Hindawi
Wireless Communications and Mobile Computing
Volume 2024, Article ID 6980514, 12 pages
https://doi.org/10.1155/2024/6980514

*Research Article*

# Reliability-Constrained Task Scheduling for DAG Applications in Mobile Edge Computing

**Liangbin Zhu,[1] Ying Shang,[2] Jinglei Li,[3] Yiming Jia,[3] and Qinghai Yang[3]**

[1]*School of Information and Electronics, Beijing Institute of Technology, Beijing, China*
[2]*Beijing Institute of Computer Technology and Application, Beijing, China*
[3]*School of Telecommunications Engineering, Xidian University, Xi'an 710071, China*

Correspondence should be addressed to Ying Shang; 15901821251@163.com

Received 28 February 2023; Revised 31 October 2023; Accepted 12 December 2023; Published 29 January 2024

Academic Editor: Carlo Giannelli



The development of the internet of things (IoT) and 6G has given rise to numerous computation-intensive and latency-sensitive applications, which can be represented as directed acyclic graphs (DAGs). However, achieving these applications poses a huge challenge for user equipment (UE) that are constrained in computational power and battery capacity. In this paper, considering different requirements in various task scenarios, we aim to optimize the execution latency and energy consumption of the entire mobile edge computing (MEC) system. The system consists of single UE and multiple heterogeneous MEC servers to improve the execution efficiency of a DAG application. In addition, the execution reliability of a DAG application is viewed as a constraint. Based on the strong search capability and Pareto optimality theory of the cuckoo search (CS) algorithm and our previously proposed improved multiobjective cuckoo search (IMOCS) algorithm, we improve the initialization process and the update strategy of the external archive, and propose a reliability-constrained multiobjective cuckoo search (RCMOCS) algorithm. According to the simulation results, our proposed RCMOCS algorithm is able to obtain better Pareto frontiers and achieve satisfactory performance while ensuring execution reliability.

## 1. Introduction

With the advent of the internet of everything, autonomous driving, immersive games based on VR or AR technology, and the industrial internet of things (IoT) are emerging [1]. However, achieving these computation-intensive and latency-sensitive applications poses a significant challenge for user equipment (UE) that is limited in terms of computation power and battery capacity [2, 3]. To solve this problem, mobile edge computing (MEC) is proposed as a new and popular computing paradigm. It provides users with abundant computing resources and reliable computing services at the edge of access networks [4–7].

In MEC, an increasingly important research concern is the scheduling of directed acyclic graph (DAG) applications. These applications are considered practical for representing multiple interdependent submodules, such as vehicle navigation [2] and object classification [8]. Offloading subtasks of a DAG to nearby MEC servers can reduce application execution latency and energy consumption of user equipment, thereby improving the execution efficiency of DAG applications.

Oriented task scheduling for DAG applications, a considerable part of the works is dedicated to solving single-objective task scheduling problems in order to optimize execution latency [2, 9–11], and energy consumption [7, 12, 13]. However, for practical applications, especially safety-critical ones, execution reliability is a crucial factor in ensuring the quality of service (QoS) for users. Due to the existing failure rate of computing devices, when a processor fails during task execution, it affects the execution reliability of the execution. As a result, re-executing the task consumes more time and energy [7]. Therefore, in addition to the indicators mentioned above, there is an increasing focus on execution reliability [7, 14–19]. Based on the above analysis, we consider execution reliability as a constraint in DAG task scheduling in this paper.

In practice, instead of focusing on a single indicator, different requirements are usually presented in various task scenarios. In addition, in MEC systems, latency and energy consumption of the entire system often conflict with each other. Therefore, it is crucial to study multiobjective task scheduling optimization problems in order to explore tradeoff solutions. This will ensure that satisfying scheduling schemes are provided to users in the different task scenarios.

As the cuckoo search (CS) algorithm [20] has the advantages of simplicity, high robustness, and high efficiency, it has been employed to solve multiobjective optimization problems in a variety of fields, including network awareness, resource allocation, and task scheduling, etc. [21–23]. These research studies indicate that it outperforms traditional optimization algorithms such as whale optimization and hybrid bee colony optimization, among others. Thus, we still utilize the CS algorithm [20] and the multiobjective cuckoo search (MOCS) algorithm [24] to solve a multiobjective task scheduling problem for DAG applications based on our previously proposed improved multiobjective cuckoo search (IMOCS) algorithm [25].

The main contributions in this paper are summarized as follows:

(1) The formulation of a multiobjective DAG task scheduling problem aims to minimize the execution latency and energy consumption of the entire MEC system while satisfying the given execution reliability constraint.

(2) The reliability-constrained multiobjective cuckoo search (RCMOCS) algorithm is proposed. Different from our previous work [25], we take into account the execution reliability constraint and make further improvements to the initialization process and update strategy of the external archive. This allows us to obtain Pareto solutions that meet the specified constraint.

Simulation results demonstrate that under the constraint of execution reliability, the proposed RCMOCS algorithm consumes less latency and energy in the entire MEC system. It also outperforms the comparison algorithms in terms of convergence and uniformity.

The structure of the paper is as follows: Section 2 introduces the related literature. Section 3 introduces the system models. Problem formulation are introduced in Section 4. Section 5 presents the proposed multiobjective task scheduling algorithm. Simulation results are shown in Section 6. Section 7 concludes the main work of this paper.

## 2. Literature Review

Recently, a large number of research studies on task scheduling have been conducted. In this section, we will classify and analyze the relevant research.

Aiming at the task offloading problem in the small cellular network scenario, plenty of studies have focused on single-objective task scheduling problems for DAG applications. Different from task scheduling for general tasks like, where Zhang et al. [26] proposed the artificial fish swarm algorithm to optimize the system's energy with limited delay, Shu et al. [2] proposed a distributed offloading policy in a multiuser and multiserver scenario to minimize execution latency while adhering to an energy consumption constraint. Hu et al. [11] converted large-scale graphs in social networks into DAGs and proposed a heterogeneous-aware cluster scheduling algorithm to minimize the processing time of a DAG. In a multiuser and multiserver scenario, Dong et al. [27] designed a quantum particle swarm optimization (QPSO) algorithm to minimize the total energy consumption of all users and MEC servers within a limited time. Aiming at minimizing the total energy consumption of all tasks, Yin et al. [28] proposed a novel task scheduling algorithm based on the firefly algorithm in a cloud-edge system.

In addition, execution reliability is receiving increasing attention as it is crucial to ensure users' QoS. For example, Liu et al. [7] designed a heuristic algorithm to minimize the energy consumption of a UE, considering constraints such as the execution latency and execution reliability of a DAG application. Hu and Cao [18] proposed a DAG application scheduling policy with reliability requirements in a heterogeneous embedded system. They priced the tradeoff between reliability and resource consumption costs and gradually adjusted the schedule to improve reliability.

Optimization objectives in the above studies are all singular, but in practice, multiple performance indicators are usually taken into consideration. It should also be noted that the improvement of one indicator always comes at the expense of the decline of other indicators. Therefore, it is crucial and meaningful to study multiobjective optimization problems in order to explore tradeoff solutions.

Aiming to minimize the execution latency and energy consumption of UE, Chen et al. [29] proposed a method of user perceptual task offloading in the IoT based on an optimizing strategy of quantum behavior particle population. This method aims to reduce execution time and energy consumption for the offloading scenario involving multiple edge center servers and smart devices used by multiple IoT users. Peng et al. [30] optimized the execution latency and energy consumption of UE in mobile cloud computing using the weighted summation method and the whale optimization algorithm (WOA). Nevertheless, both of the above studies transformed a multiobjective problem into a single-objective one by using a weighted cumulative function. However, this approach requires manual adjustment of weight parameters and cannot accommodate changes in the requirements.

Instead, taking advantage of multiobjective evolutionary algorithms, Zhou et al. [31] proposed a multiobjective workflow scheduling algorithm based on genetic algorithms (GA) and a delay transmission mechanism in mobile cloud computing. But this method requires a significant amount of computing resources and may lead to local optimal solutions. Sun et al. [32] propose an effective multiobjective immune algorithm (MOIA) to solve the multiobjective scheduling problem and generate Pareto-optimal solutions for achieving joint optimization of computational efficiency and energy efficiency in the hyperspectral image classification applications. However, the diversity maintenance methods of immune

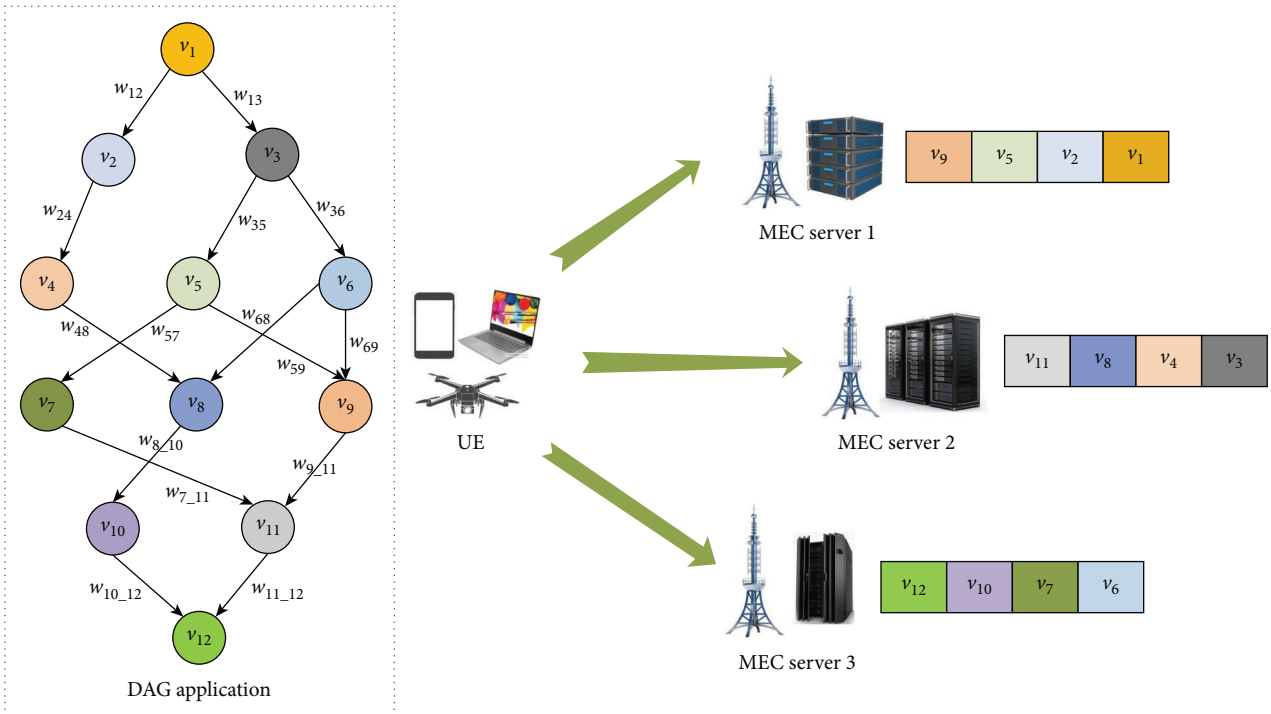


Figure 1: System model.

algorithms may not be sufficient to effectively explore and preserve the entire Pareto front. Zhou et al. [33] developed a multiobjective ant colony system algorithm (MOACS) for the airline crew rostering problem, focusing on fairness and satisfaction. Fang et al. [34] proposed an improved multiobjective evolutionary algorithm for high-quality pattern mining (MOEA-PM) to optimize three objectives in pattern mining. Although, they are not oriented toward task scheduling, the idea of their population initialization strategies is worth learning from. Zhang et al. [35] improved the NSGA-II algorithm to minimize the average energy consumption and task offloading delay of all vehicles in the context of the internet of vehicles.

Considering reliability as an indicator, in [36], a new multiobjective scheduling algorithm with fuzzy resource utilization was proposed. This algorithm was developed under the constraint of execution reliability and utilized particle swarm optimization (PSO) and Pareto dominance. Nevertheless, further optimization is needed for the management of the archive. Tang [14] proposed a fault-tolerant cost-efficient workflow scheduling algorithm that minimizes the cost and time of application execution, while also ensuring reliability. Huang et al. [17] presented an out-degree scheduling algorithm that allocates the DAG nodes based on their out-degrees, considering energy consumption, reliability, and dynamic finish time. Hu et al. [19] built a safety-guaranteed and development cost-minimized schedule for functionality modeled as a DAG running on an automotive system.

Through the analysis of the literature above, it can be observed that the majority of current studies focuses on heterogeneous computing systems and cloud computing. However, there is a lack of research on multiobjective task scheduling for DAG applications in MEC scenarios. In terms of algorithms, several existing multiobjective task scheduling algorithms are based on the GA and PSO algorithm, whose search capabilities are weaker than CS. In this paper, on the basis of our previously proposed IMOCS algorithm, we optimize the execution latency and energy consumption of the entire MEC system, taking into account the execution reliability.

## 3. System Models

In this section, we will introduce the details of the system model. As shown in Figure 1, there is a UE that is running a computation-intensive and latency-sensitive DAG application. Multiple base stations are deployed in the MEC network, each of which is embedded with an MEC server. In this system, all processors including the UE and $M$ heterogeneous MEC servers are denoted by $s_0$ and a set $S$, respectively. Specifically, the set $S$ is represented as $S = \{s_1, s_2, \ldots, s_M\}$.

In the following subsections, we will present the DAG application model, communication model, computation model, and reliability model.

*3.1. DAG Application Model.* DAG provides an efficient representation for typical practical applications that consist of a series of submodules with dependencies. An application can be described as $G = (V, E)$. Here, $V$ represents the set of $N$ subtasks in DAG $G$ and it is denoted as $V = \{v_1, v_2, \ldots, v_N\}$. Each subtask $v_i$ is described by $\{m_i, c_i, a_i\}$, where $m_i$ is the size of subtask $v_i$, $c_i$ is the required CPU cycles to complete $v_i$, and $\alpha_i$ is the scheduling decision of $v_i$. For example, $a_3 = 2$ represents that subtask $v_3$ is offloaded to MEC server $s_2$. $E = \{(e(v_i, v_j)) | v_i, v_j \in V\}$ is composed of all directed edges, where $e(v_i, v_j)$ means that $v_i$ is the predecessor of $v_j$ and $v_j$ is

the successor of $v_i$. Subtask $v_j$ does not start until $v_i$ is done. The edge e $(v_i, v_j)$'s weight $w_{ij}$ is equal to the size of $v_i$'s execution result. Subtask $v_i$'s predecessors and successors constitute pred $(v_i)$ and succ $(v_i)$, respectively. Without loss of generality, each DAG has one entry subtask $v_{\text{entry}}$ without any predecessors and one exit subtask $v_{\text{exit}}$ without any successors.

*3.2. Communication Model.* DAG-based task scheduling is equivalent to determining the execution location for each subtask. In this subsection, we will analyze the process of offloading subtasks to MEC servers and transmitting execution results in detail.

Assuming that subtask $v_i$ is offloaded to MEC server $s_m$, that is, $a_i = m$, the offloading latency is given as follows:

$$T_{im}^{\text{comm}} = m_i / r_{0m}, \tag{1}$$

where $m_i$ is the size of subtask $v_i$ and $r_{0m}$ is the uplink data rate.

After subtask $v_i$ is completed, the output of $v_i$ executed on processor $s_m$ is sent to processor $s_n$ that is executing the successor subtask $v_j$ of $v_i$. The transmission latency of the result of $v_i$ is given by:

$$T_{ij}^{\text{comm}} = \begin{cases} w_{ij}/r_{mn}, & a_i = m, a_j = n, \\ & e(v_i, v_j) \in E, m \neq n, \\ 0, & a_i = a_j \end{cases} \tag{2}$$

where $r_{mn}$ is the transmission rate between $s_m$ and $s_n$. When two subtasks with a dependency are executed on the same processor, the transmission latency can be neglected.

Besides, the energy consumption of processor $s_m$ for executing the transmission of subtask $v_i$ is denoted by:

$$E_{ij}^{\text{comm}} = P_m T_{ij}^{\text{comm}}, \tag{3}$$

where $P_m$ is the transmission power of processor $s_m$.

*3.3. Computation Model.* In DAG $G$, each subtask is performed on UE or offloaded to an MEC server for execution. The required latency in both cases is analyzed.

If subtask $v_i$ is executed locally, the execution latency and the energy that the UE consumes are given by:

$$T_{i0}^{\text{comp}} = c_i / f_0, \tag{4}$$

$$E_{i0}^{\text{comp}} = c_i \delta_0, \tag{5}$$

where $f_0$ is the computational capability of UE and $\delta_0$ is the energy consumption of the UE in a CPU cycle.

When subtask $v_i$ is offloaded to MEC server $s_m$, the execution latency is as follows:

$$T_{im}^{\text{comp}} = c_i / f_m, \tag{6}$$

where $f_m$ denotes the computational capability of the MEC server $s_m$.

To complete subtask $v_i$, the total energy that the total energy consumed by the entire system is given by:

$$E_i = E_{im}^{\text{comm}} + E_{im}^{\text{comp}} = P_0 T_{im}^{\text{comm}} + c_i \delta_m. \tag{7}$$

The first part calculates the energy consumption of the UE for offloading subtask $v_i$, while the second part determines the energy consumption of the MEC server $s_m$ for executing subtask $v_i$.

Based on the analysis above, we provide the energy consumption of the entire system when performing any subtask $v_i$.

$$E(t_i) = \begin{cases} E_{i0}^{\text{comp}} + E_{ij}^{\text{comm}}, & a_i = 0, \\ E_{im}^{\text{comm}} + E_{im}^{\text{comp}} + E_{ij}^{\text{comm}}, & a_i = m, \\ & m \neq 0. \end{cases} \tag{8}$$

For local computing, a UE consumes energy to complete a subtask $v_i$ and transmit its execution result. For edge computing, we consider the energy consumption of the UE for offloading subtask $v_i$, as well as the energy consumption of the MEC server $s_m$ for executing $v_i$ and transmitting its result.

Then, we provide the total energy consumption of the entire system for completing a DAG application as follows:

$$E(G) = \sum_{i=1}^{N} E(t_i). \tag{9}$$

*3.4. Reliability Model.* Communication reliability and execution reliability are commonly used to assess the reliability of task scheduling [37]. Considering that communication reliability has been well-researched [38–40], and the research on execution reliability is limited, we focuses on the execution reliability of a DAG application in this paper.

The execution reliability is defined by the Poisson distribution with a parameter $\rho$. For processor $s_m$, $\rho_m$ is its failure rate per unit of time. It is a random variable that follows a Gaussian distribution [7]. The reliability of subtask $v_i$ executed on $s_m$ is defined as follows:

$$R_{im} = \exp(-\rho_m c_i / f_m). \tag{10}$$

For the entire application, the reliability is the accumulated execution reliability of all subtasks in a DAG, which is denoted as follows:

$$R(G) = \prod_{i=1}^{N} R_{im}. \tag{11}$$

In this paper, the given constraint is set as follows:

$$R_{\text{given}}(G) = \prod_{i=1}^{N} e^{\overline{\rho}} * \overline{T_i}\,, \tag{12}$$

where $\overline{\rho}$ and $\overline{T_i}$ represent the average failure rate and average execution latency of subtask $v_i$, respectively. Then we constrain the execution reliability of the whole application as follows:

$$R(G) \geq R_{\text{given}}(G). \tag{13}$$

## 4. Problem Formulation

In this article, we consider the dependency among subtasks and aim to minimize the execution latency and energy consumption of the whole MEC system for the DAG application, while ensuring the reliability constraint is met. Before formulating the optimization problem, it is necessary to provide several definitions.

First of all, for subtask $v_i$ executed on processor $s_m$, its earliest start time (EST) is defined as follows:

$$\text{EST}(v_i) = \max\left\{\text{avail}(s_m), \text{arv}(i,m), \max_{\substack{v_j \in \text{pred}(v_i), \\ a_j = n}} \left[\text{EFT}\left(v_j\right) + T_{ji}^{\text{comm}}\right]\right\}, \tag{14}$$

where avail $(s_m)$ is the earliest time that processor $s_m$ is free to perform subtask $v_i$, arv $(i, m)$ is the time when subtask $v_i$ arrives at processor $s_m$, and EFT $(v_j)$ is the earliest finish time of $v_i$'s predecessor subtask $v_j$. The execution of subtask $v_i$ cannot start unless its predecessors are completed and $s_m$ receives data required for executing $v_i$. This constraint can be represented as follows:

$$\text{EST}(v_i) \geq \text{EFT}\left(v_j\right) + T_{ji}^{\text{comm}}. \tag{15}$$

Furthermore, if two subtasks $v_i$ and $v_j$ are scheduled on the same processor, their processing times do not overlap, and the execution order depends on their priority, namely:

$$\text{EST}(v_i) \geq \text{EFT}\left(v_j\right), v_j \in \text{hpri}(v_i), \tag{16}$$

where hpri$(v_i)$ is a set that contains subtasks with higher priority than $v_i$.

In addition, the arrival of subtask $v_i$ at processor $s_m$ is a prerequisite for starting the execution of $v_i$, which is denoted as follows:

$$\text{arv}(i,m) \leq \text{EST}(v_i). \tag{17}$$

Based on the definitions provided above, our multiobjective task scheduling problem can be formulated as follows:

$$\begin{aligned} &\min \text{EFT}(v_{\text{exit}}), \\ &\min E(G), \end{aligned} \tag{18}$$

$$\text{s.t.} \quad (13), (15), (16), (17), \tag{19}$$

$$\text{EST}(v_i) \geq 0, \quad \forall v_i \in V. \tag{20}$$

As the exit subtask $v_{\text{exit}}$ has the lowest priority, the execution latency of the DAG application is exactly the finish time of $v_{\text{exit}}$. Besides, Equation (20) ensures that the earliest start time of all subtasks cannot be earlier than time 0.

## 5. RCMOCS Task Scheduling Algorithm

Based on the IMOCS algorithm [25], we establish an external archive and devise a method for updating it. Through the process of evolution, we obtain more satisfying Pareto-optimal solutions. In addition, we also take into account the execution reliability of a DAG application and propose a RCMOCS task scheduling algorithm to tackle the multiobjective task scheduling problem.

*5.1. RCMOCS Algorithm Architecture.* Similar to the IMOCS algorithm, the RCMOCS algorithm also includes the same coding scheme, an update method for the direction of Lévy flight, and an improved evolutionary process. However, when considering reliability constraint, there are two main differences between the IMOCS algorithm and the RCMOCS algorithm.

(1) Instead of finding nondominated solutions in the initial population and storing them directly into the external archive [25], the RCMOCS algorithm takes into account the reliability constraint. Thus, during the initialization phase of the archive with a fixed capacity of $L$, it is necessary to evaluate each solution based on latency, energy consumption, and execution reliability. If there are nondominated solutions that meet the reliability constraint, no more than $L$ solutions are stored in the archive. Otherwise, we will select a subset of solutions that violate the given reliability constraint to a lesser extent and store them in the archive.

(2) In updating the archive, the execution reliability needs to be considered. Algorithm 1 demonstrates our well-designed update algorithm. It is assumed that populations before and after the Lévy flight or preference random walk are recorded as $P_0$ and $P_1$, respectively, and the size of the population is $N_c$. That is, $P_1(i)$ is the solution obtained by $P_0(i)$ after a Lévy flight or preference random walk. An indicator $I$ is set to determine whether $P_1$ has a solution of higher quality than $P_0$. The indicator is initialized to false (Line 1). When a solution $P_1(i)$ is capable of dominating the corresponding $P_0(i)$ and the reliability of $P_1(i)$ is higher than that of $P_0(i)$, the indicator is updated to true. Another case in which $I$ turns to true is when $P_0(i)$ dominates $P_1(i)$ and $P_1(i)$ has

1: Initialize $I =$ False and $i = 1$
2: **while** $i \leq N_c$ **do**
3: **if** $P_1[i] \preceq P_0[i]$ and $P_1[i]$'s reliability $\geq R_{\text{given}}(G)$ **then**
4: $I =$ True
5: **if** $P_0[i]$ in archive $A$ **then**
6: Replace $P_0[i]$ with $P_1[i]$
7: **else**
8: Add $P_1[i]$ into archive $A$
9: **end if**
10: **end if**
11: $i$++
12: **end while**
13: **if** $I$ is True **then**
14: Find all nondominated solutions in $A$
15: **if** the number of solutions that satisfy the reliability constraint $\geq 0$ **then**
16: Keep solutions which satisfies the reliability constraint in $A$
17: **else**
18: Keep solutions with less violation of the reliability constraint in $A$
19: **end if**
20: **if** number of solutions in $A \leq L$ **then**
21: Retain all solutions in $A$
22: **else**
23: Do crowding distance sorting and keep the top $L$ solutions in $A$
24: **end if**
25: **end if**

Algorithm 1: Update strategy of the archive in RCMOCS.

higher reliability (Lines 3–4). Then, we determine if $P_0(i)$ is in the archive $A$. If it is already in $A$, we will replace $P_0(i)$ with $P_1(i)$. Otherwise, $P_1(i)$ is added to $A$ (Lines 5–10). After the comparison process is finished, the archive $A$ needs to be reorganized. If $I$ is true, all nondominated solutions in $A$ need to be found first. Then, solutions that satisfy the given reliability constraint are stored in $A$. If no solution satisfies the constraint, solutions with fewer violations are kept in $A$ (Lines 13–19). Finally, once the number of solutions in $A$ exceeds $L$, crowding distance sorting is used to select $L$ solutions that will be stored in $A$ (Lines 20–24).

The overall framework of the RCMOCS algorithm is the same as the IMOCS [25]. During each iteration, the update of the archive and generation of the population for the next iteration, and the process of fast nondominated sorting and crowding distance sorting, occupy the main part of the time complexity. In detail, after Lévy flight and preference random walk, new solutions need to be compared with old solutions, and this process takes $O(N_c)$ time. Finding nondominated solutions costs $O(2 \cdot N_c^2)$ and crowding distance sorting takes $O(2 \cdot (2N_c) \cdot \log(2N_c))$ time [41]. Finally, the process of crowding distance sorting takes $O(4N_c \cdot \log(2N_c))$ time to select candidate solutions. Thus, each iteration takes $O(N_c + N_c^2 + 4N_c \cdot \log(2N_c))$ time, which is equivalent to $O(N_c^2)$. Therefore, the complexity of the RCMOCS algorithm is $O(N_c^2 \cdot T)$, where $T$ represents the number of iterations.

*5.2. Performance Metrics.* To evaluate the performance of the proposed RCMOCS algorithm, we utilize the Q metric [31] and the S metric [36] to evaluate its convergence and uniformity, respectively.

*5.2.1. Q Metric.* The Q metric is used to measure the convergence of the Pareto-optimal solutions obtained by multiobjective optimization algorithms $A_1$ and $A_2$. Algorithm $A_1$ is considered to be capable of finding a Pareto-optimal solution set with better convergence than $A_2$ if and only if the following condition holds:

$$Q(A_1, A_2) > Q(A_2, A_1) \text{ or } Q(A_1, A_2) > 0.5, \tag{21}$$

where $Q(A_1, A_2) = |\Phi|/|Y|$, $\Phi = Y \cap F_1$, $Y = F_1 \cup F_2$, $F_1$ represents the Pareto solutions obtained by $A_1$. Similarly, $Q(A_2, A_1) = |\Omega|/|Y|$ and $\Omega = Y \cap F_2$.

In order to conveniently describe the performance of algorithms $A_1$ and $A_2$ in terms of convergence, the subsequent simulations use $Q(A_1, A_2) =$ "True" to indicate that the convergence of Pareto-optimal solutions obtained by algorithm $A_1$ is better than that of $A_2$.

*5.2.2. S Metric.* To compare the uniformity of Pareto-optimal solution sets obtained by different multiobjective optimization algorithms, the S metric is used. The S metric is defined as follows:

$$S = \sqrt{\frac{1}{N_p} \sum_{i=1}^{N_p} (d_i' - \overline{d'})^2}, \tag{22}$$

where $N_p$ represents the number of Pareto-optimal solutions, $d_i'$ denotes the minimum distance from the $i$-th solution in the Pareto-optimal solution set to other solutions, and $\overline{d'}$ signifies the average of the minimum distances from each solution to the other solutions. The smaller the value of $S$ is, the more uniform the distribution of the optimal solution set on the Pareto frontier becomes.

## 6. Simulation Experiments

In this section, numerical results are provided to verify the performance of the proposed RCMOCS algorithm. In a single-user multiserver MEC system, our goal is to optimize the execution latency and energy consumption of the entire system, while also considering the constraint of execution reliability. To achieve this, we obtain a series of Pareto-optimal solutions using various multiobjective algorithms. Q metric [31] and S metric [36] are utilized to assess the convergence and uniformity of the multiobjective optimization algorithms,

Table 1: Parameters of the considered MEC system.

| Parameters | Values |
|---|---|
| Minimum distance between UE and MEC servers | 100 m |
| Uplink bandwidth | 20 MHz |
| Noise power | −100 dBm |
| Path loss index | 4 |
| CPU frequency of UE | 1 GHz |
| Transmission power of UE | 100 mW |
| CPU frequency of MEC servers | [2, 7] GHz |
| Size of DAG applications | [400, 700] KB |
| CPU cycles required per bit task | 50 |

respectively. In addition, we also explored the effect of the distance between UE and MEC servers on the Pareto-optimal solutions.

The task scheduling simulation experiment is conducted using Python and its third-party libraries, such as Sklearn, Numpy, and so on. The computer used for simulation is equipped with the Windows 10 operating system, with an AMD Ryzen 7 4800U processor, and 16 GB of memory.

Referring to the experimental settings in [2] and [42], we provide specific simulation parameters for the considered MEC scenario and DAG applications that are randomly generated in Table 1.

To validate the effectiveness of our proposed RCMOCS algorithm, we choose the following four classic multiobjective algorithms:

(1) MOCS algorithm: It is an extension of the CS algorithm that combines the search methods of Lévy flight and preference random walk. It also utilizes the concept of Pareto optimality to identify the Pareto solutions [24]. In the simulation experiment, the size of the cuckoo population is set to 200. The maximum number of iterations is 120. In Lévy flight, the step coefficient is 1. The probability of discovery is set to 0.5.

(2) Multiobjective particle swarm optimization (MOPSO) algorithm: It combines Pareto dominance and PSO [43] to solve multiobjective optimization problems. An external repository is used to guide the flight of particles, and a mutation operator is utilized to prevent the algorithm from converging to a suboptimal Pareto frontier. In the simulation experiment, the size of the particle swarm is 200 and the size of the repository is 40. The maximum number of iterations is 120. When updating the velocity of each particle, the inertia weight, cognitive weight, and social weight are 0.5, 0.25, and 0.25, respectively [36].

(3) Nondominated sorting genetic algorithm II (NSGA-II) [41]: In NSGA-II, fast nondominated sorting approach and crowding distance sorting were proposed to achieve an excellent spread of solutions and improve convergence near the true Pareto-optimal front. In the simulation experiment, the population size is 200. The maximum number of iterations is 120.

(4) Random algorithm: Under the premise of maintaining the dependencies among subtasks in a DAG application, all subtasks are randomly assigned to corresponding processors in the considered MEC system. The population size is set to 400.

Table 2: Execution reliability of Pareto solutions obtained by RCMOCS.

| Application scale (KB) | $R_{given}$ | $R_{min}$ | $R_{max}$ |
|---|---|---|---|
| 400 | 0.99695 | 0.99696 | 0.99796 |
| 500 | 0.99729 | 0.99731 | 0.99835 |
| 600 | 0.99675 | 0.99701 | 0.99796 |
| 700 | 0.99621 | 0.99625 | 0.99774 |

In the proposed RCMOCS algorithm, the size of the cuckoo population is 200. The capacity of the external archive is set to 40. The maximum number of iterations is 120. The step coefficient is in the range of [1, 1.2], and the discovery probability is 0.6.

*6.1. Verification of Reliability Constraint.* In this paper, we consider the constraint of execution reliability for DAG applications. Therefore, it is challenging to directly apply the aforementioned algorithms to our problem. For the MOCS algorithm and MOPSO algorithm, we select the solutions that meet the specified constraint from the solutions they generate as the ultimate Pareto solutions. For the NSGA-II algorithm, in each iteration, after performing fast nondominated sorting and crowding distance sorting, we select solutions whose execution reliability is higher than the given constraint. If the total number of solutions that meet the given constraint is less than the population size, the solutions with higher reliability will be selected in order of their reliability and added to the population.

In this subsection, our main focus is to determine whether the improved algorithms are able to obtain solutions that satisfy reliability constraints. Tables 2–6 display the execution reliability of solutions obtained using the five algorithms mentioned above. It can be found that at each application scale, all solutions satisfy the given reliability constraint $R_{given}$ because the minimum reliability $R_{min}$ is higher than the given constraint.

*6.2. Comparison of Pareto Frontiers and Performance Metrics.* Figure 2 shows the Pareto frontiers obtained by various multiobjective optimization algorithms for DAG applications of sizes 400, 500, 600, and 700 KB, respectively. It can be found that the Pareto frontiers obtained by RCMOCS are superior to those of other algorithms. In other words, task scheduling solutions obtained through RCMOCS are able to complete an application faster while consuming relatively less energy. RCMOCS provides users with more scheduling choices, as it offers the most scheduling solutions.

Pareto frontiers obtained by MOCS are the closest to those of RCMOCS, and the solutions on the Pareto frontiers are also widely distributed. But obviously, our proposed

Table 3: Execution reliability of Pareto solutions obtained by MOCS.

| Application scale (KB) | $R_{given}$ | $R_{min}$ | $R_{max}$ |
|---|---|---|---|
| 400 | 0.99695 | 0.99698 | 0.99801 |
| 500 | 0.99729 | 0.99751 | 0.99836 |
| 600 | 0.99675 | 0.99705 | 0.99794 |
| 700 | 0.99621 | 0.99646 | 0.99777 |

Table 4: Execution reliability of Pareto solutions obtained by MOPSO.

| Application scale (KB) | $R_{given}$ | $R_{min}$ | $R_{max}$ |
|---|---|---|---|
| 400 | 0.99695 | 0.99696 | 0.99817 |
| 500 | 0.99729 | 0.99751 | 0.99848 |
| 600 | 0.99675 | 0.99688 | 0.99803 |
| 700 | 0.99621 | 0.99627 | 0.99802 |

Table 5: Execution reliability of Pareto solutions obtained by NSGA-II.

| Application scale (KB) | $R_{given}$ | $R_{min}$ | $R_{max}$ |
|---|---|---|---|
| 400 | 0.99695 | 0.99696 | 0.99772 |
| 500 | 0.99729 | 0.99746 | 0.99834 |
| 600 | 0.99675 | 0.99681 | 0.99792 |
| 700 | 0.99621 | 0.99631 | 0.99749 |

Table 6: Execution reliability of Pareto solutions obtained by Random algorithm.

| Application scale (KB) | $R_{given}$ | $R_{min}$ | $R_{max}$ |
|---|---|---|---|
| 400 | 0.99695 | 0.99697 | 0.99811 |
| 500 | 0.99729 | 0.99734 | 0.99866 |
| 600 | 0.99675 | 0.99689 | 0.99828 |
| 700 | 0.99621 | 0.99621 | 0.99785 |

RCMOCS achieves better task scheduling schemes without increasing the complexity of the algorithm. This is due to the fact that we improve the update method of Lévy flight, introduce an external archive, and enhance the evolutionary process of the population. These enhancements help to identify a greater number of potential optimal solutions within a larger and more optimal range. As a result, the performance of Pareto-optimal solutions is improved. As a result of mechanisms such as the fast nondominated sorting approach and crowding distance sorting, NSGA-II is able to obtain solutions with better uniformity. Through combining the advantages of MOCS and NSGA-II, the RCMOCS is capable of obtaining higher quality solutions. Due to the stronger search capability of CS compared to PSO and numerous improvements, RCMOCS achieves Pareto-optimal solutions with superior performance to MOPSO. As the Random algorithm schedules subtasks to processors randomly, it often produces suboptimal scheduling solutions.

In order to compare the quality of the Pareto-optimal solutions obtained by different algorithms, the convergence and uniformity of the Pareto solutions are analyzed using the Q metric and S metric.

Tables 7–10 display the Q metrics and S metrics for all algorithms considered. From the values of the second row in these four tables, it can be concluded that RCMOCS performs the best in terms of convergence.

In addition, the Pareto-optimal solution sets obtained by RCMOCS correspond to the smallest S metrics, which indicates that it is capable of generating solutions that have the most uniform distribution on Pareto frontiers. Simulation results above show that our proposed RCMOCS algorithm is quite effective. It obtains a series of tradeoff solutions between execution latency and energy consumption of the entire MEC system, with better performance under the constraint of execution reliability.

*6.3. Impact of Distance on Pareto Frontiers.* In an MEC system, the execution latency of a DAG application and the energy consumption of the entire system are closely linked to the distance between UE and MEC servers. Therefore, the following is a simulation analysis on the influence of distance on Pareto-optimal solutions. In this part of the experiment, we assume that there is a DAG application with a size of 700 KB and 16 subtasks. The minimum distance between UE and

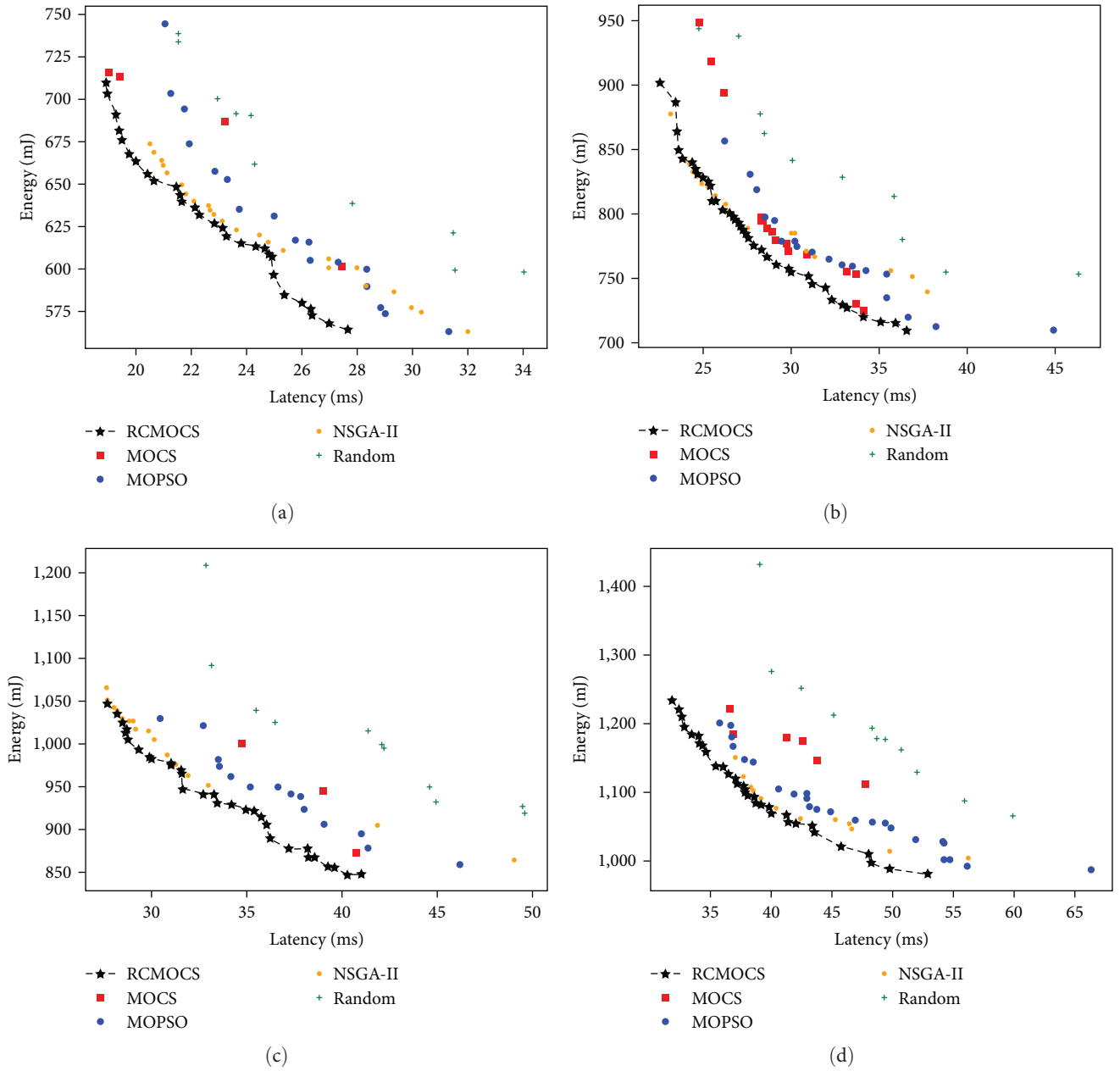


Figure 2: Pareto frontiers under different sizes of DAG applications: (a) size = 400 KB, (b) size = 500 KB, (c) size = 600 KB, and (d) size = 700 KB.

Table 7: Performance comparison while the application's size is 400 KB.

| Q metric | RCMOCS | MOCS | MOPSO | NSGA-II | Random |
|---|---|---|---|---|---|
| RCMOCS | — | True | True | True | True |
| MOCS | False | — | False | False | False |
| MOPSO | False | True | — | False | True |
| NSGA-II | False | True | True | — | True |
| S metric | 0.00210 | 0.03391 | 0.00922 | 0.00219 | 0.01048 |

Table 8: Performance comparison while the application's size is 500 KB.

| Q metric | RCMOCS | MOCS | MOPSO | NSGA-II | Random |
|---|---|---|---|---|---|
| RCMOCS | — | True | True | True | True |
| MOCS | False | — | False | False | True |
| MOPSO | False | True | — | False | True |
| NSGA-II | False | True | True | — | True |
| S metric | 0.00216 | 0.00506 | 0.01215 | 0.00320 | 0.01383 |

Table 9: Performance comparison while the application's size is 600 KB.

| Q metric | RCMOCS | MOCS | MOPSO | NSGA-II | Random |
|---|---|---|---|---|---|
| RCMOCS | — | True | True | True | True |
| MOCS | False | — | True | False | True |
| MOPSO | False | False | — | False | True |
| NSGA-II | False | True | True | — | True |
| S metric | 0.00437 | 0.00681 | 0.00958 | 0.00610 | 0.01884 |

Table 10: Performance comparison while the application's size is 700 KB.

| Q metric | RCMOCS | MOCS | MOPSO | NSGA-II | Random |
|---|---|---|---|---|---|
| RCMOCS | — | True | True | True | True |
| MOCS | False | — | True | False | True |
| MOPSO | False | False | — | False | True |
| NSGA-II | False | True | True | — | True |
| S metric | 0.00445 | 0.00721 | 0.02949 | 0.00475 | 0.03426 |

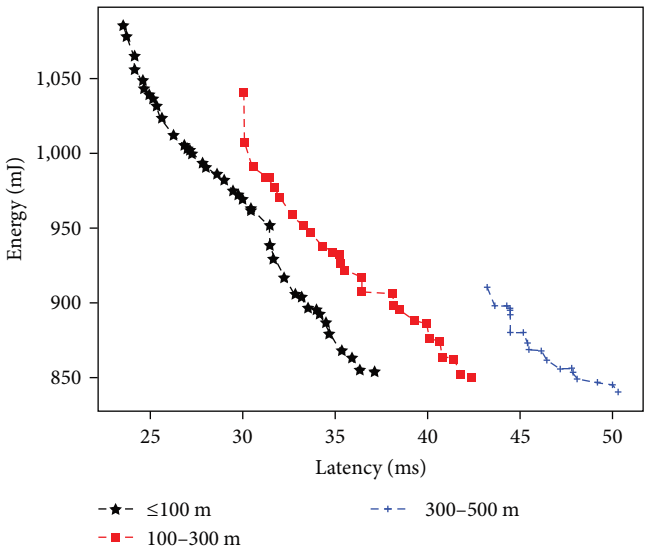


Figure 3: The impact of minimum distance between UE and MEC servers on Pareto solutions.

MEC servers is set to (0, 100], (100, 300], and (300, 500] m, respectively.

Figure 3 demonstrates the Pareto frontiers obtained by our proposed RCMOCS algorithm. As the minimum distance between UE and MEC servers increases, the resulting Pareto frontier gradually moves to the right. This is because, the longer distance consumes more time and energy. Besides, the RCMOCS algorithm tends to choose more MEC servers to perform the DAG application together when the distance between UE to MEC servers is relatively close, which explains why solutions on the black Pareto frontier are the most dispersed among three frontiers.

## 7. Conclusion

In this paper, we consider a scenario in which there is a single UE and multiple heterogeneous MEC servers. We focus on practical applications that are modeled as DAGs, and aim to optimize the execution latency and energy consumption of a DAG application within the entire MEC system, while also satisfying a given execution reliability constraint. Based on our previous work and considering the reliability constraint, we propose the RCMOCS algorithm. This algorithm improves the initialization process and the update strategy of the external archive. Simulation experiments demonstrate that the proposed RCMOCS task scheduling algorithm achieves Pareto-optimal solutions with better convergence and uniformity than the comparison algorithms, while also considering reliability constraints. Finally, we also investigate the impact of the distance between the UE and MEC servers on the Pareto solutions. The future work will focus on optimizing DAG task scheduling problems in MEC scenarios with multiple users and multiple MEC servers. With the vigorous development of the internet of vehicles and vehicle-edge

computing, combined with our research on task scheduling and intelligent optimization algorithms, we can also take content distribution technology [44, 45] as one of the new directions.

## Data Availability

The data used to support the findings of this study are included in the article.

## Disclosure

This work is the result of further research and improvement on our previously published paper entitled "Multiobjective Oriented Task Scheduling in Heterogeneous Mobile Edge Computing Networks".

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## Authors' Contributions

This research was jointly completed by Liangbin Zhu, Ying Shang, Jinglei Li, Yiming Jia, and Qinghai Yang. Liangbin Zhu and Jinglei Li are mainly responsible for algorithm design. Ying Shang and Yiming Jia mainly complete algorithm implementation and paper writing. Qinghai Yang mainly modifies and refines this article.

## Acknowledgments

The research was supported by the Guangdong Basic and Applied Basic Research Foundation (2022A1515240032), the National Natural Science Foundation of China (61971327).

## References

[1] L. Hou, M. A. Gregory, and S. Li, "A survey of multi-access edge computing and vehicular networking," *IEEE Access*, vol. 10, pp. 123436–123451, 2022.

[2] C. Shu, Z. Zhao, Y. Han, G. Min, and H. Duan, "Multi-user offloading for edge computing networks: a dependency-aware and latency-optimal approach," *IEEE Internet of Things Journal*, vol. 7, no. 3, pp. 1678–1689, 2020.

[3] L. Qin, H. Lu, and F. Wu, "When the user-centric network meets mobile edge computing: challenges and optimization," *IEEE Communications Magazine*, vol. 61, no. 1, pp. 114–120, 2023.

[4] J. Chen, X. Cao, P. Yang et al., "Deep reinforcement learning based resource allocation in multi-UAV-aided MEC networks," *IEEE Transactions on Communications*, vol. 71, no. 1, pp. 296–309, 2023.

[5] C. Regueiro, I. Gutierrez-Agueero, and O. Lage, "5G MEC deployments: low-latency services context synchronization in mobility," in *AIIPCC 2022; The Third International Conference on Artificial Intelligence, Information Processing and Cloud Computing*, pp. 1–7, VDE, Online, June 2022.

[6] Z. Sharif, L. T. Jung, I. Razzak, and M. Alazab, "Adaptive and priority-based resource allocation for efficient resources utilization in mobile-edge computing," *IEEE Internet of Things Journal*, vol. 10, no. 4, pp. 3079–3093, 2023.

[7] H. Liu, L. Cao, T. Pei, Q. Deng, and J. Zhu, "A fast algorithm for energy-saving offloading with reliability and latency requirements in multi-access edge computing," *IEEE Access*, vol. 8, pp. 151–161, 2020.

[8] C. Wisultschew, A. Pérez, A. Otero, G. Mujica, and J. Portilla, "Characterizing deep neural networks on edge computing systems for object classification in 3D point clouds," *IEEE Sensors Journal*, vol. 22, no. 17, pp. 17075–17089, 2022.

[9] S. Liu, Y. Yu, X. Lian et al., "Dependent task scheduling and offloading for minimizing deadline violation ratio in mobile edge computing networks," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 2, pp. 538–554, 2023.

[10] A. Yano and T. Azumi, "Deadline miss early detection method for mixed timer-driven and event-driven DAG tasks," *IEEE Access*, vol. 11, pp. 22187–22200, 2023.

[11] K. Hu, G. Zeng, S. Ding, and H. Jiang, "Cluster-scheduling big graph traversal task for parallel processing in heterogeneous cloud based on DAG transformation," *IEEE Access*, vol. 7, pp. 77070–77082, 2019.

[12] B. Hu, Y. Shi, and Z. Cao, "Adaptive energy-minimized scheduling of real-time applications in vehicular edge computing," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 5, pp. 6895–6906, 2023.

[13] M. Pan, Z. Li, and J. Qian, "Energy-efficient multiuser and multitask computation offloading optimization method," *Intelligent and Converged Networks*, vol. 4, no. 1, pp. 76–92, 2023.

[14] X. Tang, "Reliability-aware cost-efficient scientific workflows scheduling strategy on multi-cloud systems," *IEEE Transactions on Cloud Computing*, vol. 10, no. 4, pp. 2909–2919, 2022.

[15] Y. Han, J. Liu, W. Hu, and Y. Gan, "High-reliability and energy-saving DAG scheduling in heterogeneous multi-core systems based on task replication," in *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 2012–2017, IEEE, Melbourne, Australia, October 2021.

[16] Y. Xie, F. Wu, K. Zhang, and S. Leng, "A DAG-based secure cooperative task offloading scheme in vehicular networks," in *2021 IEEE 21st International Conference on Communication Technology (ICCT)*, pp. 870–875, IEEE, Tianjin, China, October 2021.

[17] J. Huang, R. Li, X. Jiao, Y. Jiang, and W. Chang, "Dynamic DAG scheduling on multiprocessor systems: reliability, energy, and makespan," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3336–3347, 2020.

[18] B. Hu and Z. Cao, "Minimizing resource consumption cost of DAG applications with reliability requirement on heterogeneous processor systems," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 12, pp. 7437–7447, 2020.

[19] B. Hu, S. Xu, Z. Cao, and M. Zhou, "Safety-guaranteed and development cost-minimized scheduling of DAG functionality in an automotive system," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 4, pp. 3074–3086, 2022.

[20] X.-S. Yang and Suash Deb, "Cuckoo search via lévy flights," in *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, pp. 210–214, IEEE, Coimbatore, India, 09–11 December 2009.

[21] N. Venkata Subramanian and V. S. Shankar Sriram, "An effective secured dynamic network-aware multi-objective cuckoo search optimization for live VM migration in sustainable data centers," *Sustainability*, vol. 14, no. 4, 2022.

[22] Q. Ma, X. Tong, Y. Huang, J. Li, and G. Xiong, "IMOCS based EV charging station planning optimization considering stakeholders' interests balance," *IEEE Access*, vol. 10, pp. 52102–52115, 2022.

[23] C. Liu, J. Wang, and L. Zhou, "Solving the multi-objective problem of IoT service placement in fog computing using cuckoo search Algorithm," in *Neural Process Letters*, pp. 1823–1854, 2022.

[24] X.-S. Yang and S. Deb, "Multiobjective cuckoo search for design optimization," *Computers & Operations Research*, vol. 40, no. 6, pp. 1616–1624, 2013.

[25] J. Li, Y. Shang, M. Qin et al., "Multiobjective oriented task scheduling in heterogeneous mobile edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 8, pp. 8955–8966, 2022.

[26] D.-G. Zhang, W.-M. Dong, T. Zhang et al., "New computing tasks offloading method for MEC based on prospect theory framework," *IEEE Transactions on Computational Social Systems*, pp. 1–12, 2022.

[27] S. Dong, Y. Xia, and J. Kamruzzaman, "Quantum particle swarm optimization for task offloading in mobile edge computing," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 8, pp. 9113–9122, 2023.

[28] L. Yin, J. Sun, J. Zhou, Z. Gu, and K. Li, "ECFA: an efficient convergent firefly algorithm for solving task scheduling problems in cloud-edge computing," in *IEEE Transactions on Services Computing*, vol. 16, pp. 3280–3293, 2023.

[29] L. Chen, D.-G. Zhang, J. Zhang, T. Zhang, W.-J. Wang, and Y.-H. Cao, "A novel offloading approach of IoT user perception task based on quantum behavior particle swarm optimization," *Future Generation Computer Systems*, vol. 141, pp. 577–594, 2023.

[30] H. Peng, W.-S. Wen, M.-L. Tseng, and L.-L. Li, "Joint optimization method for task scheduling time and energy consumption in mobile cloud computing environment," *Applied Soft Computing*, vol. 80, pp. 534–545, 2019.

[31] Y. Zhou, Z. Li, J. Ge, C. Li, X. Zhou, and B. Luo, "Multi-objective workflow scheduling based on delay transmission in mobile cloud computing," *Journal of Software*, vol. 29, no. 11, pp. 3306–3325, 2018.

[32] J. Sun, H. Li, Y. Zhang et al., "Multiobjective task scheduling for energy-efficient cloud implementation of hyperspectral image classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 14, pp. 587–600, 2021.

[33] S.-Z. Zhou, Z.-H. Zhan, Z.-G. Chen, S. Kwong, and J. Zhang, "A multi-objective ant colony system algorithm for airline crew rostering problem with fairness and satisfaction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 11, pp. 6784–6798, 2021.

[34] W. Fang, Q. Zhang, J. Sun, and X. Wu, "Mining high quality patterns using multi-objective evolutionary algorithm," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 8, pp. 3883–3898, 2022.

[35] J. Zhang, M.-J. Piao, D.-G. Zhang, T. Zhang, and W.-M. Dong, "An approach of multi-objective computing task offloading scheduling based NSGS for IOV in 5G," *Cluster Computing*, vol. 25, no. 6, pp. 4203–4219, 2022.

[36] M. Farid, R. Latip, M. Hussin, and N. A. W. Abdul Hamid, "Scheduling scientific workflow using multi-objective algorithm with fuzzy resource utilization in multi-cloud environment," *IEEE Access*, vol. 8, pp. 24309–24322, 2020.

[37] L. Zhao, Y. Ren, and K. Sakurai, "Reliable workflow scheduling with less resource redundancy," *Parallel Computing*, vol. 39, no. 10, pp. 567–585, 2013.

[38] Y. Cai, X. Jiang, M. Liu, N. Zhao, Y. Chen, and X. Wang, "Resource allocation for URLLC-oriented two-way UAV relaying," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 3, pp. 3344–3349, 2022.

[39] C.-F. Liu, M. Bennis, M. Debbah, and H. V. Poor, "Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing," *IEEE Transactions on Communications*, vol. 67, no. 6, pp. 4132–4150, 2019.

[40] J. Liu and Q. Zhang, "Offloading schemes in mobile edge computing for ultra-reliable low latency communications," *IEEE Access*, vol. 6, pp. 12825–12837, 2018.

[41] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[42] P. Sun, H. Zhang, H. Ji, and X. Li, "Task allocation for multi-APs with mobile edge computing," in *2018 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*, pp. 314–318, IEEE, Beijing, China, 2018.

[43] C. A. C. Coello, G. T. Pulido, and M. S. Lechuga, "Handling multiple objectives with particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 256–279, 2004.

[44] D. Zhang, W. Wang, J. Zhang, T. Zhang, J. Du, and C. Yang, "Novel edge caching approach based on multi-agent deep reinforcement learning for internet of vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 8, pp. 8324–8338, 2023.

[45] Z. Degan, W. Shuo, Z. Jie, Z. Haoli, Z. Ting, and Z. Xiumei, "A content distribution method of internet of vehicles based on edge cache and immune cloning strategy," *Ad Hoc Networks*, vol. 138, no. 2023, Article ID 103012, 2023.