,4}\}$ and 7 methods $\{m_{i,1}, ..., o_{m,7}\}$, where $\{m_{i,1}, m_{i,7}\}$, $\{m_{i,2}, m_{i,6}\}$, $\{m_{i,3}, m_{i,4}\}$ and $m_{i,5}$ belong to $o_{i,1}$, $o_{i,2}$, $o_{i,3}$ and $o_{i,4}$, respectively.

Each method $m_{i,j} \in \boldsymbol{M}_i$ in an object-oriented application $\boldsymbol{T}_i$ can be expressed as

$$m_{i,j} = (\lambda_{i,j}, \boldsymbol{M}_{i,j}^P, \boldsymbol{M}_{i,j}^S, \boldsymbol{D}_{i,j}, o_{i,k}), \quad (5)$$

where $\lambda_{i,j}$ is the calculated amount of $m_{i,j}$, $\boldsymbol{M}_{i,j}^P = \left\{m_{i,j'} | e_i^{j',j} = 1\right\}$ is the set of precursor methods of $m_{i,j}$, $\boldsymbol{M}_{i,j}^S = \left\{m_{i,j''} | e_i^{j,j''} = 1\right\}$ is the set of successor methods of $m_{i,j}$, $\boldsymbol{D}_{i,j} = \left\{d_i^{j',j} | e_i^{j',j} = 1\right\}$ is the input data set of method $m_{i,j}$, and $o_{i,k}$ indicates the object the method $m_{i,j}$ belongs to.

In addition, many time-points related to object-oriented applications and their methods will be elaborated. The generation time of object-oriented application $\boldsymbol{T}_i$ generated from its corresponding terminal is represented as t_i^{gen}. The serial processing model [22] is adopted in the execution process, which means that a server can only execute one method at the same time and the whole method is executed on the same server. Therefore, the start time of a method, $t_{i,j}^{start}$, in an object-oriented application $\boldsymbol{T}_i$ executed on server s_k is represented as

$$t_{i,j}^{start} = \begin{cases} t_{i,j}^{arr}, & \text{if } s_k \text{ is idle} \\ t_{s_k}^{idle}, & \text{otherwise} \end{cases}, \quad (6)$$

where $t_{i,j}^{arr}$ is the arrival time of datasets the method $m_{i,j}$ needs before being executed, which is described in detail as (8). When server s_k is busy, the method $m_{i,j}$ has to start until all methods on s_k have been completed. $t_{s_k}^{idle}$ is the time when s_k completes all methods before $m_{i,j}$. Note that all methods scheduled to the same server have to comply with queuing theory [23]. It means the method that arrives the server first is executed first. Therefore, the start time of object-oriented application $\boldsymbol{T}_i$, t_i^{start}, is represented as

$$t_i^{start} = \min t_{i,j}^{start}. \quad (7)$$

$$t_{i,j}^{arr} = \max_{m_{i,j'} \in \boldsymbol{M}_{i,j}^P} (t_{i,j'}^{end} + \frac{d_i^{j',j}}{r_{k,l}}) + \frac{d_{i,m}}{r_{u,l}}, m_{i,j} \in \boldsymbol{M}(o_{i,m}), \quad (8)$$

where $m_{i,j'}$ is the precursor method of $m_{i,j}$, and $t_{i,j'}^{end}$ is the end time of method $m_{i,j'}$, which is described in detail as (9). When the precursor method $m_{i,j'}$ and method $m_{i,j}$ are executed on different servers, the data transmission time $\frac{d_i^{j',j}}{r_{k,l}}$ from server s_k to server s_l should be considered. If server s_k and server s_l are the same server, there is no data transmission. Moreover, when the object $o_{i,m}$ the method $m_{i,j}$ belongs to is on server s_u before executing $m_{i,j}$, it has to be transferred from server s_u to server s_l. If server s_u and server s_l are the same server, there is no object migration. $\boldsymbol{M}(o_{i,m})$ is the set of methods belonging to object $o_{i,m}$.

$$t_{i,j}^{end} = t_{i,j}^{start} + \frac{\lambda_{i,j}}{p_k}, \quad (9)$$

where method $m_{i,j}$ is executed on server s_k. The end time of a object-oriented application $\boldsymbol{T}_i$, is represented as

$$t_i^{end} = \max_{m_{i,j} \in \boldsymbol{M}_i} t_{i,j}^{end}. \quad (10)$$

Therefore, the average execution time of g object-oriented applications, t^{aver}, in the UAV-based edge-cloud environment is represented as

$$t^{aver} = \frac{1}{g} \sum_i^g (t_i^{end} - t_i^{gen}). \quad (11)$$

C. Time-Driven Offloading Strategy

The purpose of our offloading strategy is to minimize the average execution time of object-oriented applications mainly caused by method executions and data transmission between them, while considering the 'encapsulation' characteristic of object-oriented applications during the offloading process. Any method execution has to satisfy the following conditions: (1) The method has been offloaded to a specific server; (2) the datasets required by this method have been transferred to the same server; (3) The object containing this method has been offloaded to the same server with its datasets.

The offloading strategy for object-oriented applications in a UAV-based edge-cloud environment can be described as $\boldsymbol{S}= (\boldsymbol{U}, \boldsymbol{T}, \boldsymbol{N}, t^{aver})$, where $\boldsymbol{N} = \{(m_{i,j}, s_k)|m_{i,j} \in \boldsymbol{T}_i, s_k \in \boldsymbol{U}\} \cup \{\Gamma_k\}$ indicates the map set from methods to the participated servers with a certain execution order. $(m_{i,j}, s_k)$ indicates that method $m_{i,j}$ is executed on server s_k, and Γ_k indicates the corresponding execution order of the concurrent methods on server s_k. When the map set $\boldsymbol{N}$ is determined, the execution location and order for each method is determined accordingly. Therefore, the time-driven offloading for object-oriented applications in a UAV-based edge-cloud environments can be formalized as

$$\boldsymbol{Minimize}\ t^{aver} \tag{12}$$

The core purpose is to pursue a minimum average execution time for all object-oriented applications.

IV. Offloading Strategy based on PGFE

The core of our offloading strategy, $\boldsymbol{S}= (\boldsymbol{U}, \boldsymbol{T}, \boldsymbol{N}, t^{aver})$, is to explore a optimal map $\boldsymbol{N}$ from all methods $\boldsymbol{M}_i$ to different types of servers $\boldsymbol{U}$. Its purpose is to reduce the average execution time of object-oriented application mainly caused by method executions and data transmission between them, while considering the 'encapsulation' characteristic of object-oriented applications during offloading. A method can be offloaded to different servers, and an available server can execute many methods from different applications. Therefore, exploring an optimal map from methods to servers is an NP-hard problem [24], [25]. PSO algorithm is one of the effective approaches to solve such issues, and we proposed an offloading strategy based on PGFE algorithm.

PSO algorithm is an artificial intelligence technique based on population, which was first presented by Eberhart and Kennedy in 1995 [26]. A population contains multiple particles, and a particle $Q_i^t = (P_i^t, V_i^t)$ has its own position, $P_i^t = \left(p_{i,1}^t, p_{i,2}^t, ..., p_{i,h}^t\right)$, and velocity, $V_i^t = \left(v_{i,1}^t, v_{i,2}^t, ..., v_{i,h}^t\right)$, in a continuous space at t^{th} iteration. The position of a particle represents a candidate solution for the optimization problem, and the velocity can adjust each particle searching for better positions. The global best particle in the population at t^{th} iteration is defined as $gBest^t$, and the personal best particle in its own history at t^{th} iteration is defined as $pBest_i^t$.

The initial prototype of PSO is used to solve continuous optimization problems. However, the time-driven offloading for object-oriented applications in a UAV-based edge-cloud environment is a discrete problem. Moreover, the traditional PSO algorithm has the disadvantage of premature convergence. In order to address the above issues, a particle swarm optimization algorithm employing the operators of a genetic algorithm with floating encoding is proposed.

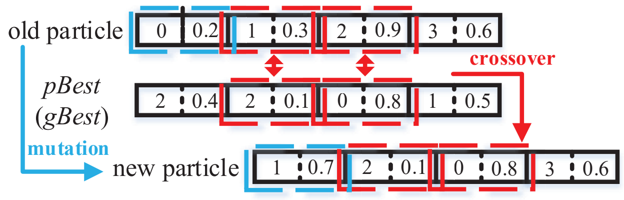


Fig. 3. The crossover operator and mutation operator.

A. Problem Encoding

Inspired by [27], we use the server-order nesting mechanism to encode the time-driven offloading problem for object-oriented applications in a UAV-based edge-cloud environment. The i^{th} particle in the t^{th} iteration in a population, P_i^t, represents a potential offloading result, which is decoded as follows.

$$P_i^t = \left(p_{i,1}^t, p_{i,2}^t, ..., p_{i,\vartheta}^t\right), \tag{13}$$

$$p_{i,j}^t = \left(\chi_{i,j}^t, \varepsilon_{i,j}^t\right), \tag{14}$$

where ϑ indicates the total number of methods from all applications during offloading. $p_{i,j}^t$ indicates the j^{th} dimension of particle P_i^t, which corresponds to the offloading decision of the j^{th} method. $\chi_{i,j}^t \in \{1, 2, ..., n\}$ represents the executing server for j^{th} method, and $\varepsilon_{i,j}^t$ is the corresponding execution order of j^{th} method on server $\chi_{i,j}^t$. $\left(\chi_{i,j}^t, \varepsilon_{i,j}^t\right)$ indicates that the j^{th} method is executed on server $\chi_{i,j}^t$ with a specified order $\chi_{i,j}^t$ in t^{th} iteration. When more than two concurrent methods are scheduled to the same server in the same time, the smaller ordered methods are executed first.

B. Fitness Function

The fitness function is used to evaluate the performance of particles. In general, a particle with a smaller fitness function value indicates a better candidate solution. The purpose of this study is to minimize the average execution time of object-oriented applications in a UAV-based edge-cloud environment. Therefore, the average execution time of all object-oriented applications can be directly used as the fitness value of a particle. The fitness function of particle P_i^t is defined as follows.

$$F\left(P_i^t\right) = t_i^{aver}\left(P_i^t\right), \tag{15}$$

where $t_i^{aver}\left(P_i^t\right)$ represents the average execution time of all object-oriented application from the offloading results corresponding to particle P_i^t.

C. Update Strategy

In our previous work [28], the update strategy of traditional PSO has been described in detail. The update of a particle in PSO, affected by its personal best particle $pBest_i^t$ and the global best particle $gBest^t$ in current population, mainly includes three parts: *inertia*, *personal cognition* and *social cognition* [26]. In order to avoid the traditional PSO's defect that is easy to fall into the local optimum and enhance the search ability of our algorithm, the crossover and mutation operators of GA are introduced. In the $(t+1)^{\text{th}}$ iteration, the update of the i^{th} particle is shown as (16).

$$P_i^{t+1} = c_2 \oplus C_r\left(c_1 \oplus C_r\left(\omega \oplus M_u\left(P_i^t\right), pBest_i^t\right), gBest^t\right), \tag{16}$$

where $C_r()$ and $M_u()$ represent the crossover and mutation operators of GA, respectively. ω is the inertia weight, c_1 and c_2 are acceleration coefficients in PSO [29].

For the *inertia* part, the mutation operator of GA is introduced to update the corresponding part of a particle as follows.

$$A_i^{t+1} = \omega \oplus M_u\left(P_i^t\right) = \begin{cases} M_u\left(P_i^t\right) & r_1 < \omega \\ P_i^t & \text{else} \end{cases}, \tag{17}$$

where r_1 is a random factor between 0 and 1, and ω denotes the mutation probability. The mutation operator randomly selects a location in a particle, and then randomly changes the corresponding value of the server-order couple in their respective value ranges. The mutation operation is shown as the blue rectangle in Figure 3. It randomly selects the first location of the old particle, and then changes the corresponding value of the server-order couple from $<0, 0.2>$ to $<1, 0.7>$.

For the *personal cognition* and *social cognition* parts, the crossover operator of GA is introduced to update the corresponding parts of a particle, as shown in (18) and (19), respectively.

$$B_i^{t+1} = c_1 \oplus C_r\left(A_i^{t+1}, pBest_i^t\right) = \begin{cases} C_r\left(A_i^{t+1}, pBest_i^t\right) & r_2 < c_1 \\ A_i^{t+1} & \text{else} \end{cases} \tag{18}$$

$$C_i^{t+1} = c_2 \oplus C_r\left(B_i^{t+1}, gBest^t\right) = \begin{cases} C_r\left(B_i^{t+1}, pBest_i^t\right) & r_3 < c_2 \\ B_i^{t+1} & \text{else} \end{cases} \tag{19}$$

where r_2 and r_3 are random factors between 0 and 1, c_1 and c_2 denote the crossover probability of the corresponding particle with its personal best particle $pBest_i^t$ and the global best particle $gBest^t$, respectively. Crossover operator $C_r(A, B)$ randomly selects two locations in particle A (*i.e.*, the particle to be updated), and then replaces the server-order segment between the two locations of A with the same interval in particle B (*i.e.*, $pBest_i^t$ or $gBest^t$). The crossover operation of the *personal cognition* and *social cognition* are shown as the red rectangle in Figure 3.

Acceleration coefficients c_1 and c_2 denote the crossover probability during the update for particles in this study, and their value is set according to [30]. Note that c_1^s, c_2^s are the start value of c_1, c_2, and c_1^e, c_2^e are the end value of c_1, c_2.

Algorithm 1: Offloading results corresponding a particle

```
Input: $\boldsymbol{U}, \boldsymbol{T}, P_i^t$
Output: $t_i^{aver}\left(P_i^t\right)$
1  for each $\boldsymbol{T}_i \in \boldsymbol{T}$ do
2      waiting queue = $\emptyset$ ;
3      for each $m_{i,j} \in \boldsymbol{T}_i$ do
4          if $\boldsymbol{M}_{i,j}^P = \emptyset$ then
5              Append $m_{i,j}$ to waiting queue;
6          end
7      end
8      while waiting queue $\neq \emptyset$ do
9          Execute method $m_{i,\tilde{j}}$ with the highest
             execution priority in waiting queue;
10         Remove $m_{i,\tilde{j}}$ from waiting queue;
11         Update the related parameters and status of
             $m_{i,\tilde{j}}$ based on Formulas (6)-(9);
12         for each $m_{i,l} \in \boldsymbol{M}_{i,\tilde{j}}^S$ do
13             if all method in $\boldsymbol{M}_{i,l}^S$ are executed then
14                 Append $m_{i,l}$ to waiting queue;
15             end
16         end
17     end
18 end
19 Calculate $t_i^{aver}\left(P_i^t\right)$ based on Formula (11);
20 return $t_i^{aver}\left(P_i^t\right)$
```

The encoding and update schemes for the order part $\varepsilon_{i,j}^t$ in a location of a particle are important for the algorithm efficiency. In this study, the floating encoding mechanism [31] is redesigned, which guarantees that the order value is different from each other in a particle. The related phases about the encoding and update of the order part in a particle mainly include *Initialization*, *Crossover operator*, and *Mutation operator*.

- *Initialization*: With the traditional scheme [27], there are ϑ different values corresponding to the ϑ order values in a particle. There is a one-to-one correspondence between them. With the floating encoding scheme, we construct a library with $1.5 \cdot \vartheta$ different values. The ϑ order values are extracted from this library, which also guarantees that the order value is different from each other in a particle.
- *Mutation operator*: With the traditional scheme [27], mutation operator randomly selects a location *ind* in a particle, and changes the corresponding order value to a value in the range of ϑ order values. Note that the order value in a particle is different from each other, so the algorithm has to take more time to scan the particle, and finds the location with the same order. Then it changes the order value of such location to the original value of location *ind*.
- *Crossover operator*: With the traditional scheme [27], crossover operator randomly selects two locations in a particle, and then replaces the server-order segment between the two locations with the same interval in another particle. Note that the order value of the crossover

particle should be adjusted.

D. Offloading Results Corresponding a Particle

Algorithm 1 shows the time-driven offloading results corresponding a particle for object-oriented applications in a UAV-based edge-cloud environment. The inputs contain all feasible servers $\boldsymbol{U}$, the object-oriented applications $\boldsymbol{T}$, and an encoded particle P_i^t. The output is the average execution time of all object-oriented application from the offloading results corresponding to particle P_i^t. The algorithm first initializes a waiting queue and adds all entry methods from application $\boldsymbol{T}_i$ to the waiting queue (line 2-7). Then, it judges whether all the methods in application $\boldsymbol{T}_i$ are executed completely. When the waiting queue is not empty, the first method in the waiting queue is executed and removed. The related parameters and status of such method is updated based on the Formulas (6)-(9) (line 9-11). After executing the first method, the waiting queue needs to be updated (line 12-16). Finally, the average execution time of all object-oriented application from the offloading results corresponding to particle P_i^t is calculated and returned (line 19-20).

E. Detailed Algorithm Process

The detailed process of PGFE algorithm is described as follows.

- *Step 1*: The relevant parameters of the PGFE, such as initial population size Ψ, the maximum number of iterations Θ, inertia weight ω, are initialized. Then the initial population is randomly generated.
- *Step 2*: According to the time-driven offloading results corresponding to a particle (Algorithm 1), the fitness of each particle based on Formula (15) is calculated. Each particle is set as its own personal best particle, and the particle with the best fitness in the initial population is set as the global best particle in the current generation.
- *Step 3*: Each particle is updated based on Formula (16), and the fitness of each new particle is recalculated.
- *Step 4*: If the fitness of the updated particle is better than its personal best particle, the updated particle is set as its own personal best particle. Otherwise, go to *Step 6*.
- *Step 5*: If the fitness of the updated particles is better than the global optimal particles, the updated particle is set as the global best particle.
- *Step 6*: If the stop condition is not satisfied, it returns to *Step 3*. Otherwise, the algorithm is determined.

V. Experiment Results and Analysis

To validate the effectiveness of the offloading strategy based on PGFE for object-oriented applications in a UAV-based edge-cloud environment, experimental evaluations are carried out. In particular, the following research questions (**RQs**) are checked with the experiments conducted:

- **RQ1**: How effective is our offloading strategy based on PGFE compared with the benchmark strategies? (Section V-C)
- **RQ2**: PGFE is a meta-heuristic algorithm, so what is the optimal maximum number of iterations, Θ, according to its performance trade-off? (Section V-D)
- **RQ3**: In running offloading with respect to given different calculated amounts of methods, does PGFE help reduce the average execution time of object-oriented applications in comparison to other algorithms? (Section V-E)

A. Basic Experimental Setup

All experiments are run on the Win10 64-bit operating system with an Intel(R) Core(TM) i5-8500 CPU at 3.00 GHz and 16GB RAM. Both PGFE and all compared algorithms are implemented in Python 3.7. The related parameters of PGFE are set according to the common practice in [29], such that $\Psi = 100$, $\omega = 0.5$, $c_1^s = 0.9$, $c_1^e = 0.2$, $c_2^s = 0.3$, and $c_2^e = 0.8$.

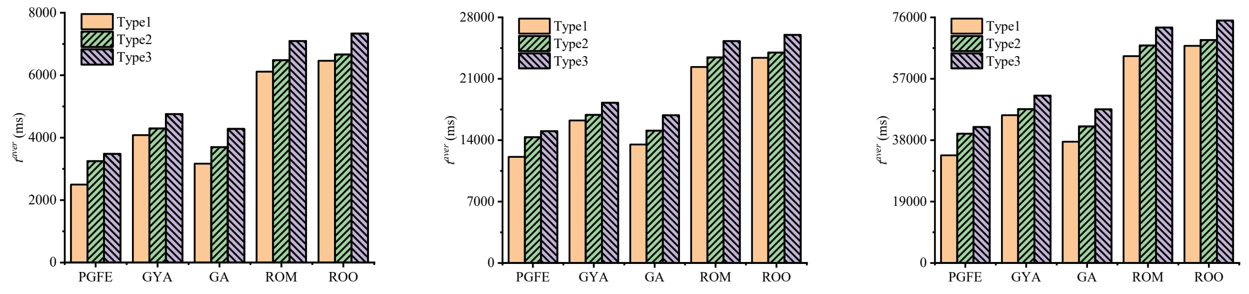
The UAV-based edge-cloud environment is a three-dimensional space, and the unit of measurement in each dimension is meter (m). For simplicity, the locations of all servers are fixed. There are 25 terminals in the space of 1000 m $\times$ 1000 m $\times$ 0 m with uniform distribution, and their computing capacity p_i is 1 gigacycles/s. There are 3 UAVs in the locations of (0 m, 0 m, 100 m), (0 m, -250 m, 100 m), and (0 m, 250 m, 100 m). Their computing capacity is 4 gigacycles/s, 10 gigacycles/s, and 6 gigacycles/s, respectively. There is only one cloud server in the location of (0 m, 0 m, 3000 m) with the computing capacity of 20 gigacycles/s. We assume that all wireless channels are dominated by LoS links [32], so the Rayleigh Fading factor $\beta_{i,j}$ and the transmission power $p_{i,j}$ between server s_i and server s_j are set to -50 dm and 0.5 W, respectively. In addition, the channel bandwidth B, channel noise power σ^2, and path attenuation index α are set to 20 MHz, 2×10^{-13} W, and 2, respectively.

We assume that each terminal generates an object-oriented application concurrently, so there are 25 object-oriented applications in our experiments, and the generation time of each object-oriented application t_i^{gen} is 0s. We conduct our experiments with the help of three types of partly synthetic applications (*i.e.*, Type1, Type2, and Type3). Each type of object-oriented application has a different number of methods, number of objects, data dependencies, and structures. There are three different sizes for each type of object-oriented application (*i.e.*, Small, Medium, and Large), whose detailed information for our experiments are recorded in github.com [33]. Note that, the types and sizes of 25 object-oriented applications are the same in a single test experiment.

B. Basic Compared Algorithms

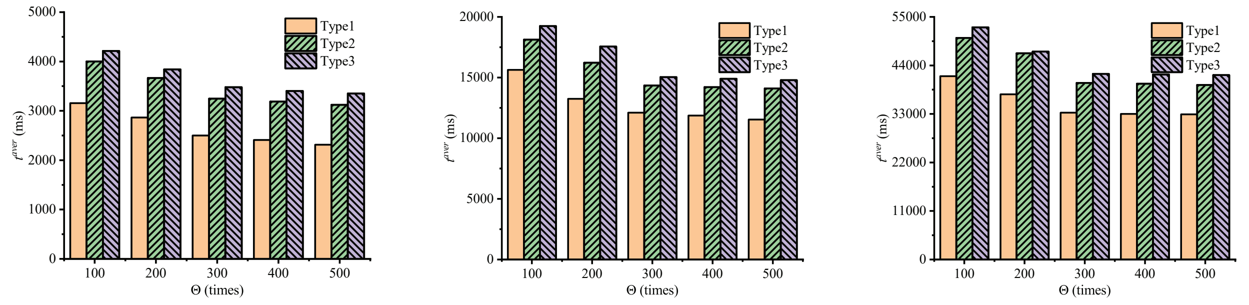
To verify the superiority of the offloading strategy based on PGFE, the following basic compared algorithms are designed.

- Random offloading for methods (ROM): ROM randomly offloads methods on different servers. It doesn't consider that an object in an application contains many methods. Therefore, there may be object migrations.
- Random offloading for objects (ROO): ROO randomly offloads objects on different servers. The object is regarded as the smallest offloading unit. Therefore, there are no object migrations.



(a) Small. (b) Medium. (c) Large.

Fig. 4. The average execution time of three type applications with different sizes through different offloading strategies.



(a) Small. (b) Medium. (c) Large.

Fig. 5. The average execution time of three type applications with different sizes through the offloading strategy based on PGFE with different maximum number of iterations.

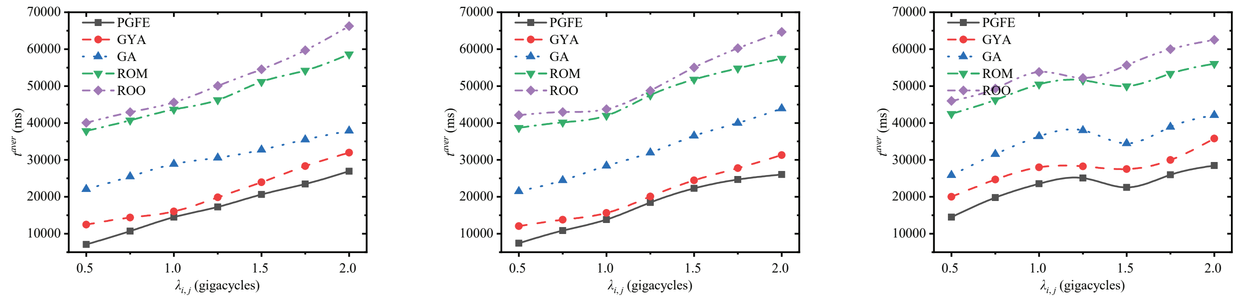
- Greedy algorithm (GYA): GYA offloads each method on the servers with the shortest execution time.
- Genetic algorithm (GA): GA employs the same order-server encoding strategy as PGFE. Following the classical update strategy of GA, the population is updated through binary tournament selection, two-point crossover operator, and mutation operator [34]. Also, this algorithm takes the elitist preservation mechanism and completely copies the elite individuals (with the most significant fitness value in the population) into the next generation. The crossover probability and mutation probability of GA are set as 0.7 and 0.1, respectively, as commonly done in the literature.

C. ***RQ1.*** *Efficiency of our offloading strategy based on PGFE*

Figure 4 shows the average execution time of three application types with different sizes through different offloading strategies. Note that the maximum number of iterations for PGFE Θ is set to 300. The detailed reasons can be explained in Section V-C. As the size of object-oriented applications increases, the number of methods in an application increases, which leads to an increase in the total calculated amount of methods and the total amount of data transmission between method executions in the application. Therefore, the average execution time of object-oriented applications increases significantly as the size of such applications increases. From the experimental results, we find that PGFE has obvious performance advantages compared with several other benchmark strategies. For the object-oriented applications with Small sizes, our proposed PGFE can reduce the average execution time of such applications by 29.98%, 17.33%, 53.33%, and 55.07%, compared with GYA, GA, ROM, and ROO, respectively. For the object-oriented applications with Medium sizes, our proposed PGFE can reduce the average execution time of such applications by 19.41%, 8.70%, 41.72%, and 43.57%, compared with GYA, GA, ROM, and ROO, respectively. For the object-oriented applications with Large sizes, our proposed PGFE can reduce the average execution time of such applications by 20.63%, 9.32%, 43.58%, and 45.43%, compared with GYA, GA, ROM, and ROO, respectively.

GYA considers the execution time of each method in an application from a local perspective, and ignores the impact of data transmission on the offloading result, so its optimization effect is limited. GA has a certain optimization effect, but its update strategy makes it easy to fall into a local optimum, which results in failing to find an optimal solution for our offloading problem. ROM uses the method as the granularity for random offloading, which is a more fine-grained algorithm and has a better performance compared with ROO. However, ROM and ROO are both random algorithms, and their performance is relatively limited. PGFE uses the method as the granularity for task offloading, which introduces the randomly two-point crossover and mutation operator of genetic algorithm to effectively avoid converging on local optima.

In addition, the Type2 applications with Medium sizes are chosen as the representative applications in the rest of our experiments.



(a) Uniform distribution. (b) Normal distribution. (c) Pareto distribution.

Fig. 6. The average execution time of Type2 applications with Medium sizes based on various calculated amount of methods through different strategies.

D. ***RQ2.*** *Optimal Maximum Number of Iterations for PGFE According to Its Performance Trade-Off*

Figure 5 shows the average execution time of three application types with different sizes through the offloading strategy based on PGFE with the different maximum number of iterations. As the maximum number of iterations for PGFE increases, the average execution time of object-oriented applications gradually decreases. In the process of increasing the maximum number of iterations of PGFE from 50 to 300, the average execution time of object-oriented applications is significantly reduced. However, the average execution time of object-oriented applications still maintains a decreasing trend with an extremely small magnitude during increasing the maximum number of iterations from 300 to 500.

When the maximum number of iterations is 300, PGFE has found a solution close to the optimal solution for our problem. Increasing the maximum number of iterations greater than 300 has little meaning to improve the quality of the offloading results based on PGFE. PGFE is essentially a meta-heuristic searching algorithm. When the maximum number of iterations tends to infinity, a solution that is infinitely close to the optimal solution can be obtained [35]. However, the increased maximum number of iterations of PGFE will generate additional algorithm execution time. Infinitely increasing the maximum number of iterations of PGFE is not advisable when dealing with actual problems. Under the current experimental configurations, the running time of the offloading strategy based on PGFE is about 20 seconds while the maximum number of iterations is 300. With the improvement of hardware performance for the experiments, the running time of our strategy will be greatly reduced. The UAV-based edge-cloud environment may change, especially the location of UAVs. The running time of the offloading strategy based on PGFE can be better adapted to such environment. Therefore, the maximum number of iterations of PGFE is set to 300 in the rest of our experiments.

E. ***RQ3.*** *Impact of Calculated Amount of Methods on Offloading Performance*

Figure 6 shows the average execution time of Type2 applications with Medium sizes based on various calculated amount of methods through different strategies. The calculated amount of each method in an application is set to satisfy different probability distributions (*i.e.*, uniform distribution $U(a = x, b = x + 0.2)$, normal distribution $N(\mu = x, \sigma^2 = 0.2^2)$ and Pareto distribution $P(x_{min} = x, k = 1.5)$), where x is the abscissa of the corresponding subgraph in Figure 6. The offloading results are measured as the average of 50 repeated experiments, whose details are shown in [33]. It can be seen that the greater the calculated amount of methods in an object-oriented application, the longer the average execution time of object-oriented applications. In addition, compared with the offloading results based on the benchmark algorithms, the offloading result based on PGFE has a shorter average execution time of object-oriented applications.

In Figure 6(a) with the calculated amount of methods following a uniform distribution, our proposed PGFE can reduce the average execution time of object-oriented applications by 19.72%, 45.56%, 64.99%, and 67.61%, compared with GYA, GA, ROM, and ROO, respectively. When the calculated amount of methods follows a normal distribution in Figure 6(b), the offloading results are similar to that with a uniform distribution in Figure 6(a). In Figure 6(c) with the calculated amount of methods following the Pareto distribution, the curve trend of the average execution time is not as smooth as that with the uniform distribution in Figure 6(a) and the normal distribution in Figure 6(b). This is because that the Pareto distribution has a long tail effect [36]. The experimental results show that no matter how the calculated amount of methods changes, the performance of PGFE is significantly better than other benchmark algorithms.

VI. CONCLUSIONS AND FUTURE WORK

A computation offloading strategy based on PGFE for object-oriented applications in a UAV-based edge-cloud environment is proposed, which aims to reduce the average execution time of object-oriented applications mainly caused by method executions and data transmission between them, while considering the 'encapsulation' characteristic of objects. The experimental results show that the proposed strategy can reduce the average execution time of object-oriented applications by 23.34%, 11.78%, 46.21%, and 48.02%, compared with GYA, GA, ROM, and ROO benchmark solutions, respectively. In addition, we discover that the data transmission caused by object migration, has a significant impact on the average execution time when offloading object-oriented applications.

In the future, the impact of environmental fluctuations (*i.e.*, network delay, bandwidth fluctuation, and server failure) on offloading strategies will be considered, and the current strategy will be improved to adapt to the fluctuating environment. In addition, each method has its unique characteristic, and the execution cost for each method on servers is different. Hence, we will comprehensively design a system cost model, which takes into account the execution cost of servers executing various methods.

REFERENCES

[1] H. Zeng, H. Zhang, J. Chen, and W. Yang, "UAV target detection algorithm using GNSS-based bistatic radar," in *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, 2019, pp. 2167–2170.

[2] Pajares and Gonzalo, "Overview and current status of remote sensing applications based on unmanned aerial vehicles (UAVs)," *Photogrammetric Engineering & Remote Sensing*, vol. 81, no. 4, pp. 281–330, 2015.

[3] J. Park, S. Choi, I. Ahn, and J. Kim, "Multiple UAVs-based surveillance and reconnaissance system utilizing IoT platform," in *International Conference on Electronics, Information, and Communication (ICEIC)*, 2019, pp. 1–3.

[4] C. Wang and H. Lan, "An expressway based TSP model for vehicle delivery service coordinated with truck + UAV," in *IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2019, pp. 307–311.

[5] R. Lee and Y. Chen, "Dual UAV PM2.5 pollution source tracking system," in *IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)*, 2019, pp. 384–386.

[6] Y. Cai, K. Guan, E. Nafziger, G. Chowdhary, B. Peng, Z. Jin, S. Wang, and S. Wang, "Detecting in-season crop nitrogen stress of corn for field trials using UAV- and cubesat-based multispectral sensing," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 12, no. 12, pp. 5153–5166, 2019.

[7] T. Zhang, Y. Xu, J. Loo, D. Yang, and L. Xiao, "Joint computation and communication design for UAV-assisted mobile edge computing in IoT," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5505–5516, 2020.

[8] Z. Xu, W. Liang, M. Jia, M. Huang, and G. Mao, "Task offloading with network function requirements in a mobile edge-cloud network," *IEEE Transactions on Mobile Computing*, vol. 18, no. 11, pp. 2672–2685, 2019.

[9] E. El Haber, T. M. Nguyen, and C. Assi, "Joint optimization of computational cost and devices energy for task offloading in multi-tier edge-clouds," *IEEE Transactions on Communications*, vol. 67, no. 5, pp. 3407–3421, 2019.

[10] X. Chen, J. Chen, B. Liu, Y. Ma, and H. Zhong, "AndroidOff: Offloading android application based on cost estimation," *Journal of Systems and Software*, vol. 158, p. 110418, 2019.

[11] X. Chen, S. Chen, Y. Ma, B. Liu, and Y. Zhang, "An adaptive offloading framework for android applications in mobile edge computing," *Science China*, vol. 62, no. 08, pp. 114–130, 2019.

[12] X. Zhang, J. Zhang, J. Xiong, L. Zhou, and J. Wei, "Energy-efficient multi-UAV-enabled multiaccess edge computing incorporating NOMA," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 5613–5627, 2020.

[13] J. Hu, M. Jiang, Q. Zhang, Q. Li, and J. Qin, "Joint optimization of UAV position, time slot allocation, and computation task partition in multiuser aerial mobile-edge computing systems," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 7, pp. 7231–7235, 2019.

[14] D. Callegaro and M. Levorato, "Optimal computation offloading in edge-assisted UAV systems," in *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–6.

[15] B. Liu, W. Zhang, W. Chen, H. Huang, and S. Guo, "Online computation offloading and traffic routing for UAV swarms in edge-cloud computing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 8, pp. 8777–8791, 2020.

[16] T. Bai, J. Wang, Y. Ren, and L. Hanzo, "Energy-efficient computation offloading for secure UAV-edge-computing systems," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 6, pp. 6074–6087, 2019.

[17] C. Xian, Y. Lu, and Z. Li, "Adaptive computation offloading for energy conservation on battery-powered systems," in *International Conference on Parallel and Distributed Systems (ICPDS)*, 2007, pp. 1–8.

[18] J. Kwak, Y. Kim, J. Lee, and S. Chong, "DREAM: Dynamic resource and task allocation for energy minimization in mobile cloud systems," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 12, pp. 2510–2523, 2015.

[19] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal & Information Processing Over Networks*, vol. 1, no. 2, pp. 89–103, 2015.

[20] F. Zhou, Y. Wu, R. Q. Hu, and Q. Yi, "Computation rate maximization in UAV-enabled wireless-powered mobile-edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 9, pp. 1927–1941, 2018.

[21] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 1, no. 1, pp. 623–656, 1948.

[22] B. Lin and C. Wu, "Mathematical modeling of the human cognitive system in two serial processing stages with its applications in adaptive workload-management systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 1, pp. 221–231, 2011.

[23] M. Foruhandeh, N. Tadayon, and S. Aïssa, "Uplink modeling of k-tier heterogeneous networks: A queuing theory approach," *IEEE Communications Letters*, vol. 21, no. 1, pp. 164–167, 2017.

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

[25] K.-H. Loh, B. Golden, and E. Wasil, "Solving the maximum cardinality bin packing problem with a weight annealing-based algorithm," *Operations Research/Computer Science Interfaces*, pp. 147–164, 2009.

[26] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *International Conference on Neural Networks (ICNN)*, 1995, pp. 1942–1948.

[27] B. Lin, F. Zhu, J. Zhang, J. Chen, X. Chen, N. N. Xiong, and J. Lloret Mauri, "A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 4254–4265, 2019.

[28] B. Lin, Y. Huang, J. Zhang, J. Hu, X. Chen, and J. Li, "Cost-driven offloading for DNN-based applications over cloud, edge, and end devices," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5456–5466, 2020.

[29] Y. Shi and B. Obaiahnahatti, "A modified particle swarm optimizer," in *IEEE Conference on Evolutionary Computation (ICEC)*, 1998, pp. 69 – 73.

[30] L. Cui, J. Zhang, L. Yue, Y. Shi, H. Li, and D. Yuan, "A genetic algorithm based data replica placement strategy for scientific applications in clouds," *IEEE Transactions on Services Computing*, vol. 11, no. 4, pp. 727–739, 2018.

[31] J. Matos, R. P. Faria, I. B. Nogueira, J. M. Loureiro, and A. M. Ribeiro, "Optimization strategies for chiral separation by true moving bed chromatography using particles swarm optimization (PSO) and new parallel PSO variant," *Computers & Chemical Engineering*, vol. 123, pp. 344 – 356, 2019.

[32] Y. Zeng and R. Zhang, "Energy-efficient UAV communication with trajectory optimization," *IEEE Transactions on Wireless Communications*, vol. 16, no. 6, pp. 3747–3760, 2017.

[33] J. Zhang, "Taskhub," [EB/OL], 2020, https://github.com/JamesZJS/ApplicationHub.git.

[34] L. Cui, J. Zhang, L. Yue, Y. Shi, H. Li, and D. Yuan, "A genetic algorithm based data replica placement strategy for scientific applications in clouds," *IEEE Transactions on Services Computing*, vol. 11, no. 4, pp. 727–739, 2018.

[35] D. Li, R. Zhan, D. Zheng, M. Li, and I. Kaku, "A hybrid evolutionary hyper-heuristic approach for intercell scheduling considering transportation capacity," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 1072–1089, 2016.

[36] A. Xu and Y. Tang, "Bayesian analysis of pareto reliability with dependent masked data," *IEEE Transactions on Reliability*, vol. 58, no. 4, pp. 583–588, 2009.