Intelligent Automation & Soft Computing
DOI: 10.32604/iasc.2023.032883
Article

Tech Science Press

E-MOGWO Algorithm for Computation Offloading in Fog Computing

Jyoti Yadav* **and Suman**

CSE Department, Deenbandhu Chhotu Ram University of Science and Technology, Murthal, 131039, India
*Corresponding Author: Jyoti Yadav. Email: yadav.jyoti007@gmail.com
Received: 01 June 2022; Accepted: 08 July 2022

Abstract: Despite the advances mobile devices have endured, they still remain resource-restricted computing devices, so there is a need for a technology that supports these devices. An emerging technology that supports such resource-constrained devices is called fog computing. End devices can offload the task to close-by fog nodes to improve the quality of service and experience. Since computation offloading is a multiobjective problem, we need to consider many factors before taking offloading decisions, such as task length, remaining battery power, latency, communication cost, etc. This study uses the multiobjective grey wolf optimization (MOGWO) technique for optimizing offloading decisions. This is the first time MOGWO has been applied for computation offloading in fog computing. A gravity reference point method is also integrated with MOGWO to propose an enhanced multiobjective grey wolf optimization (E-MOGWO) algorithm. It finds the optimal offloading target by taking into account two parameters, i.e., energy consumption and computational time in a heterogeneous, scalable, multi-fog, multi-user environment. The proposed E-MOGWO is compared with MOGWO, non-dominated sorting genetic algorithm (NSGA-II) and accelerated particle swarm optimization (APSO). The results showed that the proposed algorithm achieved better results than existing approaches regarding energy consumption, computational time and the number of tasks successfully executed.

Keywords: Fog computing; computation offloading; computational time; metaheuristic; grey wolf optimization

1 Introduction

Today's devices have an array of sensors embedded to continuously monitor their surroundings in real-time. The internet of things (IoT) devices generally has inadequate battery power, computational capability, and memory capacity, which leads to poor performance and reduces the quality of service (QoS) for computational-intensive applications [1]. Cisco predicts that 500 billion IoT devices will be in use by 2025 [2]. The enormous increase in connected devices and their demands enforce a gigantic paucity in communication networks and computational resources [3]. Computation offloading (CO) is used to improve the performance of IoT devices by executing computation-intensive parts of the application on remote machines or servers [4]. For CO, the end devices send computational tasks to the remote server over resource-limited communication channels such as long-term evolution (LTE), wireless channels, 5G

This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

systems etc [5]. Then the remote server executes the tasks and returns the results to the IoT devices. Using this technique, the resource-intensive tasks are executed in coordination with the offloading server. Therefore, CO strategy design is critical for selecting the right offloading server. During offloading decisions, we have to deal with many questions such as when, what, why, and where to offload [6]. Suppose we choose the where factor, the target can be a fog or cloud layer. When we offload to the cloud, the task is completed, and the cloud server returns the results. However, new research indicates that offloading to the cloud is never a good idea when some delay-sensitive applications must be completed by a certain deadline. Moreover, cloud computing (CC) has higher communication costs and transmission delay [7]. For latency-sensitive tasks, the availability of computational resources via fog computing (FC) is advantageous [8]. It offers a distributed infrastructure, which results in faster response and lower latency to application requests [9]. The computational capability and resources of fog nodes are lesser compared to CC. Hence the offloading destination should be wisely selected based on different conflicting parameters like delay, energy consumption, computational time, throughput etc. Only properly offloaded tasks can result in a higher quality of experience and services. Developing a task offloading scheme in the FC environment is a little more difficult than in normal CC, where an IoT device simply selects whether or not to transmit a task to the cloud [10]. In FC, the number of offloading destinations increases in proportion to the number of fog nodes. Moreover, fog nodes have limited computing capabilities and resources compared to cloud resources [11]. Furthermore, earlier studies assumed single-user single fog node scenarios, single-user single with single cloud server scenarios, or multi fog server with single cloud server scenarios, which are impractical. Few studies have focussed on task offloading in multi-user, multi-fog, and cloud server scenarios.

The full potential of fog computing can be realized using the offloading technique. Moreover, the offloading problem is np hard, and its complexity increases with the number of tasks. In this article, we use the MOGWO technique for the CO problem. MOGWO is integrated with gravity reference points to enhance the performance further. The proposed strategy significantly contributes by identifying the best computing server for every task based on two QoS parameters, i.e., computational time and energy consumption.

The following is a summary of the paper's main contribution:

- We investigate a multiobjective CO problem for a heterogeneous multi-user, multi-fog server and cloud environment.
- MOGWO algorithm is used to select the best offloading server on the basis of minimum energy consumption and computational time.
- A gravity reference point method is also integrated with MOGWO to improve its performance as it has a better solution spread set.
- Lastly, the simulation of the proposed model and comparison of results with two other multiobjective algorithms have been carried out in a three-layer architecture.

The rest of the article is as follows: Section 2 of this paper details the literature review related to CO using metaheuristic techniques. Section 3 goes over the system model in detail, including the computational and energy models and the problem definition. The proposed algorithm is described in Section 4. Section 5 describes the simulation's findings, and Section 6 describes the paper's conclusion.

2 Literature Review

Various studies have been conducted for CO problems in FC till now. Various authors published review papers [12–16] on task offloading in FC as a research direction. Many researchers use the queuing theory [17–19], game theory [20–22], dynamic programming and clustering techniques [18,23,24] to solve CO

problems. It has been observed that each has separately investigated essential parameters, for example, delay, execution time, service time, communication cost, computational cost, quality of service and energy consumption etc. In reality, the offloading problem is multiobjective and np-hard. As a result, only some researchers have investigated meta-heuristic techniques for CO in FC. However, metaheuristic offloading techniques in FC are new and require extensive augmentation to maximize the QoS and end-user experience. These algorithms are motivated by natural, physical phenomena and animal behaviour [25]. Authors in [26] proposed two swarm intelligence-based schedulers named ant colony optimization (ACO) and particle swarm optimization (PSO) to efficiently balance the workload of IoT devices by taking into account response time and cost over fog nodes. Finally, the authors declared that ACO outperformed PSO and round-robin (RR) in case of response time. Authors in reference [27] proposed adaptive GA-PSO (Genetic algorithms and particle swarm optimization) to find near-optimal scheduling algorithms for scheduling resource-intensive components of an application. The authors combined genetic algorithm (GA) and PSO, where GA is used for exploration, and PSO is used for exploitation for optimal offloading with deadline constraints. GA was also used in [28] to improve response time and load balancing while finding the best destination for the offloaded task. Here queuing model is also utilized to calculate waiting time and service time. Authors in [29] also proposed a multiobjective optimization non-dominated sorting genetic algorithm that finds an optimal offloading approach for all workflow applications. Authors in [30] proposed a fruit-fly-based CO algorithm which improves resource allocation and offloading to gain nominal energy usage.

Most of the problems were either single objectives or converted to a single objective by assigning weight to the parameters. As a multiobjective problem, literature has no sufficient discussion to solve the offloading problem. Also, offloading is a multiobjective problem and its complexity increase with the number of tasks. Thus, developing a multiobjective grey wolf to solve the CO problem is an open issue.

3 System Model

We propose a three-layer system model in our proposed work: IoT devices layer, fog server (FS) layer, and cloud server (CS) layer. The FS layer is at the network's edge and connects end devices with the cloud server. Suppose there are X IoT users $U = \{U_1,\ U_2,\ U_3,\ U_4 \ldots\ U_x\}$. Each IoT user submits a T number of computation-intensive or resource-sensitive tasks. The task set in a time interval δt is represented as $T = \{T_1,\ T_2,\ T_3, \ldots, T_x\}$. Here, we define $T_x = \{L_x,\ D_{x\_in},\ D_{x\_o},\ \tau_x\}$ where L_x represents task length, D_{x_in} represents request size, D_{x_o} represents response size, τ_x represents the maximum delay a task can tolerate. It is also considered that there are M fog servers $FS = \{FS_1,\ FS_2,\ FS_3,\ FS_4,\ \ldots\ FS_M\}$ and N cloud servers $CS = \{CS_1,\ CS_2,\ CS_3,\ CS_4,\ \ldots\ CS_N\}$. Let there are R computation servers deployed in an area $S = \{S_1,\ S_2,\ S_3,\ S_4,\ \ldots\ S_R\}$ where $R\varepsilon\{CS,\ FS\}$. FS have a higher computing capacity than IoT devices, but less than cloud servers $CC_{IoT}^{CPU} < CC_{fog}^{CPU} < CC_{cloud}^{CPU}$. We'll utilize a binary version of CO in which a computational task is either calculated locally or submitted to a remote computing server for processing. A task cannot be divided and must be accomplished entirely on a single server. Our objective is to find the best computational server for tasks to be offloaded based on the multiobjective optimization problem.

3.1 Computation Model

This model is used to compute computational time, which is the time it takes to execute a task. It is determined by the computational capabilities of the remote server or local device and the communication medium's latency. It is calculated using Eq. (1).

$$CT^{u}_{i,j} = ET^{u}_{i,j} + L_{i,j} \quad where \ \text{j}\epsilon\text{S}, \quad \text{i}\epsilon\text{X} \tag{1}$$

Here $CT^{u}_{i,j}$ represents the computational time of task i generated from device u and executed on device j. Here j can be the IoT device itself, or it can be a remote server belonging to set S. $ET^{u}_{i,j}$ represents the execution time of task i, which is calculated as $ET^{u}_{i,j} = L_x / CC^{CPU}_{j}$.

The latency between end device i and the remote computing server j is calculated on the basis of the network's bandwidth and distance.

$$L_{i,j} = PD_{i,j} + SD_{i,j} \ \forall(i,j) \tag{2}$$

Here $PD_{i,j}$ denotes propagation delay and $SD_{i,j}$ denotes serialization delay. The $PD_{i,j}$ is the ratio of distance ($D_{i,j}$) between device i and j and bandwidth ($BW_{i,j}$) of the network i.e., $D_{i,j}/BW_{i,j.}$ Where $D_{i,j} = \sqrt{(X_j - X_i)^2 + (Y_j - Y_i)^2}$ in 2-D area $(X,\ Y)$. However, $SD_{i,j}$ is calculated by dividing the size of the task by the transmission rate i.e., L_x/T_r. Here we consider two cases for the calculation of computational time.

Case1: Local Execution

Here the task is processed at the end device itself, so the computational time is calculated using Eq. (3).

$$\boldsymbol{CT^{u}_{i,j} = ET^{u}_{i,j}} \tag{3}$$

The task is processed locally, so the latency ($L_{i,j} = 0$) is zero.

Case 2: Remote Execution

The computational task is forwarded to the fog or cloud layer depending on the task's latency and resource sensitivity. The computational time required to execute an application on a remote server is calculated using Eq. (4).

$$CT^{r}_{i,j} = ET^{u}_{i,j} + L_{i,j} \tag{4}$$

3.2 Energy Consumption Model

Two factors primarily determine the energy utilization: (i) computational energy, which is consumed when the device is busy in executing the task, while (ii) transmission energy is the energy consumed during to and fro transmission of the task to a remote computational server. The energy consumed during execution is calculated as follows:

Case1: Local Execution

The energy consumed when a task executes at a local device is calculated using Eq. (5).

$$EC^{u}_{i,j} = ET^{u}_{i,j} * \mathrm{E}^{\mathrm{c}}_{\mathrm{r}} \tag{5}$$

where E^{c}_{r} represents the energy consumption rate while executing a task.

Case 2: Remote Execution

The energy consumed when a task executes at a remote device is calculated using Eq. (6).

$$EC^{r}_{i,j} = \frac{D_{x\_in}}{T_r} * \mathrm{E}^{\mathrm{c}}_{\mathrm{r\_up}} + \frac{D_{x\_o}}{T_r} \ * \ \mathrm{E}^{\mathrm{c}}_{\mathrm{r\_down}} + ET^{u}_{i,j} * \mathrm{E}^{\mathrm{c}}_{\mathrm{r}} \tag{6}$$

$EC^{r}_{i,j}$ represents energy consumption when a task executes at the remote device. In Eq. (6) $\mathrm{E}^{\mathrm{c}}_{\mathrm{r\_up}}/\mathrm{E}^{\mathrm{c}}_{\mathrm{r\_down}}$ represents the rate of energy consumption for sending the request and receiving back the results.

3.3 Problem Definition

$$\boldsymbol{Min\ F}(\boldsymbol{x}) = (\boldsymbol{F1}(\boldsymbol{x}),\ \ \boldsymbol{F2}(\boldsymbol{x})) \tag{7}$$

where $F1(x)$ refers computational time

$F2(x)$ refers energy consumption

$$F1(x) = \sum_{i=o}^{x}\left(CT_{i,j}^{u} + CT_{i,j}^{r}\right)$$

$$F2(x) = \sum_{i=o}^{x}\left(EC_{i,j}^{u} + EC_{i,j}^{r}\right)$$

Subject to

$\boldsymbol{C1}$: $\boldsymbol{T_{x,i,j}} = \mathbf{1}, \quad \boldsymbol{T_{x,i,j}}\varepsilon[\mathbf{0},\ \mathbf{1}]$// Constraint is used for binary offloading

4 MOGWO for Computation Offloading Problem

4.1 Multiobjective Optimization Problem

Multiobjective optimization (MOO), which was introduced by Vilfredo Pareto, determines the best solution values for multiple goals [19]. The MOO's use is justified by the fact that complex equations are no longer required for problem-solving. Objective functions here are vectorized. The vector corresponding to an objective function is also a function of the solution vector. Moreover, multiple solutions exist corresponding to each function in the problem definition. The two techniques, i.e., Pareto technique and scalarization, are used for solving the MOO problem. However, these two techniques are different to each other. Suppose we consider different desired outcomes and performance matrices. The Pareto technique then gives a conciliation solution (trade-off) that may be represented as a Pareto front (PF) [31]. The scalarization technique produces a scalar function, which is then implemented using the fitness function. So, to tackle the described problem here, we employed the MOGWO approach.

4.2 GWO Algorithm

GWO is a nature-inspired meta-heuristic approach motivated by the grey wolf's hunting style and social hierarchy [30]. Grey wolves live in packs of five to twelve members. Four rankings are assigned to wolves $(\alpha,\ \beta,\ \delta,\ \omega)$ in a particular pack. α wolf is the decision-makers about hunting, living places etc. The $\beta,\ \delta,\ \omega$ wolves give feedback to the alpha wolf and aid in decision-making. The best candidate for an optimal solution is α. In contrast, β and δ are the second and third best solutions. Grey wolves hunt by encircling their victim and attacking it from all sides. The first phase of encircling the prey is represented using Eqs. (8) and (9)

$$\vec{D} = \left|\overrightarrow{C_v}.\ \overrightarrow{X_P}(t) - \overrightarrow{X_G}(t)\right| \tag{8}$$

$$\overrightarrow{\mathbf{X_G}}(\mathbf{t}+1) = \left|\overrightarrow{\mathbf{X_P}}(\mathbf{t}) - \overrightarrow{\mathbf{A_v}}.\ \vec{\mathbf{D}}\right| \tag{9}$$

where $\vec{D}$ denotes distance-vector, $\vec{X}_P$ indicates the prey's location, t represents the current iteration, $\vec{X}_G$ denotes the grey wolf's location and $\overrightarrow{A_v}$, $\overrightarrow{C_v}$ represents the coefficient vectors calculated using Eqs. (10) and (11):

$$\overrightarrow{A_v} = 2.\vec{a}.\overrightarrow{r_1} - \vec{a} \tag{10}$$

$$\overrightarrow{C_v} = 2.\overrightarrow{r_2} \tag{11}$$

where $\overrightarrow{r_1}$ and $\overrightarrow{r_2}$ are random vectors with value between [0, 1]. The value of $\vec{a}$ decreases from 2 to 0 during the course of the experiment. Other wolves alter their positions based on α, β, and δ wolves. The hunting behaviour of grey wolves is simulated using Eqs. (12)–(18).

$$\overrightarrow{D_\alpha} = \left|\overrightarrow{C_{v1}}.\overrightarrow{X_\alpha} - \overrightarrow{X_G}(t)\right| \tag{12}$$

$$\overrightarrow{D_\beta} = \left|\overrightarrow{C_{v2}}.\overrightarrow{X_\beta} - \overrightarrow{X_G}(t)\right| \tag{13}$$

$$\overrightarrow{D_\delta} = \left|\overrightarrow{C_{v3}}.\overrightarrow{X_\delta} - \overrightarrow{X_G}(t)\right| \tag{14}$$

Position if we follow alpha, then

$$\overrightarrow{X_1} = \left|\overrightarrow{X_\alpha} - \overrightarrow{A_{v1}}.\overrightarrow{D_\alpha}\right| \tag{15}$$

Position if we follow beta, then

$$\overrightarrow{X_2} = \left|\overrightarrow{X_\beta} - \overrightarrow{A_{v2}}.\overrightarrow{D_\beta}\right| \tag{16}$$

Position if we follow delta, then

$$\overrightarrow{X_3} = \left|\overrightarrow{X_\delta} - \overrightarrow{A_{v3}}.\overrightarrow{D_\delta}\right| \tag{17}$$

The final position of the wolf is calculated using Eq. (18).

$$\overrightarrow{X_G}(t+1) = \left(\overrightarrow{X_1} + \overrightarrow{X_2} + \overrightarrow{X_3}\right)/3 \tag{18}$$

The final stage of attacking the prey has been formulated using the vector $\vec{a}$. The value of $\vec{a}$ lies between $[-a,\ a]$, and its value decreases from 2 to 0 during the iteration and is calculated using Eq. (19) as follows:

$$\vec{a} = 2 - t.\frac{2}{maxIter} \tag{19}$$

The *maxIter* represents the total number of iterations, t indicates the current iteration.

4.3 Multiobjective Grey Wolf Optimization Technique

The GWO approach was developed to tackle one optimization problem at a time. Because of this, it cannot be used to solve multiobjective problems directly. MOGWO was proposed in 2016 by Mirjalili et al. [32]. Although it has been used for a variety of real-world problems, it has yet to be utilized to offload issues in FC. It has good exploration and exploitation and has a faster convergence rate with fewer parameter tuning. Therefore MOGWO is used to solve CO problems in FC. Two new mechanisms were integrated into the original GWO. Initially, storage was utilized to store the solutions called archives. Then a leader selection strategy is utilized to select the first, second, and third leaders from the archive.

Here we assume that all IoT devices' probable CO decisions create a single position for the grey wolf as follows:

Step 1: Randomly initialize the wolf population

$$GW = \begin{bmatrix} GW_1 \\ GW_2 \\ GW_3 \\ \cdots \\ GW_n \end{bmatrix} = \begin{bmatrix} GW_{1,1} & .. & GW_{1,d} \\ GW_{2,1} & .. & GW_{2,d} \\ GW_{3,1} & .. & GW_{3,d} \\ .. & .. & .. \\ GW_{n,1} & .. & GW_{n,d} \end{bmatrix}$$

Here n represents the number of wolves, and d represents the dimension

Step 2: Calculate the objective function value of the wolf population. For each wolf, there are two objective functions to evaluate, i.e., $F1$ and $F2$.

$$FW = \begin{bmatrix} F_1(GW_1) & F_2(GW_1) \\ F_1(GW_2) & F_2(GW_2) \\ .. & .. \\ F_1(GW_n) & F_2(GW_n) \end{bmatrix}$$

$$FW = \begin{array}{c|cc} & Time & Energy \\ GW_1 & 2 & 0.7 \\ GW_2 & 3 & 0.6 \\ GW_3 & 3 & 0.7 \\ GW_4 & 4 & 0.5 \\ GW_5 & 5 & 0.45 \\ GW_6 & 5 & 0.6 \end{array}$$

It can be clearly observed that GW_1 dominate GW_3; GW_2 dominate GW_3 and GW_6; GW_4 dominate GW_6; and GW_5 dominate GW_6. Here $\{GW_1,\ GW_2,\ GW_4,$ and $GW_5\}$ form a non-dominating set and are also called Pareto optimal front. However, no clear winner is identified. When two solutions have the same non-dominated rank, the one in a less crowded region is selected.

$$FW = \begin{array}{c|cc} & Time & Energy \\ GW_1 & 2 & 0.7 \\ GW_2 & 3 & 0.6 \\ GW_4 & 4 & 0.5 \\ GW_5 & 5 & 0.45 \end{array} \qquad \begin{array}{l} F_1(GW_{max}) = 5 \\ F_1(GW_{min}) = 2 \\ F_2(GW_{max}) = 0.7 \\ F_2(GW_{min}) = 0.45 \end{array}$$

The crowding degree (CD) of all the non-dominated solutions is computed using Eq. (20).

$$\boldsymbol{CD_i} = \sum_{j=1}^{2} \frac{\boldsymbol{f_j^{i+1}} - \boldsymbol{f_j^{i-1}}}{\boldsymbol{f_j^{max}} - \boldsymbol{f_j^{min}}} \tag{20}$$

where i denotes the ith solution, all the values are arranged in decreasing order of CD and represented as set S^t. The optimal solution is then chosen using the roulette wheel selection method. It does not ensure that the fittest solution will be chosen, but it does increase the probability that it will be chosen [33]. The probability of selection is calculated using Eq. (21).

$$P(F_i|S^t) = \frac{k - i + 1}{\sum_{j=1}^{k} j} \tag{21}$$

where F_i is the destination function of the ith solution, S^t is the set of solutions, and k denotes the number of the S^t.

Algorithm 1: MOGWO Algorithm

1. Randomly initialize the grey wolf population of size n and set it as $GW_p[\ ]$
2. Initialize a, A_v and C_v
3. Compute the objective function value for each search agent
4. $EA_P \leftarrow [\ ]$ // EA_P denotes external archive
5. Find non dominated solution and initialize the archive
6. Choose the best solution from the archive as X_α
7. Temporarily eliminate α from archive to prevent the selection of the same solution
8. Choose second-best solution from the archive X_β
9. Temporarily eliminate β from an archive to avoid selection of the same solution
10. Choose the third-best solution from an archive X_δ
11. Add selected solution back to the archive
12. t = 1
13. **While** ($t < max\ number\ of\ iterations$) **do**
14. **For** each wolf do
15. Update the location of the current wolf by using the Eqs. (12) to (19)
16. End **for**
17. Update a, A_v and C_v
18. $\left(EA_P,\ \overrightarrow{X_P}\right) = CD\_sort(EA_P,\ GW_P,\ capacity)$
19. Select the best solution from storage as X_α
20. temporarily eliminate α from the archive to prevent the selection of the same solution
21. Select second-best solution from the storage X_β
22. temporarily eliminate β from storage to avoid selection of the same solution
23. Select the third-best solution from the storage X_δ
24. Add α and β back to the storage
25. t = t + 1
26. **end while**
27. return EA_P

Algorithm 2: CD_sort

Function : (EA_P, $\overrightarrow{X_P}$) = CD_sort (EA_P, GW_P, capacity)

Input: EA_P, GW_P, capacity

Output: EA_P, $\overrightarrow{X_P}$

1. Add GW_P to EA_P
2. update EA_P with non dominated solution of GW_P based on objective functions in Eq. (7).
3. The solutions in EA_P are sorted by crowding degree using Eq. (20).
4. If size ($EA_P > capacity$) then

(Continued)

Algorithm 2 (continued)

5. Poor crowding degree solutions are excluded
6. End if
7. Then the roulette wheel selection method is used to select the optimal solution using Eq. (21).

4.4 E-MOGWO for Computation Offloading in Fog Computing

Since the solution set of the gravity reference point technique is more evenly distributed, integrating it with MOGWO can boost its performance even more. E-MOGWO is a new hybrid method that combines the gravity reference point technique with MOGWO as shown in Fig. 1.

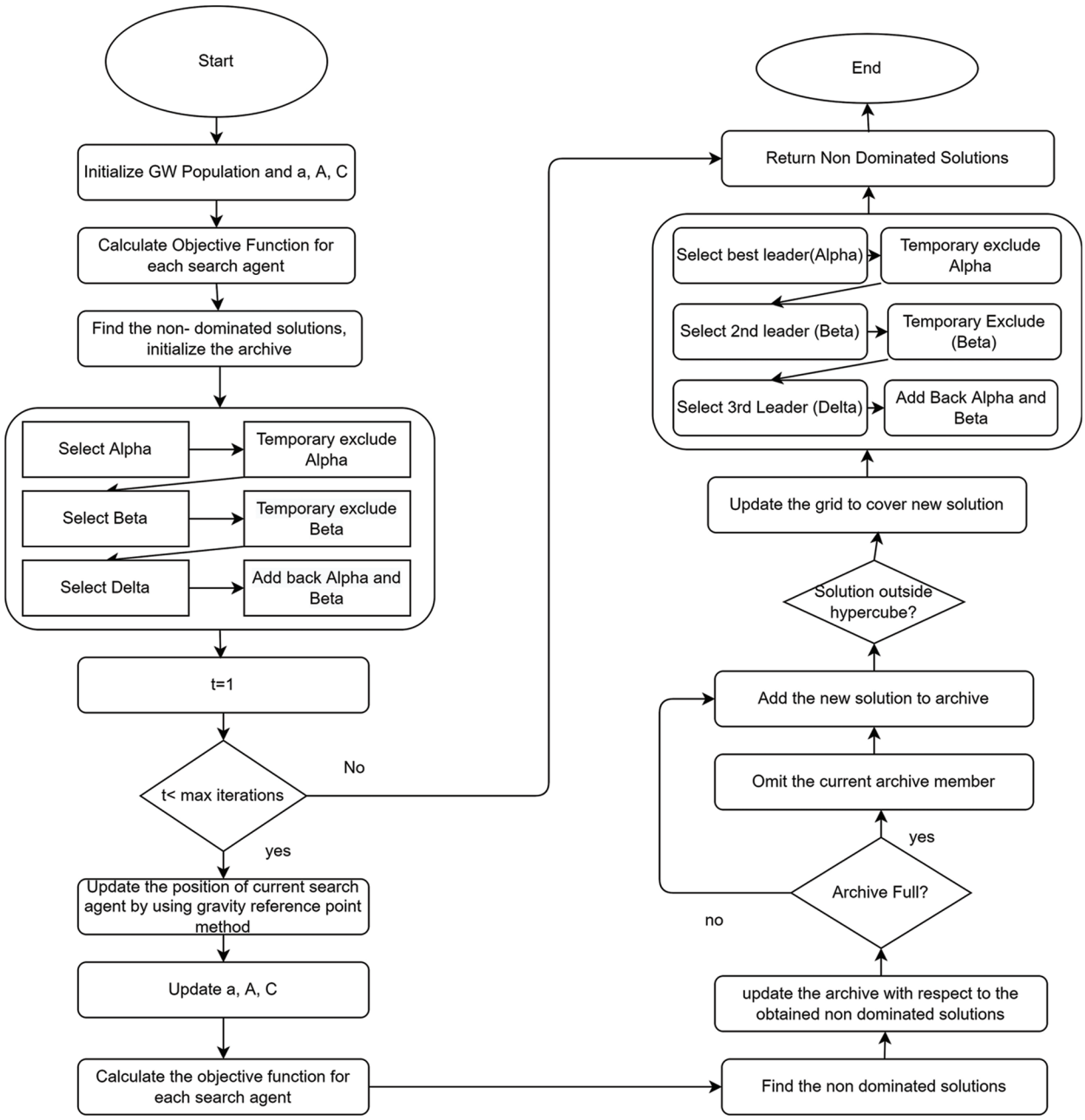

Figure 1: Flow chart of E-MOGWO

The weight coefficient increases when the gravity reference point is closer to the current solution. This work defines two functions of computational time and energy consumption usage. When $F1(X)$ takes the min value, then the resultant solution is called $\vec{X_1}$. When $F2(X)$ takes the min value, then the resultant solution is called $\vec{X_2}$. When $X = \vec{X_1}$ then $F1(X)$ and $F2(X)$ are calculated based on Eq. (7), known as CT_1 and EC_1. When $X = \vec{X_2}$ The value corresponding to functions $F1(X)$ and $F2(X)$ are called CT_2 and EC_2. Then compute the weight and distance parameter for each set as follows:

Calculate the energy consumption and computational time of the current solution, and then estimate the distance between $\vec{X_1}$, $\vec{X_2}$ and the current solution.

$$D_1 = \sqrt{(CT - CT_1)^2 + (EC - EC_1)^2} \tag{22}$$

$$\boldsymbol{D_2} = \sqrt{(\boldsymbol{CT} - \boldsymbol{CT_2})^2 + (\boldsymbol{EC} - \boldsymbol{EC_2})^2} \tag{23}$$

Compute the weight parameters

$$\mathrm{WT}_1 = \frac{D_1 + D_2 - D_1}{D_1 + D_2} = \frac{D_2}{D_1 + D_2} \tag{24}$$

$$\mathrm{WT}_2 = \frac{\boldsymbol{D_1} + \boldsymbol{D_2} - \boldsymbol{D_2}}{\boldsymbol{D_1} + \boldsymbol{D_2}} = \frac{\boldsymbol{D_1}}{\boldsymbol{D_1} + \boldsymbol{D_2}} \tag{25}$$

The current solution is updated using Eq. (28).

$$\vec{D_1} = \left| \overrightarrow{C_v} . \overrightarrow{X_1} - \vec{X}(t) \right| \tag{26}$$

$$\vec{D_2} = \left| \overrightarrow{C_v} . \overrightarrow{X_2} - \vec{X}(t) \right| \tag{27}$$

$$\vec{X}(t+1) = WT_1 * \left| \overrightarrow{X_t} - \overrightarrow{A_v} . \overrightarrow{D_1} \right| + WT_2 * |\overrightarrow{X_t} - \vec{A}_v . \overrightarrow{D_\alpha}| \tag{28}$$

Algorithm 3: Gravity Reference Algorithm

Function $EA_G = Gravity(EA_P, \ Capacity)$

1. Input EA_P and capacity
2. Output EA_G
3. $EA_G \leftarrow EA_P$
4. **For** each entry in EA_P, do
5. Revise the location of the current wolf by using Eq. (28)
6. **End for**
7. $\left(EA_G, \ \overrightarrow{X_P} \right)$ = CD_sort (EA_P, EA_G, capacity)

5 Results and Analysis

This part presents a simulation scenario for evaluating the proposed algorithm's performance. The simulation has been performed using a pure edge simulator. All experiments are run on a system with an

intel i5 processor, 8 GB RAM, 1TB hard disk and windows 10 operating system. The proposed E-MOGWO is compared with two multiobjective metaheuristic algorithms APSO [34] and NSGA-II [35]. The parameters of each algorithm are settled according to related references, as shown in Tab. 1. For the experiment, 100 search agents are investigated for 50 iterations. Each algorithm runs independently 50 times. The archive size of 50 is considered for all algorithms.

Table 1: Parameter settings

Algorithm	Parameters
E-MOGWO	Grid inflation parameter (GIP) $\alpha = 0.1$ Leader selection parameter (LSP) $\beta = 4$
MOGWO	Grid inflation parameter (GIP) $\alpha = 0.1$ Leader selection parameter (LSP) $\beta = 4$
APSO	w = 0.5 Personal learning coefficient (PLC) $c1 = 1$ Global learning coefficient (GLC) $c2 = 2$ GIP: $\alpha = 0.1$, LSP: $\beta = 4$
NSGA-II	Cross over rate (CR) = 0.5 Mutation rate (MR) = 0.1

The simulator has been designed to simulate a three-layer system, i.e., IoT devices, fog server and cloud server. In the first layer, three different types of IoT applications of health, intelligent transport systems and augmentation reality are deployed on IoT devices. Each device generates a task according to deployed application and task generation rate. The experimental setup of IoT devices, fog nodes and cloud are shown in Tab. 2.

Table 2: Simulation parameters settings

Parameters	Values
Minimum number of IoT nodes	100
Maximum number of IoT nodes	1000
WLAN bandwidth	500 (Mbits/s)
End device counterstep	100
Task generation rate	5 Task/Device/Minutes
Propagation delay	0.2(s)
WAN bandwidth	500 (Mbits/s)
Request length (MI)	50–2000
Request size (megabits)	1–16
Processing power of IoT devices (MIPS)	1000–2000
RAM memory size of IoT devices (MB)	256–512
Processing power of fog nodes (MIPS)	4000–8000
RAM memory size of fog nodes (MB)	512–1024
Processing power of each cloud server (MIPS)	12000–30000
RAM memory size of cloud server (MB)	2048–4096

Comparison Analysis: To evaluate the proposed technique, We analyzed energy consumption, computational time, the overall percentage of successful tasks, and the number of tasks completed at each level (local, fog, cloud).

5.1 Computational Time

It is the time required to execute a task either on a local IoT device or a fog or cloud server. It is defined by both the time required to complete a task on the device and the time required to transmit the task to the remote server. A comparative analysis of computational time is represented using Fig. 2. The performance improvement rate (PIR) [36] has been calculated to determine the improvement of the proposed algorithm over the existing algorithm using Eq. (29).

$$\boldsymbol{PIR}(\%) = \left(\frac{\sum_{old} \boldsymbol{Value} - \sum_{New} \boldsymbol{Value}}{\sum_{old} \boldsymbol{Value}}\right) * \mathbf{100} \tag{29}$$

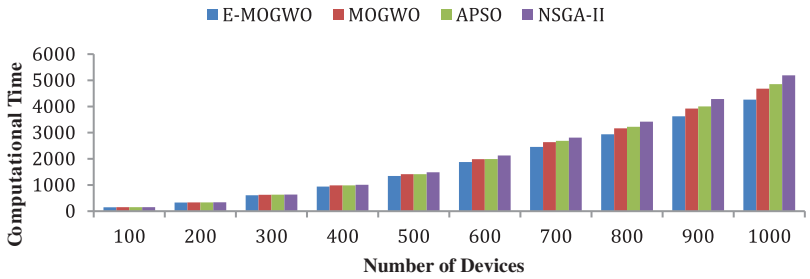

Figure 2: Comparison of computational time

E-MOGWO algorithm reduces computational time by 5.10% over MOGWO, 6.10% over APSO and 9.72% over NSGA-II, as presented in Tab. 3.

Table 3: PIR (%) of computational time

Number of devices	E-MOGWO	MOGWO	APSO	NSGA-II	PIR (%) over MOGWO	PIR (%) over APSO	PIR (%) over NSGA-II
100	150.05	152.32	152.68	153.56	1.49028361	1.72255698	2.2857515
200	330.23	336.45	336.98	340.42	1.84871452	2.00308624	2.99336114
300	610.24	625.42	630.2	634.45	2.42716894	3.16724849	3.81590354
400	940.87	984.52	985.08	1008.36	4.43363263	4.48796037	6.69304613
500	1345.98	1412.44	1413.12	1488.66	4.70533262	4.75118886	9.58445851
600	1878.02	1988.22	1992.88	2128.48	5.54264619	5.76351812	11.7670826
700	2456.03	2638.42	2686.36	2812.14	6.91284936	8.5740556	12.6633098
800	2937.78	3164.3	3223.22	3422.22	7.15861328	8.85574053	14.1557235
900	3623.54	3922.42	4002.38	4286.36	7.61978574	9.46536811	15.4634702
1000	4261.35	4680.42	4854.32	5188.22	8.95368364	12.2153051	17.8648939
Mean % improvement					5.10927105	6.10060284	9.72870008

5.2 Energy Consumption

The total amount of energy used is the sum of computational and transmission energy. Most tasks in the simulation are handled at the fog layer or local device rather than the cloud layer. The cloud is only used for the most resource-intensive tasks. As fog nodes are deployed close to the end device, they assist in reducing transmission energy and total energy usage.

The proposed E-MOGWO algorithm significantly improved energy consumption over MOGWO, APSO and NSGA-II Algorithms. This algorithm reduces energy usage by 15.10% over MOGWO, 17.29% over APSO, and 21.91% over NSGA-II, respectively, as presented in Tab. 4. A comparative analysis of energy consumption is displayed in Fig. 3.

Table 4: PIR (%) of energy consumption

Number of devices	E-MOGWO	MOGWO	APSO	NSGA-II	PIR (%) over MOGWO	PIR (%) over APSO	PIR (%) over NSGA-II
100	1.5	1.9	1.98	2.07	21.0526316	24.2424242	27.5362319
200	2.08	2.6	2.69	2.78	20	22.6765799	25.1798561
300	2.92	3.58	3.66	3.89	18.4357542	20.2185792	24.9357326
400	3.87	4.68	4.82	5.12	17.3076923	19.7095436	24.4140625
500	5.18	6.22	6.32	6.82	16.7202572	18.0379747	24.0469208
600	5.98	7.02	7.12	7.84	14.8148148	16.011236	23.7244898
700	7.89	9.12	9.38	9.97	13.4868421	15.8848614	20.8625878
800	9.54	10.66	10.98	11.68	10.5065666	13.1147541	18.3219178
900	11.23	12.54	12.89	13.54	10.446571	12.8782002	17.0605613
1000	13.89	15.13	15.46	15.96	8.19563781	10.1552393	12.9699248
Mean % improvement					15.0966768	17.292939	21.9052285

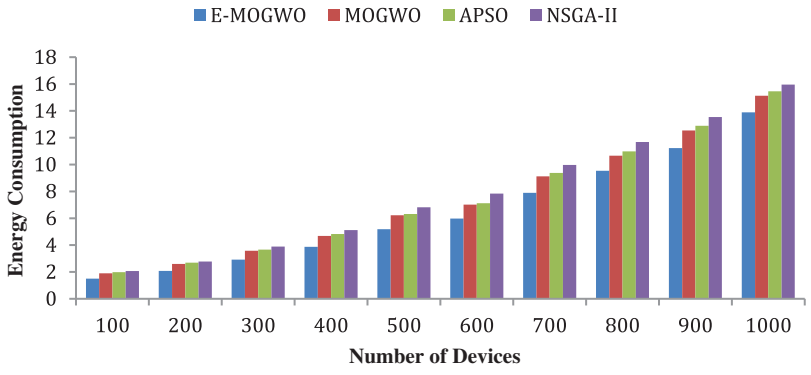

Figure 3: Comparison of energy consumption

5.3 Number of Tasks Successfully Executed

The analysis of the percentage of the number of tasks successfully executed using the various algorithms is presented in Fig. 4. The percentage of tasks successfully executed by E-MOGWO is 0.63% more than MOGWO, 1.06% more than APSO and 1.80% more than NSGA-II. A task may fail due to high delay,

unavailability of resources and low remaining battery. The task failure rate in E-MOGWO is lower than in other algorithms. Fig. 5 shows the number of tasks executed at different levels either to the fog node, cloud node and IoT devices.

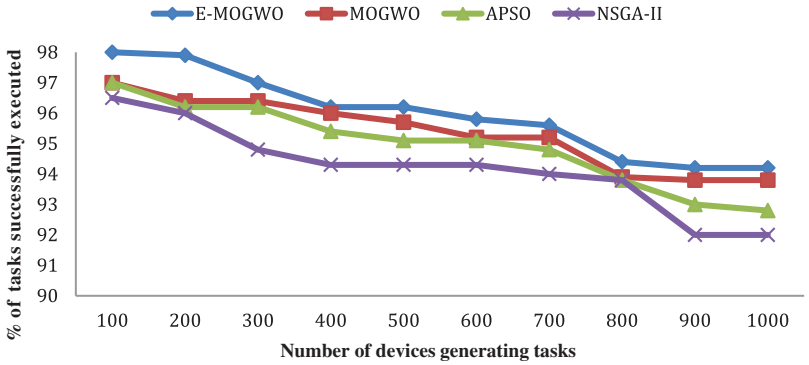

Figure 4: Comparison of percentage of tasks successfully executed

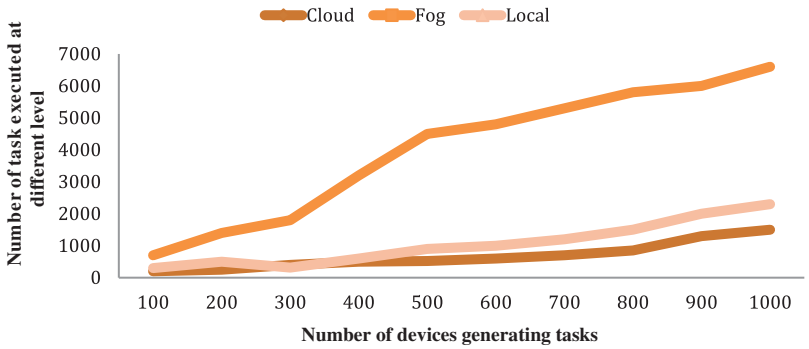

Figure 5: Number of tasks executed at different levels

E-MOGWO provides good results than other algorithms. Because this algorithm has a good balance between exploration and exploitation, which allows the escape of local optima stagnation. This method also has fewer parameters than others, which is advantageous. Moreover, it has only one position vector compared to APSO, which has two vectors, i.e., velocity and position. Hence it requires less memory space. Moreover, when the total number of iterations is finished, the obtained solutions are stored as non-dominated solutions set. These are chosen from the less crowded region of the solution space using a leader selection algorithm. APSO and NSGA-II, on the other hand, save previously discovered solutions, resulting in solution duplication and premature convergence due to the rapid loss of diversity. This can be avoided with a grid mechanism that removes the current solution from the archive when it fills up and replaces it with a better one. The computational complexity of E-MOGWO is $O(MN^2)$ where N is the population size, M is the number of objectives, and the complexity is the same as other well-known multiobjective algorithms.

6 Conclusion

A metaheuristic-based multiobjective offloading technique is proposed in this work. The optimal computing destination for each task is selected based on minimum energy consumption and minimum computational time. The E-MOGWO algorithm allocates delay-sensitive applications to local devices or

fog nodes. In contrast, resource-sensitive applications are assigned to the fog or cloud layers. It decreases the energy consumption and processing time of offloaded applications, as specified in the results section. The simulation results showed that the proposed offloading approach reduces energy consumption by 15.10% over MOGWO, 17.29% over APSO, and 21.91% over NSGA-II, and reduces computational time by 5.10% over MOGWO, 6.10% over APSO and 9.72% over NSGA-II. In the future, we will be examining the alternative service level agreement constraints and other parameters to enhance the algorithm's performance further.

Acknowledgement: Authors also acknowledge the CSE Department of Deenbandhu Chhotu Ram University of Science and Technology for providing various facilities in the research lab.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding this research work.

References

[1] S. R. Hassan, I. Ahmad, J. Nebhen, A. U. Rehman, M. Shafiq *et al.,* "Design of latency-aware IoT modules in heterogeneous fog-cloud computing networks," *Computers, Materials and Continua*, vol. 70, no. 3, pp. 6057–6072, 2022.

[2] W. Paper, "Prepare to succeed with the internet of things," CISCO, White Paper, pp. 1–9, 2017.

[3] M. G. R. Alam, M. M. Hassan, M. Zi. Uddin, A. Almogren and G. Fortino, "Autonomic computation offloading in mobile edge for IoT applications," *Future Generation Computer Systems*, vol. 90, pp. 149–157, 2019.

[4] J. Fang, J. Shi, S. Lu, M. Zhang and Z. Ye, "An efficient computation offloading strategy with mobile edge computing for IoT," *Micromachines*, vol. 12, no. 2, pp. 204, 2021.

[5] D. D. Lieira, M. S. Quessada, A. L. Cristiani, and R. I. Meneguette, "Algorithm for 5G resource management optimization in edge computing," *IEEE*, vol. 19, no. 10, pp. 1772–1780, 2020.

[6] J. Yadav and S. Sangwan, "Computation offloading in fog computing : Use cases, techniques & issues," *Turkish Journal of Qualitative Enquiry*, vol. 12, no. 6, pp. 1480–1488, 2021.

[7] X. Gao, S. Member, X. Huang, S. Member, S. Bian *et al.,* "PORA : Predictive offloading and resource allocation in dynamic fog computing systems," *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 72–87, 2019.

[8] S. R. Hassan, I. Ahmad, S. Ahmad, A. Alfaify and M. Shafiq, "Remote pain monitoring using fog computing for e-healthcare: An efficient architecture," *Sensors (Switzerland)*, vol. 20, no. 22, pp. 1–21, 2020.

[9] L. F. Bittencourt, J. D. Montes, R. Buyya, O. F. Rana and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 26–35, 2017.

[10] K. Gasmi, S. Dilek, S. Tosun and S. Ozdemir, "A survey on computation offloading and service placement in fog computing-based IoT," The Journal of Supercomputing, vol. 78, no. 2, 1983–2014, 2022.

[11] G. Baranwal and D. P. Vidyarthi, "Computation offloading model for smart factory," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 8, pp. 8305–8318, 2021.

[12] M. M. Alqarni, A. Cherif and E. Alkayal, "A survey of computational offloading in cloud/edge-based architectures: Strategies, optimization models and challenges," *KSII Transactions on Internet and Information Systems*, vol. 15, no. 3, pp. 952–973, 2021.

[13] R. C. Palacios, "A systematic literature review of fog computing," in *Proc. the Norwegian Conf. on Information Systems*, Bergen, vol. 24, no. 1, pp. 28–30, 2016.

[14] M. Aazam, S. Zeadally and K. A. Harras, "Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities," *Future Generation Computer Systems*, vol. 87, pp. 278–289, 2018.

[15] A. Bhattacharya and P. De, "A survey of adaptation techniques in computation offloading," *Journal of Network and Computer Applications*, vol. 78, pp. 97–115, 2017.

[16] P. Habibi, M. Farhoudi, S. Kazemian, S. Khorsandi and A. Leon-Garcia, "Fog computing: A comprehensive architectural survey," *IEEE Access*, vol. 8, pp. 69105–69133, 2020.

[17] A. Yousefpour, G. Ishigaki, R. Gour and J. P. Jue, "On reducing IoT service delay via fog offloading," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 998–1010, 2018.

[18] I. Ullah and H. Y. Youn, "Task classification and scheduling based on K-means clustering for edge computing," *Wireless Personal Communications*, vol. 113, no. 4, pp. 2611–2624, 2020.

[19] L. Liu, Z. Chang, X. Guo, S. Mao and T. Ristaniemi, "Multiobjective optimization for computation offloading in fog computing," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 283–294, 2018.

[20] N. Shan, Y. Li and X. Cui, "A multilevel optimization framework for computation offloading in mobile edge computing," *Mathematical Problems in Engineering*, vol. 2020, pp. 1523–1534, 2020.

[21] Z. Xiong, S. Feng, W. Wang, D. Niyato, P. Wang *et al.,* "Cloud/Fog computing resource management and pricing for blockchain networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4585–4600, 2018.

[22] H. S. Mansouri and V. W. S. Wong, "Hierarchical fog-cloud computing for IoT systems: A computation offloading game," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 3246–3257, 2018.

[23] L. Dong, M. N. Satpute, J. Shan, B. Liu, Y. Yu *et al.,* "Computation offloading for mobile-edge computing with multi-user," in *Proc. Int. Conf. on Distributed Computing Systems*, Dallas, TX, USA, vol. 2019-July, pp. 841–850, 2019.

[24] E. Balevi and R. D. Gitlin, "A clustering algorithm that maximizes throughput in 5G heterogeneous F-RAN networks," in *Proc. IEEE Int. Conf. on Communications*, Kansas City, MO, USA, vol. 2018-May, 2018.

[25] M. Kumar and S. Suman, "Scheduling in IaaS cloud computing environment using sailfish optimization algorithm," *Trends in Sciences*, vol. 19, no. 10, pp. 4204, 2022.

[26] M. K. Hussein and M. H. Mousa, "Efficient task offloading for IoT-based applications in fog computing using ant colony optimization," *IEEE Access*, vol. 8, pp. 37191–37201, 2020.

[27] R. Ezhilarasie, M. S. Reddy and A. Umamakeswari, "A new hybrid adaptive GA-PSO computation offloading algorithm for IoT and CPS context appplication," *Journal of Intelligent & Fuzzy Systems*, vol. 36, no. 5, pp. 4105–4113, 2019.

[28] S. A. Zakaryia, S. A. Ahmed and M. K. Hussein, "Evolutionary offloading in an edge environment," *Egyptian Informatics Journal*, vol. 22, no. 3, pp. 257–267, 2020.

[29] K. Peng, "An energy and cost-aware computation offloading method for workflow applications in mobile edge computing," *Eurasip Journal on Wireless Communications and Networking*, vol. 2019, no. 1, pp. 207, 2019.

[30] K. Lin, S. Pankaj and D. Wang, "Task offloading and resource allocation for edge-of-things computing on smart healthcare systems," *Computers and Electrical Engineering*, vol. 72, pp. 348–360, 2018.

[31] A. Khalili and S. M. Babamir, "Optimal scheduling workflows in cloud computing environment using pareto-based grey wolf optimizer," *Concurrency Computation*, vol. 29, no. 11, pp. 1–11, 2017.

[32] S. Mirjalili, S. Saremi, S. M. Mirjalili and L. D. S. Coelho, "Multiobjective grey wolf optimizer: A novel algorithm for multi-criterion optimization," *Expert Systems with Applications*, vol. 47, pp. 106–119, 2016.

[33] O. Shahryari, H. Pedram and V. Khajehvand, "Energy and task completion time trade-off for task offloading in fog-enabled IoT networks," *Pervasive and Mobile Computing*, vol. 74, pp. 101395, 2021.

[34] M. Adhikari, S. N. Srirama and T. Amgoth, "Application offloading strategy for hierarchical fog environment through swarm optimization," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4317–4328, 2020.

[35] M. Keshavarznejad, "Delay-aware optimization of energy consumption for task offloading in fog environments using metaheuristic algorithms," *Cluster Computing*, vol. 24, no. 3, pp. 1825–1853, 2021.

[36] M. Kumar and S. Sangwan, "Hybrid cuckoo search algorithm for scheduling in cloud computing," *Computers, Materials and Continua*, vol. 71, no. 1, pp. 1641–1660, 2022.