An Improved Genetic Algorithm with Chromosome Replacement and Rescheduling for Task Offloading

Hui Fu, Guangyuan Li, Fang Han*, Bo Wang
Faculty of Engineering, Huanghe Science and Technology College, Zhengzhou, China 450006

***Abstract*—End-Edge-Cloud Computing (EECC) has been applied in many fields, due to the increased popularity of smart devices. But the cooperation of end devices, edge and cloud resources is still challenge for improving service quality and resource efficiency in EECC. In this paper, we focus on the task offloading to address the challenge. We formulate the offloading problem as mixed integer nonlinear programming, and solve it by Genetic Algorithm (GA). In the GA-based offloading algorithm, each chromosome is the code of a offloading solution, and the evolution is to iteratively search the global best solution. To improve the performance of GA-based task offloading, we integrate two improvement schemes into the algorithm, which are the chromosome replacement and the task rescheduling, respectively. The chromosome replacement is to replace the chromosome of every individual by its better offspring after every crossing, which substitutes the selection operator for population evolution. The task rescheduling is rescheduling each rejected task to available resources, given offloading solution from every chromosome. Extensive experiments are conducted, and results show that our proposed algorithm can improve upto 32% user satisfaction, upto 12% resource efficiency, and upto 35.3% processing efficiency, compared with nine classical and up-to-date algorithms.**

***Keywords*—Genetic algorithm; task offloading; task scheduling; edge computing; cloud computing**

I. Introduction

Smart devices, such as smartphones, Internet of Things (IoT) devices, drones, and so on, have become ubiquitous in our life and their number continues to grow rapidly [1], as communications and information technology advance and our quality of life improves. Unfortunately, due to their small physical space, most devices have limited resource capacity and battery life. As a result, devices frequently lack the processing power required by user requests, especially for complex applications like facial recognition and intelligent driving, which become more and more common.

To address the above issue, several works make use of cloud computing, which provides "infinite" computing resources, to extend the processing capacity of devices [2], [3]. However, cloud computing has poor network performance because it typically provides services via a Wide Area Network (WAN), such as the Internet. To address this issue, edge computing brings a few computing resources (edge servers) close to devices to provide low latency services [4], [5], [6]. Combining advantages of end devices, edge servers and cloud, end-edge-cloud computing (EECC) has attracted much attention from both academia and industry, as it can effectively and efficiently provide various services to end users [7], [8], [9].

*Corresponding authors.

In EECC environments, it is challenge to efficiently utilize the collaboration of devices, edge servers and cloud. To address the challenge, several works have designed task offloading or scheduling algorithms for EECC, to improve service performance or/and resource efficiency. The task offloading is to decide the computing node for each task's processing. Existing works have made some assumptions to simplify the offloading problem for EECC, which limits their application scope. For example, some works ignored the heterogeneity between edge and cloud resources, which can lead to resource inefficiency [10], [11]. Some works didn't exploit the resource capacity of end devices, even though many modern devices are equipped with a wealth of hardware resources, and thus wasted zero-delay local resources for task processing.

There are mainly two categories algorithms used for task offloading, heuristics and meta-heuristics. Heuristics exploit some local optimum search strategies tailored to the specific problem. Heuristics generally have rapid solving processes but limited performances. In contrast, meta-heuristics are general problem solvers. Meta-heuristics apply both local searches and global searches, inspired by natural and social rules. Usually, compared with heuristics, meta-heuristics can achieve better performance, but cost more time.

Therefore, in this paper, we exploit hybridization of heuristics and meta-heuristics, to exploit their complementary strengths for the task offloading in EECC, considering the resource heterogeneity. Even though some works have proposed hybrid heuristic offloading algorithms, most of them simply perform two or more algorithms sequentially, leading to a poor performance of hybridization. Specifically, we use genetic algorithm (GA) due to its representativeness and extensive application. GA has powerful global search ability, but slow convergence sometimes. To make up for this shortcoming, we propose to use chromosome replacement instead of selection operator for GA. To improve the performance of offloading solutions decoded from chromosomes, we reschedule failed tasks by a heuristic algorithm. The contributions of this paper can be summarized as follows:

- The task offloading problem of EECC is formulated into a mixed integer nonlinear programming problem (MINLP), with deadline constraints. The optimization objectives are maximizing the finished task number and the overall resource utilization, which are commonly used for quantifying user satisfaction and resource efficiency, respectively.
- A task offloading algorithm is proposed based on GA and first fit heuristic scheduling (FF). The proposed algorithm uses an integer coding approach for mapping

between task offloading solutions and chromosomes. GA is employed for searching the global best solution. To improve the quality of task offloading solutions, FF is used to reschedule failed tasks to available resources in EECC, for every offloading solution. To speed up the convergence of GA, the selection operator is replaced by a replacement operator that replaces every chromosome with its better offspring produced by crossover.

- Extensive simulated experiments are conducted for evaluating the performance of our proposed algorithm. Simulation parameters are set referring to related works. Experiment results verify that our proposed algorithm can finish more tasks than nine of classical and up-to-date offloading algorithms. The efficiencies of the replacement and the task rescheduling are also verified by the results.

The content below is organised as follows. Section II formulates the task offloading problem in EECC. Section III illustrates our proposed offloading algorithm. Section IV evaluates the performance of the proposed algorithm. Section V presents the works related to task offloading for EECC. Section VI concludes this paper.

II. Problem Statement

A. Resource and Task Model

In this paper, we consider the EECC system consisting of D end devices, E edge servers (ES), and V cloud servers (CS). We use $s_i, 1 \leq i \leq D+E+V$ to represent these computing nodes, where devices include $s_i, 1 \leq i \leq D$, ES are $s_i, D+1 \leq i \leq D+E$, and CS are $s_i, D+E+1 \leq i \leq D+E+V$. For computing node s_i, there are n_i computing cores each with g_i capacity. The network connection between two computing nodes, say s_i and s_j, is represented by constants $b_{i,j}$ which is its data transfer rate. If there is no connection between s_i and $s_{i'}$, $b_{i,i'} = 0$. For each computing node, there is no data transmission delay within it, i.e., $b_{i,i} = +\infty, 1 \leq i \leq D+E+V$.

T tasks ($t_k, 1 \leq k \leq T$) are launched by D devices. Binary constants $o_{i,k}, 1 \leq i \leq D, 1 \leq k \leq T$ are used to indicate the ownerships of these tasks, where $o_{i,k} = 1$ means t_k is launched by s_i, and $o_{i,k} = 0$ means not. Task t_k has c_k computing size, i.e., it requires c_k computing resource for its processing. The input data amount of t_k is a_k. In this paper, we ignore the transmission delay of the output data, because the output data amount usually is very small [12]. The deadline of t_k is d_k, which means t_k must be finished before d_k. For every task, if its deadline constraint cannot be satisfied, it will be rejected, because there will be no profit for processing the task.

A task offloading solution is the mapping/assignments of tasks to computing cores for their processing, which can be represented by a set of binary variables $x_{i,j,k}$, as shown in Eq. 1. $x_{i,j,k}$ is 1 if t_k is assigned to jth core in computing node s_i, and 0 otherwise. In this paper, we consider the resource granularity as computing core instead of computing node, because considering fine granularity of resources helps to improve the resource efficiency [13].

$$x_{i,j,k} = \begin{cases} 1, & \text{if } t_k \text{ is assigned to } j\text{th core in } s_i \\ 0, & \text{else} \end{cases}, \quad 1 \leq i \leq D+E+V, 1 \leq j \leq n_i 1 \leq k \leq T. \tag{1}$$

For each task, it can be only assigned to one core for its processing. In this paper, we don't consider to use the redundant execution for the performance improvement due to its huge resource costs. Thus, Eq. 2 holds.

$$\sum_{i=1}^{D+E+V} \sum_{j=1}^{n_i} x_{i,j,k} \leq 1, 1 \leq k \leq T. \tag{2}$$

And when t_k is accepted and processed by a computing core, $\sum_{i=1}^{D+E+V} \sum_{j=1}^{n_i} x_{i,j,k} = 1$. When t_k is rejected, $\sum_{i=1}^{D+E+V} \sum_{j=1}^{n_i} x_{i,j,k} = 0$. Then, the number of accepted tasks can be achieved by Eq. 3.

$$N = \sum_{k=1}^{T} \sum_{i=1}^{D+E+V} \sum_{j=1}^{n_i} x_{i,j,k}. \tag{3}$$

B. Task Processing Model

For a task assigned to a core, its computing can be started only when its input data transfer finishes and the core is available. Then Eq. (4) must be satisfied, where ft_k^A and st_k are respectively the completion time of data transfer and the start time of computing for t_k.

$$ft_k^A \leq st_k, 1 \leq k \leq T. \tag{4}$$

When t_k is assigned to jth core in the computing node s_i, its computing consumes c_k/g_i time. Its input data is transferred from its device ($s_{i'}$ where $o_{i'} = 1$) to s_i. Then the data transfer rate is $\sum_{i'}^{D}(o_{i',k} \cdot b_{i',i})$, and the transfer time is $a_k / \sum_{i'}^{D}(o_{i',k} \cdot b_{i',i})$. Therefore, for each task, the start time and the finish time of data transfer and computing satisfy constraints is Eq. (5) and (6), where st_k^A and ft_k represent the start time of t_k's input data transfer and the finish time of its computing, respectively. Noticing that Eq. (5)–(6) also hold for rejected tasks, as both sides of inequality operators are 0 for these tasks. Then, the deadline constraints can be formulated as Eq. (7).

$$st_k^A + \frac{a_k}{\sum_{i=1}^{D+E+V} \sum_{j=1}^{n_i} (x_{i,j,k} \cdot \sum_{i'}^{D}(o_{i',k} \cdot b_{i',i}))} \leq ft_k^A, \quad 1 \leq k \leq T. \tag{5}$$

$$st_k + \frac{c_k}{\sum_{i=1}^{D+E+V} \sum_{j=1}^{n_i} (x_{i,j,k} \cdot g_i)} \leq ft_k, 1 \leq k \leq T. \tag{6}$$

$$ft_k \leq d_k, 1 \leq k \leq T. \tag{7}$$

When multiple tasks are assigned to one computing core, they cannot be computed simultaneously. There are two cases

for the computing order of two tasks (t_k and $t_{k'}$) in every computing core. If t_k is computing before $t_{k'}$, $ft_k \leq st_{k'}$, and otherwise, $ft_{k'} \leq st_k$. Therefore, Eq. (8) formulates the exclusiveness of tasks' computing in every computing node.

$$x_{i,j,k} \cdot x_{i,j,k'} \cdot (st_{k'} - ft_k) \cdot (st_k - ft_{k'}) \leq 0, \quad 1 \leq k, k' \leq T. \tag{8}$$

For each computing core, its occupied time is the latest finish time of tasks assigned to it, which is $\tau_{i,j} = \max_{k=1}^{T}(x_{i,j,k} \cdot ft_k)$ for jth core of s_i. The occupied time of a computing node is the maximal occupied time of its cores, which is $\tau_i = \max_{j=1}^{n_i} \max_{k=1}^{T}(x_{i,j,k} \cdot ft_k)$ for s_i. Thus, the amount of occupied computing resources is $\tau_i \cdot n_i \cdot g_i$ on s_i. While, the resources effectively use for computing tasks are $R = \sum_{k=1}^{T} \sum_{i=1}^{D+E+V} \sum_{j=1}^{n_i}(x_{i,j,k} \cdot c_k)$ in overall system. Then, the overall computing resource can be calculated by Eq. (9).

$$U = \frac{R}{\sum_{i=1}^{D+E+V}(\tau_i \cdot n_i \cdot g_i)}. \tag{9}$$

C. Task Offloading Problem Model

Now, based on above formulations, the task offloading problem can be modelled as follows:

$$\text{Maximizing } N + U, \tag{10}$$

subject to,

$$\text{Eq. } (1)\text{–}(9). \tag{11}$$

The objective is to maximize the number of accepted tasks plus the overall computing resource utilization. Because the total number of tasks is fixed in EECC system, the maximization of accepted task number is identical to maximizing the accepted ratio which is one commonly used metric for quantifying user satisfaction, service level agreement, and quality of service. As resource utilization is not greater than one, user satisfaction maximization is the major optimization objective in the model. Noticing that U is nonlinear and non-convex, and decision variables including binary ($x_{i,j,k}$) and continuous variables (st_k^A, ft_k^A, st_k, and ft_k), the task offloading problem belongs to mixed integer non-linear programming (MINLP), which is hard to be solved exactly. In fact, the task offloading problem has been proved NP-hard [14]. Therefore, in the next section, we propose a hybrid heuristic algorithm for efficiently solving the offloading problem with polynomial time.

III. Improved Genetic Algorithm for Offloading

In this section, we design an improved genetic algorithm with chromosome replacement and task rescheduling method, GRRS, to solve the task offloading problem presented in the previous section, which is outlined in Algorithm 1.

At first, we design a solution representation method (or encoding/decoding approach) to create the map between task offloading solutions and chromosomes used for search in GA-based algorithms. For GA inspired by Charles Darwin's theory of evolution, the population consists of multiple individuals and is evolved by changing these individuals' chromosomes each with multiple genes. In the solution representation method, there is a one-to-one relationship between genes and tasks in EECC. The value of each gene is integer, which identifies the computing core where the corresponding task is assigned. Thus, the possible value of a gene is 1 to the number of cores which can be used for processing the corresponding task. Then, we get the assignments of all tasks from a chromosome or an individual.

Algorithm 1 GRRS: The genetic offloading algorithm with replacement and rescheduling

Input: The information of tasks, and EECC resources;
Output: A task offloading strategy;
1: Initializing chromosomes of individuals randomly;
2: Evaluating fitness of every individual using Algorithm 2;
3: Initializing the best chromosome (bc) as one with the best fitness in all individuals;
4: **while** the terminal condition is not reached **do**
5: **for** each individual (Y) **do**
6: Crossing Y with another individual which is randomly selected, with a certain probability, and producing two offspring, i.e., new chromosomes;
7: Evaluating the fitnesses of two offspring;
8: Replacing Y's chromosome with the better offspring;
9: **if** Y's chromosome has better fitness than bc **then**
10: Updating bc as Y's chromosome;
11: **end if**
12: mutating Y with a certain probability;
13: Evaluating fitness of Y;
14: Updating bc as done in lines 9–11.
15: **end for**
16: **end while**
17: **return** the offloading strategy decoded from bc by Algorithm 2;

For example, considering an EECC system consisting two devices, two ES, and two CS, where each node has one computing core and each device launches one task. Then, the number of genes is 2, corresponding to these two tasks. The possible value in each dimension is 1 to 5, respectively representing the cores of the device, two ES, and two CS for the corresponding task.

A fitness function is needed to evaluate how goodness of every chromosome/individual. We use the optimization objective (10), $N + U$, as the fitness function of GRRS. The fitness evaluation of every chromosome is given in Algorithm 2. Given a chromosome, it can be easily to achieve a task assignment solution by the solution representation (lines 3–5 in Algorithm 2). To achieve a complete offloading solution, the computing order of tasks assigned to every core needs to be decided. In this paper, we use the most simple algorithm, First Fit (FF), for order decisions, and will study more efficient algorithms to improve the performance of GRRS. With FF order, we can calculate the finish time of each task one-by-one (line 7 in Algorithm 2). If the finish time fits deadline constraint for a task, it is accepted, and otherwise, rejected (lines 8–12 in Algorithm 2). After all tasks are decided to be accepted or rejected, GRRS tries to reschedule every rejected task using FF, to improve the overall user satisfaction (lines 14–22 in Algorithm 2). Now, we achieve a task offloading solution from a chromosome. The accepted task number and the overall resource utilization can be easily calculated based on the offloading solution, and the fitness is achieved for the chromosome.

Algorithm 2 Decoding a chromosome into a task offloading with rescheduling

Input: A chromosome;
Output: A task offloading solution, Ω, and the fitness;
1: $\Omega \leftarrow \phi$; /*the set including assignments of accepted tasks to computing cores*/
2: $\Phi \leftarrow \phi$; /*the set including rejected tasks*/
3: **for** each gene of the chromosome **do**
4: Per-assigning the corresponding task into the computing core identified by the gene value;
5: **end for**
6: **for** each task t **do**
7: Calculating its finish time in the scheme of first fit scheduling on the core (c) to which it is per-assigned;
8: **if** the finish time is earlier than the deadline **then**
9: $\Omega \leftarrow \Omega \cup \{< t, c >\}$; /*deciding to assign the task to the core*/
10: **else**
11: $\Phi \leftarrow \Phi \cup \{t\}$; /*the deadline being violated*/
12: **end if**
13: **end for**
14: **for** each task $t \in \Phi$ /*rescheduling rejected tasks*/ **do**
15: **for** each core c **do**
16: Calculating its finish time as line 7;
17: **if** the finish time is earlier than the deadline **then**
18: $\Omega \leftarrow \Omega \cup \{< t, c >\}$; /*rescheduling t to c*/
19: **break**; /*rescheduling another rejected task*/
20: **end if**
21: **end for**
22: **end for**
23: Calculating overall resource utilization U by Eq. 9;
24: **return** Ω and $|\Omega| + U$; /*the accepted task number $N = |\Omega|$*/

Based on the solution representation and the fitness evaluation, GRRS exploits the main idea of GA to iteratively search for the optimal solution for task offloading, as shown in Algorithm 1. First, GRRS initializes the population, i.e., randomly sets the value of every gene for every individual's chromosome, as done by standard GA. Then, GRRS evaluates the fitness for every initialized chromosome, and sets the best chromosome as the chromosome with the best fitness. After these initialization steps, GRRS uses some operators to evolve individuals by updating their chromosomes (lines 4–16 in Algorithm 1). The evolution procedure is as follows.

First, for every individual, GRRS uses crossover operator to create new chromosomes/offspring. GRRS randomly selects another individual, and performs the crossover operator on them, with a certain possibility (the crossover possibility). To ensure the individual diversity for large-scale offloading problems, GRRS exploits the uniform crossover operator, which swaps values of two chromosomes in every gene location with a certain probability. After crossing an individual, two new chromosomes are produced. For the individual, GRRS evaluates the fitness of its two offspring, and replaces its chromosome by the offspring with better fitness than another one. By such replacement, GRRS can increase the diversity by retaining new produced chromosomes, and speed the convergence rate by transmitting good genes of the better offspring to the next generation. If a new offspring has better fitness than the best chromosome, the best one is uprated as the offspring.

To further enhance exploration ability, GRRS applies the uniform mutation operator on each individual, to increase the diversity by creating new genes. The uniform mutation operator is to change each gene with the mutation possibility for an individual. After mutating an individual, if the new chromosome has better fitness than the best one, the best one is updated as the new one.

GRRS repeats the above evolution procedure until the terminal condition is reached. There are two approaches for the set of terminal condition. One is setting the maximal number of iterations, and another is setting the most times that the fitness of the best chromosome has no (significant) change. After the evolution procedure, GRRS decodes the best chromosome into the task offloading solution, and return it as the global best solution.

In this paper, we focus on the improvement of GA by chromosome replacement and task rescheduling. Undoubtedly, the crossover and mutation operators as well as the parameters have impact on the performance of GA. These opportunities will be studied on our future works.

IV. Performance Evaluation

In this section, we conduct extensive simulated experiments to verify the efficiency of GRRS by comparing with several classical and up-to-date offloading algorithms. The experiment environment is illustrated in Section IV-A, and the results are discussed in Section IV-B.

A. Experiment Environment

In simulated EECC systems, where the simulation parameters are set referring to [18], [21], [23], [15] and reality, there are ten devices, five ES, and ten types of CS. The core number of devices, ES, and CS are set randomly in ranges of [2,8], [4,32], and [1,8], respectively. The computing capacities of each core in every device, ES, and CS are respectively set as [1.8,2.5]GHz, [1.8,3.0]GHz, and [1.8,3.0]GHz, randomly. The network transfer rate between a device and an ES/CS is in the range of [80,120]/[10,20] Mbps. There are 1000 tasks generated, and the device for lunching each task is randomly allocated. The computing resource required by a task is in the range of [0.5, 1.2]GHz, and the input data amount is [1.5, 6]MB. The deadline of every task is set between one and five seconds.

The algorithms used for the performance comparison with GRRS to confirm the performance include FF, FFD, EDF, RAND, GA, GAR, PSO, PSOM, and GAPSO.

- First Fit (FF) iteratively schedules the first task to the first computing node meeting its requirements.
- First Fit Decreasing (FFD) iteratively schedules the task requiring maximal amount of computing resources to the first computing node meeting its requirements.
- Earliest Deadline First (EDF) iteratively schedules the task with the earliest deadline to the first computing node meeting its requirements.
- Random method (RAND) randomly generates a population as done by GRRS, and provides the solution corresponding to the best individual.

- GA [15], [16] uses the uniform crossover, the uniform mutation, and the roulette wheel selection operators for the population evolution.
- GA with replacement (GAR) [17] is same to GRRS without the rescheduling.
- Particle Swarm Optimization (PSO) [18] uses the idea of particle movement. PSO initializes a population with multiple particle (individual), and iteratively moves each particle toward its personal best position and the global best position for the particle position updates (the population evolution).
- PSO with mutation operator (PSOM) [19] added a mutation operator on each particle at the end of each iteration.
- GAPSO [20] first initializes a population, and then sequentially performs GA and PSO on the population evolution.

The performance of above algorithms are evaluated as following, and all performance metrics are better when their values are greater.

- User satisfaction is the experience of users, which has great influence on the profit and the reputation of service providers. In this paper, we use the number of tasks with deadline met, i.e., the accepted task number (N), for the quantification.
- Resource efficiency is the workload processed by a unit of resources, which determines the cost-performance of service provision. The metrics used for measuring the resource efficiency are the computing resource utilization (U, the completed computing size per unit of computing resource) and the data processing efficiency (the processed data amount per unit of computing resource, $\sum_{k=1}^{T}\sum_{i=1}^{D+E+V}\sum_{j=1}^{n_i}(x_{i,j,k}\cdot a_k)/\sum_{i=1}^{D+E+V}(\tau_i\cdot n_i\cdot g_i)$).
- Processing efficiency is the processing speed of a computing system, which is quantified by the completed computing size and the processed data amount by a time unit ($R/\max_i\{\tau_i\}$ and $\sum_{k=1}^{T}\sum_{i=1}^{D+E+V}\sum_{j=1}^{n_i}(x_{i,j,k}\cdot a_k)/\max_i\{\tau_i\}$).

The experiment process are as follows. We first generate a EECC system, and then sequentially measure various performance metrics for all of comparison algorithms and GRRS. For each measured value for every algorithm and every metric, we normalize it by dividing it into that of FF, to focus on the relative performance between different algorithms. These previous experiment steps are repeated more than 100 times, and we report the average value for each metric in the follows. Noticing that, in each measurement, there is a new EECC system generated randomly. Thus, the statistical information of every algorithm in each metric is meaningless without the normalization. GRRS has statistically significant difference with other algorithms in every performance metric.

Besides comparing the performance of GRRS with other offloading algorithms, we verify the efficiency of the task rescheduling, by comparing the performance between GA/PSO/GAPSO/GAR with and without task rescheduling. The results are presented and discussed in section IV-B4.

B. Experiment Results

1) User Satisfaction: Fig. 1 gives the relative number of accepted tasks when applying different task offloading methods, on average. As shown in the figure, GRRS achieves the most accepted tasks, which completes 7.98%–32% more tasks than other algorithms. This verifies that GRRS performs good on the optimization of the user satisfaction. The main reasons are as follows.

For heuristics, FF, FFD, and EDF, the priority order of resources used for task processing is devices, ES, and CS. This can complete more task in low network latency but scarce resources of devices and ES. As shown in experiment results, GRRS accepts 12%–29.4% and 32%–33.7% less tasks than these heuristic algorithms at devices and ES, respectively, as shown in Fig. 2 and Fig. 3. But this can result in some tasks with loose deadline assigned to devices or ES at first. This leads to insufficient resources for processing subsequent tasks with tight deadline, and thus can drastically decrease the number of tasks processed by CS and reduce the overall user satisfaction. As shown in Fig. 4, meta-heuristics process more than 100% more tasks than heuristics by the cloud. As shown in Fig. 2 - 4, GRRS processes not the most tasks in one tier of devices, edges, and cloud. But GRRS has the best overall satisfaction, as shown in Fig. 1. This phenomenon verifies the powerful global search ability of GRRS.

Compared with other meta-heuristic algorithms (GA, GAR, PSO, PSOM, GAPSO), GRRS achieve better performance in optimizing the user satisfaction, as shown in Fig. 1. The main advantages of GRRS is the replacement replacing the selection operator for GA and the rescheduling for improving the quality of the solution corresponded to an individual. GAR can complete more tasks than GA, which verifies that the replacement improves the evolution effectiveness by substituting the selection operator. The improvement of the rescheduling strategy on the task offloading will be illustrated in Section IV-B4.

In addition, we can see that some meta-heuristics (GA, PSO, PSOM, and GAPSO) has poorer performance than heuristics, even though they are designed for pursuing the global best and heuristics are aiming at the local best, in such a large-scale offloading problem. This inspires us that meta-heuristics should be carefully designed for a good performance.

2) Resource Efficiency: Fig. 5 and Fig. 6 show resource utilization and data processing efficiency achieved by different offloading algorithms. From the figure, we can see that heuristics achieve higher resource utilization and higher data processing efficiency than meta-heuristics. This is mainly because that heuristics process tasks using scarce local and edge resources at first. This can provide a good performance for accepted tasks, and a high computing resource efficiency, because there is no or a low latency for data transfer. But with prioritization of device and edge resources, much less tasks can be completed by the cloud resources, leading to a low user satisfaction as illustrated above. Contrary to heuristics, meta-heuristics try to find the global best solution, which can schedule every task to

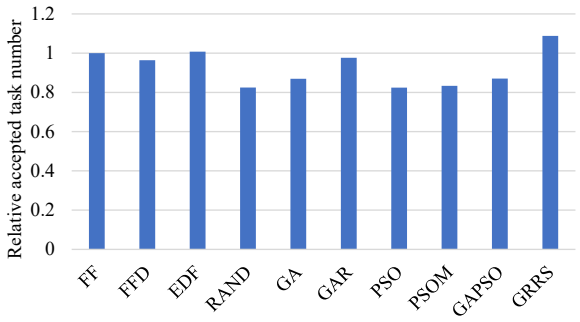

Fig. 1. The relative accepted task number achieved by various task offloading methods in overall.

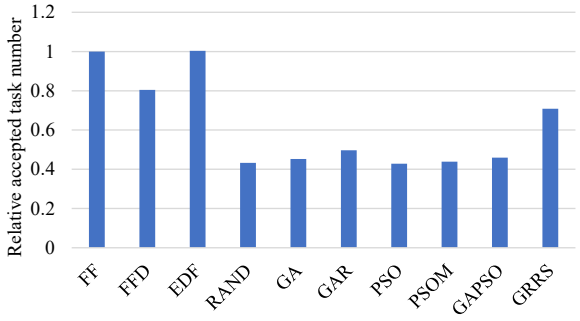

Fig. 2. The relative accepted task number achieved by various task offloading methods in device tier.

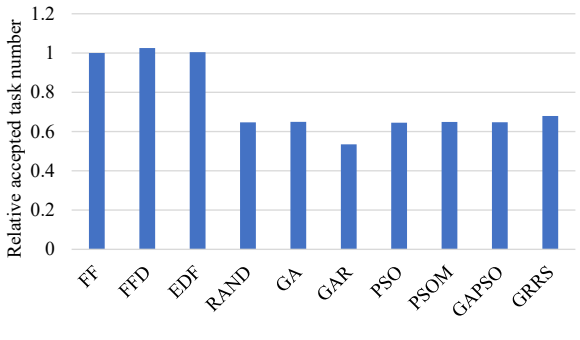

Fig. 3. The relative accepted task number achieved by various task offloading methods in edge tier.

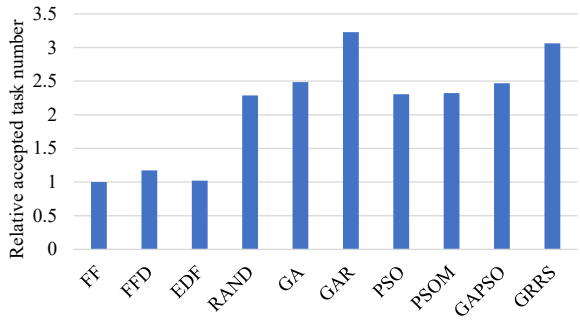

Fig. 4. The relative accepted task number achieved by various task offloading methods in the cloud.

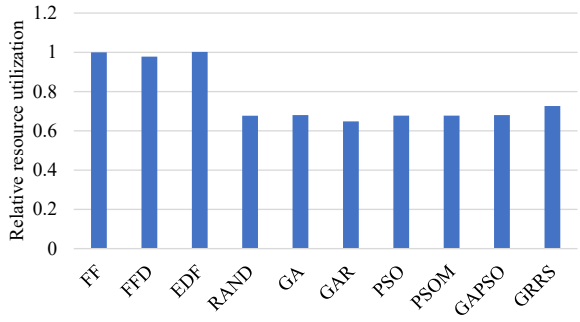

Fig. 5. The relative resource utilization achieved by various task offloading methods.

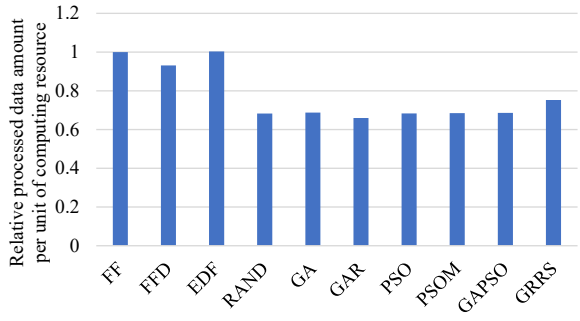

Fig. 6. The relative processed data amount per unit of computing resources achieved by various task offloading methods.

any resource at first. This provides opportunities for processing more tasks in overall. Therefore, GRRS improves the user satisfaction by sacrificing the resource efficiency, compared with heuristics. It is worth it because the user satisfaction usually decides the income and reputation of service providers.

In all of these meta-heuristics, GRRS has the highest resource utilization and data processing efficiency, which are 6.74%–12% and 9.5%–14.1% higher than that of other meta-heuristics, respectively, as shown in Fig. 5 and 6. This demonstrates that GRRS performs good at the optimization of both the user satisfaction and the resource efficiency, and further confirms the high effectiveness of GRRS.

3) Processing Efficiency: Fig. 7 and Fig. 8 present the processing rates or efficiencies in computing and data processing in EECC when applying various task offloading methods. In a distributed system, the processing rate reflects the parallelism, and thus the throughput, which is one of the most used metrics quantifying overall performance. As shown in these two figures, we can see that GRRS has the highest processing rates, which are 7.6%–31.5% and 11.4%–35.3% higher than other methods in the computing and data processing, respectively. This illustrates that GRRS achieves good processing efficiency for EECC systems.

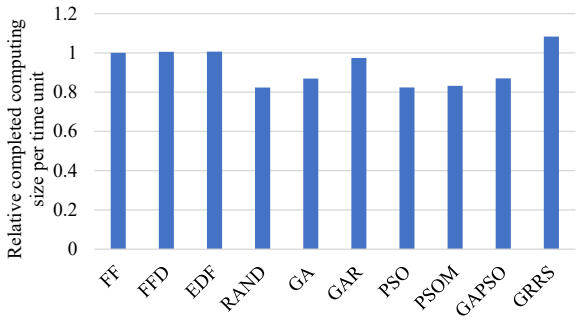

Fig. 7. The relative computing efficiency by various task offloading methods.

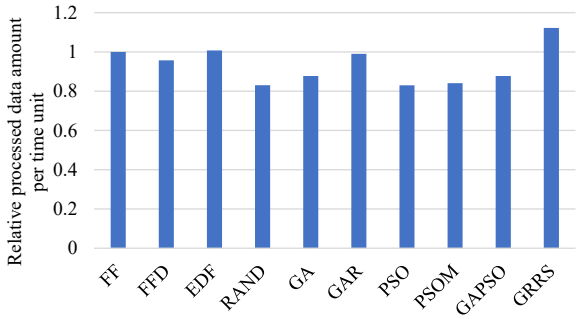

Fig. 8. The relative data processing efficiency achieved by various task offloading methods.

4) Performance of Improvements: One of our improvement schemes for meta-heuristics is task rescheduling, which can be applied by any meta-heuristic. In this section, we test the performance of rescheduling in the improvement of user satisfaction, resource efficiency, and processing efficiency. The experiment results are shown in Fig. 9-13, where x_RS means x improved with the rescheduling. From these figures, we can see that rescheduling can improve above 11% performance in every metric for meta-heuristic algorithms on task offloading. This verifies the high efficiency of rescheduling in improving the performance of meta-heuristic-based offloading algorithms. The main reason why rescheduling can improve the performance of meta-heuristics is that meta-heuristics make decision of task assignment without considering the load balance between computing cores. This leads to some cores are overloaded while some others are underloaded, giving a opportunity for performance improvement by rescheduling.

V. Related Works

As the development of IoT, EECC has been applied to various fields for improving the performance of various data processing applications. To improve service quality and resource efficiency in EECC environments, several works focused on addressing the task offloading problem.

Sang et al. [21] proposed a heuristic offloading algorithm to improve the cooperativeness of EECC resources. They used cloud resources for processing offloaded tasks at first, and rescheduled some tasks from the cloud to ES and devices to improve overall performance. This can improve overall user satisfaction, but negatively affect the overall performance of task processing. Wang et al. [22] presented two offloading al-

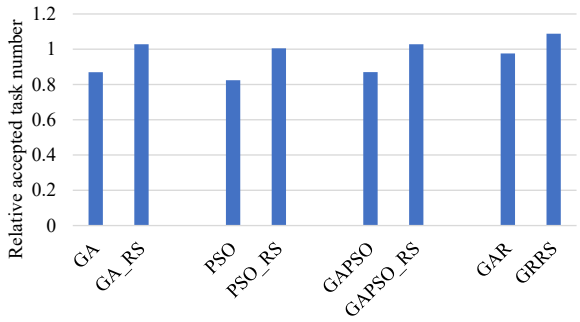

Fig. 9. The accepted task number improved by rescheduling.

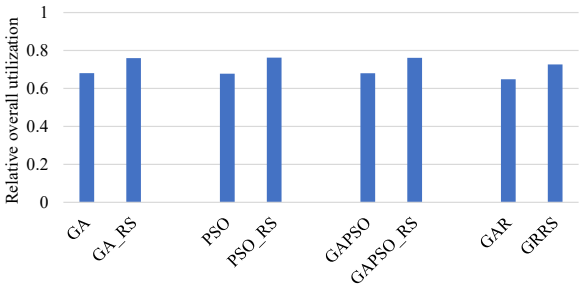

Fig. 10. The resource utilization improved by rescheduling.

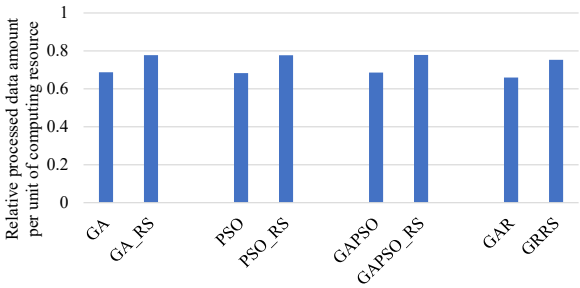

Fig. 11. The data processing efficiency improved by rescheduling.

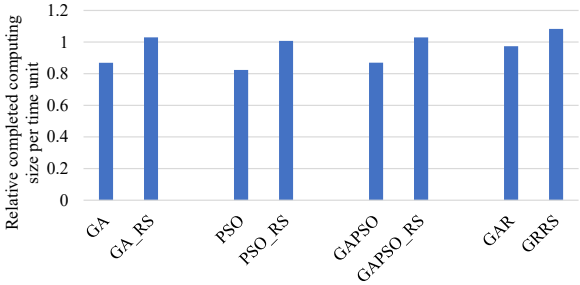

Fig. 12. The overall computing rate improved by rescheduling.

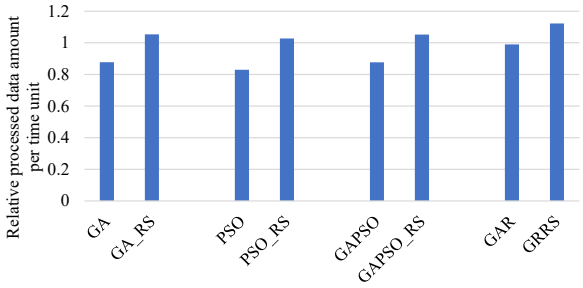

Fig. 13. The overall data processing rate improved by rescheduling.

gorithms, named as FRFOA and LBOA, respectively. FRFOA was to offload a task to an ES such that the response time is minimum every time. LBOA iteratively assigned a task to an ES which can satisfy requirements of the most tasks. These heuristic-based algorithms generally consume few resources but have limited performance, because they only exploit local search strategies.

Therefore, some works used meta-heuristics to pursue the global best offloading solution. Both Wang et al. [18] and Gao et al. [23] applied PSO with same solution representation method to this paper. In addition, to improve exploration ability, Gao et al. [23] used Lévy Flight movement pattern for updating particle positions. Wang et al. [15] used GA for optimizing user satisfaction and resource efficiency. Chakraborty and Mazumdar [16] employed GA to reduce energy consumption with latency constraints. Bali et al. [24] used NSGA-II to optimize energy and queue delay for offloading data to ES and CS.

To further improve performance, some works considered to exploit complementary advantage of different algorithms, proposed hybrid heuristic algorithms. Hussain and Al-Turjman [17] replaced the chromosome by its better offspring generated by the crossover operator for each individual, which is similar to population evolution behavior of PSO. Nwogbaga et al. [19] performed mutation operator for every individual at the end of each evolutionary iteration for PSO to improve diversity for avoiding premature convergence. Farsi et al. [20] sequentially performed GA and PSO for the population evolution. Zhang et al. [25] presented a dynamic selection mechanism to combine multiple meta-heuristics, which selected offspring generated by these meta-heuristics to be passed on to next generation. All of above works just performed two or more meta-heuristics separately, which leads to a poor performance of combination.

Therefore, in this paper, we exploit a combination approach to integrate the swarm intelligent into the evolutionary algorithm for a better offloading solution on EECC. In addition, we propose to use heuristic rescheduling approach to further improve the solution quality.

VI. Conclusion

In this section, we focus on the task offloading problem for EECC systems. We first formulate the problem into MINLP, which has been proofed as NP-hard. Then, to solve the problem with reasonable time complexity, we design a task offloading algorithm, GRRS, based on GA which is one of the most representative meta-heuristics and performs well on solving various optimization problems in many fields. To enhance exploration and exploitation of GA, we integrate two improvement scheme into it. One is replacing each individual with its better offspring during the population evolution, to pass on good genes. Another is rescheduling rejected tasks to take full advantage of available EECC resources. Extensive experiments are conducted, and results verify efficiency and effectiveness of GRRS.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions. The research was supported by the key scientific and technological projects of Henan Province (Grant No. 232102211084, 232102210023, 232102210125), the Key Scientific Research Projects of Henan Higher School (Grant No. 22A520033), Zhengzhou Basic Research and Applied Research Project (ZZSZX202107) and China Logistics Society (2022CSLKT3-334).

References

[1] Cisco Systems, Inc., Cisco Annual Internet Report (2018–2023) White Paper, https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html, Mar. 2020.

[2] X. Jin, W. Hua, Z. Wang and Y. Chen, "A survey of research on computation offloading in mobile cloud computing," Wireless Networks, 2022, 28(4): 1563–1585, May 2022, doi: 10.1007/s11276-022-02920-2.

[3] C. Bi, J. Li, Q. Feng, C.-C. Lin and W.-C. Su , "Optimal deployment of vehicular cloud computing systems with remote microclouds," Wireless Networks, February 2023, In Press, 13 pages, doi: 10.1007/s11276-023-03268-x.

[4] W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, "Edge Computing: Vision and Challenges," IEEE Internet of Things Journal, 2016, 3(5): 637-646, doi: 10.1109/JIOT.2016.2579198.

[5] X. Wang, J. Li, Z. Ning, Q. Song, L. Guo, S. Guo, and M. S. Obaidat. "Wireless Powered Mobile Edge Computing Networks: A Survey," ACM Computing Surveys, January 2023, In Press, doi:10.1145/3579992.

[6] M. Reiss-Mirzaei, M. Ghobaei-Arani, L. Esmaeili, "A review on the edge caching mechanisms in the mobile edge computing: A social-aware perspective," Internet of Things, 2023, vol. 22, Article ID: 100690, 22 pages, doi:10.1016/j.iot.2023.100690.

[7] J. Ren, D. Zhang, S. He, Y. Zhang and T. Li, "A Survey on End-Edge-Cloud Orchestrated Network Computing Paradigms: Transparent Computing, Mobile Edge Computing, Fog Computing, and Cloudlet," ACM Computing Surveys, 2019, vol. 52, no. 6, Article ID: 125, 36 pages, doi: 10.1145/3362031

[8] T. Wang, Y. Liang, X. Shen, X. Zheng, A. Mahmood, and Q. Z. Sheng, "Edge Computing and Sensor-Cloud: Overview, Solutions, and Directions," ACM Computing Surveys, February 2023, In Press, doi: 10.1145/3582270

[9] B. Kar, W. Yahya, Y. -D. Lin and A. Ali, "Offloading Using Traditional Optimization and Machine Learning in Federated Cloud–Edge–Fog Systems: A Survey," IEEE Communications Surveys & Tutorials, vol. 25, no. 2, pp. 1199-1226, Secondquarter 2023, doi: 10.1109/COMST.2023.3239579.

[10] M. I. Khaleel, "Efficient job scheduling paradigm based on hybrid sparrow search algorithm and differential evolution optimization for heterogeneous cloud computing platforms," Internet of Things, 2023, vol. 22, Article ID: 100697, 29 pages, doi: 10.1016/j.iot.2023.100697.

[11] W. Khallouli and J. Huang, "Cluster resource scheduling in cloud computing: literature review and research challenges," The Journal of Supercomputing, vol. 78, pp. 6898–6943, 2022. doi: 10.1007/s11227-021-04138-z.

[12] W. Lu, Y. Mo, Y. Feng, Y. Gao, N. Zhao, Y. Wu, and A. Nallanathan, "Secure Transmission for Multi-UAV-Assisted Mobile Edge Computing Based on Reinforcement Learning," IEEE Transactions on Network Science and Engineering, vol. 10, no. 3, pp. 1270-1282, 1 May-June 2023, doi: 10.1109/TNSE.2022.3185130.

[13] Z. Liu, C. Chen, J. Li, Y. Cheng, Y. Kou, D. Zhang, "KubFBS: A fine-grained and balance-aware scheduling system for deep learning tasks based on kubernetes," Concurrency and Computation: Practice and Experience, vol. 34, no. 11, Article ID: e6836, 16 pages, 2022. doi:10.1002/cpe.6836

[14] J. Du, and J. Y.-T. Leung, "Complexity of Scheduling Parallel Task Systems," SIAM Journal on Discrete Mathematics, vol. 2, no. 4, pp. 473-487, 1989, doi: 10.1137/0402042.

[15] B. Wang, B. Lv, and Y. Song, " A Hybrid Genetic Algorithm with Integer Coding for Task Offloading in Edge-Cloud Cooperative Computing," IAENG International Journal of Computer Science, vol. 49, no. 2, pp. 503-510, 2022.

[16] S. Chakraborty, K. Mazumdar, "Sustainable task offloading decision using genetic algorithm in sensor mobile edge computing," Journal of King Saud University - Computer and Information Sciences vol. 34, no. 4, pp. 1552-1568, 2022, doi: 10.1016/j.jksuci.2022.02.014.

[17] A. A. Hussain and F. Al-Turjman, "Hybrid Genetic Algorithm for IOMT-Cloud Task Scheduling," Wireless Communications and Mobile Computing, vol. 2022, Article No. 6604286, 14 pages, 2022, doi: 10.1155/2022/6604286.

[18] B. Wang, J. Cheng, J. Cao, C. Wang and W. Huang, "Integer particle swarm optimization based task scheduling for device-edge-cloud cooperative computing to improve SLA satisfaction," PeerJ Computer Science, vol. 8, Article ID: e893, 22 pages, 2022. doi:10.7717/peerj-cs.893

[19] N. E. Nwogbaga, R. Latip, L. S. Affendey, and A. R. Abdul Rahiman, "Attribute reduction based scheduling algorithm with enhanced hybrid genetic algorithm and particle swarm optimization for optimal device selection," Journal of Cloud Computing, vol. 11, Article ID: 15, 17 pages, 2022. doi: 10.1186/s13677-022-00288-4.

[20] A. Farsi, S. Ali Torabi, and M. Mokhtarzadeh, "Integrated surgery scheduling by constraint programming and meta-heuristics," International Journal of Management Science and Engineering Management, July 2022, In Press, doi: 10.1080/17509653.2022.2093289.

[21] Y. Sang, J. Cheng, B. Wang and M. Chen, "A three-stage heuristic task scheduling for optimizing the service level agreement satisfaction in device-edge-cloud cooperative computing," PeerJ Computer Science, vol. 8, Article ID: e851, 24 pages, 2022, doi:10.7717/peerj-cs.851

[22] C. Wang, R. Guo, H. Yu, Y. Hu, C. Liu, C. Deng, "Task offloading in cloud-edge collaboration-based cyber physical machine tool," Robotics and Computer-Integrated Manufacturing, vol. 79, Article ID: 102439, 13 pages, 2023, doi: 10.1016/j.rcim.2022.102439.

[23] T. Gao, Q. Tang, J. Li, Y. Zhang, Y. Li and J. Zhang, "A Particle Swarm Optimization With Lévy Flight for Service Caching and Task Offloading in Edge-Cloud Computing," IEEE Access, vol. 10, pp. 76636-76647, 2022, doi: 10.1109/ACCESS.2022.3192846.

[24] M. S. Bali, K. Gupta, D. Gupta, G. Srivastava, S. Juneja, and A. Nauman, "An effective technique to schedule priority aware tasks to offload data on edge and cloud servers," Measurement: Sensors, vol. 26, Article ID: 100670, 9 pages, 2023, doi:10.1016/j.measen.2023.100670.

[25] J. Zhang, Z. Ning, R. H. Ali, M. Waqas, S. Tu and I. Ahmad, "A Many-objective Ensemble Optimization Algorithm for the Edge Cloud Resource Scheduling Problem," IEEE Transactions on Mobile Computing, In Press, January 2023, 18 pages, doi: 10.1109/TMC.2023.3235064.