J Grid Computing (2023) 21:21
https://doi.org/10.1007/s10723-023-09656-z

New Improved Multi-Objective Gorilla Troops Algorithm for Dependent Tasks Offloading problem in Multi-Access Edge Computing

Khalid M. Hosny · Ahmed I. Awad · Marwa M. Khashaba · Ehab R. Mohamed

Received: 18 January 2023 / Accepted: 1 March 2023 / Published online: 1 April 2023

Abstract Computational offloading allows lightweight battery-operated devices such as IoT gadgets and mobile equipment to send computation tasks to nearby edge servers to be completed, which is a challenging problem in the multi-access edge computing (MEC) environment. Numerous conflicting objectives exist in this problem; for example, the execution time, energy consumption, and computation cost should all be optimized simultaneously. Furthermore, offloading an application that consists of dependent tasks is another important issue that cannot be neglected while addressing this problem. Recent methods are single objective, computationally expensive, or ignore task dependency. As a result, we propose an improved Gorilla Troops Algorithm (IGTA) to offload dependent tasks in the MEC environments with three objectives: 1-Minimizing the execution latency of the application, 2-energy consumption of the light devices, 3-the used cost of the MEC resources. Furthermore, it is supposed that each MEC supports many charge levels to provide more flexibility to the system. Additionally, we have extended the operation of the standard Gorilla Troops Algorithm (GTO) by adopting a customized crossover operation to improve its search strategy. A Max-To-Min (MTM) load-balancing strategy was also implemented in IGTA to improve the offloading operation. Relative to GTO, IGTA has reduced latency by 33%, energy consumption by 93%, and cost usage by 34.5%. We compared IGTA with other Optimizers in this problem, and the results showed the superiority of IGTA.

Keywords Multi-access edge computing · Computational offloading · Multi-objective gorilla troops · Dynamic needs · Task dependency

K. M. Hosny (✉) · A. I. Awad · M. M. Khashaba · E. R. Mohamed
Department of Information Technology, Faculty of Computers and Informatics, Zagazig University, Zagazig 44519, Egypt
e-mail: k_hosny@yahoo.com

1 Introduction

Mobile Edge Computing or multi-access Edge computing (MEC) is an evolution of the centralized cloud server that aims at decentralizing the cloud computing services close to end users at the network edge or Radio Access Networks (RAN) of mobile systems [1, 2]. The tremendous advances in Information and Communication Technologies (ICT), such as Internet of Things (IoT) technologies and Mobile Devices (MD), have widened the horizons of programmers towards discovering new and innovative applications that can automate and enhance our life. Examples of such applications are virtual and augmented reality, face detection and recognition applications, surveillance systems, smart agriculture applications, smart healthcare applications, etc. [3, 4]. These applications call for high processing and storage requirements that cannot be fully provided by IoT and MD [5]. The first solution was to utilize cloud services, which suffered from bandwidth limitations and public network conditions. A more powerful solution was the MEC server due to its proximity to the end devices that get it away from the public network limitations [6, 7].

MEC can be used in the client–server computing paradigm, where a client requests a service from a nearby MEC server. This model reduces the burden on the public network by getting cloud resources near the users [8]. MEC also can be used in the three layers computing paradigm to support high computing and storage-intensive applications where the computing power can be divided into three locations, including MD, MEC, and the central cloud [9]. The computation offloading is a big challenge in the two mentioned computing paradigms. Computation offloading is the transmission of whole/some tasks of a target application to a powerful server for processing [10]. The modularity of the fine-grained programming principle facilitates the offloading process [11]. However, deciding which tasks to be offloaded to which server is a critical part of the MEC and cloud paradigm [12, 13]. Offloading some parts of an application while executing some parts locally with respecting the dependency relationship among tasks is still an open research point for which researchers try to find more efficient solutions. Since this is an NP-hard problem, many heuristics [14], metaheuristics [15], game-theoretic [16], mathematical, machine learning [17], and other algorithms were proposed to solve this problem. However, these methods are single objective, complex to implement, computationally expensive, or ignore the dependency among tasks.

Metaheuristic algorithms have recently been developed to solve many optimization problems and confirm near-optimal solutions [18, 19]. One of the metaheuristics is The Gorilla Troops Optimization algorithm (GTO) [20], inspired by the social intelligence of gorilla troops in the wild. In this paper, we formulate a multi-objective offloading problem of a set of dependent tasks composing an application to multiple nearby MECs. Despite the extensive research on this point, our proposed work is different. We seek to provide a new innovative solution to the dependent task offloading problem in the MEC environment. Our proposed work considers minimizing three main objectives, including energy consumption (E), MEC charge cost (C), and completion time latency (CT). We added the charge level factor to MEC server for providing more flexibility to the model. We depended on an improved version of the GTO algorithm, which is the first time to be used in this problem. Also, we extensively tested our algorithm with generated and standard topologies to ensure its efficiency. The comparison with other algorithms has shown the superiority of our IGTA method. The results showed that Relative to GTO, IGTA has reduced latency by 33%, energy consumption by 93%, and cost usage by 34.5%. The main contributions can be summarized as follows:

- We proposed an Improved Gorilla Troops Algorithm (IGTA) for real-time dependent tasks offloading in the MEC environment by optimizing three conflicting objectives simultaneously. In this regard, the Standard GTO algorithm is improved using a crossover operation, and the results have confirmed this improvement. Additionally, Each MEC is supposed to support multiple charge levels to add more flexibility to the system. We also adopted two vector values mapping methods to map the continuous values of the produced vectors into discrete and bounded values. A detailed discussion of the IGTA steps is presented in Section 4.
- A Max-To-Min (MTM) load balancing strategy was implemented in IGTA to improve the offloading operation. This important step is introduced in Section 4.5.
- Using a set of Standard and derived test cases with various topologies and task data, we have performed simulation experiments to test the performance of IGTA. The results demonstrate the superiority of IGTA, as in Section 6.

The following sections are organized as follows. Section 2 introduces a brief discussion of the related works. In Section 3, the system model and problem formulation are discussed. Section 4 discusses the proposed IGTA algorithm steps in detail. Section 5 provided an illustrative example to demonstrate the IGTA steps clearly. Experiments and results are discussed in Section 6. Finally, we conclude our work in Section 7.

2 Related work

MEC is the introduction of cloud services at the network edge. The introduced storage, computing, and networking services are used by small capabilities devices like smartphones and IoT devices for executing latency-sensitive tasks, computing hungry, and consuming large energy for execution. Unlike cloud servers, the tasks can be scheduled among multiple nearby MEC servers [21]. Efficient allocation tasks to the available computing resources can reduce task processing time and energy consumption and maintain server load balance [22]. In this regard, effective task offloading utilizes the available computing resources and high bandwidth to improve the quality of service by reducing the system latency and minimizing the energy consumption and charge cost. Therefore, several research attempts have been conducted in this regard.

The work in [23] uses two edge servers alternatively for offloading the Mobile's tasks to minimize both latency and power consumption. Storing a copy of the file after the first time processing is another strategy implemented in this research for better results. Some attempts have been made using game-theoretic techniques in this regard. Authors in [24] formulated a Stackelberg game for multi-user computational offloading problems in the MEC system to take an offloading decision that can minimize execution time and energy consumption goals. Another attempt in this regard was done [25], where the authors formulated a stochastic game for modeling the offloading problem under a dynamic environment. The latency and energy consumption were the two objectives that the authors of this research had considered.

Recently, heuristic and metaheuristic algorithms are gaining a lot of attention in solving optimization problems, so some attempts have been made in this regard. Authors in [1] suggested a heuristic algorithm for obtaining an efficient solution to minimize the application execution cost under a pre-allocated tasks' finish times' constraint. Authors in [12] and [26] depended on the whale optimization algorithm (WOA) to develop a task scheduling method to reduce completion time and energy consumption at the same time. In a homogeneous MEC case, the work of [27] formulated the dependent task offloading problem with pre-specified service caching, then introduced a favorite successor-based heuristic algorithm to reduce application completion time. Liu et al. [28] proposed a one-dimensional search algorithm to reduce task execution delay, considering the application buffer queuing status and the processor state. Authors of [17] proposed a genetic algorithm (GA) based scheme to reduce task offloading time and failure risk. In the work of [29], Security, energy consumption, and application completion time were all factors considered by Huang et al. In their research, a GA was used to reduce the energy consumption of MDs while meeting application deadlines. Authors in [30, 31] developed a particle swarm optimization (PSO) algorithm-based method to reduce the makespan and cost of the task scheduling problem. Another work by [32] suggested a load-balancing heuristic method to offload tasks and optimize task execution time.

Some attempts were done using supervised deep learning and Deep Reinforcement (RL) techniques and introduced innovative solutions to task offloading problems. First, the problem of dependent task offloading is also defined in the literature [33], and the goal is to reduce total execution costs while meeting application completion time constraints. Moreover, the Authors in [34–36] proposed a single objective RL-based offloading method to reduce the application completion time. The Works in [37–40] proposed two objective RL-based offloading methods to optimize completion time and energy consumption. Furthermore, the work in [21] proposed a multi-objective RL method to optimize energy consumption, completion time, and cost of MEC charge. Authors in [41] suggested a supervised deep learning approach for the single-user task offloading problem. This research used a mathematical model to generate the proposed model's learning dataset. They tend to provide an energy-efficient solution to the problem. The authors in [42] developed a Markov decision procedure to represent the joint user association and offloading decision of the MEC-based SAT-IoT networks. They suggested using deep RL to reduce energy usage and delay goals.

After reviewing the previous works in this area, we can say that our proposed solution has the flowing advantages over the developed ones:

- Optimize three objectives (CT, C, and E) at the same time.
- Consider the dependency among tasks.
- Use a new algorithm to solve this problem.
- Consider more than MEC and more than charge level for cost.

3 System model and problem formulation

In this section, we provide the system model for the computational offloading problem in the multi-access edge computing environment considering the dependency among tasks. Table 1 provides an overview of the key notations used in this paper.

Table 1 Summary of Used Notations

Notation	Description
M	Set MEC servers
V	Set of application Tasks
v_i	Task number i in the application
DAG	Directed acyclic graph to represent the task dependency
pre_set(v_i)	Set of successors of task v_i
post-set(v_i)	The set of predecessors of task v_i
E_{ij}	It is an edge in the DAG between task v_i and task v_j
CC_i	The number of CPU clocks needed to process the task v_i
Q_{in}	Input size
Q_{out}	Output size
P	Set of processing locations, including PoC and the M set
P_k	The kth processing location, where p_0 is the PoC
L	Set of charge levels or cost levels
L_r	The rth charge level
R	The number of supported charge levels
CT_i^{local}	Completion time of executing task v_i locally
FT (p_0)	Finishing time of p_0 task queue
ET_i^{local}	Execution time of task v_i on the local processor
MaxPred(v_i)	Max predecessor completion time of task v_i
WT_i^{local}	Waiting time for task v_i to be executed locally
F_{local}	Operating frequency of the local processor of the PoC
CT_j^{MEC}	Completion time of task v_j on one of the MEC servers
μ	Coefficient expressing the operating chip's capacitance
$CT_i^{k,r}$	Completion time for processing task v_i remotely on MEC server P_k under charge level l_r
$ET_i^{k,r}$	Execution time of task v_i on p_k, where $p_k != p_0$ under level l_r
WT_i^k	Waiting time for task v_i to be executed on p_k
F_l^k	Operating frequency of p_k under the charge level l_r
TT_i^k	The transmission time of Q_{in} of task v_i to MEC server p_k
RT_i^k	Receiving time of Q_{out} of task v_i from MEC server p_k
H_K	Achievable uplink and downlink transmission rate between the PoC and the MEC server p_k
B	Channel bandwidth
WP	Wireless transmission and receiving the power of the PoC
A_k	Gain of the channel between p_0 and p_k
σ^2	Channel noise
E_ik	Energy consumption for offloading task v_i to p_k, where $p_k != p_0$
E_ik^{trans}	Consumed energy consumed for transmitting Q_{in} of task v_i to p_k
E_ik^{rec}	Consumed energy consumed for receiving Q_{out} of task v_i from p_k
WTP	The wireless transmission power of the PoC
WRP	Wireless receiving the power of the PoC
C_i^r	Charge cost for processing task v_i remotely at level l_r
$\bigtriangledown$	Coefficient expressing the cost per unit of time
a_1, a_2, and a_3	Weighting parameters are used to identify the relative importance of each objective of the three
CT(v_i)	Total completion time of task v_i
E(v_i)	Total energy consumption of task v_i
C(v_i)	Total used cost for completing task v_i
Z	Optimization function

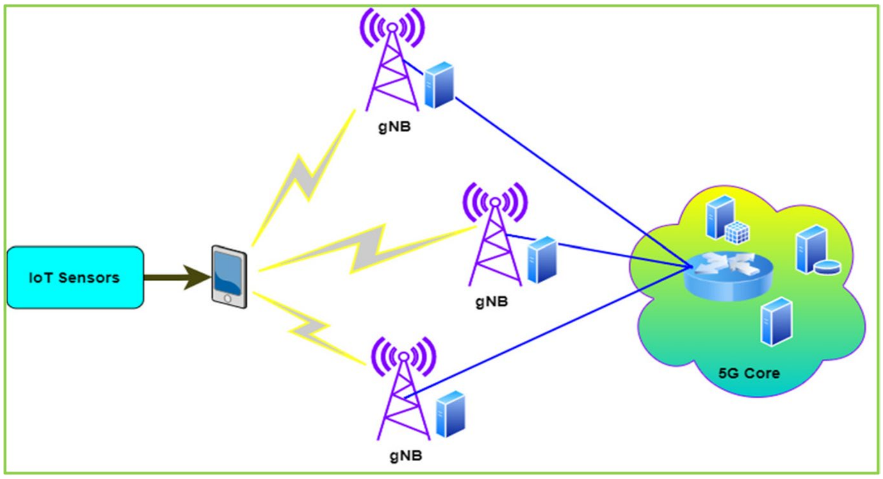


Fig. 1 MEC system architecture

3.1 System Model

The architecture of the MEC system is shown in Fig. 1. The IoT sensors sense the data and transmit it to the point of Collection (PoC) device, which can be a Personnel Digital assistant (PDA) or smartphone. This PoC runs the application that analyzes the sensed data. Such applications are computationally intensive and contain complex tasks that require powerful resources. To meet such resources' requirements, the PoC device can perform partial offloading for some of its tasks to a nearby MEC server installed in the base station (gNB) at the network edge, as in the 5G network architecture [43]. We consider a system that consists of M MEC Nodes and only one PDA/ smartphone that runs an application composing a set of V tasks. We denote the set of dependent tasks composing the application as V = {v1, v2, v3, …, vN}, where N represents the number of tasks in the application. The analysis result is then transferred to the central cloud through the backhaul network of the 5G network core for further analysis and storage purposes.

To clarify the dependency relationship among the tasks, we modeled them as a Directed Acyclic graph (DAG), as shown in Fig. 2. Each vertex in the graph vi represents a task in the model, and i represent the task ID, where each edge E_{ij} represents a dependency relationship between task v_i and v_j, such that task v_j cannot be executed before task v_i finishes its execution. We denote the pre_set(v_i) and post-set(v_i), which are the set of predecessor and successor tasks of $v_{i,}$ respectively. Predecessors express the set of tasks that must be executed before v_i starts. In contrast, successor means the set of tasks that cannot start its execution before completing the execution of v_i. Figure 2 depicts an application composing 11 tasks. Task v_0 is the start task because the number of elements in its pre_set is zero, and task v_{10} is the end task as the number of elements in its post_set is zero.

We maintain three parameters for each task in the system. The number of CPU clocks needed to process the task CC_i in a million instructions per second (MIPS), the input size Q_{in} in megabytes, and the output size Q_{out} in megabytes. Each task can be executed locally with the local device's full processing capacity or offloaded to one of the neighboring MECs at a specific charge level. Each MEC device support set of charge level with different processing capabilities and usage cost. Each MEC server can easily support this strategy by allocating its tasks to its cores under different Dynamic Voltage and Frequency Scaling (DVFS)

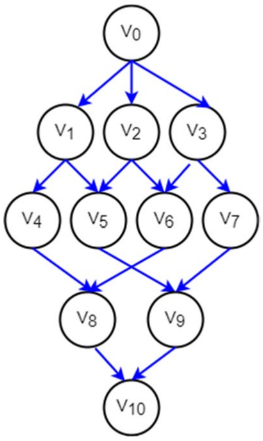


Fig. 2 Example of tasks DAG

levels [44]. We denote the set of processing locations is $P = \{p_0, p_1, p_{2\ldots} p_M\}$, such that $0 \le p_k \le M$, where M is the number of MEC servers in the system and P_0 is the PoC. The set of charge levels is denoted as $L = \{L_1, L_2 \ldots L_R\}$, such that $0 \le L_r \le R$, where R is the number of supported charge levels.

3.2 Problem Formulation

As mentioned earlier, MEC offers Cloud services at the network edge, a big technology revolution for the IoT and mobile applications. For this reason, we try to help maximize the benefit of this new technology paradigm. As a result, we seek in this research to build an optimization framework that can be used by mobile and IoT application developers to develop better MEC-based applications in terms of execution time, energy consumption, and usage cost.

This subsection will formulate the task offloading optimization problem considering M MECs and a PoC device. We seek to develop a framework that IoT and Mobile application developers can use in designing their applications and programs. As mentioned in the previous sections, there exist two offloading decisions in this problem:

- Executes the task locally using its resources.
- Offloads the task to one of the nearby MEC servers.

The extra intensive-computing tasks can be offloaded to cloud computing, but this is not included in our model, as it is a special case.

3.2.1 Task Local Processing

In the case of local processing, task v_i is not transmitted to any MEC server, so the task completion time can be defined as follows:

$$CT_i^{local} = FT(\mathrm{P}_0) + ET_i^{local} + WT_i^{local}. \tag{1}$$

FT (P_0) is the finishing time of the P_0 queue before the addition of v_i. It is also the start time of v_i. ET_i^{local} is the task execution time on the local processor. It is calculated as CC_i/F_{local}. The CC_i is the required CPU Clock cycle for executing task v_i, and F_{local} is the operating frequency of the PoC device. The WT_i is the waiting time for task v_i to be executed. WT_i^{local} is defined as follows:

$$WT_i^{local} = \mathrm{MaxPred}(\mathrm{v_i}) - \mathrm{FT}(\mathrm{P_0}). \tag{2}$$

$\mathrm{MaxPred}(v_i)$ is the max predecessor completion time of task v_i. it can be defined as:

$$\mathrm{MaxPred}(v_i) = \mathrm{Max}\{\mathrm{Max}(CT_j^{local}, CT_j^{MEC})\forall v_j \in \mathrm{pre\_set}(v_i)\}. \tag{3}$$

Where CT_j^{MEC} is the completion time of executing task v_j on one of the MEC servers. Each task is executed in one location. If the task is executed locally, then $CT_j^{MEC} = 0$. Conversely, if the task is offloaded to one of the MEC servers, then $CT_j^{local} = 0$. The value of WT_i^{local} is neglected if it is a negative value or zero.

According to [18], energy consumption is defined as follows:

$$E_i^{local} = \mu * CC_i * (F_{local})^2. \tag{4}$$

Where μ is a coefficient expressing the operating chip's capacitance, the cost of processing task v_i on the local processor naturally equals zero.

3.2.2 Task offloading to Nearby MEC

In case of offloading task v_i to a MEC server p_k, where $1 < k \le M$, the PoC will transmit the input Q_{in} including program code, to the MEC server, then receive the output Q_{out} from the server using the uplink and downlink wireless channels, respectively. We denote $CT_i^{k,r}$ as the completion time for processing task v_i remotely on MEC server P_k under charge level L_r, where $1 < r \le R$. it can be calculated as:

$$CT_i^{k,r} = ET_i^{k,r} + WT_i^k + TT_i^k + RT_i^k. \tag{5}$$

Where $ET_i^{k,l}$ is calculated as:

$$ET_i^{k,r} = CC_i / F_r^k \tag{6}$$

Where F_l^k WTik is waiting for task vi to be executed on PK; it can be calculated in Eqs. (2) and (3), but for the Pk. Is the operating frequency of MEC server pk under the charge level Lr. TT_i^k and RT_i^k express the transmission time for the input data of task v_i to MEC p_k and the receiving time for the output data of task v_i from MEC p_k, respectively. They can be defined as follow:

$$\mathrm{TT}_i^k = \mathrm{Q}_{in}/\mathrm{H}_K., \mathrm{RT}_i^k = \mathrm{Q}_{out}/\mathrm{H}_{K.} \quad (7)$$

Where H_K is the achievable uplink and downlink transmission rate between the PoC and the MEC server p_k, we assumed the symmetry property for the two wireless channels. According to Shannon–Hartley theory [45], the H_K is defined as:

$$H_k = B * LOG_2(1 + \frac{WP * A_k}{\sigma^2}) \quad (8)$$

Where B is the channel bandwidth, WP is the wireless transmission and receiving the power of the PoC, A_k is the channel's gain, and σ^2 is the channel noise. The propagation delay can be discarded as the distance is very small relative to the light speed, which generates a tiny number [21].

The energy consumption for offloading task v_i to MEC server p_k can be calculated as follows:

$$\mathrm{E}_{ik} = \mathrm{E\_ik}^{trans} + \mathrm{E\_ik}^{rec}. \quad (9)$$

$\mathrm{E\_ik}^{trans}$ and $\mathrm{E\_ik}^{rec}$ express the energy consumed for transmitting input data and receiving the output data of task v_i to/from the MEC $\mathrm{p}_{k,}$ respectively, and can be defined as following [46]:

$$\mathrm{E\_ik}^{trans} = \mathrm{WTP} * \mathrm{TT}_i^k, \ \mathrm{E\_ik}^{rec} = \mathrm{WRP} * \mathrm{RT}_i^k. \quad (10)$$

The WTP is the wireless transmission power, and WRP is the wireless receiving power. Offloading task v_i to a MEC server P_k at charge level L_r imposes a cost C_i^r on the PoC for using the MEC computing resources for some time [47]. This usage cost can be obtained by:

$$\mathrm{C}_i^r = \nabla * \mathrm{ET}_i^{k,r}. \quad (11)$$

The symbol $\bigtriangledown$ refers to a coefficient expressing the cost per unit of time.

3.2.3 Objective Function

Task offloading problem in the MEC environment is a multi-objective problem that minimizes the completion time or makespan, energy consumption, and cost usage of remote MEC servers.

The total completion time of task v_i CT (v_i), the energy consumption E (v_i), and the total used cost C (v_i) for completing this task can be obtained using the following equations:

$$CT(v_i) = CT_i^{local} + CT_i^{k,r} \quad (12)$$

$$E(v_i) = E_i^{local} + E\_ik \quad (13)$$

$$C(v_i) = +C_i^r \quad (14)$$

The total completion time CT, energy consumption E, and cost usage C for all tasks, including in the DAG of tasks, can be defined as follow:

$$CT = Max\{CT(v_i)\forall v_i \ in \ task \ DAG\} \quad (15)$$

$$E = \sum_{i=1}^{v} \mathrm{E}(v_i) \quad (16)$$

$$C = \sum_{i=1}^{v} \mathrm{C}(v_i) \quad (17)$$

The objective function is defined as follows:

$$\text{Minimize Z } = a_1 * \mathrm{CT} + a_2 * \mathrm{E} + a_3 * \mathrm{C}. \quad (18)$$

Where a_1, a_2, and a_3 are weighting parameters in the mathematical period [0, 1], they must sum to 1. They are used to identify the relative importance of each objective in the problem.

4 IGTA Algorithm for Solving the Proposed Model

The proposed algorithm IGTA deal with optimizing the problem of offloading a single application composed of multiple dependent tasks using multiple neighboring MEC servers under different charge levels. The IGTA algorithm's overall flow is depicted in Fig. 3. For simplicity, we called the operations the vector values mapping, evaluated the fitness function, updated the current solution, updated the silverback position (Xsb) with the name, and updated the IGTA parameters. In this section, IGTA will be described in detail using several subsections. Firstly, the Gorilla vector initialization process is introduced. Then, the modeling of GTO Operations is discussed. After that, we introduce the mapping operation of the resultant continuous vector to a discrete one. Finally,

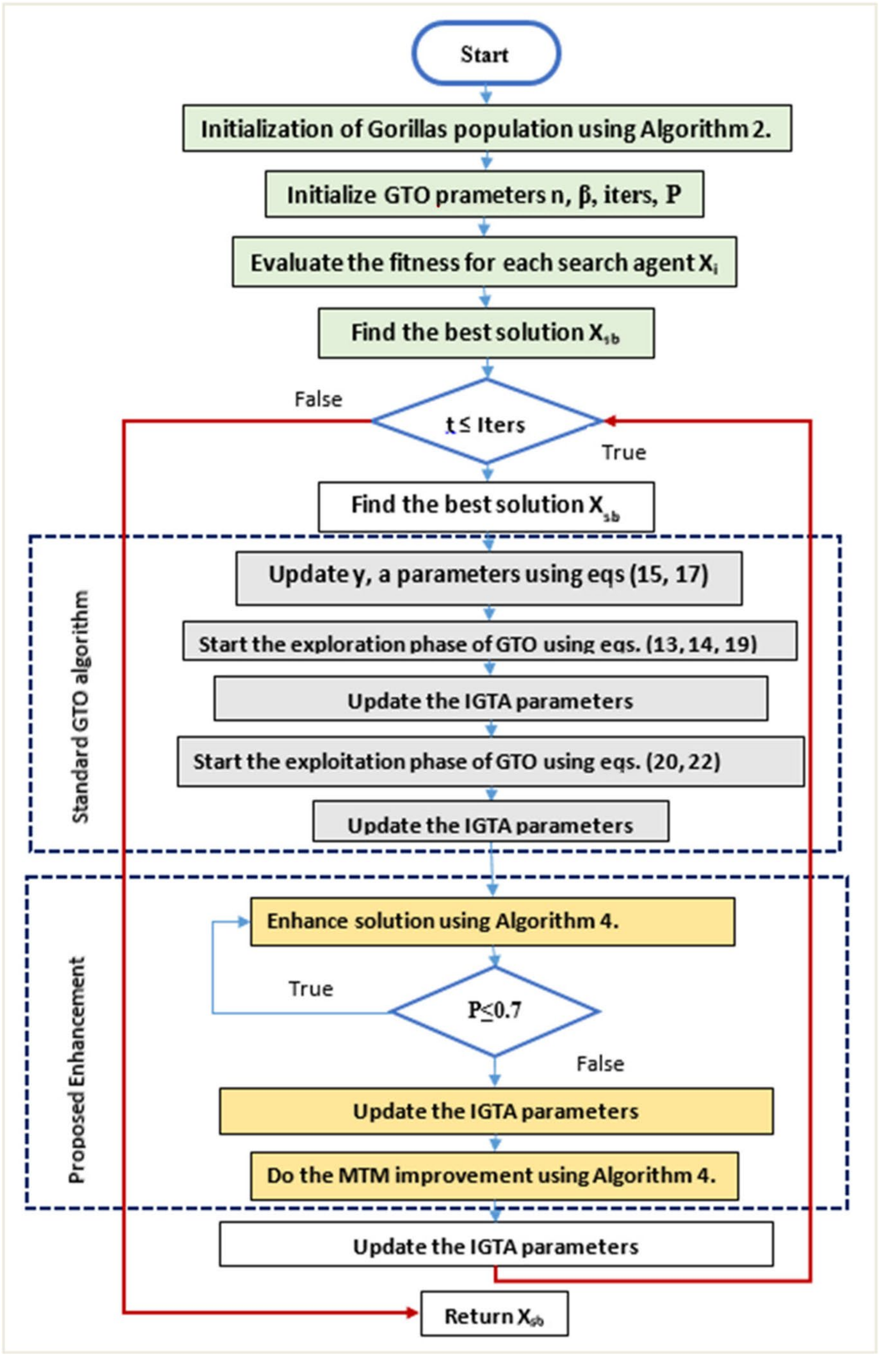

Start
Initialization of Gorillas population using Algorithm 2.
Initialize GTO prameters n, β, iters, P
Evaluate the fitness for each search agent X_i
Find the best solution X_{sb}
False
t ≤ Iters
True
Find the best solution X_{sb}
Standard GTO algorithm
Update γ, a parameters using eqs (15, 17)
Start the exploration phase of GTO using eqs. (13, 14, 19)
Update the IGTA parameters
Start the exploitation phase of GTO using eqs. (20, 22)
Update the IGTA parameters
Proposed Enhancement
Enhance solution using Algorithm 4.
True
P≤0.7
False
Update the IGTA parameters
Do the MTM improvement using Algorithm 4.
Update the IGTA parameters
Return X_{sb}

Fig. 3 Flow chart of IGTA algorithm

we end our discussion with two subsections introducing the two improvement operations added to the standard GTO steps. These two enhancement operations are the crossover operation and the Maximum-to-minimum (MTM) load balancing method.

4.1 Gorilla Vector Initialization

As we explained in the system model, each task can be executed locally with the full processing capacity of the local device or can be offloaded to one of the neighboring MECs at a specific charge level. We maintain a single vector to represent the offloading of tasks to different processing locations under different charge levels that will be optimized using our algorithm. The vector is divided into three sets as follows:

- The first V items of the vector represent the allocated tasks.
- The next V items represent the processing location of the tasks, respectively.
- The last V items represent the charge level used in processing the task on a MEC server. The charge level is the node processing cost level for task processing.

We use algorithm 1 to generate a randomized task list vector. In the proposed algorithm, this vector is represented by the gorilla that needs to be optimized, where each gorilla vector represents a possible solution to the problem. So In this vector, the task at index i will be executed at the processing location at index V + i under a charge level at index V + 2i. Algorithm 1 is a recursive function that starts with the start task; the task with no predecessors then gets its post_set elements and shuffles them. The algorithm recursively does this operation for each task. This algorithm guarantees the random generation of task lists considering task interdependency relationships. We used Algorithm 2 to initialize the three parts of the gorilla. It initializes the task list by calling algorithm 1. The other two parts, the MEC list and the charge levels list, are initialized randomly. Figure 4 depicts an example of a gorilla vector (X) consisting of 11 tasks, 4 processing locations, including the PoC, and 7 charge levels. In the figure, task 0 will be offloaded to MEC 1 under charge level 1. This example is provided for a better understanding of the structure of each gorilla vector.

X = (0, 1, 3, 2, 4, 5, 6, 7, 8, 9, 10, 0, 0, 1, 2, 3, 1, 3, 3, 1, 2, 2, 0, 0, 3, 4, 5, 5, 4, 2, 3, 1, 4).

4.2 Modeling the GTO Operations

After initializing the population of the gorilla positions, the IGTA algorithm will start its operations with the GTO algorithm. The GTO algorithm simulated the Gorillas' troops' behaviors [20]. The Gorillas' group behavior was mathematically modeled using five different operations. Three operations included in the exploration phase are migration to unknown places, to known places, and other gorillas. Furthermore, the two other operations are included

Algorithm 1. Task list generation algorithm

Input: Generated list GeneratedList, Next task index $_{index}$, Next gorilla location index G_{index}, Number of tasks V, Ordered task list OrderList.
Output: GeneratedList.

```
        //base case
1:      If ($v_{index} \geq V$ )
2:          Return
3:      End if
4:      If ($v_{index} == 0$)
5:      GeneratedList[$v_{index}$]= OrderList[$v_{index}$].
6:      End If
7:      Post_set = successors of task with Index $v_{index}$
8:      Shuffle the Post_set
9:      For i = 0 : Post_set.length
10:             If (Post_set ($v_i$) is Not in the GeneratedList and all its
                predecessors have been added to GeneratedList)
11:             GeneratedList[++$G_{index}$]= Post_set [$v_i$]
12:         End if
13:     End For
14:     Recursively call this method, increasing $v_{index}$ by one in each call.
15:     End for
```

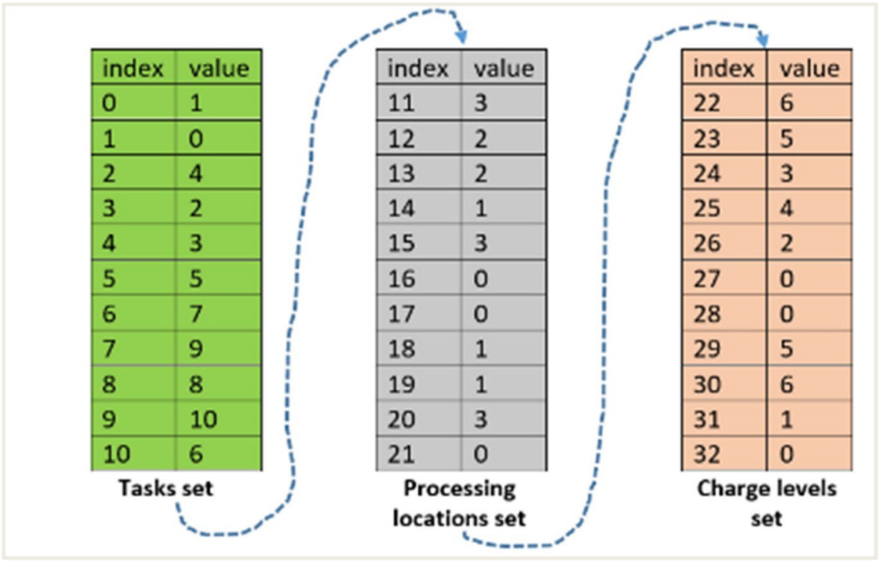
Algorithm 2. Initialization algorithm

```
Input: population Size pop_size, task number V, processing locations
Size M, charge levels number R.
Output: initialized population pop.
1:  For i = 1: pop_size
2       TaskList =generate task list using algorithm 1.
3:      For j = 1:3*V
4:          If (j < V)
5:              $P_i(j)$ = tasklist(j)
6:          Else if (j >= V and j < 2 * V)
7:              $P_i(j)$ = generate a random processing location id in
                {0,1,2,3,…, M}
8:          Else
9:              If ($P_i$(j-N) = 0 )
10:                 $P_i(j)$ = 0
11:             Else
12:                 $P_i(j)$ = generate a random charge level in
                    {1,2,3,…, R}
13:             End if
14:         End if
15:     End for
16: End for
```

in the exploitation phase. These two operations are to obey the silverback and compete with adult males. The silverback is the leader of the gorilla group. The GTO algorithm supposes that the best solution is the silverback position. The mathematical formulation of the three operators included in the exploration phase depends on a seed variable S and a randomly generated value P. thus, when P < S, the migration to an unknown place is used, and the candidate solution GX in the next i iteration can be calculated as follow:

$$GX(i+1) = (U - L) * r_1 + L \tag{19}$$

Where L and U are the lower and upper bounds for the variables, respectively, r_1 is a random variable in the range [0: 1]. r_1 is continuously updated in each iteration. The movement to other gorillas' operator is



index	value
0	1
1	0
2	4
3	2
4	3
5	5
6	7
7	9
8	8
9	10
10	6

Tasks set

index	value
11	3
12	2
13	2
14	1
15	3
16	0
17	0
18	1
19	1
20	3
21	0

Processing locations set

index	value
22	6
23	5
24	3
25	4
26	2
27	0
28	0
29	5
30	6
31	1
32	0

Charge levels set

Fig. 4 Gorilla vector example

selected when $P \geq 0.5$, and the next candidate solution is obtained by:

$$GX(i+1) = (r_2 - y) + X_r(i) + a * h \tag{20}$$

X_r is a randomly obtained position vector for an individual from the gorillas' population. y, a, and h are variables that can be defined according to the following equations:

$$y = V * (1 - \frac{i}{Mi}) \tag{21}$$

$$V = \cos(2 * r_4) + 1 \tag{22}$$

$$a = y * r_4 | r_4 \text{ is a random value in } [-1 : 1] \tag{23}$$

$$h = Z * X(i) | Z = [-y, y]. \tag{24}$$

X(i) is the current position vector. In the last case, if $P < 0.5$, the migration to known locations operator is selected, which is formulated as follows:

$$GX\,(i+1) = X(t) - a * \left(a * \left(X(i) - GX_r(i)\right) + r_3 * \left(X(i) - GX_r(i)\right)\right) \tag{25}$$

The selection from the two operations of the exploitation phase is based on the value of the y variable obtained using Eq. (21). When y is greater than or equal to a pre-settled variable w, obeying the silverback is mathematically selected and modeled.

$$GX(i+1) = a * b * (X(i) - X_{sb}) + X(i) \tag{26}$$

X_{sb} is the silverback or best solution position vector, and b is calculated using the following equation.

$$b = \left[\left(\frac{1}{N}\sum_{j=1}^{N}\left(GX_t(i)\right)\right)^{g}\right]^{\frac{1}{g}} | g = 2^{a}. \tag{27}$$

$GX_t(i)$ is a candidate position vector in iteration i for each gorilla member t. N represents the total number of gorillas. On the other side, when y is less than W, the competition for adults' behavior is selected by the GTO algorithm. This operator is modeled as follows:

$$GX_t(i) = X_{sb} - \left(X_{sb} * f - X(i) * f\right) * A \tag{28}$$

$$f = 2 * r_5 - 1 | r_5 \textit{ is a random number in } [0 : 1]. \tag{29}$$

$$A = \beta * £ \tag{30}$$

$$£ = \begin{cases} r_6 \text{ if } r_0 \geq 0.5 \\ r_7 \text{ if } r_0 < 0.5 \end{cases} \tag{31}$$

Where f is the impact force, β is a variable that should be given a value before starting the optimization operation. £ is used to determine the violent effect. £ value is determined based on a random number r_0 with two expected values in the normal distribution and the dimensions of the problem $if r_0 \geq 0.5$. We have added two operations to extend the functionality of the standard GTO algorithm. The customized crossover operation and MTM load balancing operation are the two added operations. The results showed that our modification had improved the quality of the GTO algorithm to provide better results.

4.3 Vector Mapping Operation

The generated Gorilla vector (after performing the mathematical formulas of each operation of IGTA) produces a vector of continuous values that need to be mapped to discrete bounded ones according to the value position in the gorilla vector. In this regard, we depended on two mapping methods to discretize the values of the produced gorilla position vector.

The first mapping method is used to map the values of the task list part. To respect the dependency constraint among tasks, firstly, we start with the first task from the task list and get its successors. For example, we take task v_0 and get its successors v_1, v_2, and v_3.

Suppose that the values of generated task list of GX_i are as in Table 2. We then put the start task v_0 in index 0 of the gorilla vector GX_i. We then get the values of the task successors from GX_i.

Table 2 Successors of task v_0 in the Generated Task List

0.0	0.94	3.1	2.12	5.36	4.12	6.68	6.34	9.39	8.46	9.84

Table 3 Indices of task 0 successors

vector Value	3.1	2.12	0.94
value index	2	3	1
Successors list	1	2	3

The successors of task 0 should be in indices 1, 2, and 3 with values 0.94, 3.1, and 2.12. We sort these values as in Table 3.

Then we replace each vector value in the GX_i vector with its corresponding successor's list value. So the vector will be as in Table 4 after this step.

If two or more tasks have the same successor task, this successor task will be added only once to satisfy the uniqueness constraint for executing each task only once. This operation will be repeated for each task in the task list.

The second mapping method is used to map the values of the MEC list and the charge level list. This method is simply a direct normalization of the vector values according to the following equation:

$$GX_i = \frac{GX_i - minval}{maxval - minval} \tag{32}$$

Where maxval and minval are the maximum and minimum values of the vector part (MEC list or level list) to be mapped, respectively, this equation will be multiplied by the number of MEC set in case of MEC vector value mapping and by the number of charge level in case of the level vector value mapping.

4.4 Crossover Improvement Operation

We suggested a special crossover operation to enhance the generated solution's quality and improve the performance of the GTO algorithm.

We perform the crossover operation on each generated position X_i^t and the silverback position X_{sb}. For the task list part, we then choose a random number from the task list and swap its successors between the two input vectors. Assume the chosen random index is 0 with task number 0. We then find its successors in the two Children and swap them. To swap them, we get the position of the successor list in each child. After that, we swap the positions of tasks between the two Children, as shown in Fig. 5. The Crossover operation algorithm for the task list is provided in algorithm 3.

Furthermore, for the other two parts of the gorilla vector, we choose a random index for the second and third parts and swap its value between the two Children.

4.5 MTM Load Balancing Operation

We proposed the MTM method to redistribute the load of the task among the processing locations. The MTM load balancing operation continuously updates tasks' processing locations from maximum completion time to minimum one until the completion time of the minimum location reaches or exceeds the best-obtained completion time or the maximum completion time. In the MTM load balancing method, we search for the two processing locations with maximum and minimum completion times. The pseudo-code for the MTM load balancing operation is provided in algorithm 4. The pseudo-code of the IGTA algorithm is shown in algorithm 5.

5 Illustrative Example

In this section, we provide a motivating example to clarify the offloading strategy of this research. Depending on the DAG of Fig. 2, that shows an application consisting of eleven dependent tasks $V = \{v_0, v_1 \ldots v_{10}\}$. We randomly generated the values of the three parameters maintained for each task. Table 5 shows these generated values. We suppose there are three MEC servers and a PoC in the system $p = \{P_0, P_1, P_2, P_3\}$, P_0 expresses the PoC processor. By assuming the symmetry property for the two wireless channels, the achievable uplink and downlink rates are provided in Table 6. We suppose that the transmission and receiving powers of the PoC equal 0.1 Watts. Each MEC device supports a set of charge levels with different processing capabilities. Each MEC server can support up to six charge levels $L = \{L_1, L_2 \ldots L_6\}$ in addition to L_0, which expresses the task's local processing. Each charge level has a different operating frequency

Table 4 Task list after mapping the successors of the task

0.0	3	1	2	5.36	4.12	6.68	6.34	9.39	8.46	9.84

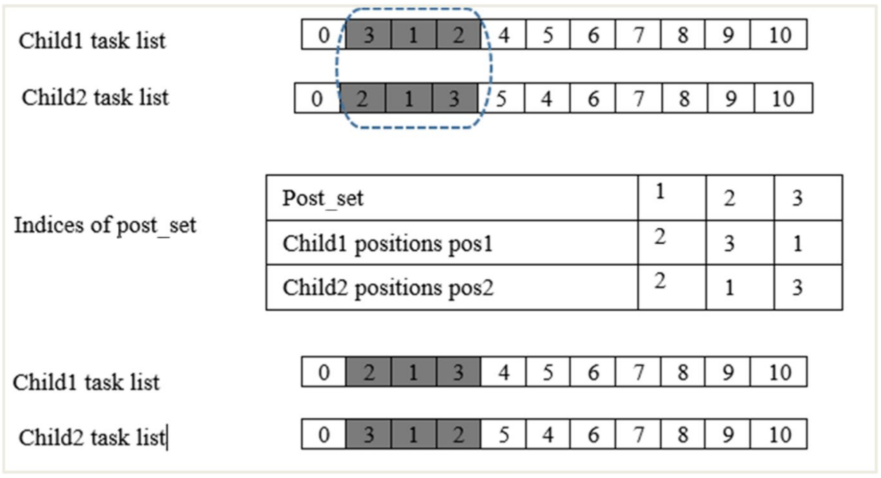


Fig. 5 Visual illustration of the crossover operation

and unit price, as shown in Table 7. The dependency among tasks is provided in Table 8. Assuming that the mobile dynamic energy is 0.125. For simplicity, we suppose sequential execution of tasks from each mobile device on MEC. As in the case of parallel execution, it is a confusing issue to distribute the operating frequency among tasks. This distribution will directly impact the task completion time, so we suppose sequential execution to avoid this. Assume we have a candidate solution:

$$X = (0, 1, 3, 2, 4, 5, 6, 7, 8, 9, 10, 0, 0, 1, 2, 3, 1, 3, 3, 1, 2, 2, 0, 0, 3, 4, 5, 5, 4, 2, 3, 1, 4).$$

Where X_i (i=0... 10) is the task id, X_i (i=11... 21) is the execution location of V_i, and X_i (i=22... 32) is the charge level.

Firstly, we calculate the execution time matrix for executing each task under each charge level, equal to the CC_i/F_{op}, where F_{op} is the operating frequency. The time execution matrix for this example is shown in Table 9.

For simplicity, we will use the equations presented in the problem formulation section to calculate the completion time of this example task. Since v_0 and v_1 are executed locally, their completion time can be obtained as follow.

$WT_0^{local}=0$. Since it is the start task.

$$CT(v_0) = CT_0^{local} = FT(P_0) + ET_0^{local} + WT_0^{local} \\ = 0 + 3.33 + 0 = 3.33.$$

$$WT_1^{local} = CT_0^{local} - FT(P_0) - 3.33 - 3.33.$$

$$CT(v_1) = CT_1^{local} = FT(P_0) + ET_1^{local} + WT_1^{local} \\ = 3.33 + 5 + 0 = 8.33.$$

We need to compute each task's waiting time, transmission time, and receiving time for remote-processing tasks. We set the H value by 1 GBps for all channels to calculate the transmission and receiving times. So the completion time of v_3 can be calculated as follow:

$$WT_3^1 = CT_0^{local} - FT(P_1) = 3.33 - 0 = 3.33.$$

$$TT_3^1 = 4.2M/1G = 0.0042.$$

$$RT_3^1 = 2.5M/1G = 0.0025.$$

$$CT(v_3) = CT_3^{1,3} = FT(P_1) + ET_3^{1,3} + WT_3^1 + tTT_3^1 + RT_3^1 \\ = 0 + 1.333 + 3.33 + 0.0042 + 0.0025 = 4.67.$$

The completion times for the other tasks were computed similarly and provided in Table 10.

Figure 6 shows the complete task offloading chart for this example. The completion time of the application is

Algorithm 3. Crossover operation algorithm for the task list

```
Input: child1, child2, pos1, pos2.
Output: improved solution $X_{sb}$
1:  posLen = get the length of the pos1 list
2:  For i=1: posLen
3:      If (pos1[i] != pos2[i])
4:          swapedTask = child1[pos1[i]]
5:          child1[pos1[i]] = child1[pos2[i]]
6:          child1[pos2[i]] = swapedTask
7:          swapedTask = child2[pos1[i]]
8:          child2[pos1[i]] = child2[pos2[i]]
9:          child2[pos2[i]] = swapedTask
10:     End If
11: End For
```

also called the makespan. It is the maximum completion time among all completion times of the processing locations. In this regard, the application completion time is 21.22 s, the energy consumption is 28.15 J, and the charging cost is 4.79 $. The fitness function value:

$$Z = 0.34 * 21.22 + 0.33 * 28.15 + 0.33 * 4.79 = 18.1$$

Algorithm 5 shows the pseudo-code for the proposed IGTA algorithm. We first initialize a gorilla vector. After that, the initialized vector is passed as input to the GTO algorithm, producing a vector of continuous values. We then discretize this vector's continuous values and improve this vector using the crossover and MTM load balancing operations.

6 Experiments and Results

Several experiments and empirical tests are performed in this section to evaluate the effectiveness of the IGTA algorithm. All tests are conducted on a laptop with the following specifications. The processor specification is Intel (R) Core (TM) i7-3540 M @ 3.00 GHz, and the RAM size was 8 GB. Windows 10 Professional 64-bit was the installed operating system. All the simulated experiments were done using the java programming language.

6.1 Data Description

For the experiment, we will suppose that M = 3 MEC servers will be available for the PoC. We suppose that the transmission and receiving powers are 0.1 watts and the mobile dynamic energy is 0.125 [21, 25, 48]. By assuming the symmetry property for the two wireless channels, the achievable uplink and downlink rates are previously shown in Table 6. Additionally, we use six distinct charge levels previously noted in Table 7. The operating frequency and unit price values rise from the first level to level six.

Algorithm 4. MTM load balancing improvement algorithm

```
Input: silverback position $X_{sb}$, tasks number V, processing locations
Size M, best completion time BCT.
Output: improved solution $X_{sb}$.
1:  Calculate the completion time for each processing location $CT_i$
2:  $W_{min}$, $W_{max}$ = processing location with min and max completion
    time, respectively
3:  While ($CT_{Wmin} < CT_{Wmax}$ && $CT_{Wmin} < BCT$)
4:      Replace $W_{max}$ with $w_{min}$ in $X_{sb}$
5:      Update the completion times for the modified processing
        locations.
6:  End while
```

Table 5 Tasks data

Tasks	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
CC MIPS	10	15	18	20	25	20	30	28	23	30	29
Q_{in}	5	10	15	4.2	20	18	50	27	20	15	26
Q_{out}	0.5	1.5	2	2.5	2	2.3	3	1	1.3	1.5	2

We adopt level zero to denote the full capacity of the PoC device.

The datasets used in our experiments will be described here. The used datasets are open and accessible at [49]. The task graphs in the datasets are divided into three sets, each with 100 graphs. The first set of graphs all has the same topology, N = 9 tasks. Three must be executed locally, and the other six can be offloaded. The second set of graphs has the same topology, N = 29 tasks, and 20 off-loadable tasks. The third set of graphs all shares the same topology, N = 23 tasks, and 19 off-loadable tasks. The number of off-loadable components per task graph is also adjustable by users. In this experiment, we assume that all tasks in each graph can be offloaded.

We selected three task graphs from each graph set from the dataset mentioned above for our experiment. Moreover, for testing purposes, we randomly generated the input data Qin in the range [5, 50] megabytes and the output data Qout in the range [0.5, 5] megabytes. Table 11 contains the full description of the experiment's task graphs (TG).

6.2 Experiment Parameters

We compare our suggested IGTA algorithm with the following algorithms:

- Standard Gorilla troops algorithm (GTO) with no refinement.
- Harris hawks optimizer (HHO) [50].
- Whale Optimization Algorithm (WOA) [51].
- Grey wolf Optimization (GWO) algorithm [52].
- Bat Algorithm (BAT) [53].
- Particle Swarm Optimization (PSO) algorithm [54].
- Genetic algorithm (GA) [55].

Table 6 Achievable uplink and downlink transmission rates

MEC servers	M1	M2	M3
Achievable uplink and downlink rates	1G	1G	1G

Since the number of tasks in the described datasets is relatively small, we set the number of iterations to 200 to allow for a fair assessment of the algorithms. Each algorithm's population size is set to 10. Each algorithm is evaluated by the results of executing it 20 times. We set the parameters of the GTO as follows:

- P parameter to control the selection of the exploration operations = 0.03;
- W parameter to control the selection of the exploitation operations = 0.8;
- The beta that is used in calculating the coefficient vector to determine the degree of violence in conflicts = 3;

The crossover probability parameter P for IGTA is set to 0.7. However, the results of some experiments on a few random datasets show that 0.7 is the most suitable probability. This procedure maintains population diversity while improving the quality of the new solutions produced by IGTA.

The parameters of the comparison algorithms are set as the authors recommend. For GA, we have set the mutation rate to 0.001. Beginning from the start task, we have implemented a crossover operation between the successors of each task in the DAG. We select the random number in the task list, mutate its

Table 7 Operating frequency and unit price for each charge level

Charge level	Operating frequency MIPS	Cost per second
L_0 for mobile	3	0
L_1	5	0.01
L_2	10	0.02
L_3	15	0.03
L_4	20	0.04
L_5	25	0.05
L_6	30	0.06

Table 8. Dependency matrix

V x V	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
v_0	0	1	1	1	0	0	0	0	0	0	0
v_1	0	0	0	0	1	1	0	0	0	0	0
v_2	0	0	0	0	0	1	1	0	0	0	0
v_3	0	0	0	0	0	0	1	1	0	0	0
v_4	0	0	0	0	0	0	0	0	1	0	0
v_5	0	0	0	0	0	0	0	0	0	1	0
v_6	0	0	0	0	0	0	0	0	1	0	0
v_7	0	0	0	0	0	0	0	0	0	1	0
v_8	0	0	0	0	0	0	0	0	0	0	1
v_9	0	0	0	0	0	0	0	0	0	0	1
v_{10}	0	0	0	0	0	0	0	0	0	0	0

Table 9 Time Execution matrix

V x L	LOCAL	L_1	L_2	L_3	L_4	L_5	L_6
v_0	3.33	2	1	0.667	0.5	0.4	0.333
v_1	5	3	1.5	1	0.75	0.6	0.5
v_2	6	3.6	1.8	1.2	0.9	0.72	0.6
v_3	6.667	4	2	1.333	1	0.8	0.667
v_4	8.33	5	2.5	1.667	1.25	1	0.833
v_5	6.667	4	2	1.333	1	0.8	0.667
v_6	10	6	3	2	1.5	1.2	1
v_7	9.33	5.6	2.8	1.867	1.4	1.12	0.933
v_8	7.667	4.6	2.3	1.533	1.15	0.92	0.767
v_9	10	6	3	2	1.5	1.2	1
v_{10}	9.667	5.8	2.9	1.933	1.45	1.16	0.967

Table 10 Tasks completion times

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
3.33	8.33	4.67	4.25	9.36	9.15	10.91	13.74	12.46	19.74	21.22

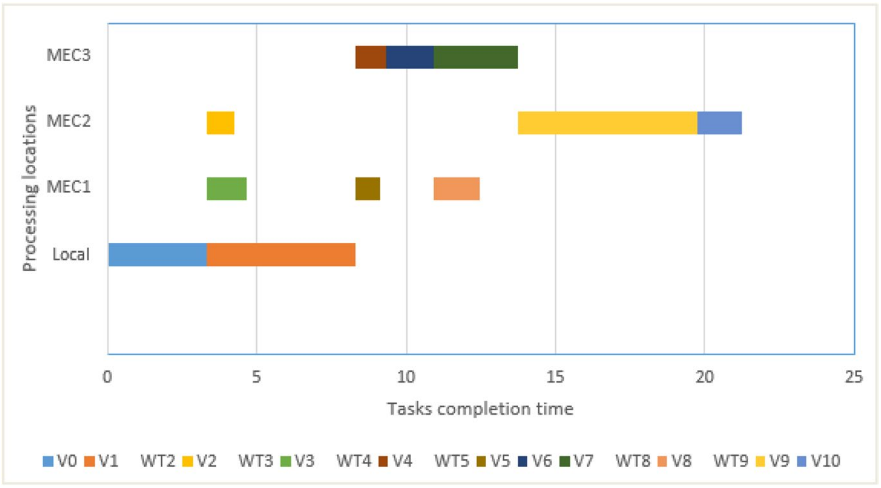


Fig. 6 Complete example tasks offloading chart

Algorithm 5. The IGTA algorithm

```
3:   t=1
4:   While(t ≤ iters)
5:       Update y, using equations (15, 17)
         %Start the exploration phase
6:       For search agent i in the population
7:           Update random variables ($r_1$,$r_2$,$r_3$,p)
8:           If (P<S)
9:               Update $GX_i(t+1)$ using equation (13)
10:          Else if (P ≥ 0.5)
11:              Update $GX_i(t+1)$ using equation (14)
12:          Else
13:              Update $GX_i(t+1)$ using equation (19)
14:          End if
15           Do Vector Values Mapping Operation for $GX_i(t+1)$
16:      End for
17:      Update X with GX, if better
18:      Update $X_{sb}$ with the best solution
         %Start the exploitation phase
19:      For search agent i in the population
20:          If (y≥w)
21:              Update $GX_i(t+1)$ using equation (20)
22:          Else
23:              Update $GX_i(t+1)$ using equation (22)
24:          End if
25           Do Vector Values Mapping Operation for $GX_i(t+1)$
26:      End for
27:      Recalculate the fitness of each search agent
28:      Update solutions if the new ones are better
29:      Update $X_{sb}$
         %start the Enhancement phase
30       r= generate a random number (0-1)
31       If (r≤0.7)
32:          $X_{sb}$= apply the enhancement phase using an algorithm (4)
33       End If
34:      $X_{sb}$= apply the MTM load balancing phase using the algorithm
         (5)
35:      t++
36:  End while
```

processing location charge level, and shuffle its successor list.

The PSO's basic parameters are set as follows as recommended in [56, 57]:

- The social coefficient is 1.57.
- The cognitive coefficient is 1.42.
- Weight inertia is 0.7298.

The other algorithms' parameters as set following recommendations made by their authors.

6.3 Comparison between GTO and IGTA

In this case, we focus on analyzing how the proposed algorithm was affected by the crossover and MTM improvement operations. Therefore, using six DVFS levels and three MECs, we will experimentally determine how different task sizes and parameters defined in Table 11 affect the performance of GTO and IGTA. In this experiment, we employ four distinct measures, including average values for completion time, energy, cost, and fitness.

Table 11 Datasets description

No	TG name	No. of tasks	No. of edges	CC in mega	Q_{in} in mega	Q_{out} in mega
1	TG1-41	9	10	[3, 60]	[5, 50]	[0.5, 4.5]
2	TG1-83	9	10	[6, 50]		
3	TG1-100	9	10	[2, 78]		
4	TG2-1	29	36	[3, 37]		
5	TG2-49	29	36	[0, 220]		
6	TG2-100	29	36	[2, 204]		
7	TG3-2	23	22	[0.0, 148.1]		
8	TG3-61	23	22	[0.1, 72.2]		
9	TG3-100	23	22	[0.0, 102.8]		
10	EX_Data	10	15	[10, 30]		

Table 12 displays the outcomes of the two algorithms (GTO and IGTA) classified based on the abovementioned measures. We can see from the outcomes that IGTA performs better than GTO across all datasets. It scored better values on all the performance measures. The results have clarified the role of the added improvement operations in enhancing the behavior of the standard GTO algorithm.

The total values of the findings from Table 12 for each of the ten datasets are shown in Fig. 7 for the GTO and IGTA algorithms. IGTA receives the minimum completion time with a value of 93.5, while GTO receives 139.5. Additionally, IGTA achieves improved energy consumption and cost savings with 71.5 and 55.46, respectively. It also scored the minimum fitness of 55.46 points, a good performance metric for distributing task loads across mobile edge computing servers at various charge levels.

In the second experiment, we used the dataset TG2-100 to examine how the performance of GTO and IGTA will change when five different numbers of MEC servers (2, 3, 5, 7, and 9) are used. The local processor is also added in each experiment. First, we determine the completion time values for executing the dataset TG2-100 twenty times using each MEC number. The average of the completion time values is then calculated. The average completion time for each MEC number is then added up. This procedure is repeated for the other three performance metrics (energy consumption, used cost, and fitness function). As seen in Fig. 8, the improvements proposed on the GTO algorithm resulted in better results across all performance metrics.

We can see in Fig. 8 that IGTA obtained better results for all performance metrics. It scored the minimum completion time with a value of 121.5 s, the minimum total Energy consumption with 0.56 J, the minimum total cost usage with 77.47$, and the minimum fitness function with a value of 66.97.

Table 12 Results of GTO and IGTA using Table 11 datasets

NO	TG name	Avg. Completion time		Avg. Energy		Avg. Cost		Avg. Fitness	
		GTO	IGTA	GTO	IGTA	GTO	IGTA	GTO	IGTA
1	TG1-41	8.17	4.96	1.15	0.03	2.55	1.68	4.00	2.25
2	TG1-83	8.48	6.27	1.04	0.02	2.43	1.83	4.03	2.74
3	TG1-100	7.13	5.07	0.14	0.03	1.76	1.16	3.05	2.12
4	TG2-1	15.28	9.18	3.45	0.08	18.14	12.05	12.32	7.13
5	TG2-49	25.63	18.96	3.33	0.08	27.61	21.44	18.93	13.54
6	TG2-100	32.72	23.59	11.27	0.09	30.92	17.9	25.05	13.96
7	TG3-2	17.78	10.16	0.29	0.07	11.29	6.96	9.87	5.77
8	TG3-61	4.59	3.56	0.38	0.08	2.94	2.17	2.66	1.95
9	TG3-100	9.06	6.96	0.50	0.07	7.14	3.90	5.60	3.68
10	EX_Data	11.04	4.74	0.02	0.02	4.34	2.12	5.19	2.32

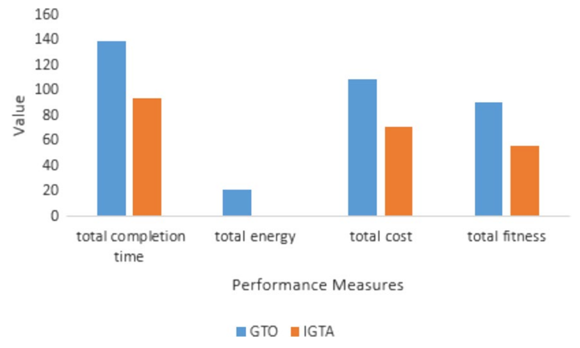


Fig. 7 Total performance measures' values for GTO and IGTA Using different datasets

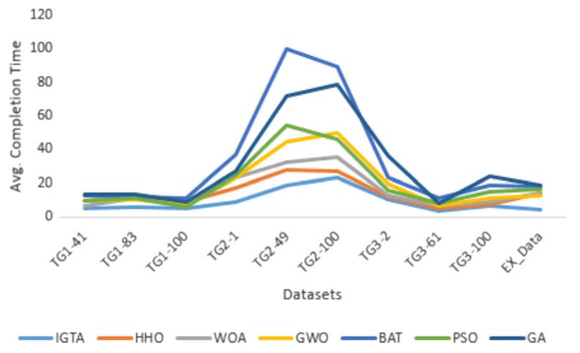


Fig. 9 Completion time results using adopted datasets and three MEC servers

6.4 Comparison with Other Algorithms

This subsection compares the performance of the proposed algorithm with HHO, WOA, PSO, BAT, GWO, and GA using the datasets defined in Table 11. Four performance metrics are used (completion time, energy consumption, cost usage, and fitness function) to evaluate the effectiveness of the algorithms. We run each algorithm on each dataset twenty times using six charge levels. As a result, we calculated the average for each used performance measure (PerM) using the following equation:

$$avg.\mathrm{PerM} = \frac{\sum_{j=1}^{20} \mathrm{PerM}_j}{20} \tag{33}$$

where PerM_j is the performance measure value obtained from running an algorithm on a dataset.

Figure 9 shows the average completion times of each algorithm on the adopted datasets. The figure shows that the proposed algorithm outperforms the other algorithms on all used datasets. It is also obvious that the ranking of other comparison algorithm change on each dataset. This observation ensures that our datasets validated the comparison approaches' performance metrics. Furthermore, Fig. 10 presents the total average completion times for the used data sets obtained by summing the average completion times produced from running each algorithm on all adopted datasets. The results showed that our algorithm provides the minimum completion time value of 93.45 s. HHO algorithm scored a completion time value of 140.31 s as the second minimum completion time value. On the other side, the maximum completion time value of 300.58 s is scored by the GA algorithm.

The second performance metric used in the comparison is the energy consumption measure. This metric is significant since batteries are the primary power source for mobile and IoT devices. Using the adopted test cases,

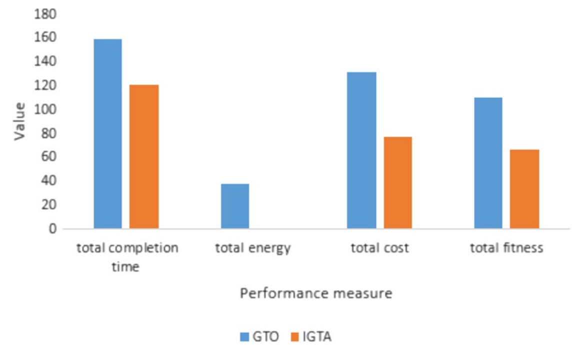


Fig. 8 Total performance measures' values for GTO and IGTA using a different number of MEC servers

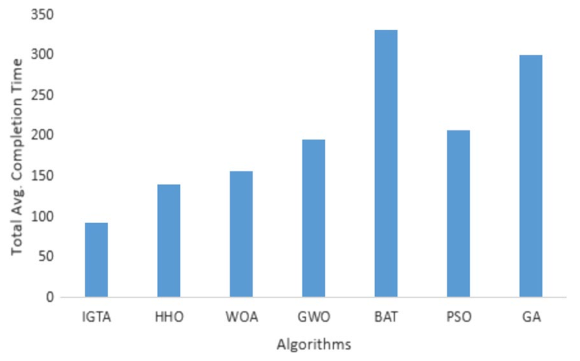


Fig. 10 Total avg. Completion times were obtained by running each comparison algorithm on the adopted datasets

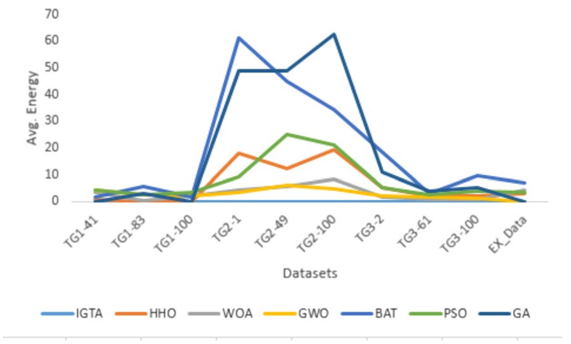


Fig. 11 Energy consumption results using adopted datasets and three MEC servers

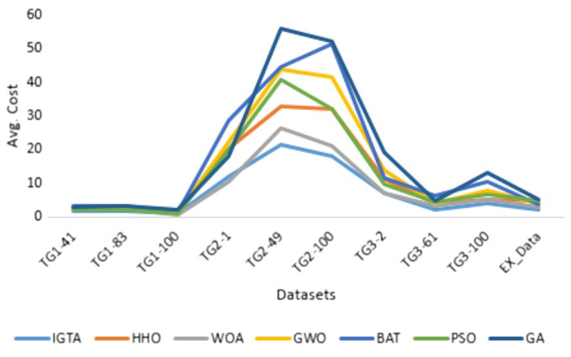


Fig. 13 Cost usage results using adopted datasets and three MEC servers

Fig. 11 shows that our algorithm achieved much more optimization according to this metric for all test cases. For more quantified values, Fig. 12 presents the total average energy consumption results of running each algorithm on each task graph set twenty times. The figure shows that the proposed algorithm scored the minimum value of 0.57 J, while GWO accomplished the second minimum value of 28.55 J. The bat algorithm reached the maximum energy consumption value of 187.26 J.

The third performance measure is the cost used to execute the application tasks, which is important for designing an economic plan. As a result of running each algorithm on the test datasets of Table 11 using the seven charge levels prices provided in Table 7, Fig. 13 shows the total average cost usage results. It is clear from the results that our algorithm introduces the minimum cost levels for all test sets. Figure 14 also did its role in providing an obvious viewpoint in quantifying the cost usage results scored by the comparison algorithms. The figure shows that the proposed algorithm hit the most optimized cost by 71.21$.WOA hit the second optimum value of 82.18$. The maximum cost usage value of 176.35 is obtained from the GA algorithm.

The fitness values are an absolutely important factor in measuring the performance of the algorithms since they present a general viewpoint that combines all of the previously mentioned metrics. Figure 15 depicts the average fitness for each algorithm on the test sets. This figure confirmed the other metrics' results that witnessed the proposed algorithm outperforming the other approaches. Figure 16 also shows that IGTA accomplishes the minimum fitness value of 55.46, and the BAT algorithm hits the maximum value of 225.29. The other algorithms hit the following fitness values: WOA with a value of 91.01, HHO with a value of 107.32, GWO with a value of 123.54, PSO with a value of 137.76, and GA with a value of 221.22.

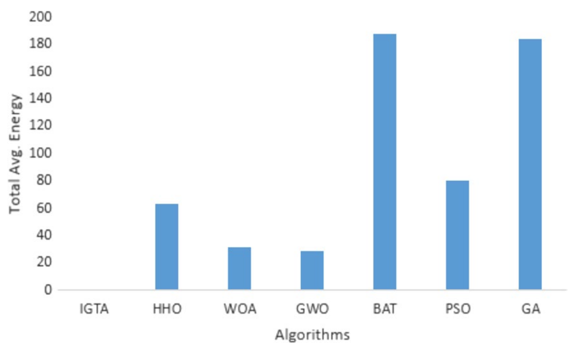


Fig. 12 Total avg. Energy consumption values were obtained by running each comparison algorithm on the adopted datasets

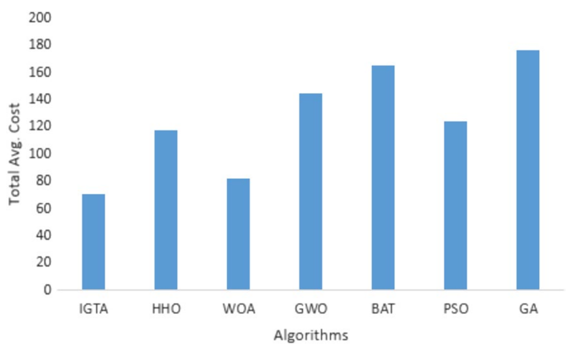


Fig. 14 Total avg. Cost usage values were obtained by running each comparison algorithm on the adopted datasets

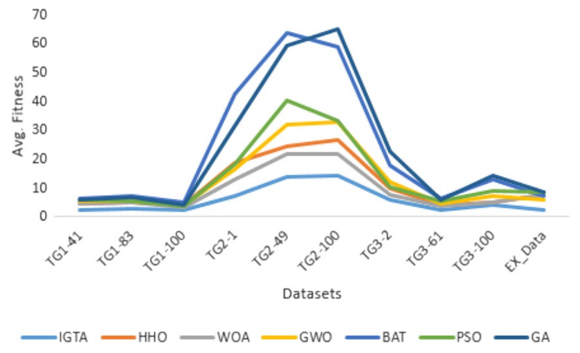


Fig. 15 Fitness function results using adopted datasets and three MEC servers

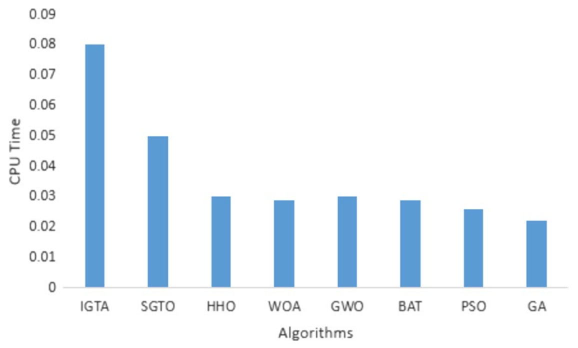


Fig. 17 Avg. Processing time for each algorithm on TG2-100

The average CPU time for each algorithm on the TG2-100 dataset is presented in Fig. 17. It is obvious in the figure that our algorithm hit the maximum CPU time with a value of 0.08 s, and GA hit the minimum value of 0.022 s. The other algorithms' CPU times are as follows: PSO with a value of 0.026 s, BAT with a value of 0.029 s, WOA with a value of 0.029 s, GWO with a value of 0.03 s, HHO with a value of 0.03 s, and standard GTO with value 0.05. But the CPU times have no impact on the performance measure of our proposed approach if this IGTA is used in the design phase of the IoT and mobile applications. As we previously mentioned in subsection 3.2, this framework aims to enable IoT and mobile application developers to build efficient applications that can benefit the MEC servers the most. Moreover, we suggest the installation of IGTA on the MEC server in the MEC environments that dynamically change the importance of the objective over time, which can reduce the impact of the relatively long CPU time of IGTA on the performance measures, especially for computationally extensive applications.

Figure 18 shows the total average performance measures' values of the algorithms using different MEC servers (2, 3, 5, 7, and 9) on the TG2-100 dataset. It can be seen from the figure that IGTA scored the minimum completion time value of 121.5 s, and the HHO algorithm scored the second minimum value of 146.2. Conversely, the BAT algorithm scored the maximum value of 349.11 s. But regarding the energy consumption measure, our algorithm also accomplished the most optimized value of 0.56 J, and the WOA scored the second optimum value of 20.96. While the maximum value of 174.02 was produced from the BAT algorithm.

Regarding used cost, IGTA also reached the minimum value of 77.47$. The second minimum value of 80.06$ was obtained by the WOA, while the BAT

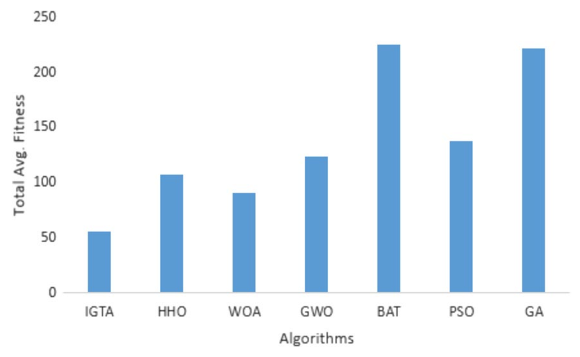


Fig.16 Total avg. Fitness function values were obtained by running each comparison algorithm on the adopted datasets

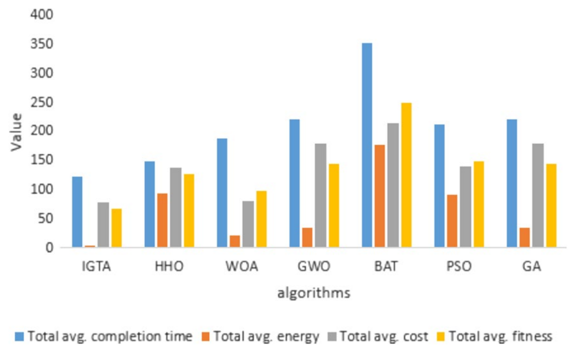


Fig. 18 Total avg. Performance measures values of the algorithms using a different number of MEC servers

algorithm reached the maximum value of 212.88$. Finally, regarding the fitness function, the results accomplished by the algorithms were as follows: IGTA with a value of 66.97, WOA with a value of 96.34, HHO with a value of 124.64, GWO with a value of 142.97, GA with a value of 142.97, PSO with value 146.93, and the BAT algorithm with value 246.39.

The results also showed that IGTA outperforms all the algorithms included in the comparison. These results confirm the stability of the IGTA performance on the change in the number of installed MEC servers in the environment.

We can infer from the extensive experiments that the proposed algorithm has performed significantly better than other meta-heuristic algorithms at saving energy, minimizing the cost, and shortening computation time.

7 Conclusion

This study examines a multi-server mobile edge computing system with multi-task dependence in which three goals were optimized simultaneously: the application completion time, MD energy consumption, and MEC server usage fee. Moreover, IGTA can be used in MEC environments that dynamically change the relative importance of objectives over time. To overcome these issues, we offered an improved multi-objective gorilla troops algorithm for solving the dependent task offloading problem in the MEC environment with three objectives. In the proposed method, each gorilla vector was broken down into three parts: a task list part, a MECs list part, and a charge levels list part. A particular initialization method was also used to create workable gorilla vectors. Since the gorilla vectors that are produced by the GTO algorithm are continuous values, two mapping techniques were used to convert such vector values into discrete and bounded ones. Additionally, we have extended the operation of the standard Gorilla Troops Algorithm (GTO) by adopting a customized crossover operation to improve its search strategy. A Max-To-Min (MTM) load-balancing strategy was also implemented in IGTA to improve the offloading operation. To validate IGTA performance, we have run extensive simulation experiments on ten test instances with various task topologies and profiles. These experiments compared IGTA with GTO, HHO, WOA, GWO, BAT, PSO, and the GA algorithm. Finally, the simulation results confirmed the superiority of IGTA's overall performance metrics. For instance, Relative to GTO, IGTA has reduced latency by 33%, energy consumption by 93%, and cost usage by 34.5%.

Authors' contributions **Khalid M. Hosny**: Conceptualization, Methodology, Writing—Review & Editing, Supervision.

Ahmed Awad: Conceptualization, Methodology, Validation, Software, Writing- Original draft.

Marwa M. Khashaba: Methodology, Validation, Supervision.

Ehab R. Mohamed: Methodology, Validation, Supervision.

Funding Open access funding provided by The Science, Technology & Innovation Funding Authority (STDF) in cooperation with The Egyptian Knowledge Bank (EKB).

Data Availability Data is available upon request.

Declarations

This work is original and not have been published elsewhere in any form or language.
No participants in this work.

Competing interest No financial and non-financial competing interests.

References

1. Mach, P., Becvar, Z.: Mobile edge computing: A survey on architecture and computation offloading. arXiv **19**(3), 1628–1656 (2017)
2. Kekki, S. et al.: 【ETSI白皮书】MEC in 5G networks. ETSI White Pap. (28), 1–28 (2018)
3. Awad, A.I., Fouda, M.M., Khashaba, M.M., Mohamed, E.R., Hosny K.M.: Utilization of mobile edge computing on the Internet of Medical Things: A survey. ICT Express. no. xxxx, (2022). https://doi.org/10.1016/j.icte.2022.05.006.
4. Goudarzi, M., Wu, H., Palaniswami, M., Buyya, R.: An Application Placement Technique for Concurrent IoT

Applications in Edge and Fog Computing Environments. IEEE Trans. Mob. Comput. **20**(4), 1298–1311 (2021). https://doi.org/10.1109/TMC.2020.2967041
5. Xia, Z., Abu Qahouq, J.A.: State-of-Charge Balancing of Lithium-Ion Batteries with State-of-Health Awareness Capability. IEEE Trans. Ind. Appl. **57**(1), 673–684 (2021). https://doi.org/10.1109/TIA.2020.3029755
6. Portilla, J., Mujica, G., Lee, J.S., Riesgo, T.: The Extreme Edge at the Bottom of the Internet of Things: A Review. IEEE Sens. J. **19**(9), 3179–3190 (2019). https://doi.org/10.1109/JSEN.2019.2891911
7. Wang, S., Zhao, Y., Xu, J., Yuan, J., Hsu, C.H.: Edge server placement in mobile edge computing. J. Parallel Distrib. Comput. **127**, 160–168 (2019). https://doi.org/10.1016/j.jpdc.2018.06.008
8. Abbas, N., Zhang, Y., Taherkordi, A., Skeie, T.: Mobile Edge Computing: A Survey. IEEE Internet Things J. **5**(1), 450–465 (2018). https://doi.org/10.1109/JIOT.2017.2750180
9. Reznik, A. et al.: Developing Software for Multi-Access Edge Computing. **20**, 1–38 (2017)
10. Islam, A., Debnath, A., Ghose, M., Chakraborty, S.: A Survey on Task Offloading in Multi-access Edge Computing. J. Syst. Archit. **118**(June), 102225 (2021). https://doi.org/10.1016/j.sysarc.2021.102225
11. Sundar, S., Liang, B.: Offloading Dependent Tasks with Communication Delay and Deadline Constraint. Proc. - IEEE INFOCOM **2018-April**, 37–45 (2018). https://doi.org/10.1109/INFOCOM.2018.8486305
12. Huang, M., Zhai, Q., Chen, Y., Feng, S., Shu, F.: Multi-objective whale optimization algorithm for computation offloading optimization in mobile edge computing. Sensors **21**(8), 1–24 (2021). https://doi.org/10.3390/s21082628
13. Aldmour, R., Yousef, S., Yaghi, M., Tapaswi, S., Pattanaik, K.K., Cole, M.: New cloud offloading algorithm for better energy consumption and process time. Int. J. Syst. Assur. Eng. Manag. **8**(s2), 730–733 (2017). https://doi.org/10.1007/s13198-016-0515-2
14. Wan, Z., Xu, D., Xu, D., Ahmad, I. Joint computation offloading and resource allocation for NOMA-based multi-access mobile edge computing systems. Comput. Netw. **196** (June), (2021). https://doi.org/10.1016/j.comnet.2021.108256
15. Shahidinejad, A., Ghobaei-Arani, M.: A metaheuristic-based computation offloading in edge-cloud environment. J. Ambient Intell. Humaniz. Comput. **13**(5), 2785–2794 (2022). https://doi.org/10.1007/s12652-021-03561-7
16. Shakarami, A., Shahidinejad, A., Ghobaei-Arani, M.: A review on the computation offloading approaches in mobile edge computing: A game-theoretic perspective. Softw. - Pract. Exp. **50**(9), 1719–1759 (2020). https://doi.org/10.1002/spe.2839
17 Shakarami, A., Ghobaei-Arani, M., Shahidinejad, A.: A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective. Comput. Networks **182**(August), 107496 (2020). https://doi.org/10.1016/j.comnet.2020.107496
18. Al-Habob, A.A., Dobre, O.A., Armada, A.G., Muhaidat, S.: Task scheduling for mobile edge computing using genetic algorithm and conflict graphs. IEEE Trans. Veh. Technol. **69**(8), 8805–8819 (2020). https://doi.org/10.1109/TVT.2020.2995146
19. Abdel-Basset, M., El-Shahat, D., Deb, K., Abouhawwash, M.: Energy-aware whale optimization algorithm for real-time task scheduling in multiprocessor systems. Appl. Soft Comput. J. **93**, 106349 (2020). https://doi.org/10.1016/j.asoc.2020.106349
20 Abdollahzadeh, B., SoleimanianGharehchopogh, F., Mirjalili, S.: Artificial gorilla troops optimizer: A new nature-inspired metaheuristic algorithm for global optimization problems. Int. J. Intell. Syst. **36**(10), 5887–5958 (2021). https://doi.org/10.1002/int.22535
21. Song, F., Xing, H., Wang, X., Luo, S., Dai, P., Li, K.: Offloading dependent tasks in multi-access edge computing: A multi-objective reinforcement learning approach. Futur. Gener. Comput. Syst. **128**, 333–348 (2022). https://doi.org/10.1016/j.future.2021.10.013
22 Fang, J., Zhang, M., Ye, Z., Shi, J., Wei, J.: Smart collaborative optimizations strategy for mobile edge computing based on deep reinforcement learning. Comput. Electr. Eng. **96**(PA), 107539 (2021). https://doi.org/10.1016/j.compeleceng.2021.107539
23. Aldmour, R., Yousef, S., Baker, T., Benkhelifa, E.: An approach for offloading in mobile cloud computing to optimize power consumption and processing time. Sustain. Comput. Informatics Syst. **31**, 100562 (2021). https://doi.org/10.1016/j.suscom.2021.100562
24. Wang, K., Ding, Z., So, D.K.C., Karagiannidis, G.K.: Stackelberg Game of Energy Consumption and Latency in MEC Systems with NOMA. IEEE Trans. Commun. **69**(4), 2191–2206 (2021). https://doi.org/10.1109/TCOMM.2021.3049356
25. Zheng, J., Cai, Y., Wu, Y., Shen, X.: Dynamic computation offloading for mobile cloud computing: A stochastic game-theoretic approach. IEEE Trans. Mob. Comput. **18**(4), 771–786 (2019). https://doi.org/10.1109/TMC.2018.2847337
26. Peng, H., Wen, W.S., Tseng, M.L., Li, L.L.: Joint optimization method for task scheduling time and energy consumption in mobile cloud computing environment. Appl. Soft Comput. J. **80**(2019), 534–545 (2019). https://doi.org/10.1016/j.asoc.2019.04.027
27. Zhao, G., Xu, H., Zhao, Y., Qiao, C., Huang, L.: Offloading Tasks with Dependency and Service Caching in Mobile Edge Computing. IEEE Trans. Parallel Distrib. Syst. **32**(11), 2777–2792 (2021). https://doi.org/10.1109/TPDS.2021.3076687
28. Liu, J., Mao, Y., Zhang, J., Letaief, K.B.: Delay-optimal computation task scheduling for mobile-edge computing systems. IEEE Int Symp. Inf. Theory - Proc. **2016-Augus**, 1451–1455 (2016). https://doi.org/10.1109/ISIT.2016.7541539
29. Huang, B., et al.: Security modeling and efficient computation offloading for service workflow in mobile edge computing. Futur. Gener. Comput. Syst. **97**, 755–774 (2019). https://doi.org/10.1016/j.future.2019.03.011
30. Xie, Y., et al.: A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud–edge environment. Futur. Gener. Comput. Syst. **97**, 361–378 (2019). https://doi.org/10.1016/j.future.2019.03.005
31. Ma, S., Song, S., Yang, L., Zhao, J., Yang, F., Zhai, L.: Dependent tasks offloading based on particle swarm

optimization algorithm in multi-access edge computing. Appl. Soft Comput. **112**, 107790 (2021). https://doi.org/10.1016/j.asoc.2021.107790

32. Jia, M., Cao, J., Yang, L.: Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing. Proc. - IEEE INFOCOM. 352–357 (2014). https://doi.org/10.1109/INFCOMW.2014.6849257
33. Liu, L., Tan, H., Jiang, S.H.C., Han, Z., Li, X.Y., Huang, H.: 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
34. Wang, J., Hu, J., Min, G., Zhan, W., Ni, Q., Georgalas, N.: Computation Offloading in Multi-Access Edge Computing Using a Deep Sequential Model Based on Reinforcement Learning. IEEE Commun. Mag. **57**(5), 64–69 (2019). https://doi.org/10.1109/MCOM.2019.1800971
35. Wu, Q., Wu, Z., Zhuang, Y., Y.C.B.: Adaptive DAG Tasks Scheduling, vol. 1. Springer International Publishing (2018)
36. Wang, J., Hu, J., Min, G., Zomaya, A.Y., Georgalas, N.: Fast Adaptive Task Offloading in Edge Computing Based on Meta Reinforcement Learning. IEEE Trans. Parallel Distrib. Syst. **32**(1), 242–253 (2021). https://doi.org/10.1109/TPDS.2020.3014896
37. Zhu, A. et al.: Computation offloading for workflow in mobile edge computing based on deep Q-learning, 2019 28th Wirel. Opt. Commun. Conf. WOCC 2019 - Proc., no. Wocc, pp. 1–5 (2019). https://doi.org/10.1109/WOCC.2019.8770689
38. Qu, G., Wu, H., Li, R., Jiao, P.: DMRO: A Deep Meta Reinforcement Learning-Based Task Offloading Framework for Edge-Cloud Computing. IEEE Trans. Netw. Serv. Manag. **18**(3), 3448–3459 (2021). https://doi.org/10.1109/TNSM.2021.3087258
39. Lu, H., Gu, C., Luo, F., Ding, W., Liu, X.: Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning. Futur. Gener. Comput. Syst. **102**, 847–861 (2020). https://doi.org/10.1016/j.future.2019.07.019
40. Yan, J., Bi, S., Zhang, Y.J.A.: Offloading and Resource Allocation with General Task Graph in Mobile Edge Computing: A Deep Reinforcement Learning Approach. IEEE Trans. Wirel. Commun. **19**(8), 5404–5419 (2020). https://doi.org/10.1109/TWC.2020.2993071
41. Ali, Z., Jiao, L., Baker, T., Abbas, G., Abbas, Z.H., Khaf, S.: A deep learning approach for energy efficient computational offloading in mobile edge computing. IEEE Access **7**, 149623–149633 (2019). https://doi.org/10.1109/ACCESS.2019.2947053
42. Cui, G., Li, X., Xu, L., Wang, W.: Latency and energy optimization for MEC enhanced SAT-IoT networks. IEEE Access **8**, 55915–55926 (2020). https://doi.org/10.1109/ACCESS.2020.2982356
43. Agiwal, M., Roy, A., Saxena, N.: Next generation 5G wireless networks: A comprehensive survey. IEEE Commun. Surv. Tutorials **18**(3), 1617–1655 (2016). https://doi.org/10.1109/COMST.2016.2532458
44 Wang, S., Qian, Z., Yuan, J., You, I.: A DVFS Based Energy-Efficient Tasks Scheduling in a Data Center. IEEE Access **5**(3), 13090–13102 (2017). https://doi.org/10.1109/ACCESS.2017.2724598
45. Song, F., Xing, H., Luo, S., Zhan, D., Dai, P., Qu, R.: A Multiobjective Computation Offloading Algorithm for Mobile-Edge Computing. IEEE Internet Things J. **7**(9), 8780–8799 (2020). https://doi.org/10.1109/JIOT.2020.2996762
46. Mach, P., Becvar, Z.: Mobile Edge Computing: A Survey on Architecture and Computation Offloading. IEEE Commun. Surv. Tutorials **19**(3), 1628–1656 (2017). https://doi.org/10.1109/COMST.2017.2682318
47. Nguyen, P. D., Le, L. B.: Joint computation offloading, SFC placement, and resource allocation for multi-site MEC systems. IEEE Wirel. Commun. Netw. Conf. WCNC.**2020-May**, (2020). https://doi.org/10.1109/WCNC45663.2020.9120597
48. Chaari, M. Z., Al-Maadeed, S.: Wireless power transmission for the Internet of Things (IoT), 2020 IEEE Int. Conf. Informatics, IoT, Enabling Technol. ICIoT 2020. 549–554 (2020). https://doi.org/10.1109/ICIoT48696.2020.9089547
49. Szymanski, T. H.: 300 Pseudo-random task graphs for evaluating mobile cloud Fog and Edge Computing Systems. https://doi.org/10.21227/kak5-8n96
50. Heidari, A.A., Mirjalili, S., Faris, H., Aljarah, I., Mafarja, M., Chen, H.: Harris hawks optimization: Algorithm and applications. Futur. Gener. Comput. Syst. **97**, 849–872 (2019). https://doi.org/10.1016/j.future.2019.02.028
51. Mirjalili, S., Lewis, A.: The Whale Optimization Algorithm. Adv. Eng. Softw. **95**, 51–67 (2016). https://doi.org/10.1016/j.advengsoft.2016.01.008
52. Mirjalili, S., Mirjalili, S.M., Lewis, A.: Grey Wolf Optimizer. Adv. Eng. Softw. **69**, 46–61 (2014). https://doi.org/10.1016/j.advengsoft.2013.12.007
53. Mirjalili, S., Mirjalili, S.M., Yang, X.S.: Binary bat algorithm. Neural Comput. Appl. **25**(3–4), 663–681 (2014). https://doi.org/10.1007/s00521-013-1525-5
54. D. Wang, D. Tan, L. Liu.: Particle swarm optimization algorithm: an overview. Soft Comput. **22**(2), 387–408 (2018). https://doi.org/10.1007/s00500-016-2474-6
55. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002). https://doi.org/10.1109/4235.996017
56. Huang, Y., Tang, C., Wang, S.: Quantum-inspired swarm evolution algorithm, Proc. - CIS Work. 2007, 2007 Int. Conf. Comput. Intell. Secur. Work., pp. 208–211, (2007). https://doi.org/10.1109/cisw.2007.4425481
57. Semnani, A., Nabi Bidhendi, M., Nadjar Araabi, B.: Detection of Low-frequency Shadow Zones using Quantum Swarm Evolutionary Matching Pursuit Decomposition (QSE-MPD). cp-363–00037, (2013). https://doi.org/10.3997/2214-4609.20131866