Egyptian Informatics Journal 22 (2021) 257–267

Contents lists available at ScienceDirect

Egyptian Informatics Journal

journal homepage: www.sciencedirect.com

# Evolutionary offloading in an edge environment

Samah A. Zakaryia *, Safaa A. Ahmed, Mohamed K. Hussein

*Department of Computer Science, Faculty of Computers and Informatics, Ismailia, Egypt*

ARTICLE INFO

*Article history:*
Received 6 May 2020
Revised 19 August 2020
Accepted 22 September 2020
Available online 9 October 2020

*Keywords:*
Mobile edge computing
Computation offloading
Genetic algorithm optimization
Evolutionary optimization
Particle swarm optimization

ABSTRACT

Due to increasing complexity of mobile applications, and limited computation resources of smart mobile devices, the quality of service requirements of mobile application can be enhanced by offloading the computation tasks of the mobile applications to edge servers, such as cloudlets, which exist at the edge of wireless networks. However, improper placement of mobile tasks on the edge servers may increase the waiting time and the transmission time. This, in turn, will increase the response time, and eventually violates the quality of service.

This paper proposes an effective offloading strategy in a mobile edge environment using the queuing networks and an evolutionary algorithm, namely the genetic algorithm (GA). The queuing network is used to model the waiting time and the service time of the mobile tasks on the edge servers. The genetic algorithm finds the best allocation of mobile tasks on the edge servers to minimize tasks response time considering the transmission times and the load conditions on edge servers represented by the waiting times and the service times which are calculated using the queuing network. The proposed GA-based offloading algorithm is compared with another evolutionary algorithm, namely particle swarm optimization (PSO). Experimental evaluations show that the GA-based offloading algorithm outperforms both of round robin offloading and the PSO-based offloading algorithms, and effectively improves mobile applications response time.



## 1. Introduction

Nowadays, there has been a major advancement in the technology of the smart mobile devices (SMD), such as smart mobiles phones and tablets. This has led to increase the functionality and the computational complexity of mobile applications, such as mobile commerce, picture searching, mobile learning, mobile games, and healthcare [1]. The form of computing where the computation is performed using mobile devices is called *mobile computing* (MC). However, SMD are poor-resource constraints, such as power, storage, memory, and processing, compared to the conventional processing devices. The limited resources of the SMD makes mobile computing unsuitable for the current computation intensive mobile applications that require better processing capabilities to improve the response time [2]. The limitation of mobile computing has led to the emergence of a promising paradigm called *mobile cloud computing* (MCC) where computationally intensive tasks of mobile applications are offloaded to the cloud for processing [3].

The mobile cloud computing is an infrastructure where both mobile data and tasks are stored and processed on powerful virtualized resources located in the cloud infrastructure which can be accessed on-demand [4,5]. As a shown in Fig. 1, computationally intensive tasks of mobile applications are offloaded to the cloud over 4G/5G connection for storage and processing, and the results are delivered to the smart mobile device [6]. This process is called *task offloading*. Consequently, MCC overcomes the constrained resources of mobile devices and improves both of the response time of intensive mobile applications and the power consumption of smart mobile devices. However, offloading tasks to the cloud increases the response time of delay sensitive mobile applications due to the low bandwidth and the high latency of the Internet [7].

To support delay sensitive mobile applications, mobile edge computing (MEC) has emerged to provide computation capability at the edge of the wireless network near the mobile devices. The edge consists of several computing servers, called *cloudlets*, which are trusted computing resources available for use by nearby mobile devices using a wireless network [8]. The main idea is to offload the computation tasks of mobile applications onto powerful

* Corresponding author.
*E-mail addresses:* samzak.2019@ci.suez.edu.eg (S.A. Zakaryia), Safaa@ci.suez.edu.eg (S.A. Ahmed), m_khamiss@ci.suez.edu.eg (M.K. Hussein).

Peer review under responsibility of Faculty of Computers and Information, Cairo University.

https://doi.org/10.1016/j.eij.2020.09.003
1110-8665/

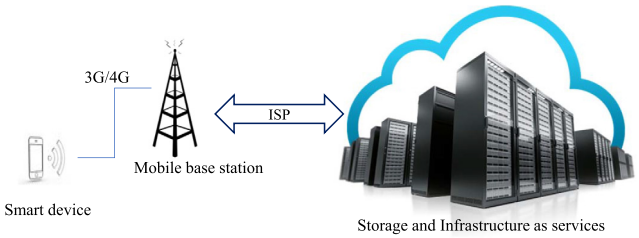


**Fig. 1.** The mobile cloud computing infrastructure.

computing resources located at the edge of the network at a one-hop distance from mobile devices [9]. Offloading mobile tasks to cloudlets aims to improve the response time by exploiting location proximity and low latency, and to reduce the power consumption by using wireless communication rather than WAN communication which consumes more energy. However, the resources capacity of cloudlet servers are limited compared to the cloud, and improper mobile task offloading and allocation decisions to the cloudlet serevers will lead to increase the response times as the number of offloaded mobile tasks increases [10]. Therefore, it is challenging to design an effective mobile tasks offloading strategy considering the constraints of the resources in a mobile edge environment, including: (1) the tasks transmission time between existing existing existing cloudlets, the waiting time, and the service time at the cloudlet servers.

The contributions of this paper are as follows:

- The mobile tasks offloading problem in an edge environment is modeled as an optimization problem that aims at minimizing the response time of independent mobile tasks. The formulation model considers the transmission time between the edge servers. The waiting time and the service time of the offloaded mobile tasks on edge servers are modeled using the queuing networks.
- The optimization problem is solved using a proposed offloading and allocation algorithm based on an evolutionary algorithm, namely the genetic algorithm (GA). The GA-based offloading algorithm aims at minimizing the response time of mobile task based on the proposed queuing network model.
- Several experiment scenarios are performed to evaluate the proposed GA-based offloading algorithm, and are compared to another evolutionary-based offloading algorithm, namely particle swarm optimization (PSO), and a greedy-based offloading algorithm, namely round robin.
- Evaluation experiments show that the proposed GA-based offloading algorithm effectively minimizes the mobile tasks response time.

The rest of the paper is organized as follows. Section 2 introduces the related work of mobile tasks offloading in an edge environment. Section 3 presents a formal system model of the optimization problem, while Section 4 outlines the proposed evolutionary-based algorithms for solving the analyzed optimization problem. Section 5 the experimental results of the proposed evolutionary offloading algorithm are analyzed and discussed. Finally, Section 6 summarizes and concludes the paper with directions of possible future research works.

## 2. Background and related work

There are several surveys which define mobile edge computing, describe existing architecture, and discuss mobile edge computing issues and challenges [11–16]. In [17], an early trial for designing an architecture that instantiates virtual machines on cloudlets to offload tasks of nearby mobiles over a wireless network. In [18], a design of cloudlet network and service architecture is proposed, and a study of the impact of cloudlet on interactive mobile cloud applications is conducted. The experimental results proved the effectiveness of the cloudlet-based approach compared with the cloud-based approach for different application scenarios. In [19], a mobile-cloudlet-cloud architecture named MOCHA for mobile face recognition applications is proposed. Further, a simple task offloading and distribution among cloudlets considering the load conditions, diverse communication latencies, and compute powers of the cloudlet nodes aiming to minimize the response time. In [20], an admission control and resource allocation for offloading mobile applications on cloudlets using a semi-Markov decision model. The proposed model considers the quality of service (QoS) for different classes of mobile applications under the resource constraints of cloudlet nodes, including bandwidth and processing times. In [21], a mobile task offloading strategy using an ad hoc mobile devices that serve as cloudlets is proposed. A dynamic tasks scheduling algorithm is proposed based on weighted bigraph and Kuhn Munkras search algorithm. The offloaded tasks and available providers are considered two independent sets and the scheduling decision is made considering the resources consumption of offloaded tasks, their deadline, and minimizing the total cost from the clients completed tasks in terms of energy consumption. In [22] an offloading strategy using Moth-Flame Optimization is proposed aiming at minimizing IoT tasks response time considering the transmission time.

Evolutionary algorithms have shown an outstanding results in tasks and virtual machines scheduling in cloud computing [23,24]. In terms of task offloading on a mobile cloud computing architecture, in [25], an offloading strategy using evolutionary algorithms is proposed. The proposed strategy uses four different evolutionary algorithms, namely genetic algorithm, differential evolution, particle swarm optimization, and shuffled frog leaping algorithm. Experimental results have shown that the genetic algorithm outperforms the other algorithms. Moreover, in [26], an evolutionary-based offloading strategy using grey wolf optimization in a mobile cloud computing environment is proposed.

In terms of using evolutionary algorithms in task offloading on a mobile edge architecture, in [27], a mobile task offloading algorithm based on ant colony optimization considering load balancing and profit maximization is proposed. However, the proposed algorithm focuses on the utilization of the resources of the cloudlets. In [28], a hybrid evolutionary algorithm is proposed to allocate offloaded mobile tasks in an edge architecture aiming at minimizing the tasks average completion time. The offloading algorithm is based on a hybrid artificial bee colony (ABC) and ant colony optimization (ACO). However, the proposed offloading algorithm focuses on modelling the edge servers using queuing networks, and does not consider the heterogeneous latencies and the corresponding transmission time. In [29], an offloading strategy based on a genetic algorithm for offloading partial mobile tasks on a single edge server is proposed. The proposed offloading strategy aims at minimizing the completion time of the mobile tasks. However, this work focuses on determining the tasks partially offloading to achieve the goal of the research, and the proposed strategy does not consider the transmission cost. In [30], an evolutionary game offloading strategy based on using reinforcement learning is proposed. However, these proposed strategy does not consider the transmission time. Further, our proposed strategy uses queuing network for modelling the waiting times of the offloaded mobile tasks on the edge servers. In [31], an offloading strategy for the tasks of IoT devices on fog nodes is proposed. The offload strategy uses two different evolutionary algorithms, namely Ant Colony

Optimization and Particle Swarm Optimization. The main goal is to load balance the fog nodes and minimizing the response times of the IoT tasks while considering the communication cost between the fog nodes and the IoT devices. Our strategy is based on queuing network and genetic algorithm that aims at minimizing the completion time of the offloaded tasks considering the communication time, waiting time, and the service time on the edge servers.

Table 1 summarizes the related research works for mobile task offloading, and compares with the proposed task offloading in terms of optimization technique, environment, advantages, and disadvantages.

## 3. MEC system model and problem formulation

This section presents a formal system model for mobile tasks offloading in an edge environment and the corresponding constraints of the mobile edge infrastructure.

### 3.1. The MEC architecture

The MEC architecture used in this paper adopts a cloud/cloudlet architecture, as shown in Fig. 2. A mobile device

**Table 1**
Comparison of works related to mobile task offloading.

| Reference | Environment | Offloading Strategy | Objectives | Disadvantages |
|---|---|---|---|---|
| [20] | MEC | Semi-Markov decision model | Maximizing the utilization of cloudlets resources | Considering only bandwidth |
| [21] | MCC | Weighted Bi-graph model and Kuhn Munkras search | Maximizing the number of completed tasks | Considering only a single cloudlet |
| [22] | MCC | Moth-Flame Optimization | Minimizing the response time | Considering only transmission time |
| [25] | MCC | Genetic algorithm, differential evolution, particle swarm optimization, and shuffled frog leaping algorithm. | Minimizing the response time | No cloudlets |
| [26] | MCC | Grey wolf optimization | Minimize the execution time | No cloudlets |
| [27] | MEC | Ant colony optimization | Load balance of cloudlets | Considers only the utilization of cloudlets resources |
| [28] | MEC | Hybrid artificial bee colony and ant colony optimization | Minimizing completion time | The strategy does not consider communication cost |
| [29] | MEC | Genetic algorithm | Minimizing completion time | The strategy does not consider communication cost |
| [30] | MEC | Reinforcement learning | Minimizing completion time | The strategy does not consider the transmission cost |
| [31] | MEC | Ant colony optimization and particle swarm optimization | Load-balance the fog nodes and minimizing the response times | The strategy does not consider waiting times on the edge servers |
| The proposed strategy | MEC | Genetic algorithm and queuing networks | Minimizing the response times considering transmission time and waiting time | |

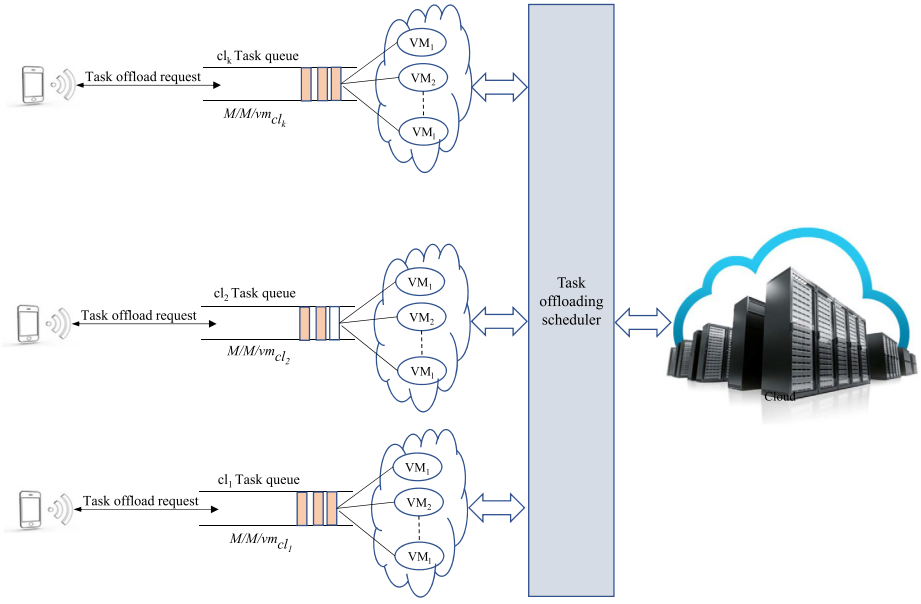


**Fig. 2.** The mobile cloud computing infrastructure.

communicate with a cloudlet server through a proxy using a wireless communication asking to offload a certain mobile task. The offloaded task is accepted at the task queue of the nearest cloudlet server. The task offloading scheduler is conducted for making scheduling decision on tasks in the queues of the cloudlets or offloading certain tasks to the cloud, and the results will be sent back to the mobile device through wireless connection. The main objective of the task offloading scheduler is to find a mapping of the set of offloaded tasks on the queues of the available cloudlet servers to minimize the tasks completion time and to load balance the tasks on the queues of the cloudlets considering the transmission times and the load conditions on the cloudlet servers.

Assume there are $N_{ts}$ offloaded mobile tasks at the queues of the cloudlet nodes. Each task is defined as $TS_i = (TL_{TS_i}, DT_{TS_i}, RC_{TS_i})$ where $TL_{TS_i}, DT_{TS_i}$, and $RC_{TS_i}$ represent the task length, the task deadline time, and resource consumption respectively. There are $N_{cl}$ cloudlet nodes. Each cloudlet node $cl_j$ has a number of virtual machines represented as $vm_{jk}$ with maximum available resource capacity *Rjk*.

### 3.2. The MEC task offloading model

The main objective of the task offloading scheduler is to find a mapping of the offloaded set of tasks on the queues of the available cloudlets to minimize the energy consumption of the mobile devices and the tasks response time while considering the tasks deadline requirement and existing loads on the queues of the cloudlet servers.

A task total execution time, $T_{TS_i}$, is calculated as the sum of the transmission time, the waiting time at the queue, and the service time, as shown in Eq. (1).

$$T_{TS_i} = Tr_{TS_i}^{(cl_{j1},cl_{j2})} + WT_{TS_i}^{cl_j} + ST_{TS_i}^{cl_j} \tag{1}$$

where $T_{TS_i}$ is the total execution time of task $TS_i$, $Tr_{TS_i}^{(cl_{j1},cl_{j2})}$ is the transmission time of task $TS_i$ between cloudlet nodes $cl_{j1}$ and $cl_{j2}$, $WT_{TS_i}^{cl_j}$ is the waiting time of task $TS_i$ on the queue of cloudlet node $cl_j$, and $ST_{TS_i}^{cl_j}$ is the service time of task $TS_i$ on cloudlet node $cl_j$. The total execution time $T_{TS_i}$, for task $TS_i$, should satisfy the following quality of service condition:

$$T_{TS_i} < DT_{TS_i} \tag{2}$$

The transmission time $Tr_{TS_i}^{(cl_{j1},cl_{j2})}$ considers the tasks lengths and the bandwidth of the wireless transmission channel between the cloudlet servers and mobile devices, and is calculated as follows:

$$Tr_{TS_i} = LC_{cl} + \frac{L_{TS_i}}{BW_{cl}} \tag{3}$$

where $LC_{cl}$ and $BW_{cl}$ are the latency and the bandwidth of the cloudlet node respectively.

The waiting time, $WT^{cl_j}$, at the cloudlet node $cl_j$ is calculated using the queuing model $M/M/vm_{cl_j}$ for cloudlet node $cl_j$ which has $vm_{cl_j}$ virtual machines as a number of servers in the queuing system, and is calculated as follows:

$$WT^{cl_j} = \frac{\mu_{cl_j}(\lambda_{cl_j}/\mu_{cl_j})^2}{(vm_{cl_j}-1)!(vm_{cl_j}\mu_{cl_j}-\lambda_{cl_j})} \tag{4}$$

where $\lambda_{cl_j}$ and $\mu_{cl_j}$ are the arrival rate and the service rate at cloudlet node $cl_j$. The service time $ST^{cl_j}$ is calculated as follows:

$$ST^{cl_j} = \frac{1}{\mu_{cl_j}} \tag{5}$$

Finally, the main objective is to minimize the overall time of the offloaded mobile tasks, and is expressed as follows:

$$T = \sum_{TS_i=1}^{N_{ts}} \sum_{cl_j=1}^{N_{cl}} \delta_{TS_i}^{cl_j} \times T_{TS_i} \tag{6}$$

where $\delta_{TS_i}^{cl_j} = 1$ is the task offloading and allocation decision where $TS_i$ is offloaded to a cloudlet node $cl_j$, and $\delta_{TS_i}^{cl_j} = 0$ represents that task $TS_i$ is not considered for offloading to cloudlet server $cl_j$.

## 4. The proposed evolutionary algorithms for MEC task offloading

This section outlines the proposed evolutionary-based mobile offloading algorithms in an edge environment, namely genetic algorithm (GA) and particle swarm optimization (PSO), used to find a placement of a set of offloaded mobile tasks on available cloudlet nodes that minimizes tasks response time while considering the transmission cost and the loads on the cloudlet servers.

### 4.1. The GA-based task offloading

The genetic algorithm (GA) is an evolutionary algorithm based on the biological concept of generating populations of random solutions and simulating the evolution of natural selection [32]. The GA is widely used for the solution of diverse NP-complete problems, including traveling sales man problem, and cloud computing scheduling and load balancing [33]. Each possible solution is encoded as a chromosome made up from a set of genes. Each set of generated possible solutions build up a population in which the solution to the problem is selected by a certain selection mechanism.

#### 4.1.1. Chromosome encoding

Each chromosome represents a possible solution for scheduling the set of all offloaded tasks on the cloudlet servers. Each chromosome is a vector of size $N_{ts}$ that is the number of offload tasks. Each element of the chromosome is a gene numbered from 1 to $N_{ts}$ in order corresponding to the IDs of offloaded tasks, and the value of each gene represents the ID of the corresponding cloudlet placement. For example, the chromosome shown in Fig. 3 represents a possible scheduling of 5 offload mobile tasks on cloudlet servers.

#### 4.1.2. Fitness function

An individual chromosome with larger fitness value is better to survive in the population evolution. A higher fitness value represents a lower total tasks response time. Therefore, the fitness function is formulated as follows:

$$f = \frac{1}{T} \tag{7}$$

where $T$ is calculated using Eq. (6).

#### 4.1.3. Population initialization

A population is a set of individual solutions where each individual solution is encoded into a chromosome. Each chromosome represents a schedule of offloaded mobile tasks onto cloudlet nodes. Initially, a number of individual solutions are randomly generated.

#### 4.1.4. Selection

In the selection operation, individual chromosomes that have good characteristics towards the optimal solution should be

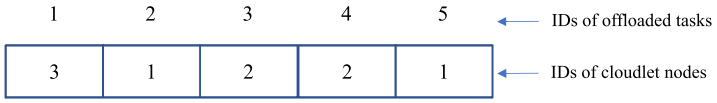


**Fig. 3.** An example of chromosome encoding.

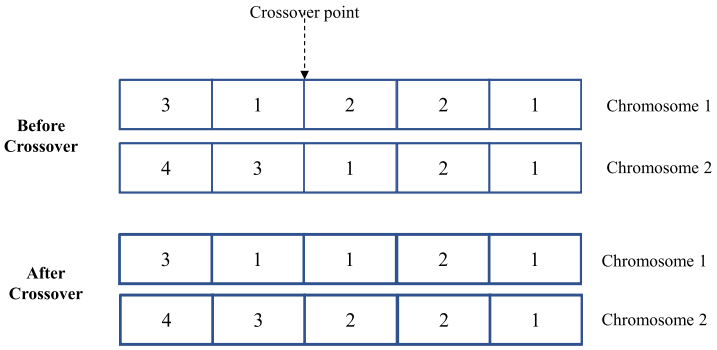


**Fig. 4.** An example of chromosome encoding.

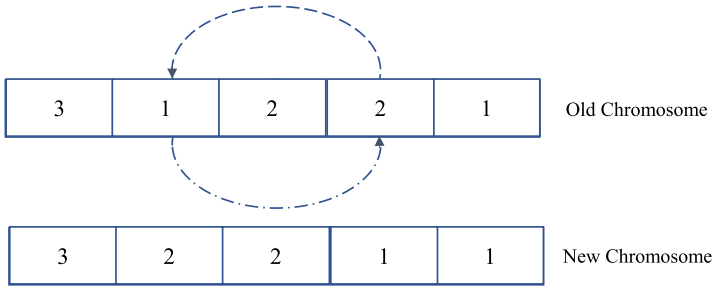


**Fig. 5.** An example of chromosome encoding.

selected for the next generation of populations. The roulette wheel method is adopted for the selection of individual chromosomes [34]. The roulette wheel method calculates the probability of each candidate solution, as shown in the following Equation:

$$P(i) = \frac{f(i)}{\sum_{i=1}^{N} f(i)} \tag{8}$$

where $f(i)$ is the fitness value of an individual chromosome $i$, and $N$ is number of individuals in the population. Then, the cumulative probabilities are calculated and sorted ascendingly. A random number $\in [0; 1]$ is created and compared to the cumulative probabilities, and the corresponding individual chromosome is selected. This ensures that the selection process selects individuals with higher fitness values and potentially useful solutions.

#### 4.1.5. Crossover

The crossover operation combines two individual chromosomes to generate a new individual. A single point cross over is applied where depending upon a randomly generated crossover point, the portion lying on one side of crossover point of a chromosome is exchanged with the same side of another chromosome. A new pair of individuals are generated, as shown in Fig. 4.

#### 4.1.6. Mutation

In the mutation process, two value of random genes are replaced with each other. Thus generate a new individual for the next generation of population, as shown in Fig. 5. The mutation introduces diversity of the generated population, and avoids that chromosomes become similar. Thus, avoid local minima which will improves the local search capability.

#### 4.1.7. The proposed GA-based offloading algorithm

The proposed algorithm commence by initializing the parameters, including number of tasks $N_{ts}$, number of cloudlet node $N_{cl}$, maximum number of iterations $N_{iter}$, number of chromosomes in the population $N_p$, probability of crossover $P_c$, and probability of mutation $P_{mu}$, as shown in Algorithm 1. A random initial population is generated, and the fitness value corresponding to each individual is calculated. The algorithm iterates for the maximum number of iterations. In each iteration, a single point crossover is applied by generating a random number, and the random number is compared with the probability of crossover $P_c$. If the random value is less than $P_c$, the crossover operation will be performed using a random cross over point. Second, a mutation operation is performed on the resulting new chromosomes between two random genes. Finally, A selection operation is performed on the population and the new individuals together, applying the roulette wheel selection. The new population is used for next round of iteration.

**Algorithm 1**: GA-based offloading algorithm.

```
Set the parameters N_ts, N_cl, N_iter, N_p, P_c, and P_mu
Randomly generate N_p chromosomes
Calculate the fitness value for particle chromosome using Equation 7
for each iter in N_iter do
    if rand() < Probabilty_c then
        Perform single point crossover by randomly selecting the
         crossover point to create new a new individuals
    end
    Apply the mutation operator on the new chromosomes as follows:
    for each gene in the chromosome i do
        if rand() < P_mu then
            gene_1 = random index
            gene_2 = random index
            Exchange genes between (gene_1 and gene_2)
        end
    end
    A new population is created using the old population and the new
     individuals by applying a roulette wheel selection
    The new population is used for next round of iteration
end
```

#### 4.1.8. Time complexity

The time complexity of the proposed genetic algorithm-based task offloading algorithm is dependent on the time complexity of the selection, the crossover, and the mutation operations. The time complexity of the selection is $O(N_{iter} \times N_p \times N_{ts})$. The time complexity of the crossover and the mutation operations are $O(N_{iter} \times N_p)$. Therefore, the overall time complexity of the proposed algorithm is $O(N_{iter} \times N_p \times N_{ts})$.

### 4.2. The PSO-based task offloading

The Particle Swarm optimization (PSO) is a bio-inspired optimization method by the behavior of bird swarm, $N_{swarm}$. Each particle, individual bird, searches for food in search space to find the best food source based on the particle position, the particle best found solution, and the swarm best found global solution [35]. For D-dimensional combinatorial problem, each particle represents a possible solution in the search space. A particle $P_i$ is expressed as D-dimensional vector $P_i = \{p_{i1}, p_{i2}, \ldots, p_{id}\}$. In the iterative search process, particle, $P_i$, modifies its new values, $P_i(t+1)$, by updating its corresponding velocity, $V_i(t+1)$ which determines the particle motion speed, as shown in Eqs. (9) and (10).

$$V_i(t+1) = \omega \times V_i(t) + C_1 \times \alpha \times (Pl_i - P_i) + C_2 \times \beta \times (Pg - P_i(t)) \tag{9}$$

$$P_i(t+1) = P_i(t) + V_i(t+1) \tag{10}$$

where $i$ is the particle number, and $t$ is the iteration number. $\omega$ is inertia weight. $C_1$ and $C_2$ are cognitive and social coefficients. $\alpha$ and $\beta$ are random numbers $\in N[0,1]$. The PSO is designed and proposed for optimization problems in continuous domain. The mobile task-cloudlet placement is a discrete problem where the search space is discrete. Therefore, a discrete version of the PSO algorithm must be adopted to suit the problem under study [36].

#### 4.2.1. The discrete PSO-based offloading algorithm

The particle encoding $p_{ij}$ represents a placement decision for an offloaded mobile task $j$ assigned to a certain cloudlet. The IDs of the offloaded mobile tasks and the cloudlet servers are discrete. The particle encoding is similar to the encoding shown in Fig. 3. In each iteration, the PSO-based offloading algorithm updates the particle velocity and its corresponding position using Eqs. (9) and (10). The continuous values of particles position are converted to discrete values, using the following equation:

$$p_{ij} = \begin{cases} \lfloor |p_{ij}| \rfloor & \text{if } 0 > p_{ij} \leqslant N_{cl} \\ \lfloor |p_{ij}| \rfloor \% N_{cl} & \text{otherwise} \end{cases} \quad (11)$$

The details of the proposed PSO-based offloading algorithm is presented in Algorithm 2. Firstly, the PSO-based offloading algorithm initializes the parameters, including $\alpha, \beta$, and $\omega$, the mutation probability $P_{mu}$, and number of iterations $N_{iter}$. Further, the PSO-based offloading algorithm initializes a random population of particles $P_i$ and corresponding velocities $V_i$. The fitness of each particle is calculated using Eq. (7), and the found local best of each particle and the found global best particle of the swarm are set. For certain number of iteration $N_{iter}$, the particle velocity and position are calculated using Eqs. (9) and (10). The continuous values of particles positions are converted into discrete values using Eq. (11). The PSO-based offloading algorithm updates the local and global best solutions. Afterwards, the mutation process is performed on the particles swarm.

**Algorithm 2**: The proposed PSO-based mobile offloading algorithm.

```
Initialize N_swarm, N_iter, α, β, ω, and P_mu
Random initialize P_i with random task allocation, and V_i
Calculate the fitness of each possible solution P_i using Equation 7
Set Pl_i for each particle P_i
Find P_g of the population
for each iter in N_iter do
    for each P_i in N_swarm do
        Calculate the velocity V_i(t) using Equations 9
        Calculate the position P_i(t) using Equations 10
        Convert P_i values to discrete particle using Equation 11
        Calculate the fitness function P_i using Equation 7
        if fitness(P_i) ¡ Pl_i then
            Update Pl_i = P_i
        end
        if Pl_i < Pg then
            Update Pg = Pl_i
        end
        Apply the mutation on P_i as follows:
        for each p_ij in the particle P_i do
            if rand() < P_mu then
                node_k = random edge index
                Mutate(node_k, node_j)
            end
        end
    end
end
```

**Table 2**
Parameter settings of the evaluation experiments

| Parameter name | Value |
|---|---|
| Bandwidth | $U[100-1000]KB$ |
| Task length | $U[100-1000]KB$ |
| Latency | $U[0.002-0.02]ms$ |
| Arrival rate $\lambda$ | $U[1-40]$ |
| Service rate | 1/20 |
| Crossover probability $P_c$ | 0.8 |
| Mutation probability $P_{mu}$ | 0.5 |
| Population size $N_p$ | 100 |
| Maximum iteration $max_{iter}$ | 50 |
| $\omega$ | 2 |
| $C_1$ | 2 |
| $C_2$ | 2 |
| $\alpha$ | U[0–1] |
| $\beta$ | U[0–1] |

#### 4.2.2. Time complexity

The time complexity of the proposed PSO-based offloading algorithm is dependent on: (1) the initialization operation of particles and velocity, (2) the updating of particles and velocity. The time complexity of initializing the population of particles is $O(N_{ts} \times N_{swarm})$. The time complexity of the updating operation is $O(N_{iter} \times N_{ts} \times N_{swarm})$. Therefore, the overall complexity of the proposed PSO-based task offloading algorithm is $O(N_{iter} \times N_{swarm} \times N_{ts})$.

## 5. Experimental evaluation

This section presents the experimental evaluation of the proposed mobile tasks offloaded strategy in a mobile edge computing environment. The objective of the proposed algorithms is to minimize offloaded mobile tasks response time considering the trans-

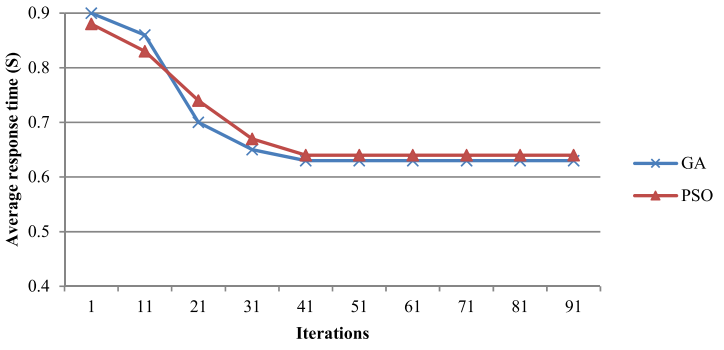


**Fig. 6.** The objective function for a single run of the proposed offloading algorithms.

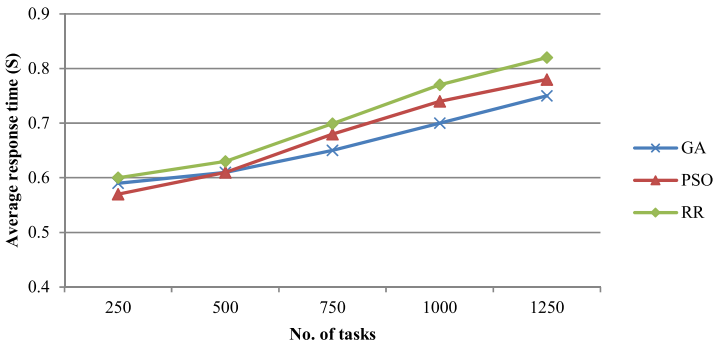


**Fig. 7.** The average response time resulting from increasing the number of tasks.

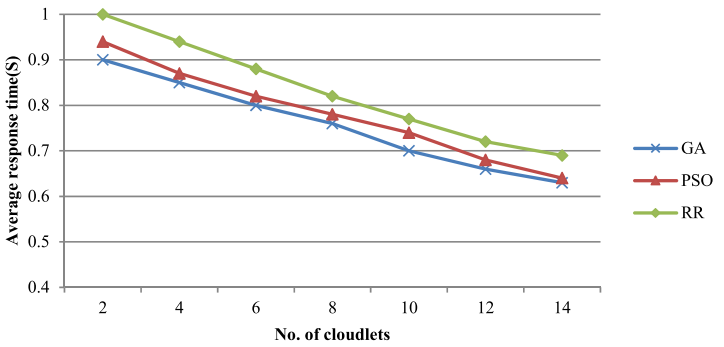


**Fig. 8.** The average response time resulting from increasing the number of cloudlets.

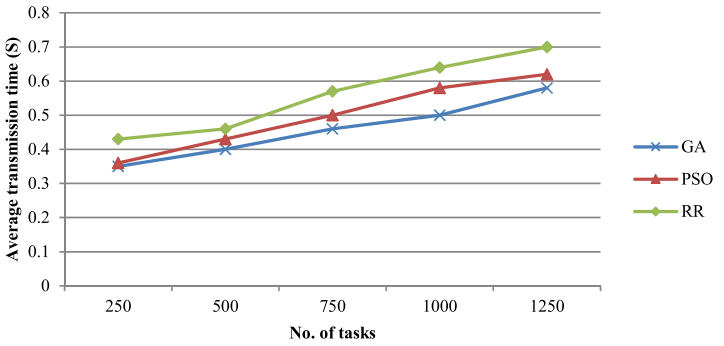


**Fig. 9.** The average transmission time resulting from increasing the number of tasks.

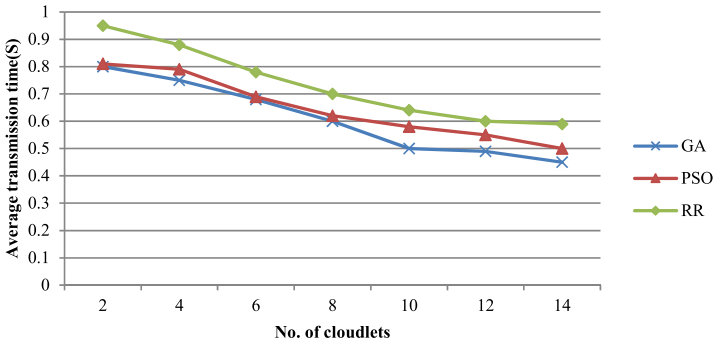


**Fig. 10.** The average transmission time resulting from increasing the number of cloudlets.

mission time and the load conditions on the cloudlet nodes. The parameter settings of the proposed offloading strategy are presented in Table 2.

The experiment analysis is conducted in terms of evaluating both tasks response time and power consumption. The results of the proposed GA-based offloading algorithm is compared to the PSO-based offloading algorithm as well as to the conventional round robin (RR) algorithm. The power consumption of a mobile device as a result of the transmission is calculated using Eq. (12).

$$P_{trans} = P_{wifi} \sum_{TS_i} \sum_{cl_j} Tr_{TS_i} \tag{12}$$

where $P_{wifi}$ is the power consumed for wireless transmission per second. The power consumption of a mobile device as a result of executing the task locally without offloading is calculated using Eq. (13).

$$P_{local} = \frac{L_{TS_i}}{f_c} p_c \tag{13}$$

where $f_c, p_c$ and $L_{TS_i}$ are frequency capacity of mobile device, energy of processing, and size of the task respectively.

For the evaluation in terms of the response time, Fig. 6 shows the output of the objective function for both of the proposed algorithms, GA-based and PSO-based offloading algorithms. It is clear that the GA-based algorithm converge faster than the PSO-based algorithm, and able to find a better solution.

Fig. 7 shows the average response time as increasing the number of tasks using 10 cloudlets for the offloading using the GA-based, and the PSO-based, as well as the RR offloading algorithms. The offloading strategy using the RR achieves highest average response time compared with the proposed evolutionary algorithms. The offloading strategy using the GA-based algorithm effectively reduces the response time.

Fig. 8 shows the average response time as increasing the number of cloudlets using 1000 tasks. The offloading strategy using the GA algorithm is effectively able to reduce the response time.

For the evaluation in terms of the transmission time, Fig. 9 shows the average transmission time as increasing the number of tasks using 10 cloudlets. The offloading strategy using the RR achieves highest average transmission time compared with the proposed evolutionary algorithms. The GA-based offloading algorithm effectively reduces the transmission time. Fig. 10 shows the average transmission time as increasing the number of cloudlets using 1000 tasks. The GA-based offloading algorithm is able to effectively reduce the transmission time compared to the other algorithms.

For the evaluation in terms of the power consumption resulting from transmission, Fig. 11 shows the average power consumption as increasing the number of tasks using 10 cloudlets for the offloading using the GA-based, the PSO-based, and the RR offloading algorithms. The offloading strategy using the RR achieves highest average power consumption compared the proposed evolutionary algorithms. The GA-based offloading algorithm effectively reduces the power consumption. Further, Fig. 12 shows the power consumption time as increasing the number of cloudlets using 1000 tasks for the offloading algorithm the GA-based, the PSO-based, and the RR. It can be concluded that the offloading strategy using the GA-based algorithm effectively reduces the power consumption. For the evaluation in terms of the power consumption resulting from processing, Fig. 13 shows a comparison of the power consumption of the mobile device using proposed GA-based task offloading against without offloading. The proposed GA-based task offloading algorithm effectively saves the energy of the mobile devices.

Table 3 shows a statistical analysis using the t-test which demonstrates the significance level of the proposed GA-based task offloading algorithm compared with the PSO-based task offloading

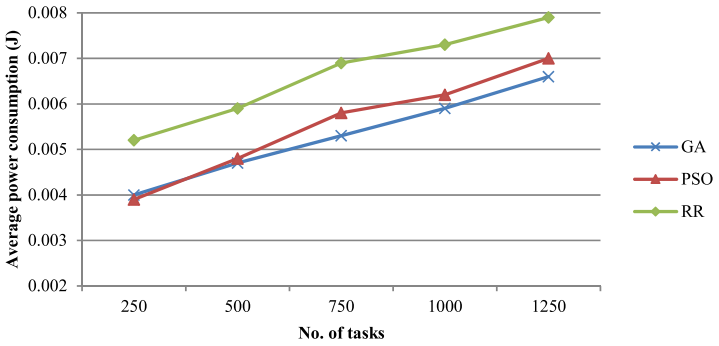


**Fig. 11.** The power consumption resulting from increasing the number of tasks.

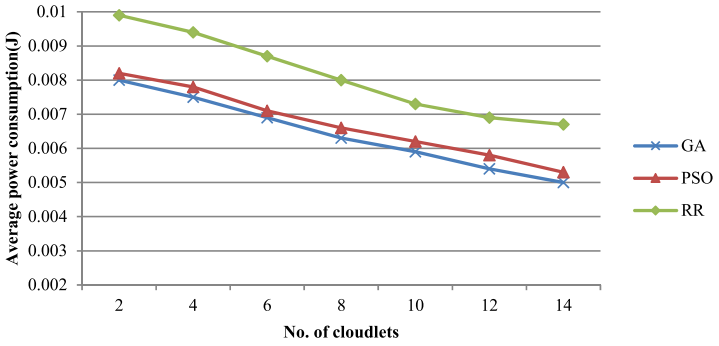


**Fig. 12.** The power consumption resulting from increasing the number of cloudlets.

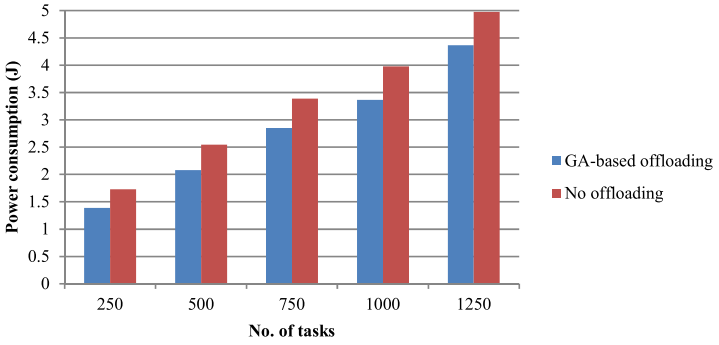


**Fig. 13.** The power consumption without offloading against GA-based offloading.

**Table 3**
Statistical analysis using t-test.

| Criteria | Algorithm | Mean | Standard deviation | Degree of Freedom | t-value | p-value |
|---|---|---|---|---|---|---|
| Response time | GA-based | 0.63464 | 0.007 | 29 | 6.4 | 0.000001 |
| | PSO-based | 0.64724 | 0.008 | | | |
| Power consumption | No offloading | 1.81 | 0.166 | 29 | 5.67 | 0.000004 |
| | GA-based | 1.56 | 0.171 | | | |

algorithm in terms of response time for 1000 tasks using 15 cloudlets. Using the t-test with a significance level $\leq 0.05$, the null hypothesis is rejected, and the proposed GA-based task offloading algorithm significantly improves the response time. In terms of power consumption, a statistical analysis using the t-test is performed to compare the significance without offloading against offloading using the proposed GA-based task offloading algorithm. For a significance level $\leqslant 0.05$, the null hypothesis is rejected, and the proposed GA-based task offloading algorithm significantly improves the power consumption of the mobile devices.

## 6. Conclusion and future work

This paper presented an effective mobile task offloading strategy in a mobile edge environment. A formal model is outlined that

aims to minimize the response time of offloaded tasks considering the transmission time and the load conditions on the cloudlet servers. A mobile task offloading algorithm is proposed based using the queuing networks and the genetics algorithm (GA). The proposed GA-based offloading algorithm is compared to round robin-based offloading and another evolutionary algorithm, namely the particle swarm optimization (PSO). The experimental evaluations show that the proposed GA-based mobile task offloading algorithm is efficient in comparison to the PSO-based offloading algorithm and the round robin offloading algorithm. In the future work, a multi-objective function, including response time, energy consumption, and cost, for mobile task offloading in mobile edge environment will be investigated. Further, a task offloading for workflows using different evolutionary algorithms, such as grey wolf optimization, cuckoo search, and sun flower optimization will be investigated.

## Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## References


[1] Abbas N, Zhang Y, Taherkordi A, Skeie T. Mobile edge computing: a survey. IEEE Internet Things J 2018;5(1):450–65.

[2] Fakhfakh F. Performance and correctness of mobile cloud computing systems: taxonomy and open challenges. In: 2019 15th International Wireless Communications Mobile Computing Conference (IWCMC), 2019. pp. 1019–1024.https://doi.org/10.1109/IWCMC.2019.8766588..

[3] Dinh HT, Lee C, Niyato D, Wang P. A survey of mobile cloud computing: architecture, applications, and approaches. Wireless Commun Mobile Comput 2013;13(18):1587–611. doi: https://doi.org/10.1002/wcm.1203.

[4] Hussein M, Mousa M-H. Quality of service management for service oriented applications in a cloud architecture. Int J P2P Network Trends Technol 2016;6 (6):12–9. URL: http://www.ijpttjournal.org/archives/ijptt-v30p401.

[5] Ahmed E, Gani A, Sookhak M, Hamid SHA, Xia F. Application optimization in mobile cloud computing. J Netw Comput Appl 2015;52(C):52–68. doi: https://doi.org/10.1016/j.jnca.2015.02.003.

[6] Shiraz M, Gani A, Khokhar RH, Buyya R. A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. IEEE Commun Surveys Tutor 2013;15(3):1294–313. doi: https://doi.org/10.1109/SURV.2012.111412.00045.

[7] Ren J, Zhang Y, Zhang K, Shen X. Exploiting mobile crowdsourcing for pervasive cloud services: challenges and solutions. IEEE Commun Mag 2015;53 (3):98–105. doi: https://doi.org/10.1109/MCOM.2015.7060488.

[8] Satyanarayanan M, Bahl P, Caceres R, Davies N. The case for vm-based cloudlets in mobile computing. IEEE Pervasive Comput 2009;8(4):14–23. doi: https://doi.org/10.1109/MPRV.2009.82.

[9] Hao P, Bai Y, Zhang X, Zhang Y. Edgecourier: an edge-hosted personal service for low-bandwidth document synchronization in mobile cloud storage services. In: Proceedings of the Second ACM/IEEE Symposium on Edge Computing, SEC '17. ACM: New York, NY, USA; 2017. pp. 7:1–7:14. https://doi.org/10.1145/3132211.3134447..

[10] Jararweh Y, Tawalbeh L, Ababneh F, Dosari F. Resource efficient mobile computing using cloudlet infrastructure, in. In: 2013 IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks. p. 373–7. doi: https://doi.org/10.1109/MSN.2013.75.

[11] Ghobaei-Arani M, Souri A, Rahmanian AA. Resource management approaches in fog computing: a comprehensive review. J. Grid Comput. 2020;18(1):1–42. doi: https://doi.org/10.1007/s10723-019-09491-1.

[12] Yousefpour A, Fung C, Nguyen T, Kadiyala K, Jalali F, Niakanlahiji A, Kong J, Jue JP. All one needs to know about fog computing and related edge computing paradigms: A complete survey. J. Syst. Arch. 2019;98:289–330. doi: https://doi.org/10.1016/j.sysarc.2019.02.009. URL: http://www.sciencedirect.com/science/article/pii/S1383762118306349.

[13] Wang S, Zhang X, Zhang Y, Wang L, Yang J, Wang W. A survey on mobile edge networks: convergence of computing, caching and communications. IEEE Access 2017;5:6757–79.

[14] Roman R, Lopez J, Mambo M. Mobile edge computing, fog et al.: a survey and analysis of security threats and challenges. Future Gen. Comput. Syst. 2018;78:680–98. doi: https://doi.org/10.1016/j.future.2016.11.009.

[15] Ahmed A, Ahmed E. A survey on mobile edge computing. In: 2016 10th International Conference on Intelligent Systems and Control (ISCO); 2016. pp. 1–8..

[16] Shakarami A, Shahidinejad A, Ghobaei-Arani M. A review on the computation offloading approaches in mobile edge computing: a game-theoretic perspective. Software: Practice and Experience 50(9); 2020: 1719–1759. https://doi.org/10.1002/spe.2839..

[17] Satyanarayanan M, Bahl P, Caceres R, Davies N. The case for vm-based cloudlets in mobile computing. IEEE Pervasive Comput. 2009;8(4):14–23. doi: https://doi.org/10.1109/MPRV.2009.82.

[18] Fesehaye D, Gao Y, Nahrstedt K, Wang G. Impact of cloudlets on interactive mobile cloud applications. In: 2012 IEEE 16th International Enterprise Distributed Object Computing Conference. p. 123–32. doi: https://doi.org/10.1109/EDOC.2012.23.

[19] Soyata T, Muraleedharan R, Funai C, Kwon M, Heinzelman W. Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In: 2012 IEEE Symposium on Computers and Communications (ISCC); 2012. pp. 000059–000066.https://doi.org/10.1109/ISCC.2012.6249269..

[20] Hoang DT, Niyato D, Wang P. Optimal admission control policy for mobile cloud computing hotspot with cloudlet. In: 2012 IEEE wireless communications and networking conference (WCNC); 2012. pp. 3145–3149. https://doi.org/10.1109/WCNC.2012.6214347..

[21] Wang T, Wei X, Liang T, Fan J. Dynamic tasks scheduling based on weighted bi-graph in mobile cloud computing. Sustain Comput: Inf Syst 2018;19:214–22. doi: https://doi.org/10.1016/j.suscom.2018.05.004. URL: http://www.sciencedirect.com/science/article/pii/S2210537917304560.

[22] Ghobaei-Arani M, Souri A, Safara F, Norouzi M. An efficient task scheduling approach using moth-flame optimization algorithm for cyber-physical system applications in fog computing. Trans. Emerg. Telecommun. Technol. 2020;31 (2):. doi: https://doi.org/10.1002/ett.3770e3770.

[23] Donyagard Vahed N, Ghobaei-Arani M, Souri A. Multiobjective virtual machine placement mechanisms using nature-inspired metaheuristic algorithms in cloud environments: a comprehensive review. Int J Commun Syst 32(14); 2019: e4068, e4068 IJCS-19-0062.R1.https://doi.org/10.1002/dac.4068..

[24] Hussein MK, Mousa MH, Alqarni MA. A placement architecture for a container as a service (caas) in a cloud environment. J Cloud Comput 2019;8(1):7. doi: https://doi.org/10.1186/s13677-019-0131-1.

[25] Mehta S, Kaur P. Efficient computation offloading in mobile cloud computing with nature-inspired algorithms. Int J Comput Intell Appl 18(04); 2019: 1950023.https://doi.org/10.1142/S1469026819500238..

[26] Kaur P, Mehta S. Efficient computation offloading using grey wolf optimization algorithm. AIP Conf Proc 2019;2061(1). doi: https://doi.org/10.1063/1.5086633. 020011.

[27] Wei X, Fan J, Lu Z, Ding K. Application scheduling in mobile cloud computing with load balancing. J Appl Math 13.https://doi.org/10.1155/2013/409539..

[28] Sundararaj V. Optimal task assignment in mobile cloud computing by queue based ant-bee algorithm. Wireless Personal Commun 2019;104(1):173–97. doi: https://doi.org/10.1007/s11277-018-6014-9.

[29] Li Z, Zhu Q. Genetic algorithm-based optimization of offloading and resource allocation in mobile-edge computing. Information 11(2).https://doi.org/10.3390/info11020083..

[30] Cui Y, Zhang D, Zhang T, Chen L, Piao M, Zhu H. Novel method of mobile edge computation offloading based on evolutionary game strategy for iot devices. AEU Int J Electron Commun 2020;118:. doi: https://doi.org/10.1016/j.aeue.2020.153134. URL: http://www.sciencedirect.com/science/article/pii/S1434841119331000153134.

[31] Hussein MK, Mousa MH. Efficient task offloading for iot-based applications in fog computing using ant colony optimization. IEEE Access 2020;8:37191–201.

[32] Holland JH. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence. Cambridge, MA, USA: MIT Press; 1992.

[33] Zhou Z, Li F, Zhu H, Xie H, Abawajy JH, Chowdhury MU. An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments. Neural Comput Appl.https://doi.org/10.1007/s00521-019-04119-7..

[34] Lipowski A, Lipowska D. Roulette-wheel selection via stochastic acceptance. Physica A 2012;391(6):2193–6. doi: https://doi.org/10.1016/j.physa.2011.12.004. URL: http://www.sciencedirect.com/science/article/pii/S0378437111009010.

[35] Kennedy J, Eberhart R. Particle swarm optimization. In: Proceedings of ICNN'95 – International Conference on Neural Networks, vol. 4; 1995. pp. 1942–1948.https://doi.org/10.1109/ICNN.1995.488968..

[36] Clerc M. Discrete particle swarm optimization, illustrated by the traveling salesman problem. Springer Berlin Heidelberg: Berlin, Heidelberg; 2004. pp. 219–239.https://doi.org/10.1007/978-3-540-39930-8_8..