WEIGHTED QUANTUM PARTICLE SWARM DISPATCHED TASK OFFLOADING OPTIMIZATION ALGORITHM FOR TIME-ENERGY MINIMIZATION IN MOBILE EDGE COMPUTING

ABSTRACT

Despite numerous advantages brought forward by edge computing, task offloading continues to present difficulties due to its dynamic nature of configuration as well the resources required. Besides, significant portion of prior literature tried to proffer solutions to the aforementioned concerns but yet, neglect optimal selection and task dependency considerations. Therefore, in this paper, firstly, we formulated a task offloading problem for reducing energy consumption and task completion time as a multi-objective function. We then propose a Weighted Quantum Particle Swarm Optimization and Dispatched (WQPSOD) task offloading optimisation algorithm for reducing completion time and energy consumption in mobile edge computing. Implementation of the proposed WQPSOD algorithm is carried out on Python programming environment with configured multiuser multi-saver edge computing environment and evaluated with metrics of completion time and energy consumption. Experimental results via simulation shows our proposed WQPSOD task offloading algorithm can achieve remarkable performance in terms of reducing the total completion time and energy consumption when compared with the benchmarked algorithms.

Keywords: *Edge Computing, Energy Consumption, Task offloading, Weighted Quantum Particle Swarm Dispatched*

1. Introduction

The development of mobile devices provides enabling technologies to support a variety of mobile applications, such as augmented reality, face recognition, social networks, gaming, and health monitoring systems[1], [2]. These mobile applications, on the other hand, necessitate High-Performance Computing environments with minimal latency, energy consumption, and completion time they may also produce delay-sensitive and computationally intensive tasks, which increases in higher energy consumption.[3]. However, mobile devices are constrained by their computing, storage, and battery capacities. These constraints make it challenging to keep up with the rapid growth of mobile computing tasks, resulting in recurrent delays, fails, and wasted batteries, which negatively impact the user experience[4]. In response to this challenge, mobile edge computing has emerged as a promising computing paradigm for handling workloads offloaded from mobile devices. Mobile edge computing plays a critical role in ensuring Quality of Service (QoS) and enhancing the overall excellence of such applications[5]. Despite mobile edge computing is a promising solution to the high-intensity computational task for mobile devices, several practical challenges related to this technique must be addressed before it can be extensively adopted[6]. Therefore, Task offloading has a favourable impact on increasing resource utilisation by minimising energy consumption, properly balancing the workload to overcome network overhead, and improving system performance. Despite the many advantages of edge computing, task offloading continues to present difficulties due to its dynamic nature of configuration, as well as the resources required[7], [8]. As the number of mobile devices increases, the energy consumption in the mobile edge computing environment also rises[9]. Some researchers used meta-heuristic methods to develop near-optimal solutions for approximate optimisation. Several well-known metaheuristics are used in task offloading, including grey wolf optimisation (GWO), Genetic algorithm (GA), ant colony optimisation (ACO), and simulated annealing (SA) [10],[11][12]. However, due to their low convergence in late iteration in terms of efficient energy consumption with the usage of task offloading approach, all of the above metaheuristic algorithms fall into the local optimum search solution.

Given this viewpoint, the research focuses on the uses of weighted quantum particle swarm optimization and dispatched algorithm (WQPSOD) task offloading optimisation because of its effective solutions to avoid falling into the local optimal solution and consider task dependency based on their calculation amount. Moreover, a significant portion of prior literature has concentrated on either single-user multi-saver or multi-saver single-user environment. Only a limited number of studies have addressed multi-user multi-saver situations, yet they often neglect optimal selection and task dependency considerations.

We suggest an improved energy optimisation technique for mobile edge computing in this paper by addressing some of the key difficulties raised previously. The following are the main contributions of our study:

i. We formulated a task Offloading problem for reducing energy consumption and task completion time as a multi-objective function.
ii. We proposed Weighted Quantum Particle Swarm Optimization and Dispatched (WQPSOD) task offloading optimisation algorithm for reducing completion time and energy consumption in edge computing.
iii. Implementation of the proposed algorithm and comparison with other benchmarked approaches.

The rest of the paper is organized as follow: Section 2 discuss the review on related work. System model is discussed in Section 3. Section 4 is centred on the problem description. Proposed approach is discussed in Section 5. Section 6 discusses the experimental setup. Results discussion in Section 7 while Section 8 wrap up the conclusion of the manuscript.

2. Related Work.

The mobile edge computing base offloading technique needs the combined allocation of communication and compute resources between mobile devices and edge servers to minimise system energy consumption and completion time. There have been numerous recent works on this problem for multi-user, multi-saver environments. Kuang et al [13] addresses the issue of partial offloading scheduling and resource allocation in mobile edge computing systems with numerous independent tasks. The goal is to reduce the weighted overall execution delay and energy consumption while maintaining the task's transmission power limitations. Zhang et al. [14] presents a parallel task offloading model and a small area-based edge offloading scheme in Mobile Edge Computing (MEC) to overcome the issues provided by device mobility and server status unpredictability in multi-server and multi-task scenarios. The authors develop an optimisation problem to minimise energy consumption and job completion time, then use a Markov decision strategy to transform it into a deep reinforcement learning-based offloading scheme. Bozor .et al [15] investigates the task offloading problem in mobile edge computing (MEC) systems, with the goal of minimising both energy usage and task processing time for mobile devices. They offer a constrained multi-objective optimisation problem (CMOP) model and an evolutionary algorithm to identify the Pareto-optimal front, which reflects the optimum trade-offs between energy consumption and task processing latency. XU et al. [15]examines the challenge of jointly optimising computation offloading, data compression, and resource allocation in a multiuser mobile-edge computing (MEC) system with delay limitations and finite MEC computation resources. The aim is to minimise energy use while achieving the delay requirement. Abbas et al [12]tackles the difficulty of performing computation-intensive tasks on mobile devices with limited energy and processing capability, which results in significant computation latency or high energy consumption. The goal of this study is to determine the best offloading tasks in mobile edge computing to minimise reaction time and energy consumption. Xu et al. [16] provides a compute offloading approach that combines a genetic algorithm and an ant colony algorithm to overcome the limitation of single heuristic algorithms and enhance system efficiency by minimising energy consumption. Qian et al [17]

developed of a task offloading method for a resource-constrained multi-user and multi-MEC system in an industrial internet of things context, the author introduces a particle swarm optimisation (PSO)-based task offloading approach that takes into account time delay, energy consumption, and task execution cost. Wen [18] provides a multi-stage particle swarm optimisation (MPSO)-based offloading strategy for vehicular edge computing (VEC) that minimises computing and resource offloading costs while completing computational tasks within given time constraints. Li [19] suggests a computation offloading strategy known as EIPSO based on a modified particle swarm optimisation (PSO) algorithm. It aims to assign computation tasks to MEC servers in a way that minimises delay while balancing energy consumption, resulting in improved system cost and load distribution. Li, Aichuan [20] presents a compute offloading technique for edge computing utilising an upgraded Particle Swarm Optimisation (PSO) algorithm, the technique intends to reduce energy consumption and time delay in the system's task allocation operations. It is compared to two other algorithms in terms of average time delay and energy consumption under various circumstances. Huynh [21] studies a two-tier computation-offloading technique for multi-user multi-MEC servers in heterogeneous networks, with the goal of reducing mobile device (MD) computing overhead in terms of completion time and energy usage. The authors offer a low-complexity algorithm based on particle swarm optimisation that guarantees convergence and gives superior solutions to the optimisation issue.

3. System Model

Figure 1 shows the system concept for a multi-user, multi-server task offloading situation. The scenario includes several devices performing a variety of delay-sensitive and computationally intensive tasks. In a manufacturing environment, where extensive data processing is necessary, algorithms in a multi-user, multi-server context with dispatched algorithms will establish two queues. These queues regulate how tasks are routed, either to MEC (Mobile Edge Computing) offloading or local offloading, based on specific criteria. Queue 1 is intended for tasks that demand large computational resources, while Queue 0 is for tasks with smaller computational needs. The edge server processes the tasks and returns the calculation results to the user after completing the computations.

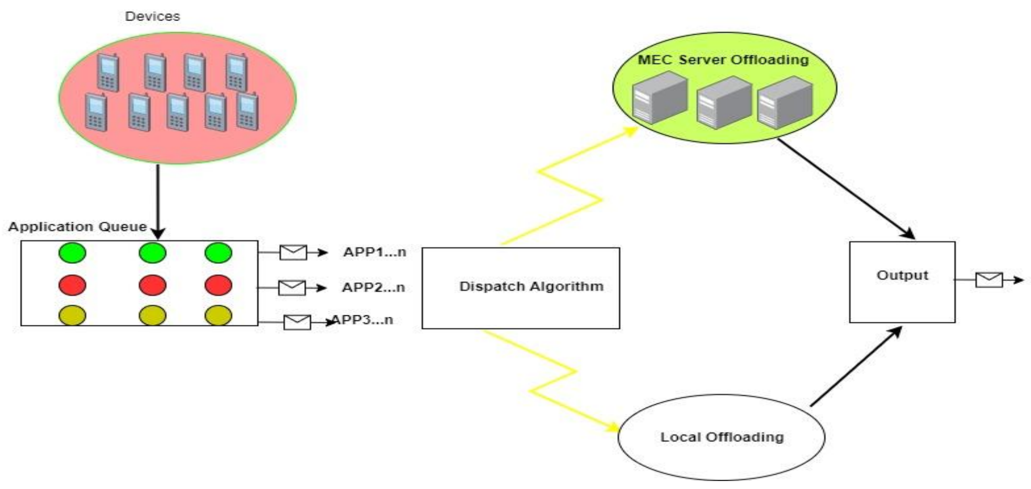

Figure 1: System Model

4. Problem Description

The task offloading method discussed in this article is adapted to multiuser and multi-server environment, assuming that the existing set of mobile devices which is v =(1, 2,…v) and S is donated the number of mobile edge computing saver in the network. Each offloaded task is handled by a MEC server. suppose the offloading task is a set N, represented as N = {1, 2, ⋯, N}. And then the probability of offloading results is Q + 1, which is Q ∈ (0,1), where Q = 0 indicates that the offloading task can be completed locally on mobile devices and Q = 1 indicates that the offloading task is executed in the edge server. Two task queues, namely Q0 and Q1, have been introduced. Q0 is designated for tasks to be executed on mobile devices' terminals, while Q1 is reserved for tasks to be executed at the edge server. The direction edge (XK_i, XK_j,) indicates the dependency between the i-th and j-th tasks in the k-th application. This means that task xk_j should be done after task xk_i is completed. A Directed Acyclic Graph (DAG) can illustrate the dependency between tasks within an application. Tasks are assigned to Q0 and Q1 using dispatched algorithms. These algorithms evaluate each task along with its dependencies to determine whether it should be executed locally or on the edge server. This process ensures that tasks requiring a substantial amount of computation are executed without encountering unnecessary delay Figure 1 illustrate the system paradigm of a task wisth dependency, multiuser, multi-sever task offloading environment. Some of the proposed services demand minimal latency and a large calculation amount. Therefore, based on the set N, the algorithm will create a queue. The queue will prioritise the complex task execution based on the higher calculation amount of each computing task; the algorithm arranges the queue in descending order. This indicates that tasks with bigger calculation amounts will be prioritised during the offloading process. To perform task offloading, first model the k users, and this can be expressed as represented by a binary (H_N, R_N), and N is an integer in the $(1, n)$ range. Consider the nth user's task and its task parameter $[H_n, R_n]$, where H_n is the computation's data size (bits) and R_n indicates the amount of CPU processing cycles needed for every 1 bit of computation. Q_n are the offloading decisions, and Q_n ∈[0, 1], means that the task of Q_n part has been allocated to the MEC server for the execution, while the rest of the $(1 - Q_n)$ task is carried out locally.

4.1. Computational Model

In the mobile edge computing task offloading model, user tasks with minimal requirements can be completed locally, while user tasks with high calculation requirements run on an edge computing server. As a result, we separate them into two distinct groups: local computing models and MEC server computing models.

4.2. Local computational model

Local computing costs generally fall into two categories: time (delay) and energy consumption (power) [22]. The computational capacity of the kth user's mobile terminal device is expressed as T_n, that is the CPU's processing power. As previously demonstrated, the number of tasks performed locally by the nth user is $H_n(1 - Q_n)$. Consequence, the user's time consumed locally, which is $RT_l(Q_n)$, is determined by the following formula:

$$RT_l(Q_n) = \frac{H_n(1-R_n)}{T_n} \tag{1}$$

The following is the energy spent throughout the computation process:

$$AT_l(Q_n) = L_c T_n^2 H_n(1 - Q_n) \tag{2}$$

where $L_c$is a variable representing the power usage component and $L_c T_n^2$ is the energy consumption of the mobile device's CPU running for a single cycle, adding first and second equation, the weighted waste caused by the nth user's mobile devices is.

$$DT_l\,(Q_n) = \; y_{nt}\, RT_l\,(Q_{n)} + y_{ne}\, AT\,(Q_n) \tag{3}$$

where y_{nt} and y_{ne} are weighted factors of local time cost and energy consumption that can be changed based on real demand. Where users have substantial actual time requirements, they may increase the y_{nt} number to satisfy the current actual requirements. When the equipment's power is low, y_{ne} this may be modified to provide the appropriate offloading technique to conserve as much electricity as feasible.

4.3. Edge saver computational Model

The most significant costs of MEC edge computing servers are time (delay) and energy consumption (power). The time cost is separated into two components: uplink transmission delay across the mobile device and the access point, and job completion delay at the MEC server. The energy consumption component consists primarily of transmission and calculation energy use.

To begin determining the time cost, we describe the transmission rate from user mobile device x to access point which is as

$$W_x = Blog_2\,(1 + \frac{RQ_x \times DV_{x,m}}{H_0 \times P}) \tag{4}$$

at which $x \in \{1, 2, \dots, n\}$ P is the channel of communication capacity among the user's devices $x\ and, AP\ m$, RQ_x is the transmission power, and H_0 denotes strength of noise spectral density of M. The amount of channel acquire among x and m is determined as $DV_{x,m} = \left(H_{x,m}\right)^{-\alpha}$ where by $H_{x,m}$ is the distance that exists among x and m and α is the channel's lost factor.

Hence, the overall time cost includes the total of uplink transmission time and MEC server task execution time. The number of tasks from user mobile devices that will be offloaded on the MEC server is $H_x \times Q_n$. The uplink transmission time overhead RB_{e1} will be determined as.

$$RT_{e1} = \frac{H_x Q_n}{W_x} \tag{5}$$

From the MEC server, the task processing time cost (delay) RT_{e2} was.

$$RT_{e2} = \frac{H_x Q_x C_x}{T_x} \tag{6}$$

in which Ru denotes the amount of CPU cycles needed to calculate a single bit of task data and fu denotes the server CPU frequency the total time cost RT_{total} is obtained by combining equation six and seven.

$$RT_{etotal} = RT_{e1} + RT_{e2} = \frac{H_x Q_x}{W_x} + \frac{H_x Q_x C_x}{T_x} \tag{7}$$

The processing energy cost of a task on the MEC server can be expressed as the total of uploading and computing energy usage $AT_e(Q_x)$ and is calculated as

$$AT_e\,(Q_x) = RQ_x \times \frac{H_x Q_x}{W_x} + H_x Q_x e_m \tag{8}$$

where e_m is the amount of energy required to calculate a single bit of task data for the MEC server.

Furthermore, we consider some tasks that can be decomposed where part of the task will be executed by the mobile device and other part of it will be uploaded to the MEC server for execution. In this scenario, data sharing may exist between the mobile device and MEC server in executing the said task. Hence the final delay (task execution time) and the final energy consumption of the task consists of delay at the mobile device and the delay at the MEC server as well as the energy consumed at the mobile device and the energy consumed at the MEC server as presented in Equation 9 and Equation.10 respectively.

$$RT_{final} = RT_l(Q_n) + RT_{total} \tag{9}$$

Where $RT_{final} = RT_l(Q_n)$ if all tasks are executed by the mobile device

$$AT_{final} = AT_e(Q_n) + AT_e(Q_x) \tag{10}$$

Where $AT_{final} = AT_i(Q_n)$ if all tasks are executed by edge saver

4.4. Task Dependence Model

Task Dependence Model. In this work, task dependency is referred to as input dependency, meaning that certain tasks rely on the results of other tasks for execution. The task dependency can be represented as a directed acyclic graph, with each node indicating a computing task. A specific task could have many precursor and successor tasks. When a task does not have a successor, it becomes the last task and executes the full service once completed. Every successor task has to wait for all precursor tasks to complete in simultaneously resulting in an execution time equal to the total of all predecessor tasks' finish times. The moment when all task begin to execute is denoted as $ST = \{st_0, st_1, \dots, st_{n-1}\}$ The initial node executes at 0. If all tasks' completion times are given as $CT = \{ct_0, ct_1, \dots, ct_{n-1}\}$ task k completion time can be calculated using the formula:

$$CT_k = ST_k + T_k \tag{11}$$

T_k represents the execution delay of task k and is determined as:

$$T_k = (1 - v_k)\, T_{k(local)} + v_k\, (T_{k(edge\, saver)} + T_{k(dispatch)}) \tag{12}$$

Task k has the following start execution time:

$$st_k == \begin{cases} max,\ ct_j\ f(k) \neq \emptyset, j \in f(k), \\ \quad 0 \qquad f(k) = \emptyset \end{cases} \tag{13}$$

where $f(k)$ is the set of the proximal predecessor tasks of task k.

Equation (13) shows that when $f(k)$ is empty, task k is the initial node with no direct predecessor, resulting in $st_k = 0$. Assuming a negligible transmission time for task computation results, task k maximum completion time was equal to its starting time.

4.5. Optimization Problem

The strategy aims to minimise energy consumption within a given timeframe. Determine if an offloading task is completed locally or at the mobile edge computing server. The problem can be described as follows [12].

$$P \ :\ \ min_x \sum_{x=1}^{n} \ [AT_l(Q_x) + AC_e(Q_x)] \tag{14}$$

$$\text{s. t:}\ \ RT_l(Q_n) + RT_{e(total)} \leq \tag{15}$$

5. Proposed Approach

The PSO algorithm can fall into a local optima, resulting in large task delays and energy costs this is its main downside. First, the PSO algorithm has been enhanced by converting the fixed inertia weight in the standard particle swarm to an evolving inertia weight, it can prevent falling into the local optimal solution, balance the local and global search, and improve global optimisation. This reduces energy consumption to meet delay requirements. Although, the Particle Swarm Optimisation algorithm is an evolutionary computer technology. Each particle has only two attributes: velocity (vi) and position (xi). Velocity represents particle speed, whereas position indicates direction of movement. The velocity and position of particles in each cycle fluctuate based on individual and buddy experience. Groups collaborate and share knowledge to guide each particle towards the best answer.

Assume r is the particle swarm size, U is the number of local tasks, S is the number of MEC servers, and M = $\{m_1\ m_2,\dots, m_u\ \}$ is the collection of all task. E denotes the task's energy consumption threshold. The periodic frequency set of all MEC servers includes the periodic frequencies of all MEC servers in the system.

$$H_{ij} = \begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,s} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,s} \\ \vdots & \vdots & & \vdots \\ h_{m,1} & h_{m,2} & \cdots & h_{m,s} \end{bmatrix}$$

The channel gain matrix H indicates the maximum transmission rate of a computing task transmitted to the MEC server. The channel gain of H_{ij} is 0, and $H_{ij}(1 \le i \le M, 1 \le j \le N, i \ne j)$ represent the channel gain needed for transmitting the task from mobile device i to server j.

Particles can accept any integer from 1 to n. The dimension of particle code corresponds to the number of task sets. Each particle provides an allocation system to offload work to the MEC server. The particle position vector X = $\{x_1, x_2,..., x_m\}$ shows that all task are offloaded to the associated MEC server. The dimension is equal to the number of tasks to be offloaded, and the value will be set dynamically. Consider the following scenario: a task set with 6 tasks. Tasks are $\{t_1, t_2, t_3, t_4, t_5,\}$. Suppose the particle code is [0, 3, 5, 6, 7,], t_1 is calculated locally, while t_2 is offloaded to the MEC server with id 3, and thereafter.

The particle velocity vector, $V = \{v_1, v_2,..., v_m\}$, indicates the span of offloading job to other servers. To begin, assign a random velocity to each particle and update the rounded velocity value before iterating. The particle velocity vector has the same dimension as the number of tasks to unload. Consider the following example: the particle position vector X is [5, 3, 1,, 6], while the velocity vector V is [1, 2, 3, 2, 1]. The first element 6 in particle position vector X indicates that task t_1 is offloaded to the MEC server with id 5. In the iterative updating process, the initial element in the particle velocity vector V is 1. This signifies that t1 has been moved to the next MEC server (id=6) for calculation. According to this rule, task ti is offloaded to the appropriate server.

5.1. Proposed Weighted Quantum Behaved Particle Swarm Optimization and Dispatched task offloading optimisation algorithm (WQPSOD)

The conventional PSO system limits the search range of particles whereby particles are not allowed to appear far away from the particle swarm, even if this position may be better than the current Gbest position. To solve the above-mentioned problem, a QPSO task offloading algorithm was proposed by [23]. Trajectory analyses in [23] demonstrated that, to guarantee convergence of PSO algorithm, each particle must converge to its local attractor $(p_{i,1}, p_{i,2}, \dots, p_{i,n})$, of which the coordinates are defined as

$$p_{ij}(t) = \varphi . p_{ij}(t) + (1 - \varphi) . p_{gj}(t), \ \ \varphi \epsilon (0,1) \tag{16}$$

It can be seen that the local attractor is a stochastic attractor of particle i that lies in a hyper-rectangle with p_i and p_g being two ends of its diagonal. We introduce the concepts of WQPSO as follows.

Assume that each individual particle move in the search space with a Ø potential on each dimension, of which the centre is the point p_{ij}. We can get the probability density function Q and distribution function F given by

$$Q(Y_{ij}(t+1) = \frac{1}{d_{ij}(t)} . e^{-2|p_{ij}(t) - Y_{ij}(t)|/d_{ij}(t)} \tag{17}$$

$$F(Y_{ij}(t+1) = \ e^{-2|p_{ij}(t) - Y_{ij}(t)|/d_{ij}(t)} \tag{18}$$

Where $d_{ij}(t)$ is the standard deviation of the function which determines search scope of each particle. Therefore, the position of the particle is obtained using the following equations:

$$Y_{ij}(t+1) = p_{ij}(t) \pm \frac{d_{ij}(t)}{2} \ln\left({}^{1}\!/_{u}\right), u = rand(0,1) \tag{19}$$

Where u is a random number generated from a uniform distribution $(0,1)$.

To valuate $d_{ij}(t)$, global point called mean best position is introduced in [24] is introduced into PSO. The global point denoted by h is defined as the mean of the p_{best} positions of all the particles given as:

$$h(t) = \left(h_1(t), h_2(t), \dots, h_n(t)\right) = \frac{1}{N}\sum_{1=1}^{N} p_{i,1}(t), \frac{1}{N}\sum_{1=1}^{N} p_{i,2}(t), \dots, \frac{1}{N}\sum_{1=1}^{N} p_{i,n}(t), \tag{20}$$

Where N the population is size and p_i is the p_{best} position of particle i. And therefore, the value $d_{ij}(t)$ is determined by the following equation:

$$d_{ij}(t) = 2\text{Ɣ}. \left|h_j(t) - Y_{ij}(t)\right| \tag{21}$$

Thus, the position of the particle as calculated as:

$$Y_{ij}(t) = p_{ij}(t) \pm \text{Ɣ}. \left|h_j(t) - Y_{ij}(t)\right|. \ln\left({}^{1}\!/_{u}\right) \tag{22}$$

Where Ɣ contraction expansion coefficient, which can be tuned to control the convergence speed of the algorithms.

The PSO with Equation 7 is called the weighted quantum behaved particle swarm optimization and Dispatch (WQPSOD) algorithm.

As mentioned above, in the WQPSO, the mean best position h is introduced to evaluate the value of d, making the algorithm more efficient than that proposed in [25]. From Equation 5, we can see that the mean best position is simply the average on the personal best position of all particles, which means that each particle is considered equal and exert the same influence on the value of h. The philosophy of this method is that the Mainstream Though, that is, mean best position h, determines the search scope or creativity of the particle [24]. The equally weighted mean position, however, is something of paradox, compared with the evolution of social culture in real world. For one thing, although the whole social organism determines the Mainstream Thought, it is not properly to consider each member equal. In fact, the elitists play more important role in culture development. With this in mind when we design a new control method for the WQPSOD in this paper, min Equation 5 is replaced for a weighted mean best position. The most important problem is to determine whether a particle is an elitist or not, or say it exactly, how to evaluate its importance in calculate the value of h. It is natural, as in other evolutionary algorithm, that we associate elitism with the particles' fitness value. The greater the fitness, the more important the particle is. Describing it formally, we can rank the particle in descendent order according to their fitness value first. Then assign each particle a weight coefficient ɲ linearly decreasing with the particle's rank, that is, the nearer the best solution, the larger its weight coefficient is. The mean best position h, therefore, is calculated as

$$\begin{aligned} h(t) &= \big(h_1(t), h_2(t), \dots, h_n(t)\big) \\ &= \frac{1}{N}\sum_{1=1}^{N} \text{ɲ}_{i,1} p_{i,1}(t), \frac{1}{N}\sum_{1=1}^{N} \text{ɲ}_{i,2} p_{i,2}(t), \dots, \frac{1}{N}\sum_{1=1}^{N} \text{ɲ}_{i,n} p_{i,n}(t), \end{aligned} \qquad (23)$$

Where ɲthe weight coefficient of every particle is, N is the number of population, based on the best parameter truing in [26] we allow ɲ to decreases linearly from 1.5 to 0.5. The WQPSOD algorithm is shown in algorithm 1.

```
    **Algorithm 1:** WQPSOD
    Input Population size $N$, dimension $n$
1.  Initialise initial solution $Y_i$ at random, find the initial
    position $p_i$
2.  **Do**
3.    Find $h_{best}$ using Eq. (23)
4.     **For** $i = 1 \ to \ N$
5.       If $Y_i < f(p_i)$ then
6.         $p_{i=Y_i}$
7.         $g = \arg\min(\mathrm{f}(p_i))$
8.         **For** $k = 1 \ to \ n$
9.             ε=rand(0,1)
10.            U=rand(0,1)
11.            If rand(0,1)>0.5 then
12.              $Y_{ij} = p_{ij}(t) - \text{Ɣ}.\left|h_j(t) - Y_{ij}\right|.\ln(1/u)$
13.            **else**
14.              $Y_{ij} = p_{ij}(t) + \text{Ɣ}.\left|h_j(t) - Y_{ij}\right|.\ln(1/u)$
15.          **End if**
16.        **End for**
17.      **End for**
18. Out put $Y_{ij}$
```

6. Experimental Setup

This section implements the proposed WQPSOD algorithm using Python in a multiuser multi-saver edge computing environment, there are N (50,100,150,200,250 and 300) devices and 10 MEC servers. Mobile devices initiate uninstallation requests to these servers. Table 1 displays the major parameters and their corresponding symbols. Moreover, we compared the WQPSOD approach with ACO [27], PSO [17], QPSO, [28] and examined how device number, task data size, and transmission rate affect system energy consumption and mean task completion time.

Table 1 Notation

$Symbol$	$Meaning$
N	$The\ number\ of\ mobile\ devices$
S	$The\ number\ of\ MEC\ saver$
H_{ij}	$Channel\ gain\ matrix$
$RT_l(Q_n)$	$The\ times\ spent\ by\ the\ user\ locally$
$AT_l(Q_n)$	$The\ energy\ consumption\ in\ the\ calculation\ process$
w_x	$The\ transmission\ rate\ from\ user\ mobile\ device\ x\ to\ acces\ point$
$d_{ij}(t)$	$The\ standard\ deviation\ of\ the\ function\ which\ determines\ search\ scope\ of\ each\ particles$
Ɣ	$Contraction\ expansion\ coefficient, which\ can\ be\ con\ be\ turned\ to\ control\ the\ convergence\ speed\ of the\ algoriths$
$ɲ_i$	$The\ weitht\ coefficient\ of\ every\ particles$
RT	$Transmission\ time$
h	$Mean\ best\ position$

7. Results Discussion

This section discusses results achieved in the experiment. To know how scalable our proposed algorithm, it is necessary to measure it performance using different number of heterogeneous devices. These devices have different computing capacity and can determined whether the performance of the results is determined by change in sizes of the resources. As can be seen in Table 2, although algorithms started at a promising point, but when the number of devices keeps increasing, the performance of our proposed algorithm. In the overall, the proposed WQPSOD performed better due to incorporation of the weighted sum approach which enable a shift that avoid the algorithm from falling into local optima. On the other hand, its enable an increase in the exploration capacity of the WQPSOD task offloading optimization algorithm while reducing the computational complexity of the algorithm.

Table 2 Results of energy consumption based on increasing number of devices

Device number	ACO	PSO	QPSO	WQPSOD
50	125	110	100	80

100	168	160	140	143
150	241	220	205	171
200	315	300	285	276
250	353	350	325	307
300	475	460	430	410

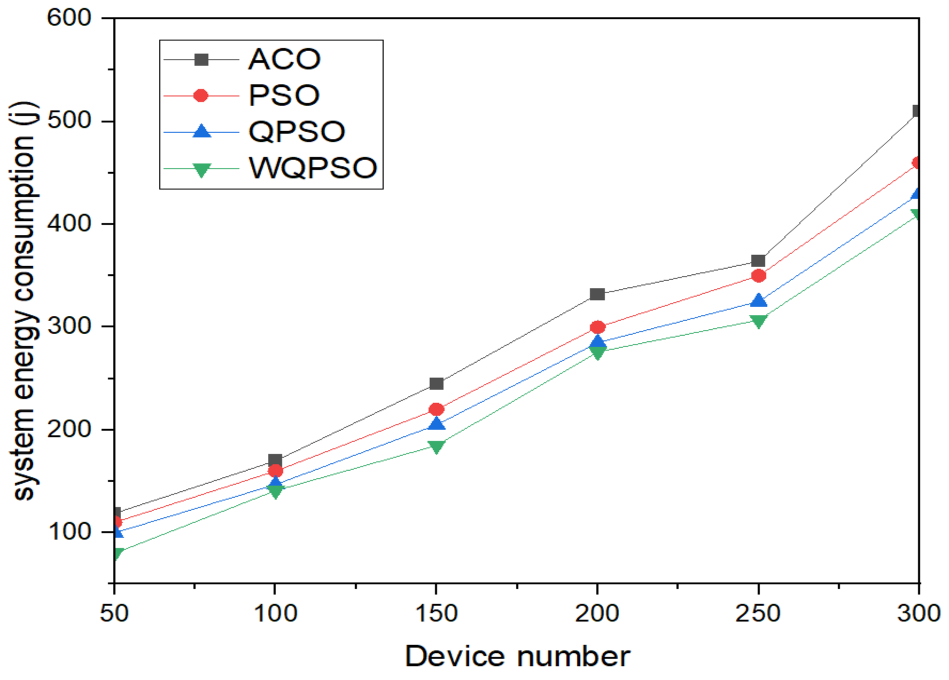

Figure 2 illustrates the system energy consumption with increasing number of devices, 45 Mbit task data size, and a constant transmission rate of 3000 kb/s.

To show trend in performances of the proposed WQPSOD task offloading algorithm against that of the benchmarked algorithms in Mobile edge computing environment, figure is used as illustrated in Figure 2 above. The figure illustrated a clear performance of the whole task offloading algorithms at the initial stages, but as the number of devices soars, it can be seen that the performance of our proposed approach outperformed that of the benchmarked approaches. This is as a result of the incorporation of the tasks dispatcher that determines the resource requirement of the tasks before routing it to the right computing resources which significantly help reducing energy consumption as illustrated in Figure 2.

Further investigation is carried out to determine the energy consumption of the whole tasks offloading algorithms with different task sizes. The essence is also to know scalable nature of the proposed WQPSOD task offloading algorithm when there is dynamic shift in task sizes. As shown in Table 3, QPSO started with a remarkable performance when compared with other tasks offloading algorithms. But as the tasks sizes increases, it can be seen that the performance of the QPSO task offloading algorithm in term of energy consumption degraded when compared to other task offloading algorithms. In the overall, our proposed WQPSOD task offloading optimization algorithm outperformed other tasks offloading algorithm as a result of the improvement carried out. The table clearly demonstrated that, our proposed task offloading algorithm can reduce computational complexity while ensuring that energy consumption level is reduce to an acceptable threshold.

Table 3. System energy consumption with different task data sizes

Task	ACO	PSO	QPSO	WQPSOD
5	215	200	188	170
10	321	310	291	282
15	445	437	436	390
20	623	582	559	530
25	770	742	718	600
30	1000	880	869	810
35	1030	1043	1030	950
5	215	200	188	170
10	321	310	291	282

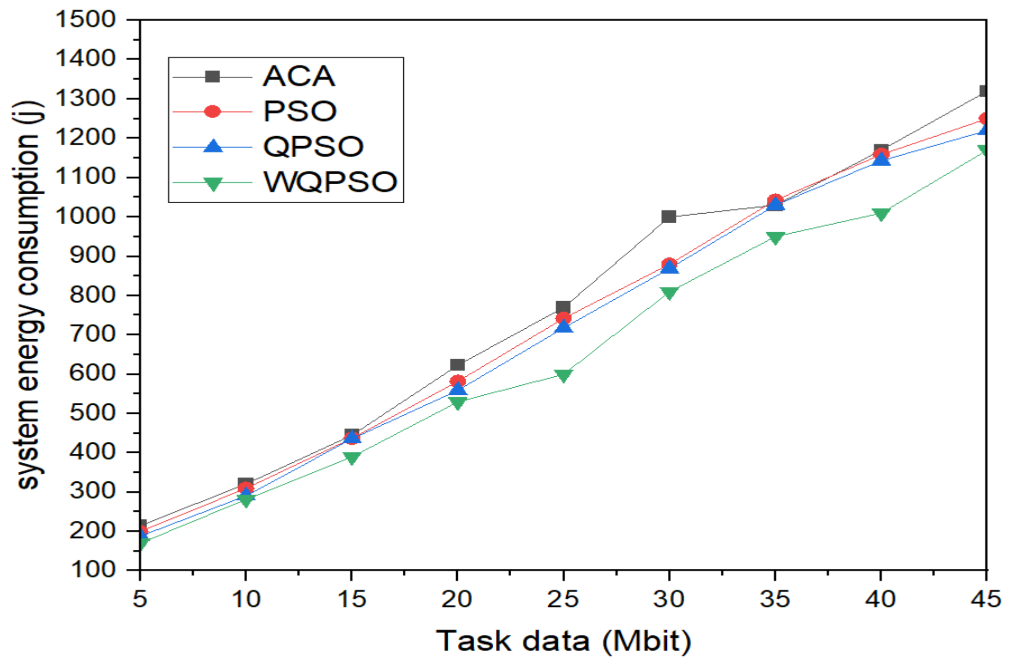

Figure 3 shows the system energy consumption with various task data sizes. The device number and transmission rate are 300 and 3000 kb/s, accordingly.

To show further show the trend on the performance of our proposed task offloading algorithm Figures were used as illustrated in Figure 3 in term of energy consumption. The uses of different tasks is to enable us determined whether the proposed algorithm can scale with respect to change in tasks sizes as well as resource sizes. Experimental results shows our approach is able start on a promising notes and as the number of tasks keeps increasing, the level of energy consumption keeps reducing as can be seen in the illustrated Figure 3 compared to that of benchmarked approaches. This is thought to be, due to the dispatched algorithm not cooperating with the local search strategy, which increases the WQPSOD exploration capacity while reducing the computational complexity of the algorithm.

Similarly, as can be seen in Table 4, the proposed WQPSOD displays a remarkable performance in achieving better results in term of completion time when compared to other benchmarked approaches. The increment in tasks instances has no effect whatsoever affecting performance of our proposed algorithm. the This remarkable achievement is made possible through the improvement of the proposed WQPSOD tasks offloading optimization algorithm. In order to understand the trend in the pattern of the results achieved by our proposed algorithm, graph is used as illustrated in Figure 4.

Table 4. Mean completion time with different task data sizes

Task	ACO	PSO	QPSO	WQPSOD
5	2.4	2	1.9	1.2
10	3.9	3.5	3	2.5
15	4.5	4.2	4	3.8
20	4.9	4.6	4.1	3.9
25	7.1	6.1	5.8	4.5
30	7.9	7.5	7.2	6.3
35	8.6	8	7.7	7.1
40	9.9	8.6	8.1	7.9
45	12.2	10.9	10	8.4

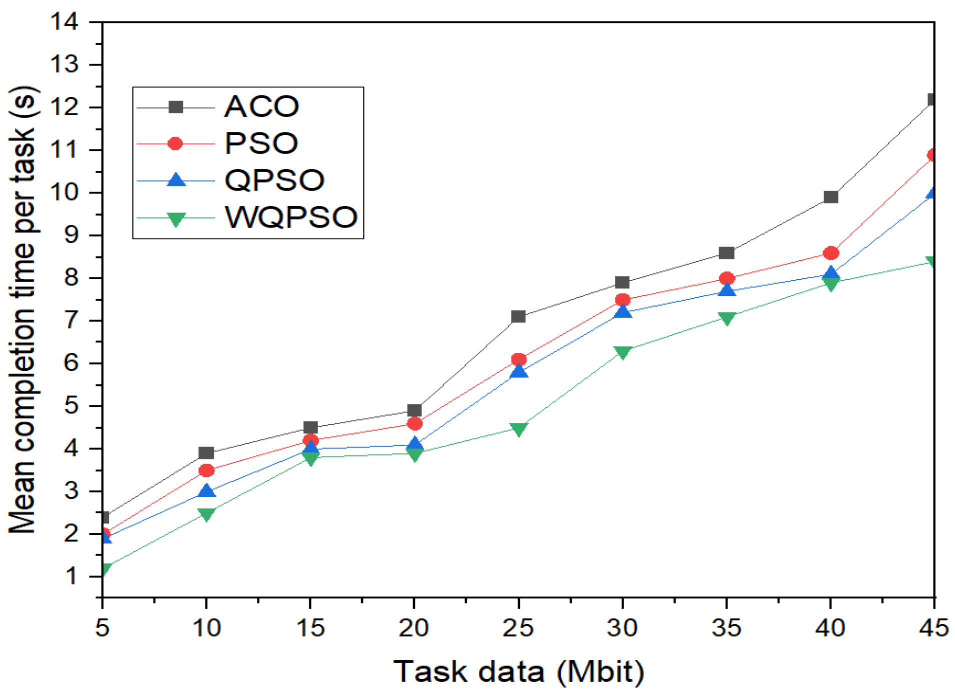

Figure 4 shows the mean completion time for various task data volumes (device number and transmission rate of 300 and 3000 kb/s).

It can be seen that the trend of the graph in Figure 4 shows that, although the whole algorithm started at promising performance, but as the number of tasks keeps increasing, the performance of our proposed algorithm is clearly seen as it achieved significant performance compared to the benchmarked algorithm. This shows the proposed algorithm is able to offload tasks properly at the most needed server. Where tasks with higher computing resource demands are channelled to the MEC serve while those with lower computing resource demand are migrated to the local server for efficient computation. The reason for the display in performance of our proposed WQPSOD tasks offloading optimization algorithm is due to the incorporation of weighted method which enable the algorithm to take a decision that enables it to scale under certain tasks condition. Besides, weighted procedure enable the algorithm overcome premature convergence during tasks selection, thereby avoiding been entrapped at the local optima.

8. Conclusion

Conventional optimization techniques like PSO, ACO and QPSO, may encounter challenges in escaping local optimization and may necessitate numerous iterations to reach the optimal solution, thereby exacerbating system energy consumption when considered. In this paper, we demonstrated via experiments how our proposed WQPSOD task offloading algorithm performs effectively for task offloading in multiuser multi saver scenarios in mobile edge computing environments. The approach outperforms the PSO and QPSO algorithms in terms of system energy consumption and task completion time across various mobile devices and workload sizes. Likewise, the research shows that the suggested WQPSOD task offloading algorithm consistently achieves the optimal global solution. This is thought to be due to the dispatched algorithm not cooperating with the local search strategy, which increases the WQPSOD exploration capacity.

Conflict of Interest

There is no conflict of interest. All authors have agreed to it submission.

Acknowledgement

This research is sponsored by Universiti Putra Malaysia under grant Number: (FRGS/1/2023/ICT11/UPM/02/3)

References

[1] C. Feng, P. Han, X. Zhang, B. Yang, Y. Liu, and L. Guo, “Computation offloading in mobile edge computing networks: A survey,” *J. Netw. Comput. Appl.*, vol. 202, no. April, p. 103366, 2022, doi: 10.1016/j.jnca.2022.103366.

[2] J. Bi, H. Yuan, K. Zhang, and M. C. Zhou, “Energy-Minimized Partial Computation Offloading for Delay-Sensitive Applications in Heterogeneous Edge Networks,” *IEEE Trans. Emerg. Top. Comput.*, vol. 10, no. 4, pp. 1941–1954, 2022, doi: 10.1109/TETC.2021.3137980.

[3] X. Deng, J. Yin, P. Guan, N. N. Xiong, L. Zhang, and S. Mumtaz, “Intelligent Delay-

Aware Partial Computing Task Offloading for Multiuser Industrial Internet of Things Through Edge Computing," *IEEE Internet Things J.*, vol. 10, no. 4, pp. 2954–2966, 2023, doi: 10.1109/JIOT.2021.3123406.

[4] S. Ghanavati, J. Abawajy, and D. Izadi, "Automata-Based Dynamic Fault Tolerant Task Scheduling Approach in Fog Computing," *IEEE Trans. Emerg. Top. Comput.*, vol. 10, no. 1, pp. 488–499, 2022, doi: 10.1109/TETC.2020.3033672.

[5] S. Azizi, M. Othman, and H. Khamfroush, "DECO: A Deadline-Aware and Energy-Efficient Algorithm for Task Offloading in Mobile Edge Computing," *IEEE Syst. J.*, vol. 17, no. 1, pp. 952–963, 2023, doi: 10.1109/JSYST.2022.3185011.

[6] H. Huang, Q. Ye, and Y. Zhou, "Deadline-Aware Task Offloading with Partially-Observable Deep Reinforcement Learning for Multi-Access Edge Computing," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 6, pp. 3870–3885, 2022, doi: 10.1109/TNSE.2021.3115054.

[7] Y. Chen, F. Zhao, Y. Lu, and X. Chen, "Dynamic Task Offloading for Mobile Edge Computing with Hybrid Energy Supply," *Tsinghua Sci. Technol.*, vol. 28, no. 3, pp. 421–432, 2023, doi: 10.26599/TST.2021.9010050.

[8] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, "Dynamic Task Offloading and Scheduling for Low-Latency IoT Services in Multi-Access Edge Computing," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 668–682, 2019, doi: 10.1109/JSAC.2019.2894306.

[9] M. P. J. Mahenge, C. Li, and C. A. Sanga, "Energy-efficient task offloading strategy in mobile edge computing for resource-intensive mobile applications," *Digit. Commun. Networks*, vol. 8, no. 6, pp. 1048–1058, 2022, doi: 10.1016/j.dcan.2022.04.001.

[10] J. Bi, H. Yuan, S. Duanmu, M. Zhou, and A. Abusorrah, "Energy-Optimized Partial Computation Offloading in Mobile-Edge Computing with Genetic Simulated-Annealing-Based Particle Swarm Optimization," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3774–3785, 2021, doi: 10.1109/JIOT.2020.3024223.

[11] Y. Li, "Optimization of Task Offloading Problem Based on Simulated Annealing Algorithm in MEC," *2021 9th Int. Conf. Intell. Comput. Wirel. Opt. Commun. ICWOC 2021*, no. 618015167, pp. 47–52, 2021, doi: 10.1109/ICWOC52624.2021.9530216.

[12] A. Abbas, A. Raza, F. Aadil, and M. Maqsood, "Meta-heuristic-based offloading task optimization in mobile edge computing," *Int. J. Distrib. Sens. Networks*, vol. 17, no. 6, 2021, doi: 10.1177/15501477211023021.

[13] Z. Kuang, L. Li, J. Gao, L. Zhao, and A. Liu, "Partial Offloading Scheduling and Power Allocation for Mobile Edge Computing Systems," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6774–6785, 2019, doi: 10.1109/JIOT.2019.2911455.

[14] H. Zhang, Y. Yang, X. Huang, C. Fang, and P. Zhang, "Ultra-Low Latency Multi-Task Offloading in Mobile Edge Computing," *IEEE Access*, vol. 9, pp. 32569–32581, 2021, doi: 10.1109/ACCESS.2021.3061105.

[15] A. Bozorgchenani, F. Mashhadi, D. Tarchi, and S. A. Salinas Monroy, "Multi-Objective Computation Sharing in Energy and Delay Constrained Mobile Edge Computing Environments," *IEEE Trans. Mob. Comput.*, vol. 20, no. 10, pp. 2992–3005, 2021, doi: 10.1109/TMC.2020.2994232.

[16] F. Xu, Z. Qin, L. Ning, and Z. Zhang, “Research on computing offloading strategy based on Genetic Ant Colony fusion algorithm,” *Simul. Model. Pract. Theory*, vol. 118, no. May 2021, p. 102523, 2022, doi: 10.1016/j.simpat.2022.102523.

[17] Q. You and B. Tang, “Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things,” *J. Cloud Comput.*, vol. 10, no. 1, 2021, doi: 10.1186/s13677-021-00256-4.

[18] Y. Wen, Q. Zhang, H. Yuan, and J. Bi, “Multi-Stage PSO-Based Cost Minimization for Computation Offloading in Vehicular Edge Networks,” *ICNSC 2021 - 18th IEEE Int. Conf. Networking, Sens. Control Ind. 4.0 AI*, vol. 1, pp. 1–6, 2021, doi: 10.1109/ICNSC52481.2021.9702184.

[19] S. Li, H. Ge, X. Chen, L. Liu, H. Gong, and R. Tang, “Computation Offloading Strategy for Improved Particle Swarm Optimization in Mobile Edge Computing,” *2021 IEEE 6th Int. Conf. Cloud Comput. Big Data Anal. ICCCBDA 2021*, pp. 375–381, 2021, doi: 10.1109/ICCCBDA51879.2021.9442609.

[20] A. Li, L. Li, and S. Yi, “Computation Offloading Strategy for IoT Using Improved Particle Swarm Algorithm in Edge Computing,” *Wirel. Commun. Mob. Comput.*, vol. 2022, 2022, doi: 10.1155/2022/9319136.

[21] L. N. T. Huynh, Q. V. Pham, X. Q. Pham, T. D. T. Nguyen, M. D. Hossain, and E. N. Huh, “Efficient computation offloading in multi-tier multi-access edge computing systems: A particle swarm optimization approach,” *Appl. Sci.*, vol. 10, no. 1, pp. 1–17, 2020, doi: 10.3390/app10010203.

[22] F. Mashhadi, S. A. S. Monroy, A. Bozorgchenani, and D. Tarchi, “Optimal auction for delay and energy constrained task offloading in mobile edge computing,” *Comput. Networks*, vol. 183, no. June, p. 107527, 2020, doi: 10.1016/j.comnet.2020.107527.

[23] M. Clerc and J. Kennedy, “The particle swarm-explosion, stability, and convergence in a multidimensional complex space,” *IEEE Trans. Evol. Comput.*, vol. 6, no. 1, pp. 58–73, 2002, doi: 10.1109/4235.985692.

[24] S. Chen, “Quantum-behaved particle swarm optimization with weighted mean personal best position and adaptive local attractor,” *Inf.*, vol. 10, no. 1, 2019, doi: 10.3390/info10010022.

[25] S. Zheng, X. Zhou, X. Zheng, and M. Ge, “Improved Quantum-Behaved Particle Swarm Algorithm Based on Levy Flight,” *Math. Probl. Eng.*, vol. 2020, 2020, doi: 10.1155/2020/3230643.

[26] M. Xi, J. Sun, and W. Xu, “An improved quantum-behaved particle swarm optimization algorithm with weighted mean best position,” *Appl. Math. Comput.*, vol. 205, no. 2, pp. 751–759, 2008, doi: 10.1016/j.amc.2008.05.135.

[27] M. K. Hussein and M. H. Mousa, “Efficient task offloading for IoT-Based applications in fog computing using ant colony optimization,” *IEEE Access*, vol. 8, pp. 37191–37201, 2020, doi: 10.1109/ACCESS.2020.2975741.

[28] S. Dong, Y. Xia, and J. Kamruzzaman, “Quantum Particle Swarm Optimization for Task Offloading in Mobile Edge Computing,” *IEEE Trans. Ind. Informatics*, vol. 19, no. 8, pp. 9113–9122, 2022, doi: 10.1109/TII.2022.3225313.