ISSN: 2710-0588

An Efficient Task Offloading Approach based on Multi-objective Evolutionary Algorithm in End-Edge-Cloud Collaborative Architecture

Zengzeng Zhang*

School of Computer Science and Technology, Henan Polytechnic University, Jiaozuo, China

*Corresponding Author: 212309020003@home.hpu.edu.cn

Abstract

The dynamic and distributed nature of the "End-Edge-Cloud" collaborative architecture increases the uncertainty of task offloading decisions, leading to higher computational latency and energy consumption. This paper proposes the Improved Puma Optimization Algorithm (IPOA) to address the multi-objective optimization problem in the "End-Edge-Cloud" collaborative architecture. The IPOA enhances the traditional Puma Optimization Algorithm (POA) by introducing an adaptive solution acceptance strategy driven by charging dynamics and the Levy flight mechanism. Additionally, a multi-objective optimization model considering both latency and energy consumption is established. The IPOA effectively solves the multi-objective optimization problem. Experimental results demonstrate that, compared to various benchmark algorithms (such as POA,GWO, ABC, KOA, DE, and GA), the proposed algorithm reduces latency by more than 14% and energy consumption by more than 31%.

Keywords

"End-Edge-Cloud"Collaborative Architecture, Multi-objective Optimization, Puma Optimization Algorithm, Latency, Energy.

1. Introduction

In recent years, smart mobile devices (SMDs) used in various industries [1], transportation [2], and the Internet of Things (IoT) [3] have generated a massive amount of computational tasks. These tasks demand exacting performance requirements, including ultra-low latency and energy consumption. Although Mobile Cloud Computing (MCC) was proposed to satisfy such needs, it has become inadequate for real-time and resource-intensive applications [4],[5]. Consequently, Mobile Edge Computing (MEC) is proposed as a complement to MCC. It involves deploying MEC servers in the Radio Access Network (RAN) to sink the computing power of cloud servers to the edge of the wireless network where the SMDs are located [6]. However, MEC servers generally offer fewer resources than MCC and may not be sufficient for highly resource-demanding tasks [7].

To address these limitations, the "End-Edge-Cloud" collaborative architecture has gained prominence as an enhancement of both MCC and MEC. By coordinating resources among SMDs, edge servers (ESs), and cloud data centers (CDCs), this architecture more effectively handles diverse workloads while providing notable gains in latency and energy consumption [8],[9]. Nonetheless, because the dynamic and distributed nature of the "End-Edge-Cloud" collaborative architecture increases the uncertainty of task offloading decisions, optimizing energy consumption and delay remains a significant challenge. To address these challenges, some studies [10],[11],[12] have focused on minimizing system delay, whereas others [13],[14],[15] have concentrated on reducing energy consumption. Different from previous research, this paper formulates the task offloading problem in the "End-Edge-Cloud"

ISSN: 2710-0588

collaborative architecture as a multi-objective optimization problem, simultaneously addressing energy consumption and delay. Furthermore, the optimal solution is derived through a multi-objective evolutionary algorithm without imposing preset weights on the objectives.

Specifically, the main contributions of this paper are as follows:

1) In order to better deal with the tasks of mobile devices, this work models a multi-objective optimization task offloading problem that considers the delay and energy consumption in the "End-Edge-Cloud" collaborative architecture.

2) By introducing an adaptive solution acceptance strategy driven by charging dynamics and the Levy flight mechanism of POA. This work propose IPOA,which is not easy to fall into the local optimum, and it further accelerayes the speed of convergence to the global optimum.

3) The simulation results show that the IPOA converges to the approximate optimal solution with high efficiency and effectiveness of the optimization objectives, and it also reflects the necessity of "End-Edge-Cloud" collaborative architecture.

2. Related Work

In this section, relevant studies are examined in terms of both task latency and energy efficiency in MEC.

Yang et al.[16] proposed an OONS strategy based on an MDP model, obtaining the optimal offloading time through VIA. Chen et al.[17] proposed an algorithm to optimize the average task delay, achieving delay minimization. Wang et al.[18] proposed a distributed computation offloading algorithm using the MFG, achieving delay minimization. Wang et al.[19] proposed a joint optimization algorithm, which significantly reduces total latency and demonstrates strong performance. Zhu et al.[20] jointly optimized the NOMA-based transmission duration and workload offloading allocation among edge computing servers, aiming to minimize task computation delay. Deng et al.[21] proposed an autonomous partial offloading system for delay-sensitive computation tasks in MEC systems, achieving minimum-delay offloading services to enhance QoS. Wu et al.[22] proposed a UAV MEC system based on URLLC offloading, optimizing UAV positioning to significantly reduce computation delay. Chen et al.[23] proposed a DAG-based multi-task computation offloading strategy to reduce average energy-time cost for all users. Pendo et al.[24] proposed a hybrid method based on PSO and the GWO, reducing overall energy consumption. Zhao et al.[25] proposed a Lyapunov-based online energy consumption optimization algorithm, which achieves better energy performance while meeting system constraints. Liu et al.[26] proposed a deep reinforcement learning approach based on multi-agent proximal policy optimization to reduce energy consumption. Jing et al.[27] proposed a particle swarm optimization method based on genetic learning, significantly reducing the total energy consumption of the entire system. Chen et al.[28] proposed an energy-efficient offloading strategy based on an adaptive particle swarm optimization algorithm, achieving notable energy reduction.

In general, the above studies do not focus on the issue of task offloading to optimize task latency and energy consumption at the same time. This paper investigates task offloading in the "End-Edge-Cloud" collaborative architecture and introduces an enhanced computational offloading method aimed at minimizing both task latency and energy consumption. Finally, a new optimization algorithm, IPOA, is proposed.

ISSN: 2710-0588

3. System Model and Problem Formalation

This section introduces the "End-Edge-Cloud" collaborative architecture, along with the communication and computational models. For the convenience of the readers, the symbols that will be used are presented in Table 1.

Table 1. Notations For System Model.

Symbol	Meaning
M	Set of smart mobile devices
E	Set of edge nodes
L	Set of tasks
$R_{i,u}$	Whether SMD i is in the service area of ES u
B	Wireless channel bandwidth
p_i	Transmission power of SMD i
p_i^{wait}	Wait power consumption of SMD i
$D_{i,u}$	Distance between SMD i and ES u
r_u	Coverage radius of ES u
α	Path loss factor
$g_{i,u}$	Wireless channel gain between SMD i and ES u
B_0	Background noise
$T_{i,j}$	Task j of SMD i
X	Offloading decision set for all SMDs computing tasks
$x_{i,j}$	Offloading decision for the computation task $T_{i,j}$
$a_{i,j}$	Input data size of task $T_{i,j}$
$b_{i,j}$	Total number of CPU cycles required for task $T_{i,j}$
$c_{i,j}$	Completion deadline of computation task $T_{i,j}$
f_i	Computing capability provided by SMD i
e_i	Energy consumption per CPU cycle by SMD i
$r_{i,j}$	Transmission rate of computation task $T_{i,j}$
$p_i^{\max}$	Maximum power that SMD i can provide
F_u	Computing capability provided by ES u
$f_u^{i,j}$	ES u provides computing capability for task $T_{i,j}$
f_c	Available computational capacity in the CDCs
$t_{i,j}$	Latency of computation ES u
W_c	Limited link bandwidth
$Npop$	Size of population
exp	Exponential function
$Iter$	Current evolutionary generation
$MaxIter$	Maximum evolutionary generation
$Puma_{male}$	Optimal solutions for population
Dim	Dimension of the problem

ISSN: 2710-0588

3.1. Overview of System Architecture

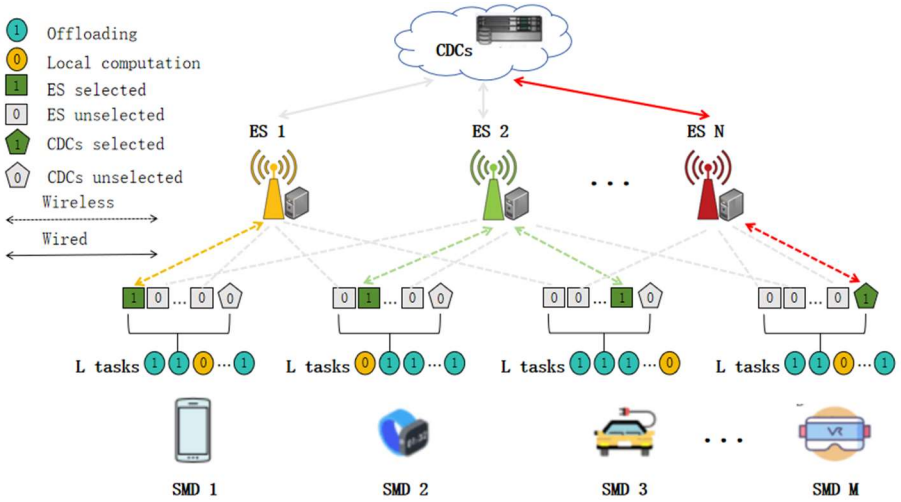

Fig. 1 "End-Edge-Cloud" collaborative architecture.

As shown in Fig.1, we consider an "End-Edge-Cloud" collaborative architecture, consisting of a CDCs, n ESs and m SMDs. The set of SMDs is denoted as $M=\{1,2,3,...,m\}$, where m denotes the $m-th$ SMD, and the set of ESs is denoted as $E=\{1,2,3,...,n\}$, where n denotes the $n-th$ ES, and each ES can communicate with SMDs or CDCs through neighboring base stations. This work argue that SMDs are capable of running multiple applications simultaneously and issuing multiple task processing requests for parallel processing. Assume that the number of computational tasks submitted by each SMD is denoted as $L=\{1,2,3,...,l\}$. For any SMD $i\in M$, the $j-th$ task submitted by it is defined as $T_{i,j}$. The processing of each computing task can be categorized into three ways: 1) local processing on SMDs, 2) offloading to ESs for processing, and 3) offloading to CDCs for processing. In general, ESs can cover a specific geographic area, and the SMDs in that area can obtain services through ESs. In the "End-Edge-Cloud" collaborative architecture, to ensure that the service area of ESs can cover more SMDs, this paper assumes an overlap in the service area of neighboring ESs. SMDs within the overlapped region can be randomly selected from multiple ESs for service. When choosing to offload computational tasks to ESs for processing, it is necessary to first determine whether the SMD i to which the computational task belongs is located in the service area of the ES u, denoted by the binary variable $O_{i,u}\in\{0,1\}$, $i\in M$, $u\in E$. If $O_{i,u}=1$, which indicates that the SMD i is in the service area of ES u, the node can be used as an offloading decision for all computational tasks of the ES u and vice versa. The SMD i optional offloading strategy vector can be expressed as $O=\{O_{i,0},O_{i,1},...,O_{i,n},O_{i,n+1}\}$. $O_{i,0}=1$ and $O_{i,n+1}=1$ since the computational tasks for each SMD can be processed locally or offloaded to CDCs. To determine whether the SMDs are within the service area of the ESs, this study assume that the area covered by the ESs is a circle whose area radius u is denoted as r_u (defined according to the computational power), and denote the locations of the ESs and SMDs by their coordinates. Calculate the distance $R_{i,u}$ between the SMD i and the ES u.The coordinates of the SMD i and the ES u are the $O_{i,u}$ distances from the (x_i,y_i) and (x_u,y_u) points to, respectively, which can be defined as follows:

$$O_{i,u}=\begin{cases}1, & if\ R_{i,u}\le r_u,\\ 0, & if\ R_{i,u}\ge r_u.\end{cases} \tag{1}$$

ISSN: 2710-0588

$$R_{i,u} = \sqrt{(x_u - x_i)^2 + (y_u - y_i)^2}. \quad (2)$$

3.2. Communication Model

A communication model for wireless access in "End-Edge-Cloud" collaborative architecture. Each SMD can send requests to ESs that can cover and provide services for it, and each ES has a dedicated base station to receive transmission-related requests. SMDs need to send requests to CDCs through ESs.

The offloading decision for the computational task $T_{i,j}$ of SMD i is denoted as $x_{i,j} \in \{0,1,...,n+1\}$,where $i \in M$, $j \in L$. In the case of all SMDs for which the computational task offloading decision vector $X = \{x_{1,1}, x_{i,j}, ..., x_{1,l}, x_{2,1}, x_{i,j}, ..., x_{m,l}\}$ is known, the uplink transmission rate of the task to the ESs via the wireless channel at the time of offloading can be computed according to Shannon's formula [29].

Assuming that the computational task $T_{i,j}$ is offloaded to ES u for processing or through ES u to CDCs for processing, the transfer rate of the computational task $T_{i,j}$ can be defined as:

$$r_{i,j} = B \log_2(1 + \frac{p_i g_{i,u}}{B_0}). \quad (3)$$

where B is the wireless channel bandwidth, p_i is the transmit power of SMD i, the wireless channel gain between SMD i and ES u is $g_{i,u}$, and B_0 is the background noise, $g_{i,u} = D_{i,u}^{-\alpha}$, $D_{i,u}$ is the distance between SMD i and ES u, and α is the path loss factor.

3.3. Computational Model

Each computational task can choose to be processed locally, offloaded to ESs covering its range, or offloaded to CDCs for processing. This work assume that a set of computation tasks are submitted by SMD i, denoted as $T_i = \{T_{i,1}, T_{i,2}, ..., T_{i,l}\}$, the computational tasks $T_{i,j} = \{a_{i,j}, b_{i,j}, c_{i,j}\}$ are submitted by SMD i. Specifically, $a_{i,j}$ denotes the size of the input data involved in the computational task $T_{i,j}$, $b_{i,j}$ denotes the total number of CPU cycles required to complete the computational task $T_{i,j}$, and $c_{i,j}$ denotes the deadline for the computational task $T_{i,j}$ to be completed. Next, the calculation of the delay and energy consumption of the computational tasks in different computing environments is discussed.

Local computation: For the case where the computational tasks of SMDs are processed locally, this paper first define the local computational power provided by SMDs and the power consumption for local computation, which are different for each SMD, respectively. It is assumed that the computational power provided by an SMD i for a task $T_{i,j}$ is f_i (in CPU cycles/second). It follows that the execution latency of the computational task $T_{i,j}$ in local processing can be defined as:

$$t_{i,j}^{local} = \frac{b_{i,j}}{f_i}. \quad (4)$$

Second, the energy consumption of the computational task $T_{i,j}$ in local processing can be defined as:

ISSN: 2710-0588

$$e_{i,j}^{local} = b_{i,j} \cdot e_i. \tag{5}$$

where e_i denotes the energy consumed by SMD i per CPU cycle. According to the measurement method in [30], $e_i = 5 \cdot 10^{-27} \cdot (f_i)^2$.

Edge computing: In the case of edge computing, SMDs must first send computation tasks to ESs that can serve them via wireless transmission. Then, ESs process the computation tasks instead of SMDs. Therefore, it is necessary to consider the transmission delay and energy consumption of the computation task, as well as the execution delay and energy consumption of the ESs to process the computation task.

Based on the transmission rate $r_{i,j}$ in the communication model and the size of the input data involved in the computational task $T_{i,j}$, the data transmission delay can be defined as:

$$t_{i,j,trans}^{edge} = \frac{a_{i,j}}{r_{i,j}}. \tag{6}$$

At the same time, data transmission energy consumption can be defined as:

$$e_{i,j,trans}^{edge} = p_i \cdot \frac{a_{i,j}}{r_{i,j}}. \tag{7}$$

Similar to SMDs, we consider different ESs with different total computing power. Assuming that the total computational power provided by ES u is F_u (in CPU cycles/second), define the computational power assigned by ES u to the computational tasks $T_{i,j}$ as $f_u^{i,j}$. The execution delay of task $T_{i,j}$ is defined as:

$$t_{i,j,exe}^{edge} = \frac{b_{i,j}}{f_u^{i,j}}. \tag{8}$$

In addition, there is idle energy consumption in SMDs during the period when ESs are processing computational tasks instead of SMDs. Therefore, the execution energy consumption is defined as:

$$e_{i,j,exe}^{edge} = p_i^{wait} \cdot \frac{b_{i,j}}{f_u^{i,j}}. \tag{9}$$

Cloud Computing: When the computational tasks of SMDs are offloaded to CDCs for processing, the SMDs first transmit the computational tasks to the ESs that can serve them via wireless links, and then the ESs transmit the computational tasks to the CDCs for processing via wired links. Therefore, it is necessary to consider the data transmission delay and energy consumption for offloading the input data of the computation task for transmission to the ESs, and the transmission delay from the ESs to the CDCs. Assuming that the bandwidth of the wired link

ISSN: 2710-0588

from ESs to CDCs is W_c, the transmission delay of the computational tasks $T_{i,j}$ when it is offloaded to the CDCs is defined as:

$$t_{i,j,trans}^{cloud} = \frac{a_{i,j}}{r_{i,j}} + \frac{a_{i,j}}{\mathrm{W}_c}. \tag{10}$$

The transmission energy consumption from ESs to CDCs is recorded as the transmission time multiplied by the idle energy consumption. Therefore, the transmission energy consumption is defined as:

$$e_{i,j,trans}^{cloud} = p_i \cdot \frac{a_{i,j}}{r_{i,j}} + p_i^{wait} \cdot \frac{a_{i,j}}{\mathrm{W}_c}. \tag{11}$$

Define the computational power of CDCs that can be allocated to the computational task $T_{i,j}$ as f_c, and it is known that the total number of CPU cycles required by the computational task $T_{i,j}$ is $b_{i,j}$. Therefore, the execution time of the computational task in the CDCs can be defined as:

$$t_{i,j,exe}^{cloud} = \frac{b_{i,j}}{f_c}. \tag{12}$$

In the same way that ESs handle computational tasks, SMDs also have idle energy consumption. Therefore, the execution energy consumption is defined as:

$$e_{i,j,exe}^{cloud} = p_i^{wait} \cdot \frac{b_{i,j}}{f_c}. \tag{13}$$

Similar to the study in [31], since the amount of data in the result of task execution is much smaller than the amount of data in the task input, the result return downlink transmission delay is often negligible. Based on the above formulas and the offloading decisions of the computational tasks, the delay and energy consumption of the computational tasks can be defined as follows:

$$t_{i,j} = \begin{cases} t_{i,j}^{local}, & if\ x_{i,j} = 0, \\ t_{i,j,trans}^{edge} + t_{i,j,exe}^{edge}, & if\ x_{i,j} = 1,2,...,n, \\ t_{i,j,trans}^{cloud} + t_{i,j,exe}^{cloud}, & if\ x_{i,j} = n+1. \end{cases} \tag{14}$$

$$e_{i,j} = \begin{cases} e_{i,j}^{local}, & if\ x_{i,j} = 0, \\ e_{i,j,trans}^{edge} + e_{i,j,exe}^{edge}, & if\ x_{i,j} = 1,2,...,n, \\ e_{i,j,trans}^{cloud} + e_{i,j,exe}^{cloud}, & if\ x_{i,j} = n+1. \end{cases} \tag{15}$$

ISSN: 2710-0588

Since each SMD has multiple computational tasks to be processed, the total latency and total energy consumption of SMDs is the sum of all computational tasks.

3.4. Problem Formula

In the case of multiple SMDs, ESs may simultaneously receive task offloading requests from several SMDs. However, due to the limited computational power of ESs, it may not be possible to guarantee the optimal level of offloading latency and energy consumption for each SMDs. Therefore, we propose to optimize the overall latency and energy consumption of all SMDs. We formulate the offloading problem for mobile device computing tasks with two optimization objectives to reduce the overall latency and the overall energy consumption of the tasks. The task offloading problem can be formulated as:

$$\begin{aligned} \min \quad & T(X) = \sum_{i=1}^{\mathrm{m}} \sum_{j=1}^{l} t_{i,j}, \\ & E(X) = \sum_{i=1}^{\mathrm{m}} \sum_{j=1}^{l} e_{i,j}. \end{aligned} \tag{16}$$

The objective function can be defined as:

$$Minimize \qquad F = T(X) + E(X). \tag{17}$$

s.t.

$$O_{i,u} = 1, i \in M, u \in N. \tag{18}$$

$$\sum_{x_{i,j}=u} f_u^{i,j} = f_u \le F_u, u \in N. \tag{19}$$

$$t_{i,j} \le c_{i,j}, i \in M, j \in L. \tag{20}$$

$$\sum_{j=1} e_{i,j} \le p_i^{\max}, i \in M. \tag{21}$$

In Eq.16 calculates the total latency and energy consumption for completing all computing tasks. The constraints include the coverage condition in Eq.18, the computational resource allocation limit in Eq.19, the maximum execution time limit in Eq.20, and the energy consumption limit in Eq.21.

4. Task Offloading Algorithm

In this section, a solution to the previously identified multi-objective optimization problem is proposed, utilizing the IPOA. The POA is recognized for its robust search capabilities and high solution rate compared to other intelligent algorithms. However, POA is limited by its global search ability and solution accuracy during the optimization process. To address these

ISSN: 2710-0588

limitations, a task offloading algorithm based on IPOA is developed by integrating various enhancement strategies into the POA.

4.1. Puma Optimizar Algorithm

The POA is a meta-heuristic algorithm inspired by the intelligence and life of the puma [32]. It proposes unique and powerful mechanisms at each stage of exploration and development that improve the performance of the algorithm for various optimization problems. In addition, a hyper-heuristic phase-change intelligence mechanism is proposed, enabling phase-change operations during the optimization process to balance the two phases effectively.

(1) Exploration phase

In POA, pumas will follow the following strategy to find food can be formulated as:

$$Z_{i,G}=\begin{cases} R_{Dim}\cdot(Ub-Lb)+Lb \quad rand>0.5, \\ X_{a,G}+G\cdot(X_{a,G}-X_{b,G})+G\cdot \\ (((X_{a,G}-X_{b,G})-(X_{c,G}-X_{d,G}))+ \\ ((X_{c,G}-X_{d,G})-(X_{e,G}-X_{f,G}))) \quad rand\le 0.5. \end{cases} \tag{22}$$

$$G=2\cdot rand-1. \tag{23}$$

where , Ub and Lb are the lower and upper bounds of the problem and R_{Dim} are randomly generated numbers in the range of 0 and 1 and are in the dimensions of the problem. $rand$ is also a randomly generated number between 0 and 1. $X_{a,G}$, $X_{b,G}$, $X_{c,G}$, $X_{d,G}$, $X_{e,G}$ and $X_{f,G}$ are solutions in the whole population that are Faces are randomly selected. G is computed using Eq.23.

$$X_{new}=\begin{cases} Z_{i,G}, \ \ j=j_{rand} \ or \ rand\le U, \\ X_{a,G}, otherwise. \end{cases} \tag{24}$$

where, $Z_{i,G}$ is a solution generated using Eq.22. j_{rand} is a randomly generated integer in the range of issue dimensions.

$$NC=1-U. \tag{25}$$

$$p=\frac{NC}{Npop}. \tag{26}$$

$$\begin{cases} X_i=X_{new}, \ \ if \ \mathrm{Cos}tX_{new}<\mathrm{Cos}tX_i, \\ U=U+p, \ \ otherwise. \end{cases} \tag{27}$$

In each iteration, according to the condition in Eq.27, the number of dimensions that are replaced by new solutions increases, and this increase is done using Eqs.(25-27). In Eq.26 $Npop$ is the total number of Pumas. Improving the solution is according to the condition in Eq.27 in which only the solutions' dimensions are updated if this condition is met.

ISSN: 2710-0588

(2) Exploitation phase
1) fast-running strategy

$$X_{new} = \frac{(\frac{mean(Sol_{total})}{Npop}) \cdot X_1^r - (-1)^\beta \times X_i}{1 + (\alpha \cdot rand\ )}. \tag{28}$$

where, $mean(Sol_{total})$ is the mean of all solutions. X_1^r is a randomly chosen solution from the entire population, and β is 0 or 1. X_i is the current solution in the current iteration. α is static parameters that must be tuned before the optimization procedure.
2) ambush strategy
Simulate pumas making short leaps toward other pumas' prey to generate new solutions:

$$X_{new} = Puma_{male} + (2 \cdot rand) \cdot \exp(rand_n) \cdot X_2^r - X_i. \tag{29}$$

$$X_2^r = round(1 + (Npop - 1) \cdot rand). \tag{30}$$

where, $Puma_{male}$ is the best solution for the entire population.
$rand_n$ is randomly generated number in the issue dimension, and X_2^r is a randomly chosen solution that is selected based on Eq.30.
Simulate pumas making long jumps towards the best Puma prey to generate new solutions:

$$X_{new} = (2 \cdot rand\ ) \cdot \frac{(F_1 \cdot R \cdot X(i) + F_2 \cdot (1 - R) \cdot Puma_{male})}{(2 \cdot rand\ \ -1 + rand_n)} - Puma_{male}. \tag{31}$$

where, R, $F1$, and $F2$ are calculated by Eqs.(32–34), respectively.

$$R = 2 \cdot rand - 1. \tag{32}$$

$$F_1 = rand_n \cdot \exp(2 - Iter \cdot (\frac{2}{MaxIter})). \tag{33}$$

where, $Iter$ depicts the present iteration number and $MaxIter$ signifies the total number of iterations to perform the optimization
operation.

$$F_2 = w \times (v)^2 \cdot \cos((2 \times rand\ ) \cdot w). \tag{34}$$

where, w, v are calculated by Eqs.(35–36).

$$w = rand_n. \tag{35}$$

ISSN: 2710-0588

$$v = rand_n. \tag{36}$$

4.2. Multi-Strategy Improved Puma Optimization Algorithm

1) Levy Flight

The complexity of the POA, particularly in the intelligent selection phase, leads to increased execution time and memory usage. To address this issue, we introduced the Levy flight strategy. Levy flight is a random walk model where step sizes follow a Levy distribution with a long-tail characteristic, enhancing the algorithm's global search capability and effectively avoiding local optima. Through this strategy, POA is able to accelerate convergence and reduce execution time. Additionally, Levy flight optimizes search paths, eliminating unnecessary computational steps and reducing memory consumption. As a result, the Levy flight strategy significantly improves POA's efficiency and scalability, successfully overcoming the performance bottlenecks caused by increased complexity and variable numbers. By adding Levy flight operations to the search for food in the exploration phase, the newly generated solutions are respectively:

$$Z_{i,G} = \begin{cases} R_{Dim} \cdot (Ub - Lb) + Lb \;\; rand \;\; > 0.5, \\ X_{a,G} + G \cdot abs(Ly(i,:)) \cdot (X_{a,G} - X_{b,G}) + \\ G \cdot abs(Ly(i,:)) \cdot (((X_{a,G} - X_{b,G}) - (X_{c,G} - X_{d,G})) + \\ ((X_{c,G} - X_{d,G}) - (X_{e,G} - X_{f,G}))) \;\; rand \;\; \le 0.5. \end{cases} \tag{37}$$

During the development phase, the Levy flight strategy improves on the short-range jumping behavior of the ambush strategy, allowing cougars to approach other cougars' prey more efficiently, thus generating new solutions.

$$X_{new} = Puma_{male} + (2 \cdot rand \;) \cdot \exp(rand_n) \cdot abs(Ly(i,:)) \cdot X_2^r - X_i. \tag{38}$$

where , $Ly()$ is the Levy flight function as shown in Eq.39.

$$Ly = \frac{u_{Npop \cdot Dim}}{|\, v_{Npop \cdot Dim} \,|^{\frac{1}{\varsigma}}}. \tag{39}$$

where, $u_{Npop \cdot Dim} \sim \mathrm{N}(0, \sigma_u)$, and $v_{Npop \cdot Dim} \sim \mathrm{N}(0, \sigma_v)$. σ_u is calculated from Eq.40, $\sigma_v = 1$.

$$\sigma_u = \left(\frac{\Gamma(1+\varsigma) \cdot \sin(\frac{\pi \cdot \varsigma}{2})}{\Gamma(\frac{1+\varsigma}{2}) \cdot \varsigma \cdot 2^{\frac{\varsigma-1}{2}}}\right)^{\frac{1}{\varsigma}}. \tag{40}$$

where, ς is a parameter used to stabilize the Levy flight.

2) Adaptive solution acceptance strategy driven by charging dynamics (ASAS-CD)

In the exploration phase of POA, the initial value of parameter U is set too low, leading to insufficient solution space exploration. As the number of iterations increases, the value of U gradually grows, making it difficult for the algorithm to escape local optima in the early stages.

ISSN: 2710-0588

When the performance of a new solution is not superior to the current solution, the algorithm cannot update the solution set, thus missing potential high-quality solutions. To solve this problem, we propose an adaptive solution acceptance strategy based on the dynamic phenomenon of capacitor charging. This strategy simulates the nonlinear variation of current and voltage with iterations during capacitor charging. It dynamically adjusts the acceptance probability of new solutions based on the cost differences, thereby enhancing the solution set's diversity and exploration ability, effectively preventing the algorithm from getting trapped in local optima during the early stages.

To more accurately describe the dynamic changes of current and voltage with iterations and effectively integrate them into the solution acceptance strategy, a mathematical model based on charging dynamics is developed. This model introduces parameters such as the nonlinear function of current variation over iterations, solution cost differences, and internal resistance, which allow for flexible adjustment of the solution acceptance probability at different stages. It ensures sufficient exploration of the solution space in the early stages of the algorithm while gradually converging to the global optimal solution in the later stages. Specifically, the acceptance probability P of this strategy is defined as:

$$P = \exp(-\frac{\mathrm{Cos}\,tX_{new} - \mathrm{Cos}\,tX_i}{k \cdot I_t \cdot V_t}). \tag{41}$$

where, I_t and V_t represent the variations of current and voltage, respectively, and are computed using Eq.(42-43), where k is a constant.

$$I_t = I_0 \cdot e^{-\frac{t}{RC}}. \tag{42}$$

$$V_t = V_{\max} \cdot (1 - e^{-\frac{t}{RC}}). \tag{43}$$

where, R represents the resistance, C represents the capacitance, I_0 is the initial current, $V_{\max}$ is the voltage when the capacitor is fully charged, and t is the current iteration number.

Meanwhile, this strategy can adaptively adjust the acceptance probability of new solutions as the number of iterations increases, complementing the evolution of parameter U and compensating for the insufficient exploration of the solution space caused by the small initial value of U. Through this mechanism, the algorithm achieves a good balance between global exploration and local exploitation, and it gradually converges to the global optimal solution in the later stages, thereby significantly enhancing overall optimization performance and global search capability. The improvements in balancing exploration and exploitation as well as convergence performance provide strong support for the algorithm in finding the global optimal solution. The new solution update scheme is defined as:

$$\begin{cases} X_i = X_{new}, \ if \ \mathrm{Cos}\,tX_{new} < \mathrm{Cos}\,tX_i, \\ \begin{cases} X_i = X_{new}, \ otherwise \ \ if \ \ p > rand_{13}. \end{cases} \\ U = U + p, \end{cases} \tag{44}$$

The process is shown in Table 2.

ISSN: 2710-0588

Table 2. IPOA's task offloading pseudo-code.

Algorithm 1: IPOA's task offloading pseudo-code
Input: $Npop$, $MaxIter$
Output: The location of Puma and its fitness value
1: Initialize the random population
2: Calculate the fitness value of each solution using Eqs.(16-17)
3: Start Unexperienced Phase
4: for $iter \leftarrow 1$ to 3 do
5: Start Exploration Phase
6: for $iter \leftarrow 1$ to $Npop$ do
7: if $rand > 0.5$
8: Update the location vector using Eq.34
9: else
10: Update the location vector using Eq.34
11: Replace the newly solution with the current solution using Eq.38
12: Start Exploitation Phase
13: for $iter \leftarrow 1$ to $Npop$ do
14: if $rand \leq 0.5$
15: if $rand > Q$
16: Update the location vector using Eq.35
17: else
18: Update the location vector using Eq.29
19: else
20: Update the location vector using Eq.26
21: if $fitness\ X_{new} < fitness\ X_i$
22: $X_i = X_{new}$
23: Start Unexperienced Phase
24: for $iter \leftarrow 4$ to $Maxiter$ do
25: if $Score_{Explore} < Score_{Exploit}$
26: Execute lines 5-11 Update the location vector
27: else
28: Execute lines 12-22 Update the location vector
29: Calculate fitness values in $X_{NewBest}$ using Eqs.(16-17)
30: if $fitness\ X_{NewBest} < fitness\ X_{Best}$
31: $X_{best} = X_{NewBest}$
32: return X_{best}

4.3. IPOA Time Complexity

The total time complexity is $O((2T+1)\times N^2 \times N \times D)$, where T is the maximum number of iterations, D is the dimension of the issue, and N is the total number of populations.

5. Experimental Evaluation

In this section, we evaluate the proposed task offloading algorithm through simulation experiments and numerical studies. All tests are performed on a server with the following specifications. Processor specification is Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz 2.29 GHz, RAM size is 64.0 GB. The installed operating system is Windows 11 Professional 64-bit. All simulation experiments were done using Matlab programming language.

ISSN: 2710-0588

5.1. Simulation Setup

In our simulation, the total number of ESs is set to 3. The ESs are distributed according to a given coordinate law and the radius of the coverage area is 50 m, 30 m and 40 m. The total number of SMDs is set to 50, and there are multiple computational tasks to be processed simultaneously by each device. SMDs are randomly allocated within the coverage area of ESs or can be repeatedly covered by multiple nodes. For computational tasks that require SMDs to process, we assume that the input data size of each task is between {500-3000} KB, the total CPU cycles are randomly allocated from {1,2,3,4,5} Gigacycles [33], and the maximum completion time of the task is 2s. The maximum energy consumed by SMDs to accomplish their own tasks is 10w. Considering the heterogeneous computational capabilities of SMDs and ESs, we define the computational capacity that SMDs can provide as {0.5,0.6,0.8,1} GHz [34]. The ESs with high computational power cover a larger area, and the total computational power of the edge nodes is {150, 90, and 120} GHz, respectively. The total computational power of the CDCs is much larger than that of the ESs. The power provided by the CDCs for each task is set to 20 GHz [35]. ESs wireless channel bandwidth B = 10MHz, background noise B_0 = -100dBm [36]. We set the channel gain $g_{i,u} = R_{i,u}^{-\alpha}$, where $R_{i,u}^{-\alpha}$ is the distance between SMD i and ES u, and the path loss factor α = 4 [37].

5.2. Experimental Parameters

We compare the IPOA with:

- Puma Optimizar Algorithm (POA)
- Genetic Algorithm [38] (GA)
- Grey Wolf Optimizer [39] (GWO)
- Artificial Bee Colony Algorithm [40] (ABC)
- Kepler Optimization Algorithm [41] (KOA)
- Differential Evolution Algorithm [42] (DE)

We set the number of iterations to 100. the population size of each algorithm is set to 30. Each algorithm is evaluated by executing the results 30 times. We set the parameters of the comparison algorithms as shown in Table 3.

Table 3. Algorithm Parameter.

Algorithm	Parameter	Value
POA	PF1	0.5
	PF2	0.5
	PF3	0.5
	U	0.2
DE	F	0.4
	CR	0.1
GWO	a	[0,2]
	r1	[0,1]
	r2	[0,1]
KOA	lambda	15
	M0	0.1
	Tc	3
ABC	CN	30
	MN	30
	GN	30
GA	cross	0.6
	mutation	0.01

ISSN: 2710-0588

5.3. Comparison of POA and IPOA

In this section, we focus on analyzing how the proposed algorithms are affected by the improved operation of the individual selection strategy with Levy flights and ASAS-CD. Therefore, we will experimentally determine how the different task sizes and parameters defined in Table 4 affect the performance of POA and IPOA. In this experiment, we use three different measures, including averages of completion time, energy consumption, and fitness.

Table 4. Description Of The Data Set.

No	SM name	No.of tasks	No.of edges	CC in mega	Q_{in} in mega
1	SM_1	600	3	[1,5]	[500,3000]
2	SM_2	800	3		
3	SM_3	1000	3		
4	SM_4	1200	3		
5	SM_5	1400	3		
6	SM_6	1600	3		
7	SM_7	1800	3		
8	SM_8	2000	3		
9	SM_9	2500	3		
10	SM_10	3000	3		

Table 5 shows the classification results of the two algorithms (POA and IPOA) based on the above metrics. We can see from the results that IPOA outperforms POA on all datasets. It scores higher on all performance metrics. The results make clear the role of the added improvement operations in enhancing the behavior of the standard POA.

Table 5. Poa And Ipoa Results Using The Table 4 Dataset.

No	TG name	Avg.Delay		Avg.Energy		Avg.Fitness	
		POA	IPOA	POA	IPOA	POA	IPOA
1	SM_1	0.4191	0.2885	0.2901	0.2178	0.7092	0.5063
2	SM_2	0.5026	0.3205	0.3998	0.3042	0.9025	0.6247
3	SM_3	0.5722	0.3153	0.5660	0.3309	1.1382	0.6462
4	SM_4	0.5672	0.3024	0.4488	0.2730	1.0160	0.5727
5	SM_5	0.5259	0.3067	0.4008	0.2415	0.9267	0.5482
6	SM_6	0.5191	0.3117	0.4752	0.3129	0.9943	0.6246
7	SM_7	0.5507	0.3540	0.5148	0.4008	1.0655	0.7548
8	SM_8	0.5618	0.3184	0.5174	0.2757	1.0793	0.5905
9	SM_9	0.6096	0.3219	0.6198	0.3159	1.2293	0.6378
10	SM_10	0.8561	0.4345	0.7899	0.3048	1.6460	0.7393

Table 5 shows the overall results of the POA and IPOA algorithms across the 10 datasets presented in Fig.2. IPOA's shortest completion time is 3.2739s, outperforming POA's 5.6843s, demonstrating greater efficiency. Additionally, IPOA's energy consumption is less than 3J, while POA's is close to 5J, indicating a significant reduction in energy consumption. IPOA's fitness score is 6.2514, compared to POA's approximately 11, showing better performance in task load distribution and system optimization. These results demonstrate that IPOA has clear advantages in multi-objective optimization, including latency, energy consumption, and fitness, making it a superior task offloading solution.

ISSN: 2710-0588

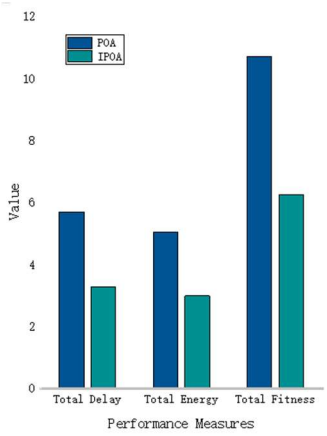

Fig. 2 Measuring total performance values of POA and IPOA using different datasets.

5.4. Comparison of IPOA with Other Algorithms

This subsection compares the performance of the proposed algorithm with GA, GWO, ABC, KOA, and DE using the dataset defined in Table 4. Three performance metrics (latency, energy consumption, and fitness function) are used to evaluate the effectiveness of the algorithms. We run each algorithm 30 times on each dataset. Therefore, we calculate the average value of each performance metric used ($PerM$) using the following formula.

$$avg.PerM = \frac{\sum_{i=1}^{30} PerM_i}{30}. \tag{45}$$

Where $PerM_i$ is the performance metric value obtained by running the algorithm on the dataset.

Fig.3 show that the proposed IPOA algorithm achieves a lower average completion time across all datasets compared to other algorithms, with a smaller increase as the dataset size grows, demonstrating higher efficiency and stability. Fig.4 further confirms this, with IPOA's total average completion time being 4402.3023s, significantly better than the second lowest, the GWO algorithm 5158.9179s, while the GA algorithm has the highest total completion time, reaching 11120.252s. This indicates that IPOA offers significant advantages in multi-objective optimization for task offloading, making it suitable for efficient processing of complex tasks.

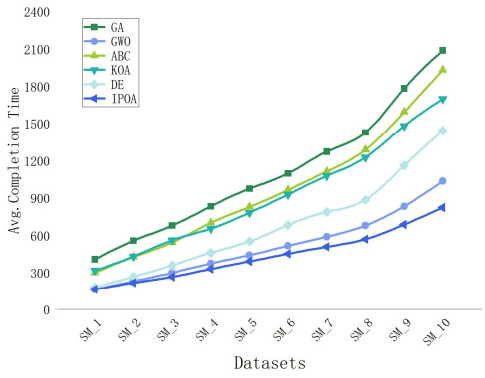

Fig. 3 Completion time results using the adoption dataset.

ISSN: 2710-0588

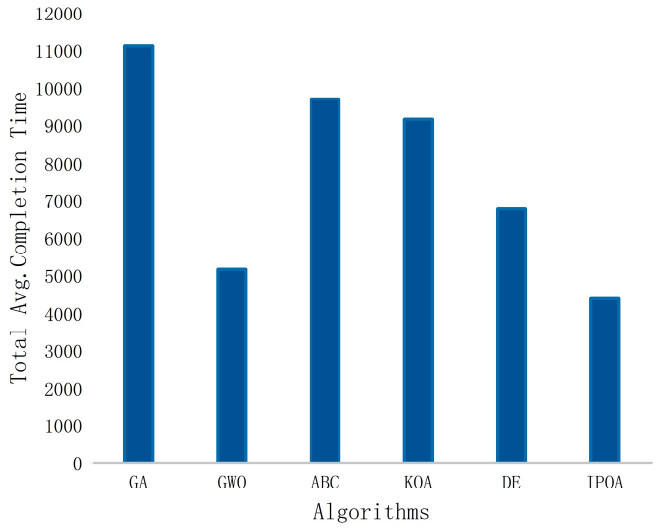

Fig. 4 The total average completion time is obtained by running each comparison algorithm on the adopted dataset.

Fig.5 shows the average energy consumption of different algorithms across multiple datasets. As dataset size increases, IPOA demonstrates the most stable and lowest energy consumption, outperforming other algorithms, especially on large-scale datasets. Fig.6 summarizes the average energy consumption after 30 runs, with IPOA having the lowest at 2193.0238J, followed by GWO at 3216.2887J, and GA with the highest at 11096.509J. Overall, IPOA's outstanding performance in energy control and operational efficiency highlights its significant advantage in multi-objective optimization tasks.

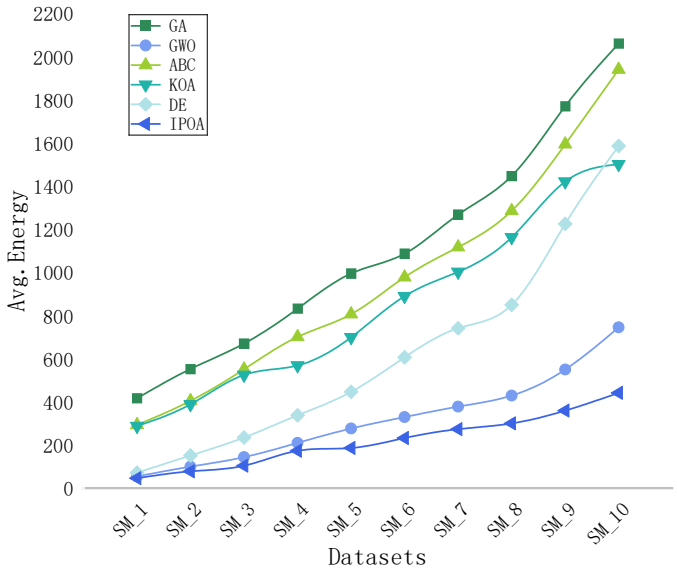

Fig. 5 Energy consumption results using the adoption dataset.

ISSN: 2710-0588

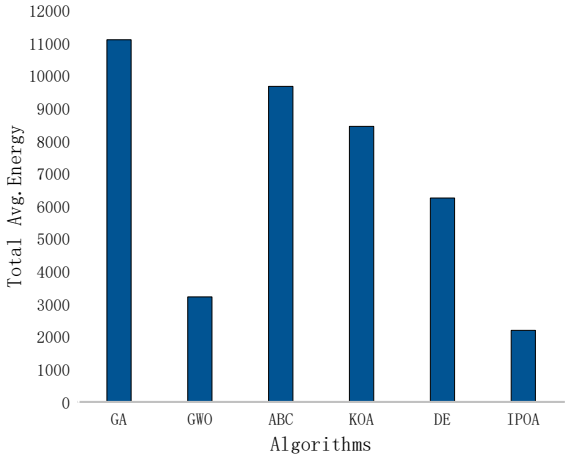

Fig. 6 The total average energy consumption value is obtained by running each comparison algorithm on the adopted dataset.

The fitness value is a key indicator of algorithm performance, reflecting multiple aspects. Fig.7 shows the average fitness values across all datasets, with IPOA consistently maintaining the lowest value, indicating its clear advantages in optimization, global search ability, and convergence characteristics. In contrast, algorithms like GA and ABC show higher fitness values, particularly GA, which may perform poorly due to local optima. Fig.8 further highlights the differences, with IPOA achieving the lowest fitness value at 6808.1261, while GA has the highest at 22216.7613, underscoring IPOA's superiority in optimization efficiency and robustness. In summary, IPOA demonstrates significant competitiveness in solving complex optimization problems.

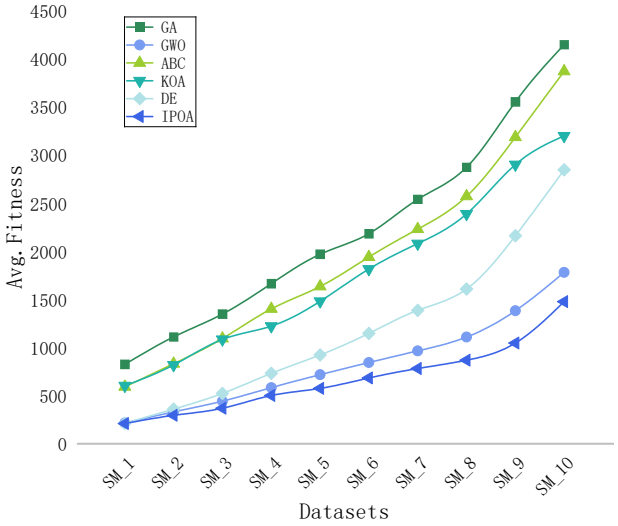

Fig. 7 Results of the fitness function using the adopted dataset.

ISSN: 2710-0588

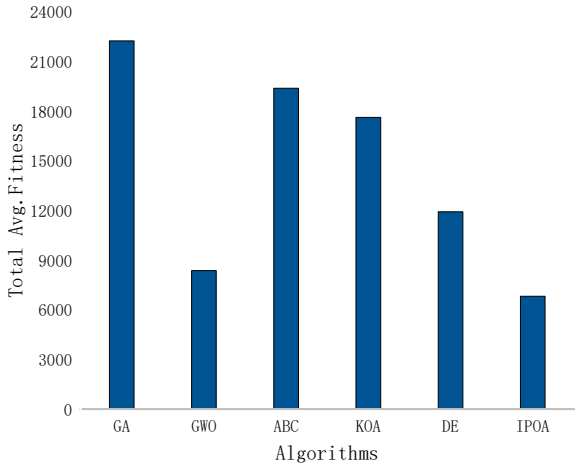

Fig. 8 Run each comparison algorithm on the adopted dataset to get the total average fitness function value.

Fig.9 illustrates the performance of each algorithm across various metrics. GA shows the highest delay at 678.4s, while IPOA has the lowest at 265.1s, confirming its advantage in reducing latency. GWO performs well with a delay of 298.5s but still falls short of IPOA. In terms of energy consumption, IPOA excels with a value of 104J, far outperforming other algorithms, while GA performs the worst at 669.7J. Regarding fitness, IPOA achieves 369.1, significantly better than GWO's 442.6 and GA's 1348.1. ABC and KOA also show higher fitness values at 1093.5 and 1084.4, indicating shortcomings in global optimal search. Overall, IPOA outperforms in delay, energy consumption, and fitness, showcasing superior global search capability and convergence, especially in energy efficiency and latency reduction, further proving its exceptional performance in complex optimization tasks.

From a large number of experiments, we can infer that the algorithm significantly outperforms other metaheuristics in terms of energy savings and delay reduction.

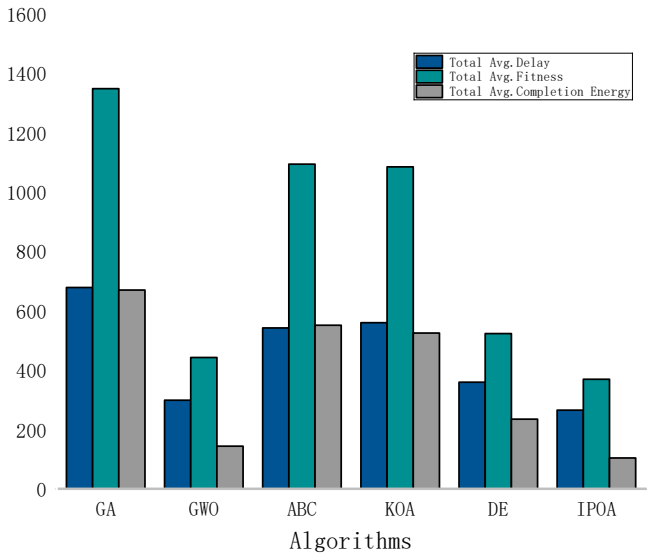

Fig. 9 Total average performance metrics for the SM_3 dataset.

ISSN: 2710-0588

6. Conclusion and Future Work

Efficient task offloading strategies are critical for optimizing performance and enhancing computational efficiency in the "End-Edge-Cloud" collaborative architecture. Given the heterogeneity, dynamics, and multi-constrained nature of task offloading scenarios, traditional algorithms struggle to effectively address complex multi-objective optimization requirements. To this end, this paper proposes IPOA that integrates multi-strategy optimization techniques, significantly enhancing its adaptability to complex task offloading scenarios. Experimental results demonstrate that the proposed algorithm outperforms traditional methods in terms of global search capability, latency optimization, energy efficiency, and stability, exhibiting superior comprehensive performance compared to other state-of-the-art optimization algorithms.

Future research will focus on how to reasonably partition tasks into multiple subtasks and based on the dependencies between subtasks, further develop task offloading optimization algorithms suitable for the End-Edge-Cloud" collaborative architecture.

References

[1] Y. Wang, S. Yang, X. Ren, P. Zhao, C. Zhao, X. Yang, Inductedge: A time-sensitive networking enabled edge-cloud collaborative intelligent platform for smart industry, IEEE Transactions on Industrial Informatics 18 (2021) 2386-2398.

[2] X. Zhou, W. Liang, J. She, Z. Yan, K. L. Wang, Two-layer federated learning with heterogeneous model aggregation for 6g supported internet of vehicles, IEEE Transactions on Vehicular Technology 70 (2021) 5308-5317.

[3] T. Yu, X. Wang, J. Hu, A fast hierarchical physical topology update scheme for edge-cloud collaborative iot systems, IEEE/ACM Transactions on Networking 29 (2021) 2254-2266.

[4] J. Liu, C. Li, Y. Luo, Efficient resource allocation for iot applications in mobile edge computing via dynamic request scheduling optimization, Expert Systems with Applications 255 (2024) 124716.

[5] H. Takabi, J. B. Joshi, G.-J. Ahn, Security and privacy challenges in cloud computing environments, IEEE Security & Privacy 8 (2010) 24-31.

[6] M. Gaggero, L. Caviglione, Model predictive control for energy-efficient, quality-aware, and secure virtual machine placement, IEEE Transactions on Automation Science and Engineering 16 (2018) 420-432.

[7] Z. Tong, X. Deng, J. Mei, B. Liu, K. Li, Response time and energy consumption co-offloading with sira algorithm in cloud-edge collaborative computing, Future Generation Computer Systems 129 (2022) 64-76.

[8] D. Yang, E. Cui, H. Wang, H. Zhang, Eh-edge-an energy harvesting-driven edge iot platform for online failure prediction of rail transit vehicles: A case study of a cloud, edge, and end device collaborative computing paradigm, IEEE Vehicular Technology Magazine 16 (2021) 95-103.

[9] Y. Wei, M. B. Blake, Service-oriented computing and cloud computing: Challenges and opportunities, IEEE Internet Computing 14 (2010) 72-75.

[10] J. Kai, H. Zhou, Y. Yi, W. Huang, Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability, IEEE Transactions on Cognitive Communications and Networking 7 (2020) 624-634.

[11] G. Zhou, L. Zhao, Y. Wang, G. Zheng, L. Hanzo, Energy efficiency and delay optimization for edge caching aided video streaming, IEEE Transactions on Vehicular Technology 69 (2020) 14116-14121.

[12] J. Ren, G. Yu, Y. He, G. Y. Li, Collaborative cloud and edge computing for latency minimization, IEEE Transactions on Vehicular Technology 68 (2019) 5031-5044.

ISSN: 2710-0588

[13] D. Wu, R. Bao, Z. Li, H. Wang, H. Zhang, R. Wang, Edge-cloud collaboration enabled video service enhancement: A hybrid human-artificial intelligence scheme, IEEE Transactions on Multimedia 23 (2021) 2208-2221.

[14] Y. Dai, D. Xu, S. Maharjan, G. Qiao, Y. Zhang, Artificial intelligence empowered edge computing and caching for internet of vehicles, IEEE Wireless Communications 26 (2019) 12-18.

[15] G. Manogaran, S. Muthu, C. X. Mavromoustakis, E. Pallis, G. Mastorakis, Artificial intelligence and blockchain-assisted offloading approach for data availability maximization in edge nodes, IEEE Transactions on Vehicular Technology 70 (2021) 2404-2412.

[16] G. Yang, L. Hou, X. He, D. He, S. Chan, M. Guizani, Offloading time optimization via markov decision process in mobile-edge computing, IEEE Internet of Things Journal 8 (2020) 2483-2493.

[17] C.-L. Chen, C. G. Brinton, V. Aggarwal, Latency minimization for mobile edge computing networks, IEEE Transactions on Mobile Computing 22 (2021) 2233-2247.

[18] D. Wang, W. Wang, H. Gao, Z. Zhang, Z. Han, Delay-optimal computation offloading in large-scale multi-access edge computing using mean field game, IEEE Transactions on Wireless Communications 23 (2024) 1684-1698.

[19] X. Wang, Y. Han, H. Shi, Z. Qian, Latency-oriented joint optimization of computation offloading and resource allocation in d2d-assisted mec system, IEEE Wireless Communications Letters 11 (2022) 1780-1784.

[20] B. Zhu, K. Chi, J. Liu, K. Yu, S. Mumtaz, Efficient offloading for minimizing task computation delay of noma-based multiaccess edge computing, IEEE Transactions on Communications 70 (2022) 3186-3203.

[21] X. Deng, J. Yin, P. Guan, N. Xiong, L. Zhang, S. Mumtaz, Intelligent delay-aware partial computing task offloading for multiuser industrial internet of things through edge computing, IEEE Internet of Things Journal 10 (2021) 2954-2966.

[22] Q. Wu, M. Cui, G. Zhang, F. Wang, Q. Wu, X. Chu, Latency minimization for uav-enabled urlc-based mobile edge computing systems, IEEE Transactions on Wireless Communications 23 (2023) 3929-3311.

[23] J. Chen, Y. Yang, C. Wang, H. Zhang, C. Qiu, X. Wang, Multitask offloading strategy optimization based on directed acyclic graphs for edge computing, IEEE Internet of Things Journal 9 (2021) 9367-9378.

[24] M. P. J. Mahenge, C. Li, C. A. Sanga, Energy-efficient task offloading strategy in mobile edge computing for resource-intensive mobile applications, Digital Communications and Networks 8 (2022) 1048-1058.

[25] Z. Tong, J. Cai, J. Mei, K. Li, K. Li, Dynamic energy-saving offloading strategy guided by lyapunov optimization for iot devices, IEEE Internet of Things Journal 9 (2022) 19003-19015.

[26] W. Liu, B. Li, W. Xie, Y. Dai, Z. Fei, Energy efficient computation offloading in aerial edge networks with multi-agent cooperation, IEEE Transactions on Wireless Communications 22 (2023) 5725-5739.

[27] J. H. Yuan, K. Zhang, M. Zhou, Energy-minimized partial computation offloading for delay-sensitive applications in heterogeneous edge networks, IEEE Transactions on Emerging Topics in Computing 10 (2022) 1401-1414.

[28] X. Chen, J. Zhang, B. Lin, Z. Chen, K. Woter, G. Min, Energy-efficient computing for dnn-based smart iot systems in cloud-edge environments, IEEE Transactions on Parallel and Distributed Systems 33 (2021) 684-697.

[29] C. Wang, W. Li, S. Peng, Y. Qu, G. Wang, S. Yu, Modeling on energy-efficiency computation offloading using probabilistic action generating, IEEE Internet of Things Journal 9 (2022) 20681-20692.

[30] H. Liu, L. Cao, T. Pei, Q. Deng, J. Zhu, A fast algorithm for energy-saving offloading with reliability and latency requirements in multi-access edge computing, IEEE Access 8 (2019) 151-161.

[31] X. Cao, F. Wang, J. Xu, R. Zhang, S. Cui, Joint computation and communication cooperation for energy-efficient mobile edge computing, IEEE Internet of Things Journal 6 (2018) 418-4200.

ISSN: 2710-0588

[32] B. Abdellatif, N. Khodadaadi, S. Barshandeh, P. Trojovský, E. S. Gharechopogh, E.-S. M. El-kenawy, L. Abualigah, S. Mirjalili, Puma optimizer (po): a novel metaheuristic optimization algorithm and its application in machine learning, Cluster Computing 27 (2024) 5235-5283.

[33] X. Zhou, S. Li, Y. Feng, Quantum circuit transformation based on simulated annealing and heuristic search, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 39 (2020) 4683-4694.

[34] M.-H. Chen, M. Dong, B. Liang, Resource sharing of a computing access point for multi-user mobile cloud offloading with delay constraints, IEEE Transactions on Mobile Computing 17 (2018) 2868-2881.

[35] X. Lyu, H. Tian, W. Ni, Y. Zhang, P. Liu, Energy-efficient admission of delay-sensitive tasks for mobile edge computing, IEEE Transactions on Communications 66 (2018) 2603-2616.

[36] S. Yang, A joint optimization scheme for task offloading and resource allocation based on edge computing in 5g communication networks, Computer Communications 160 (2020) 750-768.

[37] L. Yang, H. Zhang, M. Li, J. Guo, H. Ji, Mobile edge computing empowered energy efficient task offloading in 5g, IEEE Transactions on Vehicular Technology 67 (2018) 6398-6409.

[38] M. Xu, G. Feng, Y. Ren, X. Zhang, On cloud storage optimization of blockchain with a clustering-based genetic algorithm, IEEE Internet of Things Journal 7 (2020) 8547-8558.

[39] S. Mirjalili, S. M. Mirjalili, A. Lewis, Grey wolf optimizer, Advances in engineering software 69 (2014) 46-61.

[40] D. Karaboga, An idea based on honey bee swarm for numerical optimization, Technical report, Erciyes University, Engineering Faculty, Computer Engineering Department (2005).

[41] M. Abdel-Basset, R. Mohamed, S. A. A. Elazem, M. Jameel, M. Abouhawwash, Kepler optimization algorithm: A new metaheuristic algorithm inspired by kepler's laws of planetary motion, Knowledge-Based Systems 268 (2022) 110544.

[42] R. Storn, Differential evolution: a simple and efficient adaptive scheme for global optimization over continuous spaces, Technical report, International Computer Science Institute 11 (1995).