Alexandria Engineering Journal 97 (2024) 302–318

Contents lists available at ScienceDirect

Alexandria Engineering Journal

journal homepage: www.elsevier.com/locate/aej

Enhanced whale optimization algorithm for dependent tasks offloading problem in multi-edge cloud computing

Khalid M. Hosny [a,*], Ahmed I. Awad [a], Wael Said [b], Mahmoud Elmezain [c], Ehab R. Mohamed [a], Marwa M. Khashaba [a]

[a] Department of Information Technology, Faculty of Computers and Informatics, Zagazig University, Zagazig 44519, Egypt
[b] Computer Science Department, College of Computer Science and Engineering, Taibah University, Medina 42353, Saudi Arabia
[c] Faculty of Computer Science and Engineering, Taibah University, Yanbu 966144, Saudi Arabia

ARTICLE INFO

Keywords:
Computation offloading
Task dependency
Mobile edge computing
Multi-edge cloud computing
Multi-objective optimization, Enhanced Whale Optimization, dynamic allocation

ABSTRACT

In this paper, we introduce the Enhanced Whale Optimization Algorithm (EWA) to optimize dependent task offloading in a multi-edge cloud computing environment. Our proposed algorithm aims to identify the most suitable offloading scenario for dependent tasks, focusing on minimizing total processing latency, energy consumption, and associated costs. We operate within a system comprising many decentralized Mobile Edge Computing servers (MECs) and a centralized cloud server. Two novel improvement operations, namely Frame Shifting (FS) and Load Redistribution Strategy (LRS), are introduced to enhance the performance of the whale algorithm. Through simulation, our results demonstrate the superior performance of EWA. Specifically, compared to the Whale Optimization Algorithm (WOA), EWA achieves a remarkable reduction in latency by 22.84%, a substantial decrease in energy consumption by 78.28%, and a notable reduction in cost usage by 61.47%. These outcomes underscore the efficacy and practical significance of the proposed EWA in addressing the challenges posed by dependent task offloading in the multi-edge cloud computing landscape.

1. Introduction

Recently, the escalating utilization of advanced wireless applications on intelligent devices, such as Computer Vision (CV), virtual reality (VR), natural language processing (NLP), and augmented reality (AR), has intensified the pressing need for high-performance computing resources [1,2]. Furthermore, The rapid advancement of the Internet of Things (IoT) and Internet of Vehicles (IoV) technologies and the appearance of the sixth Generation (6 G) technology have facilitated communication among vehicles and increased their intelligence [3–5]. These characteristics make it easier for different vehicle functions to develop for easier and more secure driving. Numerous automotive tasks require enormous computational resources and stringent timeframes for the completion, such as online intelligent navigation and autonomous driving. Vehicles are furnished with onboard computers (OBCs), but their ability to provide real-time application services is limited by the high hardware cost of these devices and insufficient resources [6,7]. Additionally, smartphones and IoT devices are lightweight devices regarding processing capabilities. Moreover, it depends on temporary batteries as a power source [8].

The emergence of cloud computing empowers vehicles and IoT gadgets to offload their processing tasks to remote cloud servers. The constraints of a vehicle's limited processing capability are efficiently addressed by the cloud server's large computing resources [4,5]. Nevertheless, the latency resulting from long-distance task transfers is insufficient to fulfill the low latency needs of real-time applications [6]. Mobile Edge Computing (MEC) is deployed at the edge of the network to become a promising solution for the next generations' wireless devices [9–11] to compensate for such limitations. Road-side units (RSUs) are one example of how it has been implemented in IoV networks [12]. With the MEC in the network, the lightweight IoT and IoV devices can offload their tasks to be completed as quickly as possible [13]. Relying just on MEC servers might not be able to meet the processing demands of compute-intensive processing tasks, even though it can greatly reduce latency [14]. Consequently, collaborating between the nearby MECs can be more than one and the cloud to compute these applications can

* Corresponding author.
E-mail addresses: k_hosny@yahoo.com, k_hosny@zu.edu.eg (K.M. Hosny), ahmedawd622@gmail.com (A.I. Awad), wmohamed@taibahu.edu.sa (W. Said), mmahmoudelmezain@taibahu.edu.sa (M. Elmezain), ehab.rushdy@gmail.com (E.R. Mohamed), marwamkhashaba@gmail.com (M.M. Khashaba).

https://doi.org/10.1016/j.aej.2024.04.038
Received 19 October 2023; Received in revised form 15 March 2024; Accepted 16 April 2024
Available online 23 April 2024

optimize the requirement for IoV and IoT systems [15–18]. An effective offloading strategy should be used to achieve the most benefit from these resources.

Optimization problems are typically formulated and addressed through various optimization techniques [19,20] to tackle computation offloading challenges, including user mobility and content caching. Key optimization objectives encompass energy conservation (E), cost minimization for MEC and cloud usage (UC), and reduction of application completion latency (L) [21]. Efficiently managing these objectives is critical, especially in the context of battery-operated devices like mobile and IoT devices, where users are charged for server resource utilization, and the Quality of Service (QoS) for time-sensitive applications depends on minimizing latency [22]. Considering the typical complexity of these optimization challenges, which often encompass diverse constraints, mixed variables, and distinct function architectures, achieving optimal solutions within a reasonable timeframe can be challenging, particularly when faced with conflicting objectives and the absence of a precise solution [23,24].

Effective task execution on a cloud network aided by MEC servers is closely linked to making well-informed decisions about task offloading. For this reason, Numerous strategies, including Lyapunov optimization, heuristics [25], metaheuristics [26], game-theoretic [27,28], convex optimization, mathematical, machine learning [29], supervised deep learning (SDL) [30], Deep Reinforcement Learning (DRL) [31], and various other algorithms, have been proposed to offer a stable solution to this problem [32]. However, these solutions often serve a singular purpose, pose challenges in application, are computationally expensive, focus solely on individual user considerations, are either cloud or MEC-based, or overlook the interdependence among application tasks. Therefore, there is an urgent demand to develop more effective optimization algorithms that can achieve a higher level of depreciation for the cost function considered for this problem.

In this paper, we employed an enhanced version of the Whale Optimization Algorithm (WOA) [33] to address the dependent tasks offloading problem within the multi-edge cloud computing environment, targeting three main objectives (L, UC, E). The cooperative hunting behavior of humpback whales inspired the WOA. It is a Metaheuristic with broad applications, including engineering design optimization, data mining, image processing, and machine learning. Its versatility and effectiveness make it valuable for resolving intricate optimization problems across diverse domains [34]. WOA stands out for its optimization performance, adaptability to dynamic environments, capability in handling multi-objective optimization, and simplicity of implementation and interpretation compared to other deep learning-based algorithms and alternative heuristics [35–37].

We considered the cloud computing server in our model to fully benefit from the available resources. Additionally, We relied on an enhanced Whale Algorithm (EWA), which was used for the first time in this case. The standard WOA is improved by adding two main operations to its standard flow. The first operation is a FrameShifting (FS) operation, which is inspired by the replication behavior of the COVID-19 virus [38]. The second enhancement operation is the Load Redistribution Strategy (LRS), which enforces a fair load balancing among the included processing locations. In addition, we thoroughly tested our algorithm with both generated and standard topologies to guarantee its efficacy. Compared to other algorithms, our EWA method has proven to be better. Following a review of earlier works in this field, we can conclude that our proposed solution has the following contributions:

- Developed an integrated system composed of M MECs and a cloud server for processing the offloaded tasks, taking the tasks' interdependency into account. Section 2 presents a detailed description of the mathematical formulation of the problem.
- Dynamically and simultaneously optimize three goals (L, UC, and E) as in section 2.4.
- Used an enhanced version of the WOA algorithm called EWA, which was used for the first time in this case. Two improvement strategies were adopted to enhance the obtained results of the WOA. Section 4 contains a comprehensive discussion of the EWA steps.
- Conducted simulation experiments to test the performance of EWA using a set of Standard and derived test cases with different topologies and task data. As stated in Section 6, the results show EWA's superiority.

The rest of the document is organized as follows: Section 2, discusses the related work in this point. Section 3 introduces the proposed system model and the related problem formulation. Section 4 discusses the steps of the suggested EWA algorithm. In this section, we also discuss the two enhancement operations and the reason behind adding each operation to the model. In Section 5, we discuss the performed Experiments and obtained results. We also discussed the impact of each added enhancement operation on the model. Finally, we conclude our work in Section 6.

2. Related work

Efficient computation offloading entails leveraging the high bandwidth and processing capabilities of mobile, IoT, IoV, and cloud devices. The goal is to enhance service quality by minimizing system latency and reducing energy consumption and associated costs. Various crucial parameters must be carefully considered when addressing this issue. These parameters encompass but are not limited to, the transmission and reception times of task-related data, interdependencies among application tasks, and variations in computational capabilities among the selected computing devices, among other factors. Numerous efforts in the literature have been made to offer viable solutions to the computational offloading issue, employing various system models and formulated problems. The work in [39] employs two MECs alternately for offloading Mobile tasks to reduce latency and battery usage. Another technique used in this study to improve the obtained results is to keep a copy of the file after the first time processing.

In this regard, various endeavors have been undertaken employing game-theoretic methods. In reference [40], the authors designed a Stackelberg game for addressing multi-user computational offloading challenges within the MEC system, aiming to optimize execution time and energy consumption during offloading decisions. Another initiative, described in reference [41], involved the development of a stochastic game to model the offloading problem in a dynamic setting, focusing on minimizing latency and energy usage as primary objectives pursued by the researchers in this investigation.

Heuristic and metaheuristic algorithms have recently garnered significant interest in tackling optimization problems, prompting various endeavors in this domain. In reference [42], the authors introduced a heuristic algorithm to efficiently minimize application execution costs within the constraints of predefined task completion times. Authors in [43] proposed the multiobject evolutionary algorithm to minimize energy consumption and latency. In references [17] and [44], the authors harnessed the Whale Optimization Algorithm (WOA) to develop a task scheduling approach that effectively reduced both completion time and energy consumption. The work in [12] modeled the collaborative computational offloading problem as mixed integer nonlinear programming and suggested a heuristic approach for minimizing the execution latency. Authors in [23] depended on a mobility-aware heuristic algorithm to solve the computation offloading problem in the heterogeneous vehicular edge computing networks with many objectives. In reference [45], the research addressed the problem of dependent task offloading with predefined service caching within a homogeneous MEC scenario. They introduced a heuristic algorithm based on favorite successors to reduce application completion times.

Liu et al. [46] introduced a one-dimensional search algorithm designed to minimize task execution delays, taking into account the

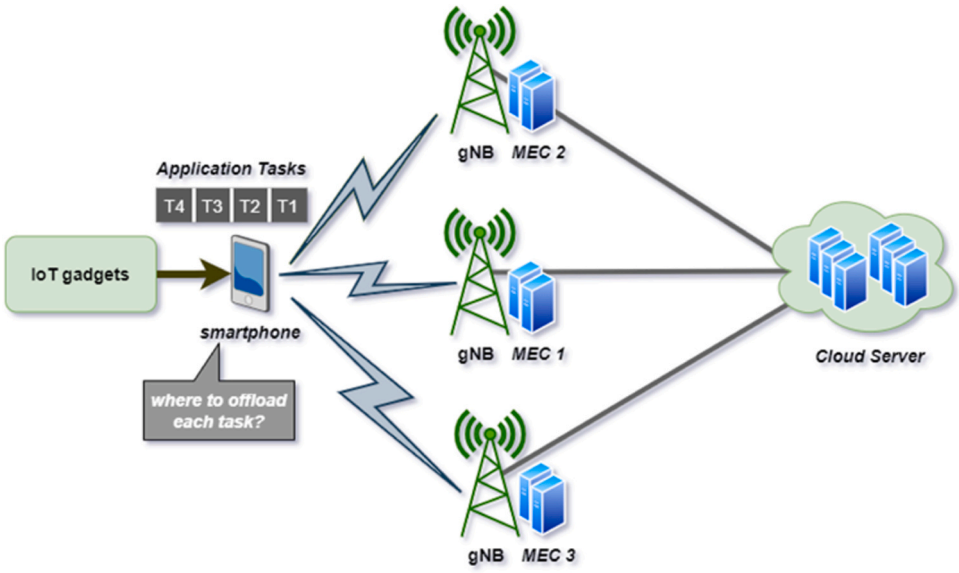

Fig. 1. : Proposed system architecture.

application buffer queuing status and the processor state. In a different approach, the authors of [47] proposed a scheme based on a Genetic Algorithm (GA) to reduce task offloading time and mitigate the risk of failure. Meanwhile, Huang et al. addressed a multifaceted challenge in their work [48], considering security, energy consumption, and application completion time. They employed a GA in their study to optimize energy consumption on mobile devices while meeting application deadlines.

Turning to optimization techniques, the authors of [49] and [50] developed a particle swarm optimization (PSO) algorithm-based approach to reduce both task scheduling time and cost efficiency. In a parallel endeavor, the work presented in [51] introduced a load-balancing heuristic technique for task offloading, focusing on optimizing task execution time. Furthermore, the research in [52] proposed an enhanced version of the Gorilla Troops algorithm to address the problem of dependent task offloading within a MEC environment, pursuing three key objectives: minimizing processing latency, reducing energy consumption, and lowering the overall usage cost. Additionally the authors in [53] optimized the three mentioned objectives. However, these works were MEC-based; they did not include the cloud server in their offloading strategy.

Numerous endeavors have been undertaken employing SDL and DRL techniques, yielding innovative solutions to address task offloading challenges. The study presented in [54] outlines the intricate problem of dependent task offloading, aiming to minimize overall execution costs while adhering to application completion time constraints. The work in [55] suggested an SDL approach for offloading the fine-grained application task to MEC with two optimization objectives: latency and energy consumption. Furthermore, researchers in references [56–58] put forth a single-objective DRL-based offloading approach, focusing on reducing total application latency. Meanwhile, in references [24,59–64], a two-objective DRL-based offloading strategy was proposed, concentrating on optimizing processing time and energy consumption. Additionally, the work by the authors of [65] introduced a multi-objective DRL method that prioritizes optimizing energy consumption, completion time, and MEC usage cost. Their proposed system does not encompass the cloud server in its framework. From a different perspective, the authors of [66] devised a Markov Decision Process to represent joint user association and offloading decisions in MEC-based SAT-IoT networks. Their approach leveraged DRL techniques to achieve energy consumption reduction and latency minimization objectives.

Despite this field's considerable body of research, our proposed work is distinct and innovative. We aim to offer a fresh and original solution to the challenge of dependent task offloading within the MEC environment. Existing approaches may encounter one or more of the following challenges:

- Disregard a few of the three goals under consideration.
- Ignores the interdependence of tasks.
- For their offloading, rely either on the cloud server or the edge, but not both.
- Depends on a single MEC server.
- Ignore the variations in computing capabilities of the adopted devices.
- Suggest symmetry in uplink and downlink wireless channels.
- Make elaborate suggestions.

We provide a novel approach to address the dependent task offloading problem in a multi-edge-cloud computing architecture by reducing three primary goals: use cost, energy consumption, and execution latency. We utilize an enhanced version of the WOA technique, which is utilized on this problem for the first time. We have extensively tested our approach with constructed topologies. It has been demonstrated that our EWA approach is better than other algorithms.

3. System model and problem formulation

In this paper, we consider a system that consists of M MECs and a single cloud server. In this system, the execution of each task can be done locally using the smartphone's resources and offloaded to one of the nearby MECs or the centralized cloud. Contrary to the current works [17,65,67], this scenario does not neglect any of the available processing locations, which can satisfy high-level optimization in the problem objectives. In this model, the IoT devices capture and pre-process the sensed data. After that, each IoT device sends its data to a smartphone, representing the central point of captured data collection. The smartphone is the central device targeted to run the application that analyzes the sensed data. Such applications are expected to be very computationally hungry and analyze large amounts of data called big data generated from connected IoT devices [68–71]. This application is divided into a set of T tasks during the design phase. Some of these tasks are executed on the local processor of the smartphone.

In contrast, some other tasks are offloaded to the nearby MECs

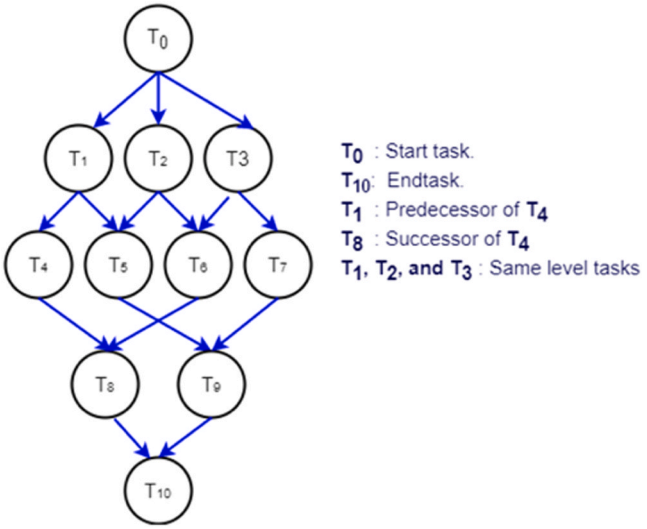

Fig. 2. application of eleven dependent tasks.

wirelessly using a wireless channel between the smartphone and the base station gNB [72]. The backhaul link can offload the last part to the cloud server. The system architecture is shown in Fig. 1, and the MEC is supposed to be directly connected to the gNB.

As we stated earlier, we propose a model composed of a set of IoT sensors connected to a smartphone as a central point of data collection. The smartphone is wirelessly connected to a set of gNBs through M channels. Each gNB (5 G New Radio Node) is equipped with its own MEC server, and these servers are securely interconnected with the central cloud via the backhaul network. Furthermore, we suppose that the smartphone runs a complex, computationally intensive application composed of dependent tasks. Each task can be executed locally, off-loaded to one of the nearby edge servers, or directly offloaded to the part of the central cloud that is maintained for this network.

We represent the set of tasks comprising the application as T = {T0, T1, T2, …, TN}, where N represents the total number of tasks in the application. Fig. 2 illustrates an application comprising eleven tasks. Each node within the graph represents an individual task within the application; for instance, T_i represents the task indexed as i. Moreover, the edges in the graph symbolize the dependencies among these tasks. For instance, the edge E_{ij} signifies a dependency relationship between task i and task j, with the arrowhead indicating that, the task i must be completed before task j can commence.

We employ two sets to model these dependency relationships among tasks: P (predecessors) and S (successors). In general, $P(T_i)$ represents the set of tasks that must be finished before task T_i can begin its execution, while $S(T_i)$ denotes the set of tasks that cannot commence until task T_i has completed its execution.

We employ a vector of three primary parameters for each task in our proposed system. These parameters encompass the following:

- The number of CPU clocks required to complete the task (CC_i) is expressed in millions of instructions per second (MIPS).
- The input size (Q_{in}) of the task is measured in megabytes.
- The task's output size (Qout) is also measured in megabytes.

The rest of this section is subdivided into three distinct subsections. Each subsection includes the mathematical formulation of one of the computational decisions adopted in the proposed system.

A. Local computation

In this case, the task is fully completed on the local device. We denote the ET_local_i is the time required for executing the task T_i locally. It is described as follows:

$$ET\_local_i = \frac{CC_i}{F\_local} \tag{1}$$

CC_i is the number of CPU clock cycles needed to complete task T_i, and F_local is the working frequency of the local processor. Since no data related to task T_i has to be transmitted on the network during local processing, the task T_i completion latency (L_local_i) can be described as follows:

$$L\_local_i = ET\_local_i + WT\_local_i + QT\_local \tag{2}$$

QT_local is the queuing time of the local processor before the scheduling of the T_i task. Besides that, it is the beginning time of task T_i. WT_local_i is the waiting time for task T_i. It is described as follows:

$$WT\_local_i = \max\{\max[L\_local_j, L\_MEC_j, L\_cloud_j] \forall T_j \in P(T_i)\} - QT\_local \tag{3}$$

where L_MEC_j and L_cloud_j the completion latencies for executing task Tj on one of the MEC servers or the central cloud, respectively. If WT_local_i has a negative or zero number, it is ignored. According to [47], The following is a definition of the energy consumption quantity:

$$E\_local_i = C * CC_i * (F\_local)^2 \tag{4}$$

Here, C represents a coefficient that quantifies the capacitance of the operational chip. It's important to note that the cost of executing task Ti on the local processor is inherently zero.

B. Offloading to the MEC server

In this case, the task is fully offloaded to one of the nearby MEC servers. Firstly, the local device sends task inputs Q_{in} to the edge server using the uplink wireless transmission channel. When the MEC server finishes the task execution, it responds with task outputs Q_{out} to the local device using the downlink wireless channel. We denote the $ET\_MEC_i^k$ is the time required for executing the task T_i on MEC_k, where $1 < k \leq M$. It is described as follows:

$$ET\_MEC_i^k = \frac{CC_i}{F\_MEC_k} \tag{5}$$

where F_MEC_k is the working frequency on MEC_k. The transmission and receiving times for the inputs Q_{in} and the outputs Q_{out} to/from MEC_k can be described as follows:

$$T\_trans_i^k = \frac{Q_{in}}{R_k} \tag{6}$$

$$T\_rec_i^k = \frac{Q_{out}}{R_k} \tag{7}$$

where R_k is the achievable uplink and downlink transmission rate between the smartphone and MEC_k. For the two wireless networks, we assumed symmetry. As in Shannon–Hartley theory [35], the R_k is described as:

$$R_k = W * LOG_2\left(1 + \frac{WP * G_k}{\sigma^2}\right) \tag{8}$$

where W is the channel bandwidth, WP is the local device's wireless transmitting and receiving powers of the local devices, G_k is the channel's gain, and σ^2 is the channel noise. The propagation delay can be ignored because the distance is very short compared to the speed of light, resulting in a very small value [65]. The total time latency required for completing task T_i on MEC_k is described as follows:

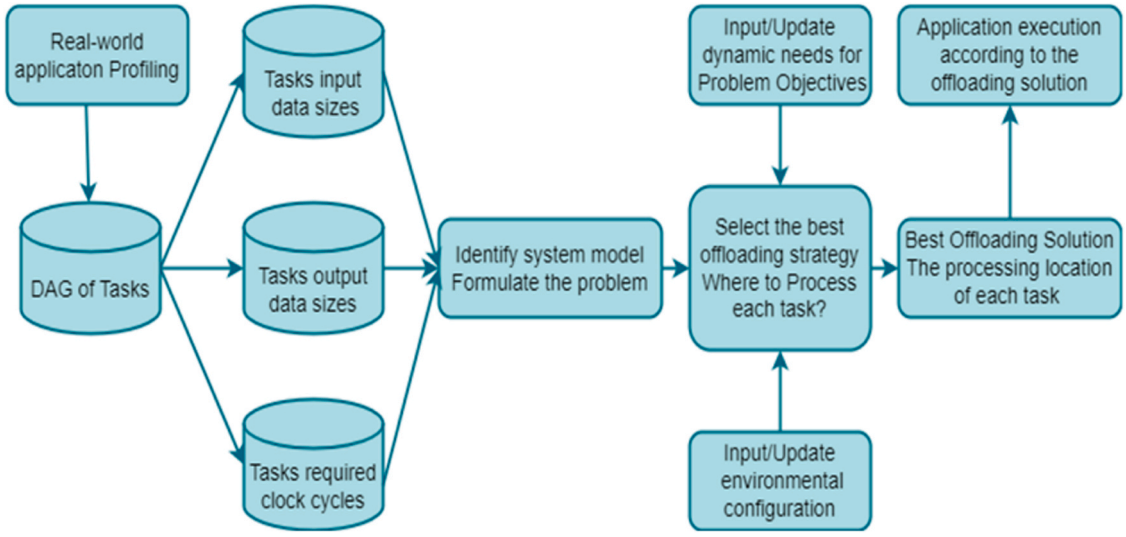

Fig. 3. : Research Methodology Steps.

$$L\_MEC_i^k = ET\_MEC_i^k + T\_trans_i^k + T\_rec_i^k + WT\_MEC_i^k \quad (9)$$

where $WT\_MEC_i^k$ is waiting times for task T_i to be executed on MEC_k. It can be computed as in Eq. (3) but for the MEC. The consumed energy during the transmission of the task inputs Q_{in} of task T_i to MEC_k is described as [73]:

$$TE\_MEC_i^k = wp * T\_trans_i^k \quad (10)$$

whereas the consumed energy during the receiving of the task outputs Q_{out} of task T_i from MEC_k can be described as follows:

$$RE\_MEC_i^k = wp * T\_rec_i^k \quad (11)$$

In the general case, the total consumed energy by the smartphone for executing task T_i on MEC_k can be calculated as follows:

$$E\_MEC_i^k = TE\_MEC_i^k + RE\_MEC_i^k \quad (12)$$

The imposed cost for executing task T_i on a specific MEC_k is described in [74] as follows:

$$UC\_MEC_i^k = \beta * E\_MEC_i^k \quad (13)$$

The symbol β is the coefficient that defines the cost per unit of time.

C. Offloading to the Cloud server

In this case, the task is fully offloaded to one of the central cloud servers. We denote the ET_cloud_i is the time required for executing the task T_i on the cloud server. It is described as follows:

$$ET\_cloud_i = \frac{CC_i}{F\_cloud} \quad (14)$$

where *F_cloud* is the working frequency of the part of the cloud server that is maintained for this network. The transmission and receiving times for the inputs Q_{in} and the outputs Q_{out} to/from the cloud can be described as follows:

$$T\_trans_i^{cloud} = \frac{Q_{in}}{R_k} + T\_prop \quad (15)$$

$$T\_rec_i^{cloud} = \frac{Q_{out}}{R_k} + T\_prop \quad (16)$$

where R_k is the achievable uplink and downlink transmission rate between the smartphone and MEC_k that is for the data flow. *T_prop* is the propagation delay taken by the data on the link between the local device and the cloud server. T_prop is defined as follows [75]:

$$T_{prop} = \frac{L}{S} \quad (17)$$

L is the distance from the MEC to the centralized cloud server. The total time latency required for completing task T_i on the cloud is described as follows:

$$L\_cloud_i = ET\_cloud_i + T\_trans_i^{cloud} + T\_rec_i^{cloud} + WT\_cloud_i \quad (18)$$

Where WT_cloud_i is waiting times for task T_i to be executed on the cloud. It can be computed as in Eq. (3) but for the cloud server. The consumed energy during the transmission of task T_i inputs TE_cloud_i and during the receiving of the same task, outputs RE_cloud_i To/From the cloud can be computed similarly to MEC offloading. The total consumed energy by the smartphone for executing task T_i on MEC_k can be calculated as follows:

$$E\_cloud_i = TE\_cloud_i + RE\_cloud_i \quad (19)$$

The imposed cost for executing task T_i on the cloud server is described as follows:

$$UC\_cloud_i = \beta * E\_cloud_i \quad (20)$$

D. Objective function

In the MEC environment, the task offloading problem is a multi-objective problem that dynamically and simultaneously minimizes remote servers' time latency, energy consumption, and cost utilization. We denote the total time required for completing task T_i is $L(T_i)$, total consumed energy $E(T_i)$, and the total used cost UC (T_i). These total quantities are computed using the following equations:

$$L(T_i) = L\_local_i + L\_MEC_i^k + L\_cloud_i \quad (21)$$

$$E(T_i) = E\_local_i + E\_MEC_i^k + E\_cloud_i \quad (22)$$

$$UC(T_i) = UC\_MEC_i^k + UC\_cloud_i \quad (23)$$

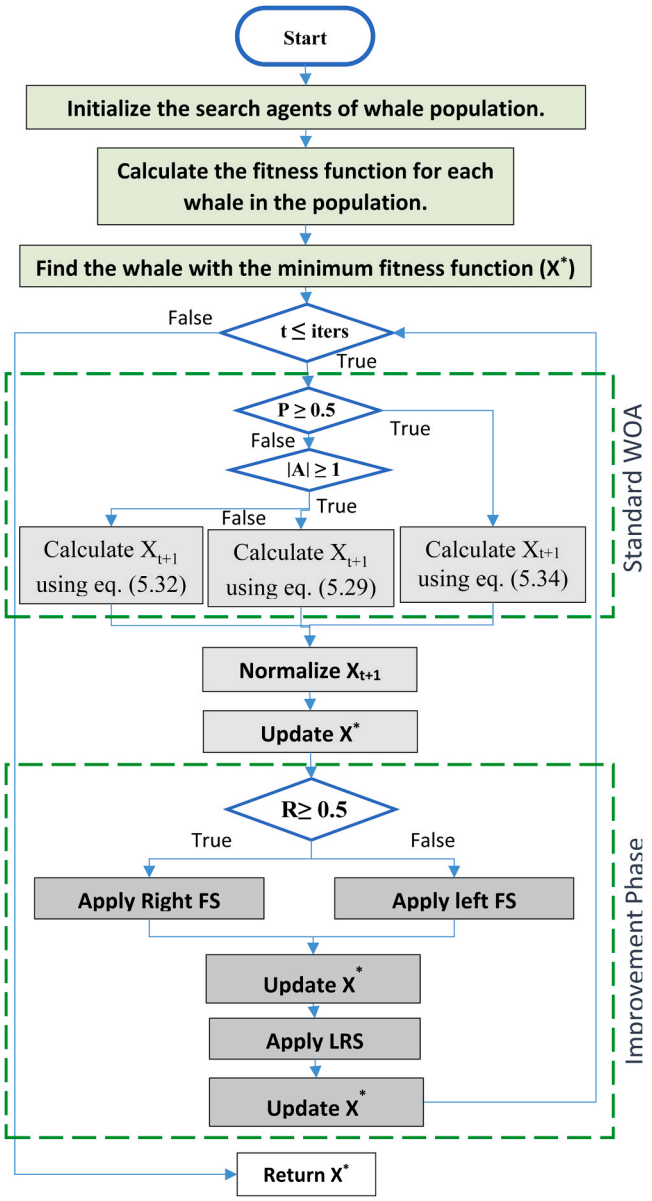

Fig. 4. Flowchart of the EWA algorithm.

We must confirm here that each task is fully executed in only one location, so if the task is executed locally, then the $L\_MEC_i^k = 0$ and$L_cloud_i = 0$. Also, $E\_MEC_i^k =$ 0and $E\_cloud_i = 0$. Furthermore, $UC\_MEC_i^k = 0$ and $UC\_cloud_i = 0$.

The total latency L, energy consumption E, and imposed cost UC for all tasks included in the application are described as:

$$L = Max\{L(T_i) \forall T_i \quad in \quad the\ application\} \quad (24)\ E = \sum_{i=1}^{N} \mathrm{E}(T_i) \quad (25)$$

$$UC = \sum_{i=1}^{N} \mathrm{UC}(T_i) \quad (26)$$

The objective function is defined as follows:

Minimize $Z = a_1 * \mathrm{L} + a_2 * \mathrm{E} + a_3 * \mathrm{UC}$. (27)

The parameters a_1, a_2, and a_3 are weighting factors that fall within the range [0,1]. It's crucial to emphasize that their sum must always equal to 1. These weighting factors play a pivotal role in the optimization process, allowing us to precisely specify the relative significance or importance of the various objectives within the problem. By adjusting these weights, we can effectively prioritize and balance the trade-offs between different objectives in problem-solving.

4. Research methodology

This paper introduces the EWA algorithm as a novel solution to tackle the optimization challenge associated with offloading a single application comprised of interdependent tasks. The optimization is achieved by leveraging multiple on-edge MEC servers and a central cloud server. Making a wise decision about task offloading among the available processing locations requires creating a DAG of tasks that accurately mimics the complexities of real-world systems. A significant amount of this complex procedure depends on advanced profiling methods. It requires properly gathering comprehensive information on network properties, local device properties, and application tasks. Program, network, and energy profilers are used expertly to acquire this massive data [76].

Furthermore, our study extends the application of the EWA algorithm to dynamically optimize three critical objectives (L, E, and UC). By deploying the EWA algorithm, we aim to provide a tailored solution to the offloading problem, ensuring adaptive optimization that aligns with the dynamic requirements of the system. Fig. 3 visually represents the steps involved in our research methodology.

The general flowchart of the EWA algorithm, which summarizes its basic steps, is shown in Fig. 4. EWA will be described in depth in this section using several subsections. This section thoroughly analyzes the EWA algorithm, broken down into several subsections for a more in-depth explanation. The introduction begins with a detailed explanation of the search agents' initialization procedure. Next, we proceed with modeling the standard phases that are part of the Whale Optimization Algorithm (WOA). An important step in the algorithmic advancement is then introduced: the production of a discrete vector from the continuous vector. Finally, we conclude our section with two subsections introducing the two enhancement operations added to the standard WOA behavior: FS operation and the LRS strategy.

A. Search agents initialization

The EWA algorithm begins by generating a population of randomly generated agents, each symbolizing a solution to the original problem. A vector of two parts represents each whale of the population. The first part expresses the tasks of the application, whereas the second part contains the processing location of each task. For example, Table 1 includes a vector that represents a whale agent in the problem population using the application in Fig. 2:

The first N elements of the example vector represent the tasks of the applications, whereas the next N elements represent the processing location of each task. You can see in the example vector that the task T_0 will be processed locally on the smartphone. As in Algorithm 1, the whale agents are randomly generated. The task list is generated by adding the start task to the vector, then getting its S set items, shuffling them, and successively adding the shuffled items to the vector. The next task is then added with its shuffled S set elements. The algorithm recursively repeats these steps until finishing the complete set of tasks in the application. The task list part of the whale agent must satisfy the following restrictions:

Table 1
Whale agent example.

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Task id	0	1	3	2	4	5	6	7	8	9	10	0	0	1	2	3	4	3	3	1	4	2

Table 2
Extracted values of task T_0.

Extracted values	1.6	3.2	0.5
Vector indices	1	2	3

1. All tasks of the application should be added to the vector.
2. Each task id should appear only one time in the vector.

The task cannot appear in the vector before all of its predecessors.

Algorithm 1. . Initialization algorithm

Input: population size pop_size, tasks number N, taskList T, number of processing locations PL
Output: initialized population pop.

```
For i = 1: pop_size
    Set K=0
    For j=1:2*N
        If (j<N)
            If (j = 0)
                pop_{i,k}=T_j
                K=k+1
            End if
            S= shuffle the elements of S(T_j)
            For each E element in S
                Add E to pop_{i,k} if it is not added before, and all its P set items were already added.
                K=k+1
            End for
        else
        pop_{i,k} = generate a random in the range [0, PL[
    End for
End for
```

B. **The WOA**

The WOA is a swarm intelligence optimization algorithm inspired by the unique hunting technique of humpback whales, known as the bubble-net attacking approach [77]. The algorithm models the chasing behavior of whale groups through three distinct operations: prey search, prey encircling, and the spiral bubble net approach. WOA operates on the belief that the intended prey or a nearby location represents the current best solution.

The core of WOA comprises three main operators, which are selected based on two fundamental parameters. The first parameter, P, is a random integer between 0 and 1. The second parameter, represented as A, constitutes a vector of coefficients obtained through the following equation:

$$A = 2 * a * r - a \tag{28}$$

Where a is a vector that decreases linearly from two to zero during rounds. r is a random number with a value between 0 and 1. The prey search behavior included in the exploration phase of WOA is selected in case $P < 0.5$ and $|A| \geq 1$. Where $||$ represents the absolute symbol. According to this operation, the whale position update is described as follows:

$$X_{t+1} = X_r - A * D \tag{29}$$

$$D = C * X_r - X_t \tag{30}$$

$$C = 2 * r \tag{31}$$

X_{t+1} is the updated whale position, X_t is the current whale position, and X_r is a randomly generated position. The prey encirculating behavior is selected when $P < 0.5$ and $|A| < 0.5$. The next position is described as follows:

$$X_{t+1} = X^* - A * D \tag{32}$$

$$D = |C * X^* - X_t| \tag{33}$$

where X^*denotes the best whale position. The last operation, bubble net attacking, is selected when $P \geq 0.5$, and it can be formulated as follows:

$$X_{t+1} = D' * e^{bL} * \cos(2\pi L) * X^* \tag{34}$$

$$D' = |X^* - X_t| \tag{35}$$

To increase the usefulness of the WOA algorithm, we have introduced two improvement operations. The mutation operation and the load redistribution operation are the two new operations. Following our adjustments, the WOA algorithm yields significantly improved outcomes, as evidenced by the results. After these adjustments, the resulting whale position vector undergoes a normalization process, transforming it into discrete and bounded values.

C. **Agents' normalization**

After applying the mathematical formulas of the WOA, the generated agents will be vectors of continuous values that cannot convey any meaning related to the offloading problem. For this reason, we normalize the resulting agents' vectors to represent a candidate solution to the problem. We depended on the following two distinct normalization strategies for converting the generated continuous vectors into a discrete one:

- Task List Restructuring Strategy (TLRS).
- Liner scaling strategy.

The first normalization strategy, TLRS, is used to normalize the task list part of the generated vectors. Using Fig. 1 as an application example,

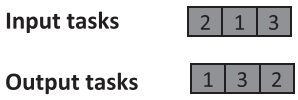

Fig. 5. : LFS technique example for shifting the task successors.

we present the basic steps of the TLRS as follows:

Step 1: Set the start task id as the first element of the normalized vector.

Step 2: Identify the elements of the S set of the start task. These elements are T_1, T_2, and T_3.

Step 3: Determine the indices of the S set in the generated vectors. Since task T_0 is the start task in index 0, the indices of the S set elements start from 1 to 3.

Step 4: Extract the values of these indices from the generated vector. Suppose that the generated vector is the following vector:

$\mathbf{X} = (0, 1.6, 3.2, 0.5, 4.6, 6.4, 3.4, 9.6, 8.7, 1.7, 1.6, 0.0, 0.4, 1.6, 2.4, 7.4, 4.9, 3.7, 4.6, 1.1, 4.9, 2.1)$.

You should notice that the start task id 0 is the first element of this vector after performing the first step. The extracted values with their indices are shown in Table 1:

Step 5: Use the extracted values to put the elements of the S set into the selected indices by setting the first element in the S set into the index of the max value of the extracted values and so on. Since the S set elements are 1, 2, and 3, the X vector will be as follows after this step.

$\mathbf{X} = (0, 2, 1, 3, 4.6, 6.4, 3.4, 9.6, 8.7, 1.7, 1.6, 0.0, 0.4, 1.6, 2.4, 7.4, 4.9, 3.7, 4.6, 1.1, 4.9, 2.1)$.

Step 6: Select the next task in the task list and add its id to the next index of the resultant vector. Repeat the steps from 1 to 5, and so on.

The linear scaling strategy is used to convert the second part into a real processing location in the range of the active processing locations of the problem according to the following relationship:

$$X_i = \frac{X_i - \min}{\max - \min} \tag{36}$$

where min and max are the max and min values of the second part, respectively.

D. FS operation

The frameshifting approach plays a specific function in optimizing processes to attain the best results in the optimization field [38]. One can use frameshifting's special properties to strategically apply it and synthesize a range of proteins from a properly selected parent source. This approach is very useful for enhancing biological systems since it adds flexibility and adaptability to the production of proteins with various functions. The optimization process depends on how precisely frameshifting is applied, which enables the modification of protein outputs according to particular needs. By providing a sophisticated tool for precisely and effectively customizing molecular processes to produce desired outcomes, this novel technique aids in improving optimization tactics. We employ two forms of the frameshifting technique to enhance the behavior of the EWA: The Right Frameshifting technique (RFS) and The Lift Frameshifting technique (LFS). The selection of the FS technique is based on a randomly generated number R. The first technique, RFS, is selected when R is more than 0.5. Otherwise, the LFS is selected. The LFS technique shifts the values of the input vector one position to the left overriding the value in the first position. The value of the last position is generated randomly in the desired range. Where the RFS shifts the values one position in the right direction, overriding the last value. The first value is then generated randomly in the desired range.

The FS technique is implemented on the processing locations part of the resultant vector as it is. But in the case of the task list part, we slightly changed the previous logic of the FS techniques. We input the S set of each task to the selected FS technique. Then, only the left FS techniques are used to shift the values to the left direction in a cyclic manner. It works as in Fig. 5.

E. LRS technique

To optimize the allocation of tasks across different processing locations, we have introduced a novel approach known as the Load Redistribution Strategy (LRS) method. The LRS method aims to enhance the system's overall performance by dynamically redistributing tasks based on the current workload status of processing locations. Here's how the LRS method works:

- **Identifying high and low load locations:** Initially, we identify the two processing locations with the highest and lowest finish times. This step is crucial for determining where the load needs to be redistributed.
- **Continuous load balancing:** We continuously update the processing locations of tasks, which involves moving tasks from the location with the maximum finish time to the one with the lowest finish time. This iterative process continues until either the finish time of the location with the lowest load reaches or surpasses the best achievable application completion time or the maximum allowable completion time.

To provide a more detailed understanding of the LRS procedure, Algorithm 2 outlines the pseudocode that guides the execution of the Load Redistribution Strategy. In Algorithm 3, we introduce the Enhanced Weighted Average (EWA) algorithm, which complements the Load Redistribution Strategy to optimize task allocation and improve system efficiency.

Algorithm 2. . LRS algorithm

Input: best whale X^*, tasks number N, processing locations Size M, best finish time FT^*.
Output: improved whale vector X^*

get the finish time for each processing location FT_i
m_{min},m_{max} = processing location with min and max finish times, respectively
While ($FT(m_{min}) < FT(m_{max})$ && $FT(m_{min}) < FT^*$)
    Replace m_{max} with m_{min} in X^*
    Update the finish times for the updated processing locations.
End while

Algorithm 3. . The EWA algorithm

Input: population size pop_size, number of tasks N, number of iterations iters
Output: best Solution X_{best}

```
Initialize the pop_size solutions using algorithm 1.
Evaluate the fitness function for each solution.
Find the whale agent with minimum fitness as X*.
t=1
Do
    For each agent X in the population
        Update WOA parameters (a, A, c, p)
        If (p < 0.5 and |A| < 0.5)
            Update current position using eq.(32)
        Else if (P < 0.5 and |A| ≥ 1)
            Update current position using eq.(29)
        Else
            Update current position using eq.(34)
        End if
        Normalize the resultant vector.
    End for
    Update X* if possible
    R=generate random number
    If(R>0.5)
      Apply RFS on X*
    Else
      Apply LFS on X*
    End If
    Update X* if possible
    Apply LRS on X*
    Update X* if possible
    t++
While ( t < iters)
```

5. Numerical experiments and results

The effectiveness of the EWA algorithm is assessed in this part through several experiments and empirical tests. The notebook used for all testing experiments has the following features. The RAM capacity was 8 GB, and the processor's specifications were Intel (R) Core (TM) i7–3540 M @ 3.00 GHz. There was a Windows 10 Professional 64-bit operating system loaded. Java was used to design all of the simulated experiments.

A. Datasets description

This subsection provides a detailed overview of the datasets utilized in our experimental investigations. The dataset files we employed are readily accessible and can be obtained from the source [78]. These datasets are instrumental in facilitating our research and evaluating the performance of our proposed methodologies. The task graphs featured in the adopted datasets are categorized into three distinct groups, each comprising 100 graphs. Let's delve into the characteristics of each group:

- **First group (N=9 Nodes):** The initial collection of graphs shares a common topology, consisting of N=9 nodes. According to the original dataset specifications, six tasks are amenable to offloading, while the remaining three must be executed locally.
- **Second group (N=29 Nodes):** The second collection of graphs retains the same network topology as the first group, with N=29 nodes. However, 20 tasks can be offloaded to remote resources in this case.
- **Third group (N=23 Nodes):** In the third group of networks, we again maintain the same underlying topology with N=23 nodes. However, this time, 19 tasks within each graph can be offloaded to external computational resources.

It's worth noting that users can adjust the number of components within each task graph that are amenable to offloading. For our experiment, we presume that all tasks within each graph can be computationally offloaded, allowing us to assess the performance of our proposed methodologies comprehensively.

From each graph group within the dataset mentioned earlier, we selected three task graphs to include in our experiment. In addition to utilizing the existing dataset, we introduced randomization to generate input data Q_{in}, ranging between 10 and 100 megabytes, and output data Q_{out}, ranging from 1 to 10 megabytes. This randomization was introduced for testing and evaluating the robustness of our methodologies. For a comprehensive understanding of the task graphs utilized in our experiment, refer to Table 3, where each graph's characteristics and specifications are thoroughly elucidated.

Table 3
Used datasets.

No	Dataset Name	No. of Nodes	No. of edges	Required CC in megabytes	Q_{in} in megabytes	Q_{out} in megabytes
1	D1–41	9	10	[3,60]	[10,100]	[1,10]
2	D1–83	9	10	[6,50]		
3	D1–100	9	10	[2,78]		
4	D2–1	29	36	[3,37]		
5	D2–49	29	36	[0220]		
6	D2–100	29	36	[2204]		
7	D3–2	23	22	[0.0, 148.1]		
8	D3–61	23	22	[0.1, 72.2]		
9	D3–100	23	22	[0.0, 102.8]		
10	D_EX	10	15	[10,30]		

B. Experiment configuration and metrics

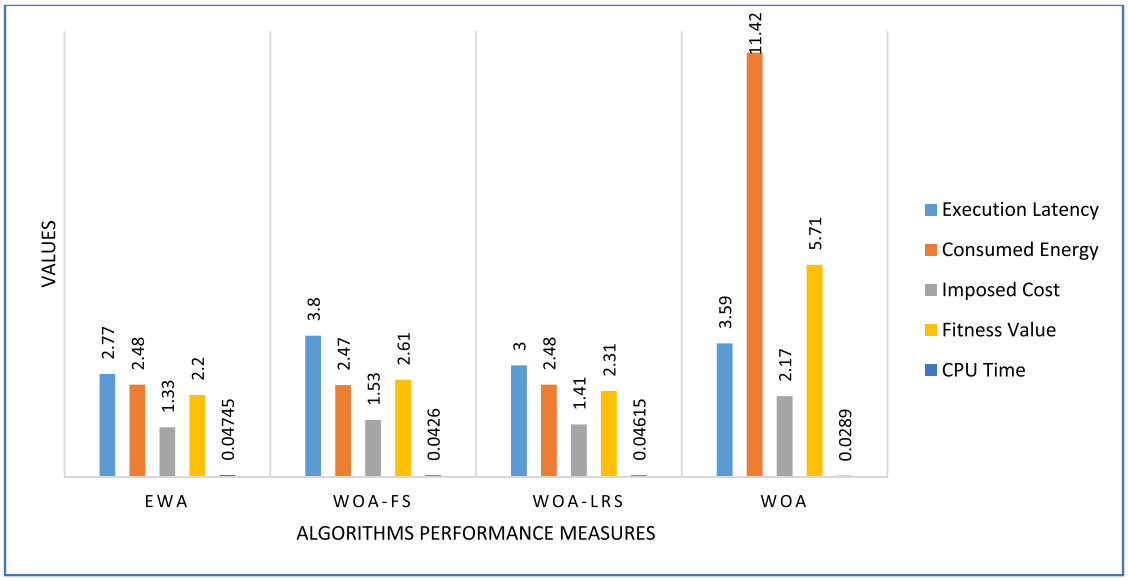

Fig. 6. : Performance measures' values for running different derivatives of the WOA algorithm on the D2–49 dataset.

Table 4
Detailed parameters of the proposed system.

Parameter	Value
Number of MECs (M)	4
The capacitance of the operating chip (C)	10^{-6}
Transmission power (TP)	0.5 Watt
Receiving power (RP)	0.1 Watt
Background noise power (σ^2)	10^{-26} Watt
Local operating frequency (Flocal)	0.5 GHZ
Operating frequencies of the MECs (FMEC)	[5,10] GHZ
Cloud operating frequency (Fcloud)	20 GHZ
System bandwidth (W)	0.5 MHZ
Distance between local device and MECs (L)	[10,30] m
Cloud propagation delay	10^{-6} s
MEC cost per second	0.01–0.05 $
Cloud cost per second	0.09 $
Tasks inputs Q_{in}	5–6 Mb
Tasks outputs Q_{out}	0.5–1.5 Mb
Tasks required clock cycles	100–500 MHZ

We'll assume that M = 4 MECs will be accessible for the experiment's proof-of-concept. We assume that the operating capacitance of the local device chip is 10^{-26}, the transmitting power is 0.5 watt, and the receiving power is 0.1 watt [41,65,79]. The channel gain G_k can be calculated using the following equation [80]:

$$G_k = \frac{1}{(L_k)^4} \tag{37}$$

L is the distance between the local device and the MEC server. We also assume that the data is sent to the cloud through the nearest base station to the local device. It is also supposed that the backhaul cable type is fiber optics cable [81]. The general experimental settings are shown in Table 3.

We compare our proposed EWA algorithm with the following Optimization Algorithms (OA):

- WOA without enhancement [34].
- COVIDOA algorithm [38].
- Gorilla troops algorithm (GTO) [82].
- Harris hawks optimizer (HHO) [83].
- Grey wolf Optimization (GWO) algorithm [84].
- Particle Swarm Optimization (PSO) [85].
- Genetic algorithm (GA) [86].

To ensure a fair and thorough evaluation of the algorithms, we established certain parameters and procedures:

- **Number of iterations:** Given the relatively small number of tasks in the described datasets, we set the number of iterations to 200. This decision allowed ample opportunities for the algorithms to converge and produce meaningful results.
- **Population size:** Each algorithm's population size was constant at 10. This parameter influences the diversity and exploration capability of the algorithms during optimization.
- **Multiple runs for evaluation:** To assess the quality and consistency of each algorithm's performance, we executed each program 20 times. This approach enables us to gather sufficient data and draw robust conclusions about the algorithms' effectiveness.

These parameters and procedures were carefully chosen to ensure a rigorous and reliable evaluation of the algorithms' performance under experimental conditions.

The parameters for the comparison algorithms were selected per the recommendations provided by the respective authors. Here's a breakdown of the parameters for each algorithm:

EWA algorithm:

Regarding EWA, the frameshifting probability factor is 0.5. This factor plays a role in the algorithm's decision-making process and helps introduce variability for improved exploration. Several frameshifting probabilities, including 0.2, 0.3, 0.4, 0.5, 0.6, and 0.7, are evaluated; nevertheless, 0.5 is shown to be the most appropriate probability based on the outcomes of several tests on various random datasets. Additionally, the parameters P and A of the WOA are set to 0.5 and 1, respectively.

Genetic algorithm (GA):

- Mutation Rate: Fixed at 0.001.
- Crossover Operation: A custom crossover operation was designed for this algorithm. It involves selecting a random task number, changing its processing location, and shuffling the task's successor list within the Directed Acyclic Graph (DAG).

Gorilla troops optimization (GTO):

- W Parameter: Set at 0.8, controlling the choice of exploitation operations.

Table 5
Total Average performance metrics.

OA APM	Latency	Energy Consumption	Usage Cost	Fitness function
Proposed EWA	**19.44**	**16.37**	**6.41**	**14.11**
WOA [34]	23.26	46.07	13.4	27.55
COVIDOA [38]	21.9	31.19	11.18	21.44
GTO [82]	22.18	29.21	12.32	21.25
HHO [83]	23.5	37.49	15.93	25.63
GWO [84]	23.34	75.38	13.55	37.3
PSO [85]	22.44	99.71	12.42	44.62
GA [86]	22.72	89.8	13.09	41.67

- P Parameter: Set at 0.03, controlling the choice of exploration operations.
- Beta Parameter: Used to calculate the coefficient vector that indicates the level of aggression or violence in conflicts, and it's set at 3.

PSO algorithm's key settings, based on guidance provided in [87,88], are as follows:

- Intellect Coefficient: Set to 1.42.
- Social Coefficient: Set to 1.57.
- Inertia Weight: Maintained at 0.7298.

These specific parameter settings for each algorithm were chosen to align with established practices and recommendations from the respective algorithm authors. They play a crucial role in shaping the algorithms' behavior and their ability to find optimal solutions in the task offloading problem. The assessment of performance involves an analysis of key metrics to gauge efficiency. Firstly, task completion delay is a pivotal metric, encapsulating the total delay incurred in data transmission and computation across all tasks. Secondly, energy consumption emerges as a critical factor, encapsulating the overall energy expenditure associated with the transmission and computation of tasks. Lastly, the evaluation extends to the resource utilization cost in both

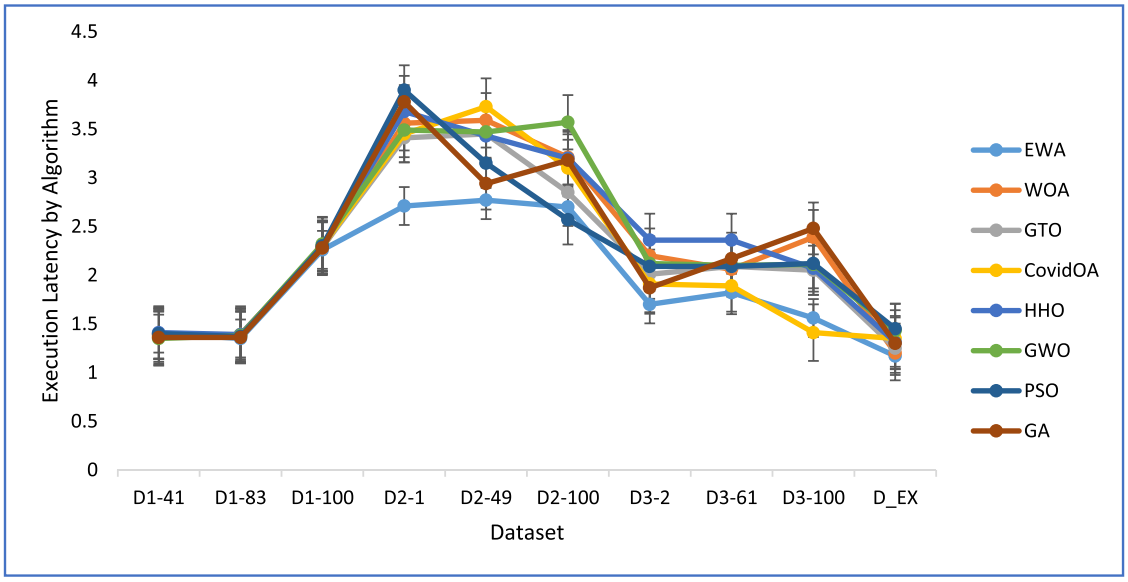

Fig. 7. : Execution latency results for running comparison algorithms on the test datasets.

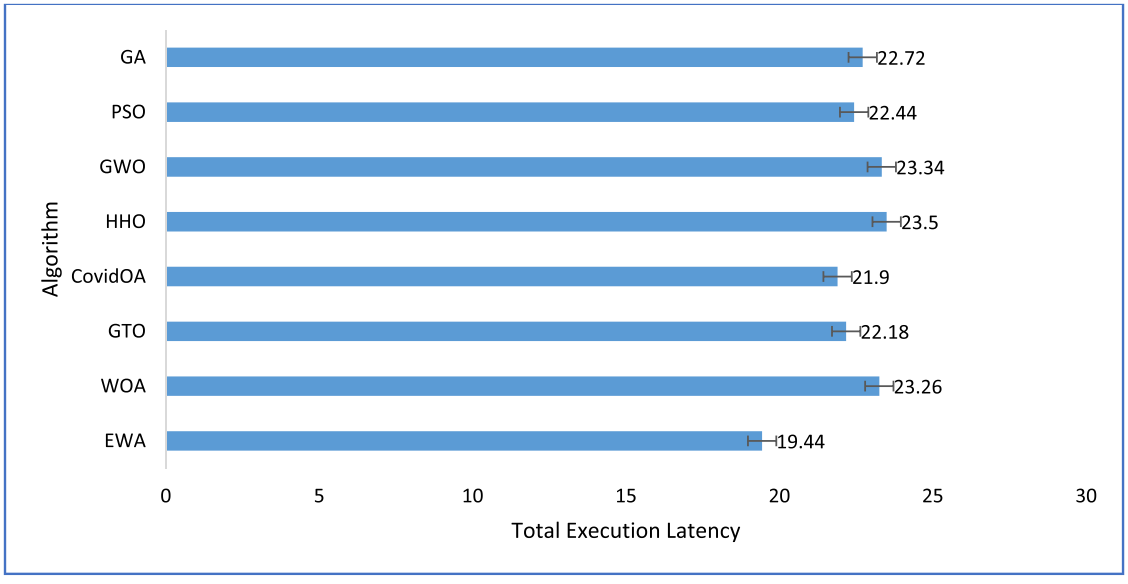

Fig. 8. : Total Execution latency results for running comparison algorithms on the test datasets.

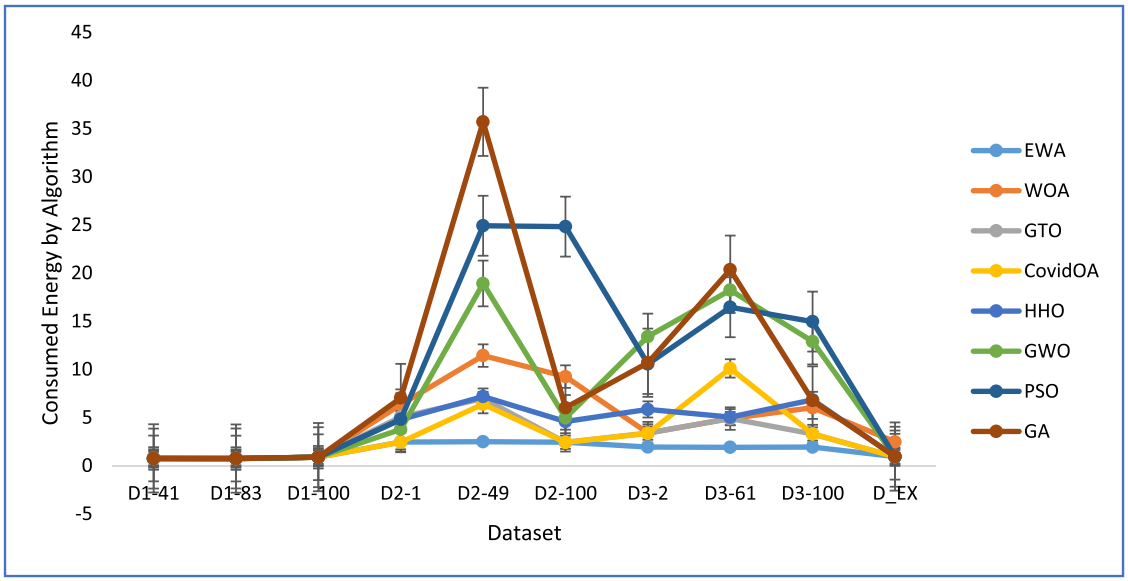

Fig. 9. : Energy consumption results for running comparison algorithms on the test datasets.

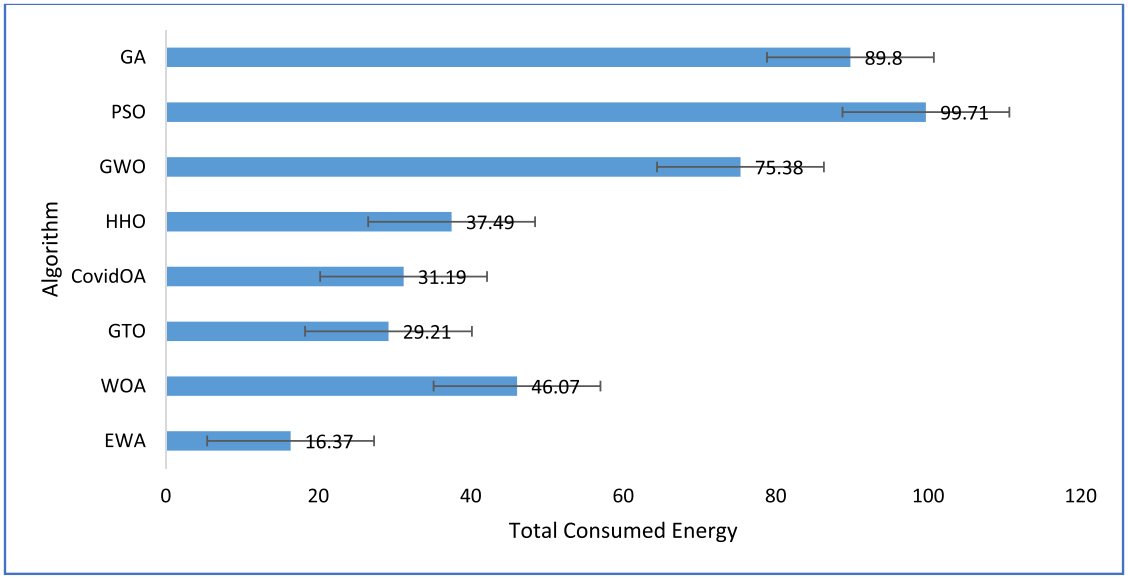

Fig. 10. : Total Energy Consumption results for running comparison algorithms on the test datasets.

Mobile MEC and cloud servers. These three metrics collectively offer a holistic perspective on the system's efficiency, energy efficiency, and economic viability under examination.

C. Testing the impact of the enhancement operations

This experiment focuses on examining how the FS and LRS enhancement operations contributed to the proposed algorithm. This experiment uses four different metrics, including average values for finishing time, energy consumption, usage cost, and fitness. The CPU time is also provided as extra information in our experiment. We used the dataset D2-49 presented in Table 3 for this experiment.

The results of four algorithm derivatives (WOA, EWA, WOA-FS, WOA-LRS) are shown in Fig. 6 and are categorized based on the previously mentioned indicators. The results demonstrate the contribution of each operation to the proposed algorithm. It can be seen from the figure that the LRS operation enhanced the standard WOA in all performance measures as follows: the execution latency is improved by 16.43%, the energy consumption is reduced by 78.28%, the Imposed cost is minimized by 35.02%, and the fitness function is improved by 59.54%. The FS improvement operation also has an impact on the produced results. It improved the energy consumption by 78.37%, the Imposed cost by 29.49%, and the fitness function by 54.29% where it fails to improve the execution latency of the application.

In examining the results, it is important to note that the FS operation, simulating COVID-19 spreading behavior within the proposed algorithm, positively influenced energy consumption, cost, and fitness function. However, the lack of improvement in execution latency raises questions about the operation's adaptability to address challenges specific to this metric. Given the unique context of simulating a pandemic scenario, inherent trade-offs may exist, and the complexities introduced by the FS operation may impact execution latency differently. A deeper exploration of these nuances is essential for understanding the operation's strengths and limitations in the context of COVID-19 simulation, guiding future optimization efforts for a more balanced algorithmic performance.

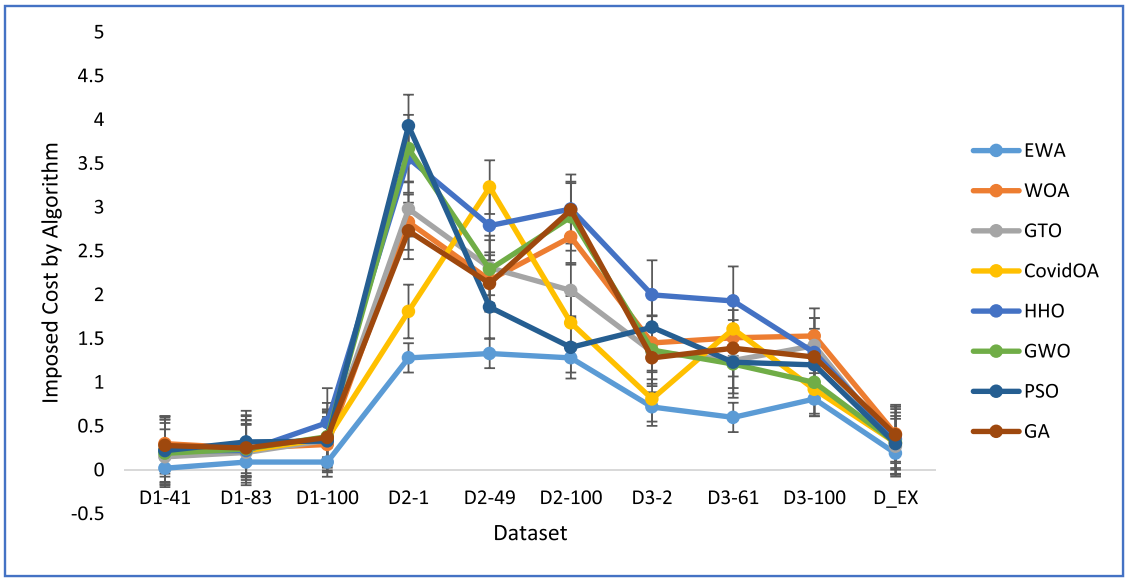

Fig. 11. : Cost usage results for running comparison algorithms on the test datasets.

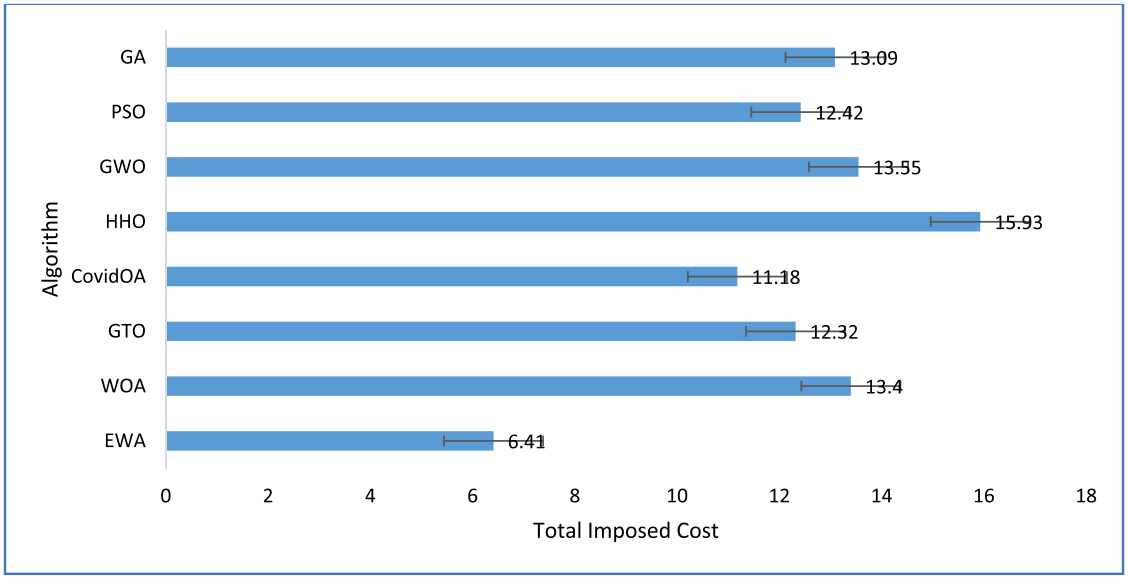

Fig. 12. : Total cost usage results for running comparison algorithms on the test datasets.

Combining these two operations has achieved an optimized EWA that enhanced the execution latency by 22.84%, the energy consumption by 78.28%, the Imposed cost by 38.71%, and the fitness function by 61.47%. The CPU time is also provided to show each operation's impact on the algorithm's running time.

This framework seeks to make it possible for developers of IoT and mobile applications to create effective applications that can most effectively benefit from the MEC and cloud servers. For this reason, our suggested algorithm is used in the design phase in most cases. Since the algorithm's running time in the design phase has no impact on the results, we do not consider it a performance indicator. Furthermore, we propose deploying the EWA algorithm on the MEC server within MEC environments where the relative importance of objectives dynamically evolves. This strategic deployment can help mitigate the potential impact of EWA's relatively long CPU processing time on overall performance metrics, particularly when dealing with computationally intensive applications. We can optimize the algorithm's efficiency and responsiveness in dynamically changing scenarios by offloading the computational burden to the MEC server.

D. Comparison with other algorithms

By using the datasets listed in Table 3 and the parameters' settings presented in Table 4, this subsection compares the efficacy of the suggested algorithm with that of WOA, GTO, HHO, GWO, COVIDOA, PSO, and GA. The efficiency of the algorithms is assessed using four performance metrics: completion time, energy consumption, cost utilization, and fitness function. We execute each algorithm twenty times on each dataset. Repeated runs allow us to assess the robustness of the optimization algorithm precisely. A good optimization algorithm should consistently perform well across multiple executions. Averaging results helps identify algorithms that consistently provide better solutions and are less sensitive to variations in initial conditions or random factors. As a result, we used the following equation to determine the average for each performance measure (APM) utilized.

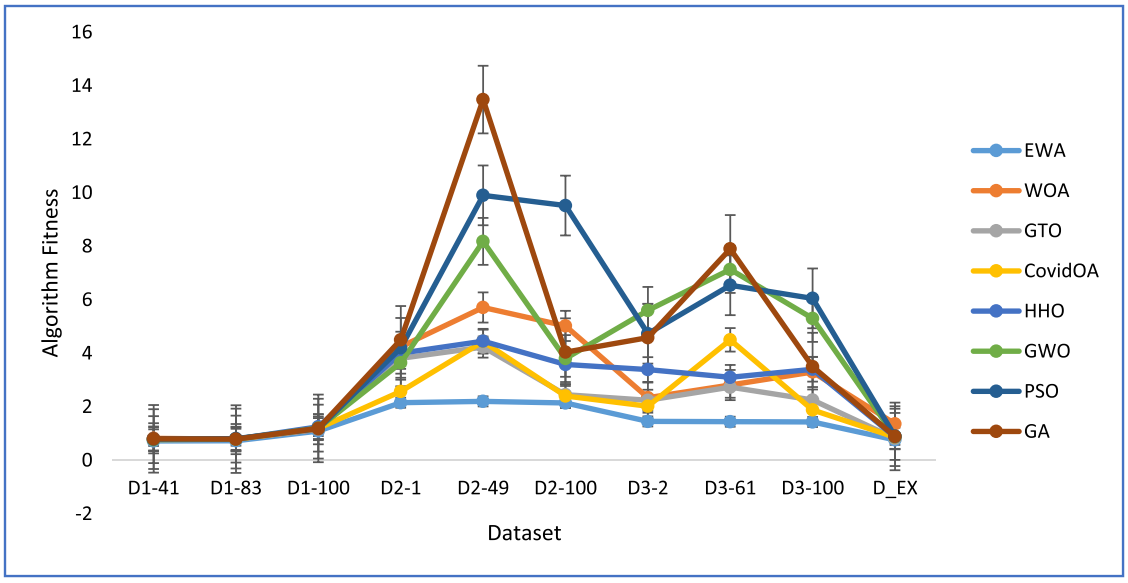

Fig. 13. : Fitness function results for running comparison algorithms on the test datasets.

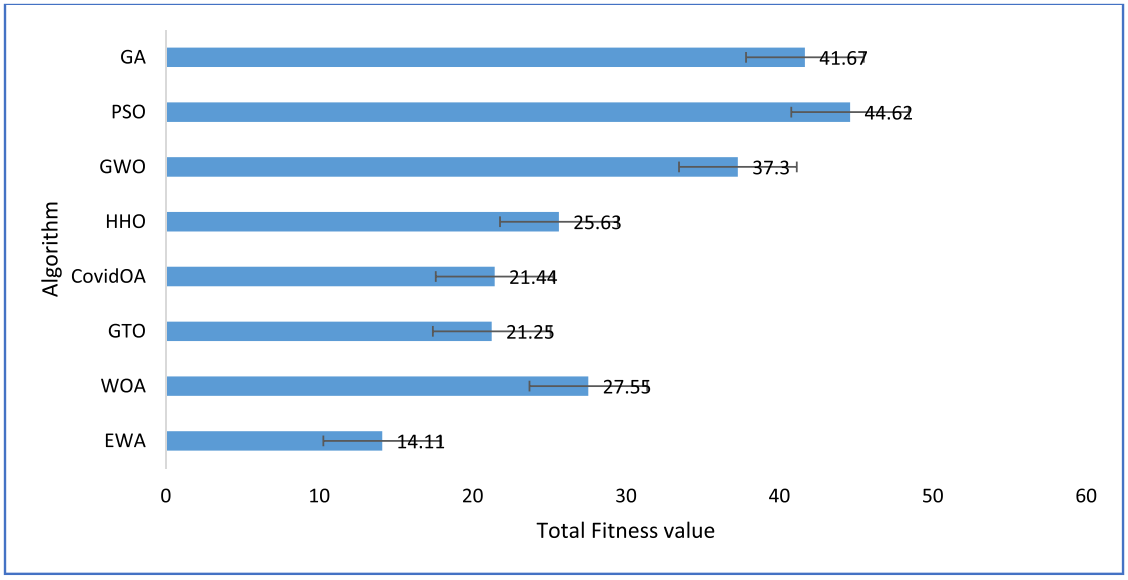

Fig. 14. : Total fitness results for running comparison algorithms on the test datasets.

$$APM = \frac{\sum_{j=1}^{20} PM_j}{20} \tag{38}$$

where PM_j is the performance indicator value produced from running an algorithm on a dataset. Table 5 shows the comparison algorithms results on the adopted datasets. The table contains average values for each performance metric over the used datasets. On the chosen datasets, each algorithm's average completion times are shown in Fig. 8. On all of the used datasets, the suggested algorithm outperforms the competition, as shown in the figure. It is also clear that different datasets affect the rankings of different comparison algorithms. This finding confirms that our datasets verified the performance metrics of the comparison approaches.

The average latency durations for each algorithm on the chosen datasets are displayed in Fig. 7. The graph demonstrates how the suggested algorithm wins the competition on most of the used datasets. These findings confirm that our datasets verified the comparison approaches' performance metrics since the comparison algorithms' outcomes change in ranking from one dataset to the other. Additionally, Fig. 8 displays the aggregate execution latencies for the employed data sets, calculated by adding the execution latencies generated by applying each method to all employed datasets. According to the findings, our algorithm offers a minimum execution latency of 19.44 seconds.

Energy consumption is the second performance metric used for the comparison. Since batteries are the main source of electricity for mobile and IoT devices, this metric is very important. Fig. 9 illustrates how our algorithm significantly improved according to this measure for almost all adopted test cases. Fig. 10 provides the findings of 20 iterations of each algorithm on each task graph set to provide more precise values. The graph demonstrates that the suggested algorithm achieved the minimum energy consumption value of 16.37 J.

The third performance metric, crucial for creating an economic strategy, is the cost incurred when carrying out application logic. Fig. 11 displays the outcomes of running each algorithm on the test datasets of Table 3 using the parameters configuration in Table 3. The outcomes

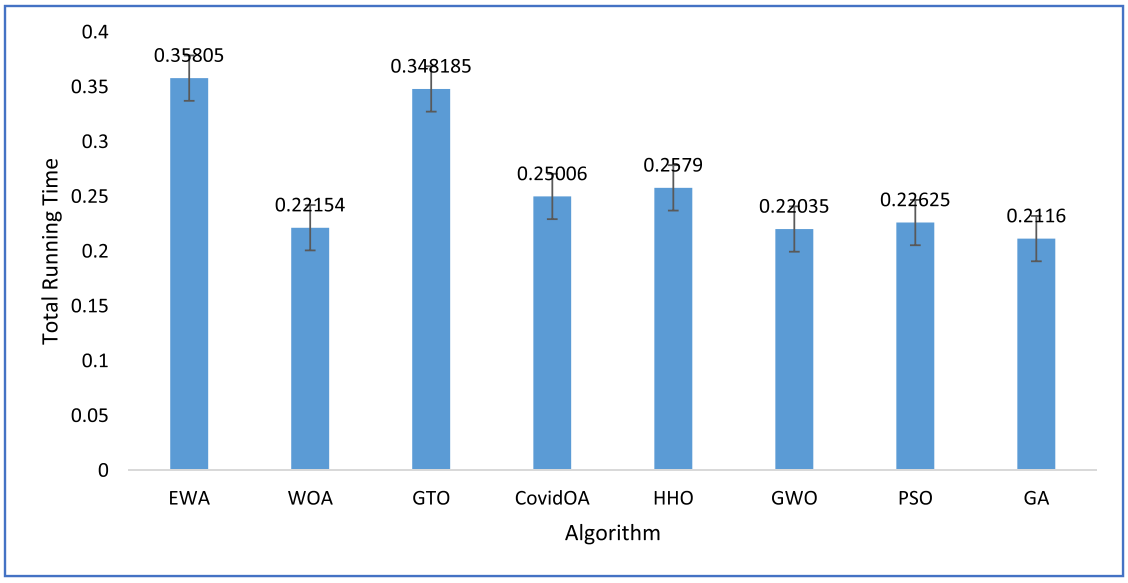

Fig. 15. : Total CPU Time results for running comparison algorithms on the test datasets.

clearly show that our algorithm incorporates the minimal cost levels for all test sets. Fig. 12 also served to quantify the cost usage outcomes as determined by the comparison algorithms from an obvious point of view. The graph demonstrates that the suggested method reached the most optimized cost of 6.41 dollars.

Since the fitness values provide a broad perspective that includes all the metrics mentioned earlier, they are crucial in evaluating the algorithmic performance. The test groups' average fitness for each algorithm is shown in Fig. 13. This outcome supported other metrics' findings that the proposed algorithm outperformed competing strategies. Additionally, Fig. 14 demonstrates that IGTA achieves the most optimized fitness value of 14.11. Fig. 15 displays the total CPU time for each method on the adopted datasets.

In presenting our research findings, we have included error bars in all result figures to provide a comprehensive and nuanced representation of the data. These error bars serve as visual indicators of the variability or uncertainty associated with each data point, offering insights into the precision and reliability of our measurements. By incorporating error bars, we aim to enhance transparency in our data presentation, enabling viewers to assess the statistical significance of observed differences and the overall robustness of our results. This approach aligns with established standards in scientific communication, facilitating a clearer understanding of the inherent uncertainties and contributing to the credibility and reproducibility of our study.

We can conclude from the thorough tests that the suggested approach has outperformed other meta-heuristic algorithms regarding energy conservation, expense reduction, and computation latency.

6. Conclusion and future directions

In this research, three objectives—the application execution latency, smartphone energy consumption, and MEC and cloud servers usage fee—were simultaneously optimized in a multi-edge cloud computing system with multi-task dependence. Additionally, in the MEC environments, goals' relative importance changes dynamically over time. We suggested an enhanced whale optimization algorithm with two improvement operations to address these challenges. The first operation is the frameshifting operation, which is inspired by the replication behavior of the COVID-19 virus. The second enhancement operation is the load redistribution strategy which inforce a fair load balancing among the included processing locations. Each agent vector in the suggested technique was divided into two parts: a task list part and a processing locations list part. Additionally, a specific initialization technique was used to produce useful agent vectors. Since the resultant vectors generated by the WOA algorithm are continuous values, we adopted mapping methods to transform them into discrete and bounded vector values.

We have conducted comprehensive simulation experiments on ten test instances with different task topologies and profiles to verify EWA performance. These tests compared EWA to the WOA, GTO, COVIDOA, HHO, GWO, Genetic algorithm, and PSO. Finally, the simulation outcomes supported the superiority of the performance measures developed by EWA. For example, EWA has decreased latency by 22.84%, energy consumption by 78.28%, and cost usage by 61.47% compared to WOA.

For future research, we recommend the following points for authors in this area: 1- considering the mobility of users in the multi-edge cloud computing system, 2- considering wireless power transfer technology for battery-operated devices, 3- providing less running times optimization algorithms with more improvement on the showed performance metrics, 4- implementing the provided algorithm for solving other optimization problems like feature selection and other areas in service computing.

CRediT authorship contribution statement

Wael Said: Funding acquisition, Resources. **Ahmed I. Awad:** Conceptualization, Methodology, Software, Writing – original draft. **Ehab R. Mohamed:** Data curation, Investigation, Supervision. **Mahmoud Elmezain:** Data curation, Formal analysis. **Khalid M Hosny:** Conceptualization, Supervision, Writing – review & editing, Methodology. **Marwa M. Khashaba:** Investigation, Project administration, Validation, Visualization.

Declaration of Competing Interest

The authors declare that they have no conflict of interest.

Acknowledgments

The authors extend their appreciation to the Deputyship for Research & Innovation, Ministry of Education in Saudi Arabia for funding this research work through the project number 445–9-571.

References

[1] A.I. Awad, M.M. Fouda, M.M. Khashaba, E.R. Mohamed, K.M. Hosny, "Utilization of mobile edge computing on the Internet of medical things: a survey,", ICT Express vol.9 (3) (2023) 473–485, https://doi.org/10.1016/j.icte.2022.05.006.

[2] M. Goudarzi, H. Wu, M. Palaniswami, R. Buyya, "An application placement technique for concurrent iot applications in edge and fog computing environments,", IEEE Trans. Mob. Comput. vol. 20 (4) (2021) 1298–1311, https://doi.org/10.1109/TMC.2020.2967041.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, "Edge computing: vision and challenges,", IEEE Internet Things J. vol. 3 (5) (2016) 637–646, https://doi.org/10.1109/JIOT.2016.2579198.

[4] X. Dai, et al., "A learning-based approach for vehicle-to-vehicle computation offloading,", IEEE Internet Things J. vol. 10 (8) (2023) 7244–7258, https://doi.org/10.1109/JIOT.2022.3228811.

[5] S. Pang, L. Hou, H. Gui, X. He, T. Wang, Y. Zhao, "Multi-mobile vehicles task offloading for vehicle-edge-cloud collaboration: a dependency-aware and deep reinforcement learning approach,", Comput. Commun. vol. 213 (October 2023) (2023) 359–371, https://doi.org/10.1016/j.comcom.2023.11.013.

[6] X. Dai, Z. Xiao, H. Jiang, J.C.S. Lui, "UAV-assisted task offloading in vehicular edge computing networks,", IEEE Trans. Mob. Comput. vol. PP (2023) 1–15, https://doi.org/10.1109/TMC.2023.3259394.

[7] Y. Guo, C. Zhang, C. Wang, X. Jia, "Towards public verifiable and forward-privacy encrypted search by using blockchain,", IEEE Trans. Dependable Secur. Comput. vol. 20 (3) (2023) 2111–2126, https://doi.org/10.1109/TDSC.2022.3173291.

[8] Z. Xia, J.A. Abu Qahouq, "State-of-charge balancing of lithium-ion batteries with state-of-health awareness capability,", IEEE Trans. Ind. Appl. vol. 57 (1) (2021) 673–684, https://doi.org/10.1109/TIA.2020.3029755.

[9] A. Sarah, G. Nencioni, M.M.I. Khan, "Resource allocation in multi-access edge computing for 5G-and-beyond networks,", Comput. Netw. vol. 227 (308909) (2023) 109720, https://doi.org/10.1016/j.comnet.2023.109720.

[10] P. Mach, Z. Becvar, "Mobile edge computing: a survey on architecture and computation offloading,", arXiv vol. 19 (3) (2017) 1628–1656.

[11] S. Kekki, et al., "【ETSI白皮书】MEC in 5G networks,", ETSI White Pap. (28) (2018) 1–28.

[12] Z. Xiao, J. Shu, H. Jiang, G. Min, H. Chen, Z. Han, "Perception task offloading with collaborative computation for autonomous driving,", IEEE J. Sel. Areas Commun. vol. 41 (2) (2023) 457–473, https://doi.org/10.1109/JSAC.2022.3227027.

[13] N. Kumari, A. Yadav, P.K. Jana, "Task offloading in fog computing: a survey of algorithms and optimization techniques,", Comput. Netw. vol. 214 (February) (2022), https://doi.org/10.1016/j.comnet.2022.109137.

[14] K. Guo, R. Zhang, "Fairness-oriented computation offloading for cloud-assisted edge computing,", Futur. Gener. Comput. Syst. vol. 128 (2022) 132–141, https://doi.org/10.1016/j.future.2021.10.004.

[15] F. Saeik, et al., "Task offloading in edge and cloud computing: a survey on mathematical, artificial intelligence and control theory solutions,", Comput. Netw. vol. 195 (January) (2021) 108177, https://doi.org/10.1016/j.comnet.2021.108177.

[16] A. Reznik *et al.*, Developing Software for Multi-Access Edge Computing, no. 20. 2017.

[17] M. Huang, Q. Zhai, Y. Chen, S. Feng, F. Shu, "Multi-objective whale optimization algorithm for computation offloading optimization in mobile edge computing,", Sensors vol. 21 (8) (2021) 1–24, https://doi.org/10.3390/s21082628.

[18] R. Aldmour, S. Yousef, M. Yaghi, S. Tapaswi, K.K. Pattanaik, M. Cole, "New cloud offloading algorithm for better energy consumption and process time,", Int. J. Syst. Assur. Eng. Manag. vol. 8 (s2) (2017) 730–733, https://doi.org/10.1007/s13198-016-0515-2.

[19] B. Cao, Z. Li, X. Liu, Z. Lv, H. He, "Mobility-aware multiobjective task offloading for vehicular edge computing in digital twin environment,", IEEE J. Sel. Areas Commun. vol. 41 (10) (2023) 3046–3055, https://doi.org/10.1109/JSAC.2023.3310100.

[20] Z. Xiao, et al., "Multi-objective parallel task offloading and content caching in D2D-aided MEC networks,", IEEE Trans. Mob. Comput. vol. 22 (11) (2023) 6599–6615, https://doi.org/10.1109/TMC.2022.3199876.

[21] H. Jin, M.A. Gregory, S. Li, "A review of intelligent computation offloading in multiaccess edge computing,", IEEE Access vol. 10 (July) (2022) 71481–71495, https://doi.org/10.1109/ACCESS.2022.3187701.

[22] A. Pashazadeh and G. Nardini, "A Comprehensive Survey Exploring the Multifaceted Interplay between Mobile Edge Computing and Vehicular Networks," 2023.

[23] X. Wu, S. Dong, J. Hu, Z. Huang, "An efficient many-objective optimization algorithm for computation offloading in heterogeneous vehicular edge computing network,", Simul. Model. Pract. Theory vol. 131 (March 2023) (2024), https://doi.org/10.1016/j.simpat.2023.102870.

[24] X. Gao, M.C. Ang, S.A. Althubiti, "Deep reinforcement learning and markov decision problem for task offloading in mobile edge computing,", J. Grid Comput. vol. 21 (4) (2023) 1–16, https://doi.org/10.1007/s10723-023-09708-4.

[25] Z. Wan, D. Xu, D. Xu, I. Ahmad, "Joint computation offloading and resource allocation for NOMA-based multi-access mobile edge computing systems,", Comput. Netw. vol. 196 (June) (2021), https://doi.org/10.1016/j.comnet.2021.108256.

[26] A. Shahidinejad, M. Ghobaei-Arani, "A metaheuristic-based computation offloading in edge-cloud environment,", J. Ambient Intell. Humaniz. Comput. vol. 13 (5) (2022) 2785–2794, https://doi.org/10.1007/s12652-021-03561-7.

[27] Y. Fu, X. Zhang, X. Qin, Q. Meng, B. Huang, "Data collection of multi-player cooperative game based on edge computing in mobile crowd sensing,", Comput. Netw. vol. 222 (November 2022) (2023) 109551, https://doi.org/10.1016/j.comnet.2022.109551.

[28] A. Shakarami, A. Shahidinejad, M. Ghobaei-Arani, "A review on the computation offloading approaches in mobile edge computing: a game-theoretic perspective,", Softw. - Pract. Exp. vol. 50 (9) (2020) 1719–1759, https://doi.org/10.1002/spe.2839.

[29] A. Shakarami, M. Ghobaei-Arani, A. Shahidinejad, "A survey on the computation offloading approaches in mobile edge computing: a machine learning-based perspective,", Comput. Netw. vol. 182 (August) (2020) 107496, https://doi.org/10.1016/j.comnet.2020.107496.

[30] Z. Ali, L. Jiao, T. Baker, G. Abbas, Z.H. Abbas, S. Khaf, "A deep learning approach for energy-efficient computational offloading in mobile edge computing,", IEEE Access vol. 7 (2019) 149623–149633, https://doi.org/10.1109/ACCESS.2019.2947053.

[31] M. Tang, V.W.S. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems,", IEEE Trans. Mob. Comput. vol. 21 (6) (2022) 1985–1997, https://doi.org/10.1109/TMC.2020.3036871.

[32] K. Sadatdiynov, L. Cui, L. Zhang, J.Z. Huang, S. Salloum, M.S. Mahmud, "A review of optimization methods for computation offloading in edge computing networks,", Digit. Commun. Netw. vol. 9 (2) (2022) 450–461, https://doi.org/10.1016/j.dcan.2022.03.003.

[33] S. Mirjalili, A. Lewis, "The whale optimization algorithm,", Adv. Eng. Softw. vol. 95 (May 2016) 51–67, https://doi.org/10.1016/J.ADVENGSOFT.2016.01.008.

[34] F.S. Gharehchopogh, H. Gholizadeh, "A comprehensive survey: whale optimization algorithm and its applications,", Swarm Evol. Comput. vol. 48 (November 2018) (2019) 1–24, https://doi.org/10.1016/j.swevo.2019.03.004.

[35] F. Song, H. Xing, S. Luo, D. Zhan, P. Dai, R. Qu, "A multiobjective computation offloading algorithm for mobile-edge computing,", IEEE Internet Things J. vol. 7 (9) (2020) 8780–8799, https://doi.org/10.1109/JIOT.2020.2996762.

[36] Z. Yan, J. Zhang, J. Zeng, J. Tang, "Nature-inspired approach: an enhanced whale optimization algorithm for global optimization,", Math. Comput. Simul. vol. 185 (2021) 17–46, https://doi.org/10.1016/j.matcom.2020.12.008.

[37] G.Y. Ning, D.Q. Cao, "Improved whale optimization algorithm for solving constrained optimization problems,", Discret. Dyn. Nat. Soc. vol. 2021 (2021), https://doi.org/10.1155/2021/8832251.

[38] A.M. Khalid, K.M. Hosny, S. Mirjalili, "COVIDOA: a novel evolutionary optimization algorithm based on coronavirus disease replication lifecycle,", Neural Comput. Appl. vol. 34 (24) (2022) 22465–22492, https://doi.org/10.1007/s00521-022-07639-x.

[39] R. Aldmour, S. Yousef, T. Baker, E. Benkhelifa, "An approach for offloading in mobile cloud computing to optimize power consumption and processing time,", Sustain. Comput. Inform. Syst. vol. 31 (2021) 100562, https://doi.org/10.1016/j.suscom.2021.100562.

[40] K. Wang, Z. Ding, D.K.C. So, G.K. Karagiannidis, "Stackelberg game of energy consumption and latency in MEC systems with NOMA,", IEEE Trans. Commun. vol. 69 (4) (2021) 2191–2206, https://doi.org/10.1109/TCOMM.2021.3049356.

[41] J. Zheng, Y. Cai, Y. Wu, X. Shen, "Dynamic computation offloading for mobile cloud computing: a stochastic game-theoretic approach,", IEEE Trans. Mob. Comput. vol. 18 (4) (2019) 771–786, https://doi.org/10.1109/TMC.2018.2847337.

[42] S. Sundar, B. Liang, "Offloading dependent tasks with communication delay and deadline constraint,", Proc. - IEEE INFOCOM vol. 2018-April (2018) 37–45, https://doi.org/10.1109/INFOCOM.2018.8486305.

[43] H. Hou, Z. Chai, X. Liu, Y. Li, Y. Zeng, "A task offloading algorithm using multi-objective optimization under hybrid mode in mobile edge computing,", Mob. Netw. Appl. (2023), https://doi.org/10.1007/s11036-023-02272-x.

[44] H. Peng, W.S. Wen, M.L. Tseng, L.L. Li, "Joint optimization method for task scheduling time and energy consumption in mobile cloud computing environment,", Appl. Soft Comput. J. vol. 80 (2019) (2019) 534–545, https://doi.org/10.1016/j.asoc.2019.04.027.

[45] G. Zhao, H. Xu, Y. Zhao, C. Qiao, L. Huang, "Offloading tasks with dependency and service caching in mobile edge computing,", IEEE Trans. Parallel Distrib. Syst. vol. 32 (11) (2021) 2777–2792, https://doi.org/10.1109/TPDS.2021.3076687.

[46] J. Liu, Y. Mao, J. Zhang, K.B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems,", IEEE Int. Symp. Inf. Theory - Proc. vol. 2016-Augus (2016) 1451–1455, https://doi.org/10.1109/ISIT.2016.7541539.

[47] A.A. Al-Habob, O.A. Dobre, A.G. Armada, S. Muhaidat, "Task scheduling for mobile edge computing using genetic algorithm and conflict graphs,", IEEE Trans. Veh. Technol. vol. 69 (8) (2020) 8805–8819, https://doi.org/10.1109/TVT.2020.2995146.

[48] B. Huang, et al., "Security modeling and efficient computation offloading for service workflow in mobile edge computing,", Futur. Gener. Comput. Syst. vol. 97 (2019) 755–774, https://doi.org/10.1016/j.future.2019.03.011.

[49] Y. Xie, et al., "A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud–edge environment,", Futur. Gener. Comput. Syst. vol. 97 (2019) 361–378, https://doi.org/10.1016/j.future.2019.03.005.

[50] S. Ma, S. Song, L. Yang, J. Zhao, F. Yang, L. Zhai, "Dependent tasks offloading based on particle swarm optimization algorithm in multi-access edge computing,", Appl. Soft Comput. vol. 112 (2021) 107790, https://doi.org/10.1016/j.asoc.2021.107790.

[51] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," Proc. - IEEE INFOCOM, pp. 352–357, 2014, doi: 10.1109/INFCOMW.2014.6849257.

[52] K.M. Hosny, A.I. Awad, M.M. Khashaba, E.R. Mohamed, "New improved multi-objective gorilla troops algorithm for dependent tasks offloading problem in multi-

access edge computing,", J. Grid Comput. vol. 21 (2) (2023), https://doi.org/10.1007/s10723-023-09656-z.
[53] J. Li, Q. Wang, S. Hu, L. Li, "Hybrid immune whale differential evolution optimization (HIWDEO) based computation offloading in MEC for IoT,", J. Grid Comput. vol. 21 (4) (2023) 1–15, https://doi.org/10.1007/s10723-023-09705-7.
[54] L. Liu, H. Tan, S.H.C. Jiang, Z. Han, X.Y. Li, H. Huang, "Dependent task placement and scheduling with function configuration in edge computing,", Proc. Int. Symp. Qual. Serv. IWQoS 2019 (2019), https://doi.org/10.1145/3326285.3329055.
[55] A. Numani, Z.H. Abbas, G. Abbas, Z. Ali, "Energy conserving cost selection for fine-grained computational offloading in mobile edge computing networks,", Comput. Commun. vol. 213 (November 2022) (2024) 199–207, https://doi.org/10.1016/j.comcom.2023.11.012.
[56] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, N. Georgalas, "Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning,", IEEE Commun. Mag. vol. 57 (5) (2019) 64–69, https://doi.org/10.1109/MCOM.2019.1800971.
[57] Q. Wu, Z. Wu, Y. Zhuang, Y. C. B, Adaptive DAG Tasks Scheduling, vol. 1, Springer International Publishing, 2018, https://doi.org/10.1007/978-3-030-05054-2.
[58] J. Wang, J. Hu, G. Min, A.Y. Zomaya, N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning,", IEEE Trans. Parallel Distrib. Syst. vol. 32 (1) (2021) 242–253, https://doi.org/10.1109/TPDS.2020.3014896.
[59] A. Zhu, et al., "Computation offloading for workflow in mobile edge computing based on deep Q-learning," (no. Wocc), 2019 28th Wirel. Opt. Commun. Conf. WOCC 2019 - Proc. (2019) 1–5, https://doi.org/10.1109/WOCC.2019.8770689.
[60] G. Qu, H. Wu, R. Li, P. Jiao, "DMRO: a deep meta reinforcement learning-based task offloading framework for edge-cloud computing,", IEEE Trans. Netw. Serv. Manag. vol. 18 (3) (2021) 3448–3459, https://doi.org/10.1109/TNSM.2021.3087258.
[61] H. Lu, C. Gu, F. Luo, W. Ding, X. Liu, "Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning,", Futur. Gener. Comput. Syst. vol. 102 (2020) 847–861, https://doi.org/10.1016/j.future.2019.07.019.
[62] J. Yan, S. Bi, Y.J.A. Zhang, "Offloading and resource allocation with general task graph in mobile edge computing: a deep reinforcement learning approach,", IEEE Trans. Wirel. Commun. vol. 19 (8) (2020) 5404–5419, https://doi.org/10.1109/TWC.2020.2993071.
[63] A. Robles-Enciso, A.F. Skarmeta, "A multi-layer guided reinforcement learning-based tasks offloading in edge computing,", Comput. Netw. vol. 220 (September 2022) (2023) 109476, https://doi.org/10.1016/j.comnet.2022.109476.
[64] Z. Aghapour, S. Sharifian, H. Taheri, "Task offloading and resource allocation algorithm based on deep reinforcement learning for distributed AI execution tasks in IoT edge computing environments,", Comput. Netw. vol. 223 (December 2022) (2023) 109577, https://doi.org/10.1016/j.comnet.2023.109577.
[65] F. Song, H. Xing, X. Wang, S. Luo, P. Dai, K. Li, "Offloading dependent tasks in multi-access edge computing: a multi-objective reinforcement learning approach,", Futur. Gener. Comput. Syst. vol. 128 (2022) 333–348, https://doi.org/10.1016/j.future.2021.10.013.
[66] G. Cui, X. Li, L. Xu, W. Wang, "Latency and energy optimization for MEC enhanced SAT-IoT networks,", IEEE Access vol. 8 (2020) 55915–55926, https://doi.org/10.1109/ACCESS.2020.2982356.
[67] S. Feng, Y. Chen, Q. Zhai, M. Huang, F. Shu, "Optimizing computation offloading strategy in mobile edge computing based on swarm intelligence algorithms,", EURASIP J. Adv. Signal Process. vol. 2021 (1) (2021), https://doi.org/10.1186/s13634-021-00751-5.
[68] E. Ahmed, et al., "The role of big data analytics in Internet of Things,", Comput. Netw. vol. 129 (2017) 459–471, https://doi.org/10.1016/j.comnet.2017.06.013.
[69] I.A. Zualkernan, M. Rashid, R. Gupta, M. Alikarar, "Smart Home Big Data," vol. 63 (4) (2017) 426–434.
[70] Y. Sun, H. Song, A.J. Jara, R. Bie, "Internet of things and big data analytics for smart and connected communities,", IEEE Access vol. 4 (2016) 766–773, https://doi.org/10.1109/ACCESS.2016.2529723.
[71] M.M. Rathore, A. Paul, W.H. Hong, H.C. Seo, I. Awan, S. Saeed, "Exploiting IoT and big data analytics: defining smart digital city using real-time urban data,", Sustain. Cities Soc. vol. 40 (2018) 600–610, https://doi.org/10.1016/j.scs.2017.12.022.
[72] M. Agiwal, A. Roy, N. Saxena, "Next generation 5G wireless networks: a comprehensive survey,", IEEE Commun. Surv. Tutor. vol. 18 (3) (2016) 1617–1655, https://doi.org/10.1109/COMST.2016.2532458.
[73] P. Mach, Z. Becvar, "Mobile edge computing: a survey on architecture and computation offloading,", IEEE Commun. Surv. Tutor. vol. 19 (3) (2017) 1628–1656, https://doi.org/10.1109/COMST.2017.2682318.
[74] P.D. Nguyen, L.B. Le, "Joint computation offloading, SFC placement, and resource allocation for multi-site MEC systems,", IEEE Wirel. Commun. Netw. Conf. WCNC vol. 2020-May (2020), https://doi.org/10.1109/WCNC45663.2020.9120597.
[75] J.F. Kurose and K.W. Ross, Computer networking: a top-down approach 7th ed. 2017.
[76] H. Wu, K. Wolter, P. Jiao, Y. Deng, Y. Zhao, M. Xu, "EEDTO: an energy-efficient dynamic task offloading algorithm for blockchain-enabled iot-edge-cloud orchestrated computing,", IEEE Internet Things J. vol. 8 (4) (2021) 2163–2176, https://doi.org/10.1109/JIOT.2020.3033521.
[77] S. Mirjalili, A. Lewis, "The whale optimization algorithm,", Adv. Eng. Softw. vol. 95 (2016) 51–67, https://doi.org/10.1016/j.advengsoft.2016.01.008.
[78] T.H. Szymanski, "300 Pseudo-Random Task Graphs for Evaluating Mobile Cloud, Fog and Edge Computing Systems," pp. 1–4.
[79] M.Z. Chaari, S. Al-Maadeed, "Wireless power transmission for the internet of things (IoT),", 2020 IEEE Int. Conf. Inform., IoT, Enabling Technol. ICIoT 2020 (2020) 549–554, https://doi.org/10.1109/ICIoT48696.2020.9089547.
[80] M. Huang, Q. Zhai, Y. Chen, S. Feng, F. Shu, "Multi-objective whale optimization algorithm for computation offloading optimization in mobile edge computing,", Sensors vol. 21 (8) (2021) 1–24, https://doi.org/10.3390/s21082628.
[81] G.K. Chang, L. Cheng, M. Xu, D. Guidotti, "Integrated fiber-wireless access architecture for mobile backhaul and fronthaul in 5G wireless data networks,", 2014 IEEE Avion. Fiber-Opt. Photonics Technol. Conf. AVFOP 2014 vol. 4 (2014) 49–50, https://doi.org/10.1109/AVFOP.2014.6999461.
[82] B. Abdollahzadeh, F. Soleimanian Gharehchopogh, S. Mirjalili, "Artificial gorilla troops optimizer: a new nature-inspired metaheuristic algorithm for global optimization problems,", Int. J. Intell. Syst. vol. 36 (10) (2021) 5887–5958, https://doi.org/10.1002/int.22535.
[83] A.A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, H. Chen, "Harris hawks optimization: algorithm and applications,", Futur. Gener. Comput. Syst. vol. 97 (2019) 849–872, https://doi.org/10.1016/j.future.2019.02.028.
[84] S. Mirjalili, S.M. Mirjalili, A. Lewis, "Grey wolf optimizer,", Adv. Eng. Softw. vol. 69 (Mar. 2014) 46–61, https://doi.org/10.1016/j.advengsoft.2013.12.007.
[85] 森, 小蔵, A. Cetinkaya, and 杉本, "Particle Swarm Optimizationを用いたマルコフ過程の補間," *第63回システム制御情報学会研究発表講演会*, 2019, [Online]. Available: 〈https://ci.nii.ac.jp/naid/40021910174/〉.
[86] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II,", IEEE Trans. Evol. Comput. vol. 6 (2) (2002) 182–197, https://doi.org/10.1109/4235.996017.
[87] Y. Huang, C. Tang, S. Wang, "Quantum-inspired swarm evolution algorithm,", Proc. - CIS Work. 2007, 2007 Int. Conf. Comput. Intell. Secur. Work. (2007) 208–211, https://doi.org/10.1109/cisw.2007.4425481.
[88] A. Semnani, M. Nabi Bidhendi, and B. Nadjar Araabi, "Detection of Low-frequency Shadow Zones using Quantum Swarm Evolutionary Matching Pursuit Decomposition (QSE-MPD)," Oct. 2013, p. cp-363-00037. doi: 10.3997/2214-4609.20131866.