Computer Modeling in Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2022.020639

ARTICLE

Efficient UAV-Based MEC Using GPU-Based PSO and Voronoi Diagrams

Mohamed H. Mousa[1,2,*] **and Mohamed K. Hussein**[2]

[1]Department of Information Technology, College of Computing & Information Technology at AlKamil, University of Jeddah, Jeddah, Saudi Arabia

[2]Department of Computer Science, Faculty of Computers and Informatics, Suez Canal University, Ismailia, Egypt

[*]Corresponding Author: Mohamed H. Mousa. Email: mohamed_mousa@ci.suez.edu.eg

Received: 04 December 2021 Accepted: 18 February 2022

ABSTRACT

Mobile-Edge Computing (MEC) displaces cloud services as closely as possible to the end user. This enables the edge servers to execute the offloaded tasks that are requested by the users, which in turn decreases the energy consumption and the turnaround time delay. However, as a result of a hostile environment or in catastrophic zones with no network, it could be difficult to deploy such edge servers. Unmanned Aerial Vehicles (UAVs) can be employed in such scenarios. The edge servers mounted on these UAVs assist with task offloading. For the majority of IoT applications, the execution times of tasks are often crucial. Therefore, UAVs tend to have a limited energy supply. This study presents an approach to offload IoT user applications based on the usage of Voronoi diagrams to determine task delays and cluster IoT devices dynamically as a first step. Second, the UAV flies over each cluster to perform the offloading process. In addition, we propose a Graphics Processing Unit (GPU)-based parallelization of particle swarm optimization to balance the cluster sizes and identify the shortest path along these clusters while minimizing the UAV flying time and energy consumption. Some evaluation results are given to demonstrate the effectiveness of the presented offloading strategy.

KEYWORDS

Task offloading; mobile-edge computing; unmanned aerial vehicles; Internet of Things; voronoi diagrams; GPU; particle swarm optimization

Table 1 describes the meaning of the common terms abbreviations and acronyms found throughout the paper.

Table 1: List of abbreviations

Abbreviation	Definition
CUDA	"Compute Unified Device Architecture"
GA	"Genetic Algorithm"
GPGPU	"General Purpose GPU"
GPU	"Graphical Processing Unit"
GPUPSO	"GPU-Based PSO"
IoT	"Internet of Things"
MEC	"Mobile Edge Computing"
PSO	"Particle Swarm Optimization"
QoS	"Quality of Service"
UAC	"UAV-Assisted Computing"
UAV	"Unmanned Arial Vehicle"
UCMEC	"UAV-Assisted Computing MEC"
VD	"Voronoi Diagram"

1 Introduction

Mobile-Edge Computing (MEC) has been become the most promising framework for meeting Quality of Service (QoS) requests, taking into account the requirements of processing cycles and energy. This is essential for computationally intensive mobile applications and the Internet of Things (IoT) to cope with capability limitations of mobile and IoT devices [1–3]. A cloud-based layer supports an edge network of servers that handles computationally intensive tasks, allowing for a reduction in the required energy and corresponding turnaround time. The close proximity of the edge servers to IoT and mobile devices provides low latency and a high bandwidth for delay-sensitive IoT applications [4–6]. However, in some locations, the deployment of edge servers may be difficult, if not impossible [7–11]. As a real-life example, monitoring and observing forests and woodlands is a difficult task because such monitoring is time-consuming and requires considerable effort along with suitable resources. Stopping dangerous undertakings that can cause great damage to nature is a major challenge and responsibility of monitoring forestlands. Because forests cover such a wide area, it is difficult for forest managers and workers to take immediate action in cases of problems such as forest fires and illegal tree felling. Communication with ground sensors, aerial surveillance, mapping, aerial photography, and thermal imaging are some approaches in which Unmanned Aerial Vehicles (UAVs) can carry out forest monitoring. Thus, monitoring by UAVs contributes to the conservation of wildlife, biodiversity, and vegetation, the balance of ecology, and other forestry-related issues.

1.1 Motivation

The UAV industry has made significant advances with regard to both technology and cost during the past decade, and many different applications have been demonstrated, such as intelligent transportation, agriculture, and wilderness monitoring [12–15]. In addition, throughout this period, communication technologies have advanced in various ways (e.g., IoT, cloud computing, etc.) [16]. UAVs have been highly successful due to their cost-effective and flexible deployment [17]. These advances in communication technologies have been used to improve the collaborative computation between UAVs and ground devices. In fact, UAV-Assisted Computing (UAC) is

regarded as computation services provided to ground devices using UAVs as flying bases [18]. To this end, UAVs can support offloading in MEC architecture in various ways such as:

- UAV-assisted communication: For IoT devices, UAVs allow quick, flexible, and cost-effective network coverage by acting as relays for distant ground base stations [19,20].
- UAV-assisted Computing MEC (UCMEC) architecture: For cloud services, UAVs equipped with edge servers allow for the offloading and processing of ground device tasks.

A unified communication network that integrates a UAV with an MEC network, as well as near line-of-sight wireless communication, are important for meeting the QoS specifications of mobile applications with regard to energy consumption and sensitivity to delay [21,22]. UAVs are constrained by their energy limitations, which lead to delays in their offloading processes. Two challenges must be taken into account in the development of any UAV-assisted computing MEC method:

1. Locally, ground devices are not distributed uniformly. Hence, the offloading and execution of tasks must be partitioned and balanced along a set of clusters in order to achieve high performance of the UCMEC offloading system.
2. Despite the energy limitations, UAVs should collect IoT data from various locations, process offloaded tasks, and report the results back. These paths have to be optimized to satisfy the energy constraints of the UAVs.

1.2 Contribution

This research designs a UCMEC framework to offload tasks with the goal of optimizing the amount of required energy and computational turnaround time for both mobile devices and UAVs. As depicted in Fig. 1, we propose a space partitioning of the offloaded tasks into different regions, with the UAV hovering over each region to process the offloaded tasks. Finally, the UAV trajectory is optimized considering the time and energy limits of the UAV. The contributions according to the proposal are as follows:

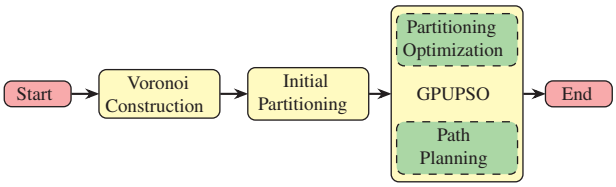


Figure 1: The proposed offloading framework

- "Voronoi Diagrams" (VDs) are used based on a formal mathematical model of the UCMEC system to partition the IoT wireless devices into set regions, where the offloaded subset of tasks in each region meets the overall time constraint.
- The initial partitioning of the wireless devices is improved using the proposed "Graphics Processing Unit" (GPU)-based Particle Swarm Optimization (PSO).
- The same GPU-based PSO (GPUPSO) steps are used for obtaining a near-optimum path minimizing the UAV trajectory and load balancing tasks in the partitioned regions according to both energy and time constraints.
- PSO has the advantage of being simple to implement and has a good capability for convergence in comparison with many population-based approaches, such as Genetic Algorithms

(GAs) [23]. The main disadvantage is that PSO may be attracted to local minima. We overcome this issue by adding an aging factor that guarantees exploration of the global search space.

The rest of the paper is organized as follows. The next section presents the related work, followed by a formulation of the optimization model in terms of the time and energy constraints as well as the affecting factors in Section 3. Section 4 explains the proposed optimization framework on the basis of clustering of the ground devices and the use of the GPU-based PSO. Finally, Section 5 presents some experimental results, followed by the conclusions in Section 6.

2 Related Work

Edge servers are always privileged by their high bandwidths and low latency. Therefore, the performance of offloaded tasks is enhanced by MEC for restricted systems in terms of energy and time [24,25]. When UAVs are equipped with MEC servers and communicating channels, UAVs contribute to computing services and instant communications. This yields a better energy consumption and rate of transmission for mobile devices. However, the UAV's energy limitations have an adverse effect on ground mobile equipment with regard to communication and computation [26]. To address these key challenges, a variety of research studies have been conducted to explore different configurations, optimization objectives, and underlying constraints.

In [27], optimum planning of the UAV trajectory, ratio of offloading tasks, and user scheduling were studied together to minimize all users' delays. In [28], UCMEC was proposed as a novel framework of agent-based task offloading. With the aim of obtaining the optimal offloading plan, the intelligence agent was led to obtain a plan that was as efficient as possible. However, this study did not address UAV trajectory optimization. In [29], the authors demonstrated that partial offloading can be optimized through the optimization of offloading ratios, local computing frequencies, transmission power, and edge server computing frequencies. However, their framework did not consider UAV mobility. In [30], the authors proposed an MEC system involving multi-UAVs and ground IoT nodes to offload computational tasks that they could not handle using their limited capabilities. To balance the computational load of the UAVs, they employed a deep reinforcement learning approach. However, stable MEC clustering with greater computing power does not guarantee generation by this method. In [31], using the Dinkelbath algorithm and "successive convex approximation" (SCA) techniques, the authors solved a nonconvex optimization problem formulated in the UAV scenario. In this system, neither the level of distribution of mobile devices on the ground level nor the timing of tasks that are to be offloaded were considered. As part of resource allocation and joint computation offloading, task assignment to MECs while reducing turnaround time and required energy is an important issue. Reference [32] focused on the number and location of UAVs in the development of algorithms for deploying multi-UAVs in a FANET. Based on whether tasks are offloaded or processed locally, a greedy algorithm was proposed to determine the optimal solution. However, it is important to determine when and how to offload tasks from IoT devices, particularly if the tasks are uncertain in order to improve accuracy while reducing cloud communications costs. Using the UAV computing platform, Chen et al. [33] proposed an intelligent task offloading algorithm (iTOA) that can solve the computationally intensive problem of task processing. The offloading operation is performed using the deep "Monte Carlo Tree Search" algorithm (MCTS), which is the principal algorithm of Alpha Go. To arrive at the optimal decision, MCTS simulates the future trajectory of the unloading decision and optimizes the rewards based on this simulation. To be effective, the proposed offloading strategy requires training data, a prediction model of the channel state, and a period of

self-learning. In [34], the UAV served as a supporting unit for helping ground devices offload their tasks to access points for further computations. They employed a greedy algorithm to optimize the required energy based on a suitable selection of UAV paths and allocation of resources. However, only the UAV velocity is taken into account in the optimization process. In [35], the relation between energy consumption and latency is considered for task offloading and for evaluating MEC system performance. Their approach used iterative optimization to determine the path of the UAV. However, this optimization does not take into account how clustered fixed slots should be partitioned or consider time delays. In [36], the UAV energy consumption was minimized by optimizing its trajectory and task scheduling. This will further shorten the operational time if some memory-intensive and computation-intensive tasks are executed on the UAV. In [37], greedy search optimization based on the uplink and downlink communications between UAVs and devices was utilized as a means of offloading and receiving data while ensuring energy-efficient operation of the system. In [38], a new mobile edge system integrating UAVs and IoT devices was developed. By launching a UAV, the system enables the provisioning of services to IoT devices by using wireless communication. Additionally, their approach formulates the problem as a nonconvex optimization problem by optimizing UAV position, resource allocation decisions, and task splitting decisions jointly to minimize the total delay and consumption of energy. However, it does not take task scheduling into account. In summary, the approach based on exhaustive and greedy searches is not suitable for such optimization. In fact, the majority of problems in consideration have many factors that must be optimized, such as the shortest path planning, ground user number and locations, energy consumption, and time delay parameters. Thus, most researchers deployed heuristic settings such as fixed partitions. However, their frameworks are not extensible due to their high complexity. In addition, learning-based approaches [39,40] request a preprocessing training step. Such a preprocessing step makes these approaches inapplicable to probabilistic problems.

Thus, the metaheuristic optimization framework is a suitable candidate for dealing with such stochastic problems. In this study, we employ a combination of a computational geometry algorithm, namely, the VD construction algorithm, and a natural phenomena simulation, namely, PSO, to carry out the required minimization of the time delay and energy consumption of the task offloading problem using UCMEC in a reasonable timeframe.

3 UAV-Based Offloading Framework

Now, we describe the employed time and energy models in this section. Table 2 lists the set of symbols and factors employed in the suggested task offloading model.

Table 2: List of symbols and parameters used in the proposed optimization framework

Symbol	Definition
N	The number of wireless devices
D	The set of wireless devices
d_i	A wireless device
d_{ik}	Euclidean distance between d_i and d_k
$\mathbb{H}$	Flying altitude of the UAV
$\mathbb{R}_c$	The communication coverage radius of the UAV

(Continued)

Table 2 (continued)

Symbol	Definition
R_i	A region containing wireless devices
$\mathbb{K}$	The number of regions
t_i^c	The computing cycle of task t_i
t_i^τ	The delay of task t_i
t_i^d	The data size of task t_i
ρ_0	The amount of power reception per 1 meter
β	The bandwidth of the uplink channel
h_i	The channel gain
$\mathbb{P}_{ik}$	The max transmission power
$\mathbb{N}_0$	The signal noise
f^u	The processing capacity of the UAV
$\boldsymbol{\kappa}_1$	The energy factor for wireless communication
$\boldsymbol{\kappa}_2$	The energy factor for computation processing
$\boldsymbol{\kappa}_3$	The energy factor for UAV hovering
v	The speed of the UAV
E	Battery storage capacity of the UAV

Two layers comprise an offloading system based on UAVs. A ground layer consists primarily of IoT and mobile devices that have N fixed-position wireless devices on the ground. Let $D = \{d_1, d_2, ..., d_N\}$ denote the set of ground devices. To maximize efficiency and reduce overall energy consumption, wireless devices are encouraged to offload their computations to a flying MEC server for faster calculations and minimum time delay. A single UAV in conjunction with an MEC form the second layer, namely, a flying MEC. This layer provides cloud-based computations for the first layer devices while ensuring a minimal response time.

For the UAV, an altitude of $\mathbb{H} > 0$ is maintained continuously. In addition, let $\mathbb{R}_c$ be the radius of communication coverage of the UAV. Several $\mathbb{K}$ regions are covered by the UAV with respect to a predefined path. Without loss of generality, let the path start at R_0 and end at $R_\mathbb{K}$, i.e., following the sequence $R_0 \cdots R_\mathbb{K}$. At the end of the path cycle, the UAV returns to the starting position after hovering over each region R_k to process the tasks that have been offloaded to this region. On the other hand, the position of each ground device d_i is specified in advance by the Cartesian coordinates (x_i, y_i). In addition, the stopping position of the UAV over the region R_k is specified by the Cartesian coordinates $u_k = (x_k, y_k)$. Therefore, by calculating the Euclidean distance, we can evaluate the distance between the UAV and each d_i as follows:

$$d_{ik} = \sqrt{(x_i - x_k)^2 + (y_i - y_k)^2 + \mathbb{H}^2} \tag{1}$$

Provided that the UAV with device d_i is in the covered region, i.e., the following condition is satisfied:

$$d_{ik} < \mathbb{R}_c \tag{2}$$

While the UAV hovers over a region R_k, device d_i requests to offload its tasks $t_i = (t_i^c, t_i^\tau, t_i^d)$, where t_i^d, t_i^τ, and t_i^c correspond to the data size, task delay, and needed computing cycles, respectively. The presented UCMEC framework divides the set of devices D into $\mathbb{K}$ clusters on each UAV to offload the requested tasks. Let $\delta_{ik} = 1$ indicate that the ground device d_i is permitted to offload task t_i by the UAV while flying over R_k. The following subsections describe the suggested optimization approach according to the energy consumption, computational, and communicating models.

3.1 Communication Model

In this section, we will describe the communicating model controlling the UAV. Suppose that device d_i is within cluster R_k. This means that when the UAV is hovering over R_k, d_i lies in the $\mathbb{R}_c$ coverage of the UAV. Therefore, the channel gain resulting from the communication between the UAV and d_i is evaluated as follows [12]:

$$h_{ik} = \frac{\rho_0}{d_{ik}^2} \tag{3}$$

where ρ_0 is the amount of power reception per 1 meter. Similarly, considering that signal noise is denoted by $\mathbb{N}_0$, the power of transmission is denoted by $\mathbb{P}_i$, and the bandwidth of the communicating channel is denoted by β. Therefore, the rate of transmission is evaluated using the following equation [41]:

$$r_{ik} = \beta \log_2(1 + \frac{\mathbb{P}_i h_{ik}}{\mathbb{N}_0}) \tag{4}$$

Let T_{ik}^{trans} be the time required to transmit a task from d_i. T_{ik}^{trans} is evaluated as follows:

$$T_{ik}^{trans} = \frac{t_i^d}{r_{ik}} \tag{5}$$

Based on Eq. (5), the time required to offload all of the tasks within R_k is as follows:

$$T_k^{trans} = \sum_i \delta_{ik} T_{ik}^{trans} \tag{6}$$

Depending on the decision value, δ_{ik}, device d_i is either permitted or not permitted to offload its task to the UAV.

3.2 Computation Model

In this section, we will formulate the time of the computation model based on the computation capacity of the UAV. Let f^u be the UAV computational capacity; therefore, the time required to execute task t_i is evaluated as follows:

$$T_{ik}^{comp} = \frac{t_i^c}{f^u} \tag{7}$$

Similar to T_k^{trans}, the total computational time is evaluated as follows:

$$T_k^{comp} = \sum_i \delta_{ik} T_{ik}^{comp} \tag{8}$$

Using Eqs. (6) and (8), the overall time required within R_k is evaluated as follows:

$$T_k = T_k^{trans} + T_k^{comp} \tag{9}$$

3.3 Energy Consumption Model

In this section, we will describe the required energy to offload the tasks in R_k. In fact, the consumption of energy results from the following:

- the energy required to send the receive tasks' data,
- the energy required to execute the received tasks, and
- the energy required to the flying operation.

For the transmission and reception of data, let the communication energy factor be denoted by κ_1. Therefore, communication energy, E_k^{trans}, is evaluated by:

$$E_k^{trans} = \kappa_1 \sum_i \delta_{ik} t_i^d \tag{10}$$

Let κ_2 be the computational energy factor. Therefore, the computational energy, E_k^{comp}, is evaluated as follows:

$$E_k^{comp} = \kappa_2 \sum_i \delta_{ik} t_i^c f^{u2} \tag{11}$$

For the UAV hovering energy, we followed the propulsion energy model of [42]. Let κ_3 denote the hovering energy factor. Therefore, the hovering energy, E_k^h, is evaluated using the following equation:

$$E_k^h = \kappa_3 T_k \tag{12}$$

Using Eqs. (10)–(12), the required energy is given by:

$$E_k = E_k^{trans} + E_k^{comp} + E_k^h \tag{13}$$

3.4 Objective Function

In this section, we present the principal factors of the objective function on which the PSO will be based. In fact, the set of ground devices is arranged into a set of clusters. This set is dynamically constructed based on the following factors: the maximum cluster capacity, $\dot{m}$, and the sum of distances, $\dot{\mathbb{D}}$, between the devices and their corresponding cluster center. These factors are evaluated as follows:

$$\dot{m} = \max_k (\sum_i \delta_{ik}) \tag{14}$$

$$\dot{\mathbb{D}} = \sum_k \sum_i \delta_{ik} d_{ik} \tag{15}$$

In the same manner, based on Eq. (13), the required energy, E, along all clusters is evaluated as follows:

$$E = \sum_k E_k \tag{16}$$

We formulate the function, $\mathbb{F}$, to be minimized as a mixed-integer, nonlinear constraints problem as follows:

$$\min: \quad \mathbb{F} = \frac{\mathbb{K}E}{\dot{m}\dot{\mathbb{D}}} \tag{17}$$

subjected to:

$$\mathbb{C}_1: d_{ik} - \mathbb{R}_c < 0 \quad \forall i = \{1, ..., N\} \tag{18}$$

$$\mathbb{C}_2: \sum_k T_k - T_u < 0 \tag{19}$$

$$\mathbb{C}_3: \sum_k E_k - E_u < 0 \tag{20}$$

$$\mathbb{C}_4: \left(\sum_k \delta_{ik}\right) - 1 = 0 \quad \forall i = \{1, ..., N\} \tag{21}$$

The first constraint, $\mathbb{C}_1$, guarantees the coverage of each device when the UAV hovers over the corresponding cluster. $\mathbb{C}_2$ guarantees that the total time does not exceed the maximum allowed flying time. Similarly, $\mathbb{C}_3$ guarantees that the total energy does not exceed the UAV battery maximum. Finally, the final constraint, $\mathbb{C}_4$, guarantees that offloading of each device is associated with one and only one cluster. Finally, we combine Eqs. (19)–(21) as a single objective function as follows:

$$\dot{\mathbb{F}} = \mathbb{F} + \gamma \sum_{i=1}^{4} \mathbb{C}_i^2 \tag{22}$$

where γ denotes the weight of the penalty sum.

4 The Optimization Model

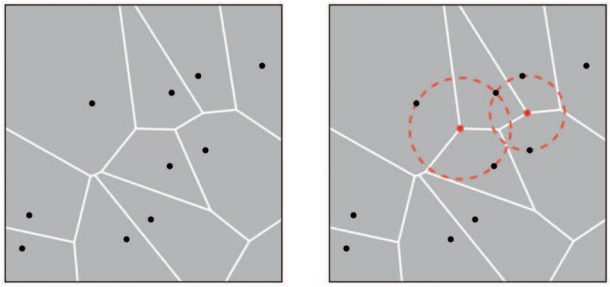
In this section, we describe our proposed optimization approach. First, we propose a partitioning of the set of wireless devices, D, using the VDs with respect to the UAV communication range $\mathbb{R}_c$. Second, optimization is performed using the GPUPSO algorithm to obtain better clustering and the shortest path for the UAV. We propose the implementation of a customized PSO on a GPU. The customized PSO is as simple as the classical PSO but shows better performance. Our aim is to speed up the computation and introduce a better local communication topology between the particles within the population as part of the global search. The parallelization of the PSO on a GPU enables a better exploration of search spaces with a high number of dimensions using a large population swarm.

4.1 Voronoi Diagram

VD partitions a two-dimensional plane containing a set of N anchors, called sites, with a set of N Voronoi polygons. Each Voronoi polygon is associated with a unique site (see Fig. 2a) such that any points inside that polygon are closer to the specified site [43,44]. In addition, the vertices of the Voronoi polygon are called Voronoi vertices, and the circles centered at these vertices that pass through the neighboring sites are called Voronoi circles [45] (see Figs. 2b). We note that the site may be shared with more than one Voronoi circle.

Given a region R containing a set of mobile devices D, where $D = \{d_1, d_2, \cdots, d_N\}$, VD partitions the region R into a set of Voronoi polygons. Each polygon contains a single mobile device and can be considered an offloading region to be visited by the UAV. We show how to merge these regions to offload multiple devices in a single visit, taking into consideration that any resulting region should satisfy the following:

1. All of the associated mobile devices in the region should be in the coverage area, $\mathbb{R}_c$, of the UAV.
2. The sum of the transmission time and the computation time of the assigned devices in the region should be less than T_u, $\sum_k T_k < T_u$, as stated in constraint C_2 in Eq. (19).



(a) (b)

Figure 2: VD construction. (a) Each Voronoi polygon corresponds to a unique site, and (b) the Voronoi circles are centered at Voronoi vertices and pass through the corresponding sites

Algorithm 1: The initial partitioning of D into a set of offloadable regions

input: A set of mobile devices $D = \{d_1, d_2, ..., d_N\}$
Construct the Voronoi diagram, $VD(D)$;
Determine the Voronoi circle for all $VD(D)$ vertices;
for $i = 1$ *to* N **do**
  $\{d_{i_1} \cdots d_{i_l}\} \leftarrow$ the set of devices sharing at least one Voronoi circle with d_i;
  $c_{d_i} \leftarrow$ the fitting circle containing $d_i \cup \{d_{i_1} \cdots d_{i_l}\}$;
end
$L \leftarrow$ Sort the set of all c_{d_i} w.r.t their radii;
$i \leftarrow 0$;
repeat
  $L' \leftarrow \{L[j] : L[j] \cap L[i] \neq \Phi\}$; // neighboring regions
  for $j \leftarrow 0 \cdots \#L'$ **do**
    if $\left(L'[j] \cup L[i] \leq \mathbb{R}_c\right)$ *and* $\left(T_{L'[j] \cup L[i]} \leq T_u\right)$ **then**
      $L[i] \leftarrow L'[j] \cup L[i]$;
      $L \leftarrow L \backslash L'[j]$;
    end
  end
  $i \leftarrow (i + 1)\%\#L$;
until *There are no more regions to merge*;

Algorithm 1 summarizes the steps for the initial partitioning of D. The input of the algorithm is the set of mobile devices D. The algorithm starts by constructing the Voronoi diagram of the set of devices, VD(D). Once VD is constructed, we identify all of the Voronoi circles of the diagram. For each device d_i, we identify the corresponding devices, $\{d_{i_1} \cdots d_{i_l}\}$, that share at least one Voronoi circle with d_i. Then, the algorithm finds the fitting circle, c_{d_i}, that contains $d_i \cup \{d_{i_1} \cdots d_{i_l}\}$. The set of all c_{d_i} is considered as the initial partitioning of the area R. Since each device is shared by at least one Voronoi circle and each Voronoi circle passes through at least three devices, c_{d_i} has at least three devices. In addition, the set of fitting circles may not be disjointed, i.e., $c_{d_i} \cap c_{d_j} \neq \Phi$. The set of all $c_{d_i}(i = 1, \cdots, N)$ are sorted in a list L with respect to their radii in ascending order. Now, we show how to merge the regions c_{d_i} with respect to the boundary condition C_2 expressed in Eq. (19).

Starting with the smallest fitting circle $L[i]$, where $i = 0$, we identify of all the fitting circles $L[j]$ that share at least one device with $L[i]$. Now, for each $L[j]$, we test whether the circle can be merged with $L[i]$ without violating the constraint specified in Eq. (19). If the merge is eligible, then $L[i]$ is updated to $L[j] \cup L[i]$, and $L[j]$ is removed from the list L. The algorithm stops when there are no more regions eligible for merging in L.

In fact, the initial partitioning of the given devices may produce unbalanced partitions. Although these initial partitions are constructed in accordance with the time constraints of the UAV, these partitions may increase the transfer time necessary to achieve all the tasks in each partition. When the UAV is in a specified region, the UAV may cover devices from neighboring partitions as shown in Fig. 3. The existence of these overlapping areas creates the possibility of improving the transfer time of such devices. In Fig. 3, device s can be covered from regions R_2 and R_3. The question here is which is better, offloading the device s within region R_2 or within region R_3? We use PSO to answer this question by associating s with the corresponding region, which optimizes the overall time, namely, the offloading time, along the UAV trajectory. In the next section, we describe the proposed implementation of GPU-based PSO that finds the optimized association of shared devices and the optimal UAV path that minimizes the overall time and energy consumption.

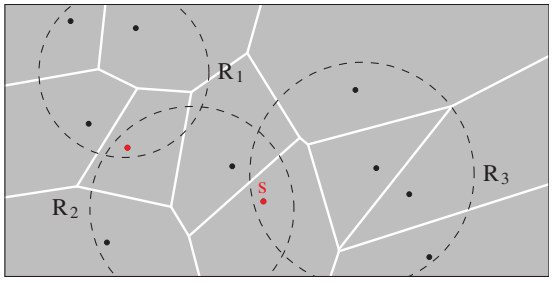


Figure 3: An example of three regions (dashed circles) with shared devices (red)

4.2 GPU-Based Particle Swarm Optimization

In this section, we illustrate how to implement PSO on a GPU architecture. We start by introducing the classical PSO. Then, we summarize the benefits of the current advances of GPUs [46]. In addition, we show how to use GPU-based PSO in optimizing the wireless device

association to improve the clustering of the set of devices D. Finally, we illustrate how to choose the shortest path for the UAV that minimizes the problem stated in Eq. (17).

4.2.1 Particle Swarm Optimization

The classical PSO algorithm is classified as a stochastic global optimization technique. PSO was developed in 1995 by Eberhard and Kennedy based on the social behavior of birds or fish [47]. Fig. 4 depicts the overall process of the classical PSO, in which each particle P_i is provided with the following information:

- x_i^t: the current position at time t,
- v_i^t: the current velocity at time t, and
- b_i: the historical best position of P_i.

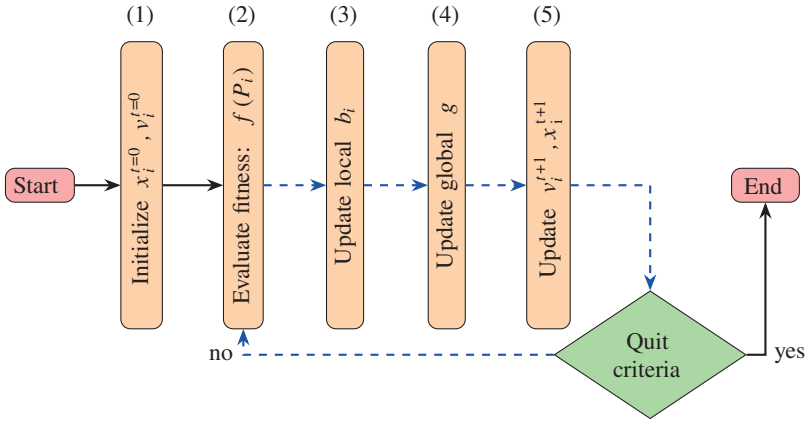


Figure 4: The classical particle swarm optimization algorithm

The algorithm shares the information of the historical global best position among all particles in the swarm, g. Using the PSO, each particle in the swarm adjusts its velocity, v_i^t, according to the following two items of information: (1) the best position at which the particle has been thus far, b_i, and (2) the historical best position along the whole swarm, g. This is accomplished using the following formula:

$$v_i^{t+1} = \omega v_i^t + c_1 r_1 \left(b_i - x_i^t\right) + c_2 r_2 \left(g - x_i^t\right) \tag{23}$$

where $\omega \in [0, 1]$ is the inertia coefficient, $c_1, c_2 \in \mathbb{R}^+$ are the acceleration coefficients, and $r_1, r_2 \in [0, 1]$ are random numbers. The first term of Eq. (23) represents the momentum part, in which the previous flight direction is memorized, preventing the particle from drastically changing direction. In the same manner, the second term is the cognitive part, in which the previous best position of the particle is memorized and the performance is quantified relative to the past performances. Finally, the third term is the social part, in which the performance is quantified relative to that of its neighbors. Once the velocity is updated, the current position of each particle is updated using the following equation:

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{24}$$

This makes all the particles align their positions in the search space to the local and global best positions in the search space. Fig. 5 shows a graphical illustration of the effect of Eqs. (23)

and (24) describe the inertia and cognitive and social behaviors of particle P_i. The choice of a convenient value for ω is crucial for the convergence of the algorithm [48]. A greater value of ω corresponds to a more global exploration of the search space. By contrast, smaller values of ω focus on the local investigation of the search space. Our experiments show that it is preferable to initialize ω with large values to encourage the global investigation of the search space and then decrease this value using an aging factor according to the running time to obtain finer solutions.

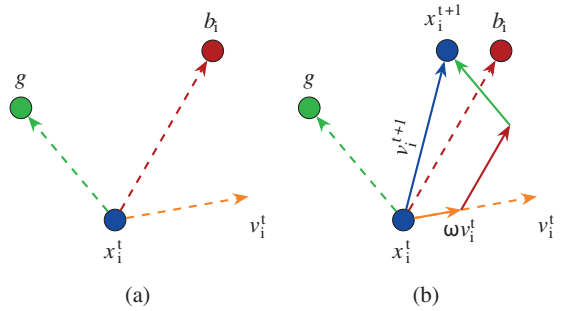


Figure 5: The alignment of the particle position according to cognitive and social behaviors. The orange, red and green solid lines correspond to the inertia and cognitive and social behaviors, respectively, of particle P_i

In fact, PSO has been considered one of the most powerful nature-inspired optimization algorithms. However, for high-dimensional space or large swarm optimization problems, PSO requires performance improvements. This is due to successive evaluations of the fitness function as well as the need to update the particle locations. The first intuitive solution for overcoming this performance degradation is to parallelize the steps from (2)–(5); see the blue shaded loop in Fig. 4 [49]. However, it is insufficient to set subjobs in a parallel calculation job by introducing a single thread for each job that includes a time-consuming evaluation task.

4.2.2 CUDA Architecture

Due to the emergence of General Purpose GPUs (GPGPUs) and their rapid arithmetic kernels, performance has dramatically improved in many fields. In fact, a GPU performs computations faster than a CPU. Due to the high transistor density, floating-point operations can be performed much faster with a GPU than with a CPU because it devotes more transistors to data processing than to data caching and flow control. The second benefit of GPUs is their data-parallel capabilities; they are particularly well suited to problem solving in data-parallel computations with a high arithmetic intensity, that is, for problems where there is a high ratio of arithmetic operations to memory operations.

Furthermore, programming on GPUs has been greatly simplified by the existing platforms, particularly by the Compute Unified Device Architecture (CUDA) [50]. For the CUDA programming concept, as shown in Fig. 6, GPUs can be considered a high-performance computing device capable of handling many threads at the same time. Kernels are the heart of CUDA, and a batch of threads is responsible for executing these kernels simultaneously. This batch is organized as a grid consisting of a number of thread blocks. Generally, a block of threads is a collection of threads. This collection collaborates by allowing data sharing using an efficient and dedicated shared memory. In addition, the collection synchronizes their execution to flawlessly optimize their

memory hits. A key feature of CUDA is its memory model that is closely related to the mechanism used by the batch of threads. Every thread has its own local memory, registers, and a unique id. Global memory can be accessed by all threads within a grid, while shared memory can only be accessed within a block. For our implementation, we mostly use shared memory and global memory.

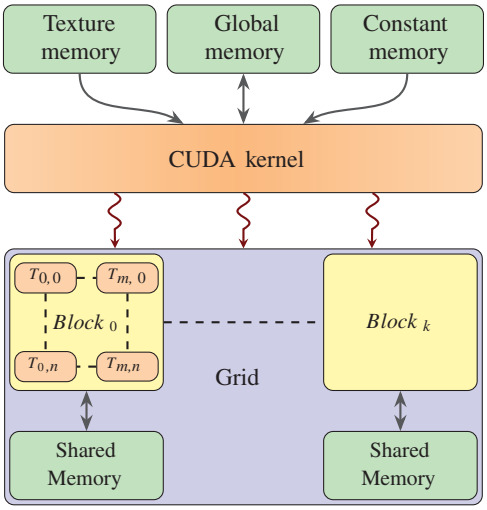


Figure 6: The programming architecture of the CUDA kernel

In this research, we propose a robust implementation of PSO using CUDA. The GPU-based PSO can enhance optimization performance, increase the swarm population, enlarge the size of the problem, and greatly speed up implementation. This allows users to solve critical time problems or complex optimization problems in a reasonable amount of time.

4.2.3 Optimization Structure

Based on the initial partitioning described in the previous section, we propose a multiobjective GPUPSO that minimizes:

1. The routing path of the UAV,
2. The energy consumption, and
3. The shared devices' offloading time.

In fact, the association of shared devices affects the overall time and transmission energy since the offloading time and energy depend on the positions of the UAV and the devices, as described in Eqs. (1), (3), (4), and (10). On the other hand, the path of the UAV affects both the time and energy consumption.

Let $R=\{R_1,R_2,\cdots,R_{\mathbb{K}}\}$ be the set of initial partitions and $T_k^{trans}=\sum_i \delta_{ik}T_{ik}^{trans}$, where T_k^{trans} is the time required to offload the devices of R_k and T_{ik}^{trans} is the transmission time of device $i\in R_k$. Consider that $D'=\{d'_1,d'_2,\cdots,d'_{n'}\}$ is the set of shared devices such that:

$$\forall d'\in D' \Leftrightarrow \left(\exists k_1,k_2:(1\le k_1\ne k_2\le \mathbb{K})\wedge(d'\in R_{k_1}\cap R_{k_2})\right) \tag{25}$$

In addition, we define the set of regions $R'=\{R'_1,R'_2,\cdots,R'_{\mathbb{K}}\}$ such that $R'_k=R_k\backslash D'$. Now, we describe the proposed particle structure. In fact, each particle P_i consists of two parts:

1. $G_i=(g_i^1,\cdots,g_i^{n'})$, which represents the association of shared devices, and
2. $C_i=(c_i^1,\cdots,c_i^{\mathbb{K}})$, which represents the routing path of the UAV.

Now, we define the structure of the part concerning the first objective. First, each g_i^j is associated with the corresponding device $d'_j\in D'$, $j=1\cdots n'$. The value of g_i^j is chosen from the set of integers k_j, $1\le k_j\le\mathbb{K}$, representing the indices of the candidate partitions R'_{k_j} such that d'_j is coverable when the UAV hovers over R_{k_j}. In addition, we define $|g_i^j|$ as the cardinality of the set of possible values of g_i^j. Therefore, the values of $G_i=(g_i^1,g_i^2\cdots g_i^{n'})$ can be, for example, as follows:

$$\begin{aligned}
&g_i^1\in\{0\equiv R'_{k_{i_1}},1\equiv R'_{k_{i_2}}\} && |g_i^1|=2\\
&\quad\vdots && \quad\vdots\\
&g_i^j\in\{0\equiv R'_{k_{j_1}},1\equiv R'_{k_{j_2}},2\equiv R'_{k_{j_3}}\} && |g_i^j|=3\\
&\quad\vdots && \quad\vdots\\
&g_i^{n'}\in\{0\equiv R'_{k_{n'_1}},1\equiv R'_{k_{n'_2}}\} && |g_i^{n'}|=2
\end{aligned}$$

The cost consists of the sum of the total time and energy transmission for devices in D'. For the second objective, we define the structure of this part as a set of floating values describing the priority of visiting the partitions R_i by the UAV; $C_i=(c_i^1,c_i^2,\cdots,c_i^{\mathbb{K}})$. The region with the highest priority is visited first. The cost consists of the flying, hovering and transmission energy of the UAV over path C_i.

The final structure of particle P_i is the concatenation of $G_iC_i\in\mathbb{R}^{n'+\mathbb{K}}$. The two parts G_i and C_i are not independent since the association of a device with a certain region affects the hovering time and energy of that region. Similarly, G does not affect the total computation time and energy. The application of the parallelized PSO to the proposed GC is summarized as depicted in Algorithm 2. The algorithm starts by initializing the values of the particle positions. Initially, the best position for each particle will be the current position, and the global position is the best position among the local best. Then, the algorithm iterates in parallel using the CUDA kernel *move*() that locates the next position for each particle P_i. After each move, the global and local best positions are updated if necessary.

Algorithm 2: The GPU-based particle swarm optimization algorithm

// initialization
foreach P_i **do** // in parallel
  initialize x_i^0 & v_i^0;
  $b_i \leftarrow x_i^0$;
end
$g = \min_i b_i$;
$a = 0.99$; // aging factor
for $t = 0$ *to* $MaxIter$ **do**
  foreach P_i **do** // in parallel
    $move(P_i, \omega a^t, C_1, C_2, t+1)$;
    $update(b_i, g)$; /* if needed */
  end
end
// CUDA kernel
__global__ $move(P_i, \omega, C_1, C_2, t+1)\{$
  $r_1 \leftarrow rand()$;
  $r_2 \leftarrow rand()$;
  $v_i^{t+1} \leftarrow \omega v_i^t + c_1 r_1 \left(b_i - x_i^t\right) + c_2 r_2 \left(g - x_i^t\right)$;
  $x_i^{t+1} \leftarrow x_i^t + v_i^{t+1}$;
  // $x_i^{t+1} \equiv G_i^{t+1} C_i^{t+1}$
  // conversion of G_i^{t+1} to a discrete value
  $g_i^{j,t+1} \leftarrow \lfloor g_i^{j,t+1} \rfloor \% |g_i^j| \quad \forall j \in (1 \cdots n')$;
}

5 Results and Discussion

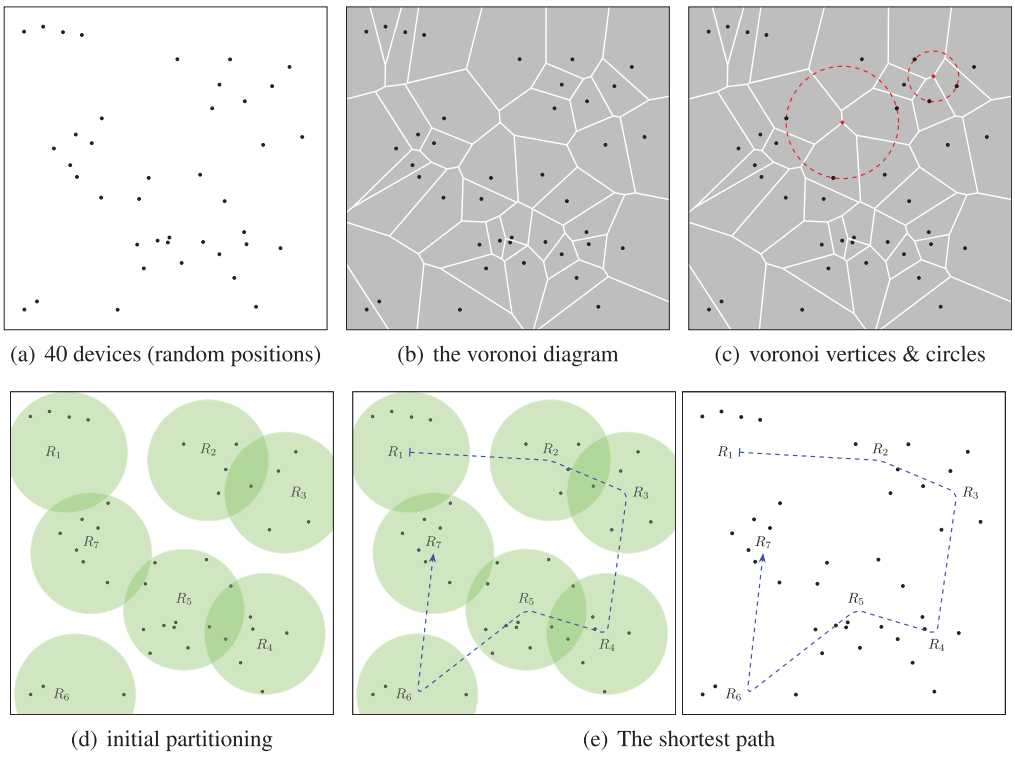
The experiments are performed on an NVIDIA CUDA Maxwell architecture embedded in an Intel Core i7-8565U CPU running at 1.8 and 1.99 GHz and a GeForce MX130 GPU with 4 GB RAM under Windows 10. Here, we present some experimental results that demonstrate the effectiveness of the proposed approach. A set of wireless devices is randomly distributed over the square region of 200×200 m^2. Many studies have been devoted to determining the flying and energy parameters for UAVs [51,52]. Similar simulation parameters are presented in Table 3. First, the partitioning of the devices is performed on the CPU, and then the proposed optimization using the PSO is applied on the GPU. The results concerning the optimization step are the average of 10 independent runs. The inertia ω, cognitive C_1, and social C_2 parts are set as 0.75, 2.5, and 1.5, respectively. The aging factor for the inertia ω is set to 0.99 and applied every 10 iterations. We have set the number of iterations to 1000.

Fig. 7 shows a graphical illustration of the proposed approach. First, a set of 40 wireless devices is randomly generated, as shown in Fig. 7a. The VDs of these devices are created using the sweep-line algorithm [53,54], as shown in Fig. 7b. For each device, the corresponding Voronoi circles are determined, as shown in Fig. 7c, to apply the merging step. In the merging step, the Voronoi vertices are merged together, as described in Algorithm 1 with respect to the UAV covering radius $\mathbb{R}_c$, as shown in Fig. 7d. Once the clusters of devices are obtained, the proposed

GPU-based particle swarm algorithm is applied to obtain the shortest path for the UAV, as shown in Fig. 7e.

Table 3: Parameter settings of the UAV for the evaluation experiments

Parameter	Value
β	40 MHz
h_i	−30 dB
$\mathbb{N}_0$	10^{-9} W
$\boldsymbol{\kappa}$	10^{-26}
$\mathbb{P}_{ik}$	0.4 W
v	30 m/s
$\mathbb{H}$	50 m
t^d	200 KB–3 MB
t^c	6×10^9–9×10^{10}
f^u	300 MHz
$\mathbf{E}$	5×10^5 J



(a) 40 devices (random positions) (b) the voronoi diagram (c) voronoi vertices & circles

(d) initial partitioning (e) The shortest path

Figure 7: (a) The set of devices, (b) the Voronoi diagram of the devices, (c) the Voronoi vertices (red) and their corresponding Voronoi circles, (d) the initial partitioning of the devices, and (e) the shortest path

Moreover, we compare the proposed optimization of our system *vs.* a GA. In fact, the GA is a metaheuristic optimization approach based on natural reproduction phenomena. The GA simulates the three main Darwinian operators of natural evolution, namely, selection of the fittest, crossover, and mutation operators. The algorithm starts by randomly initializing an initial population and selecting the fittest individuals for reproduction (applying the crossover and mutation) of the next generation offspring. The individual structure is called a chromosome. In our experiments, we use a chromosome structure similar to the proposed particle structure, as described in Section 4.2.3, and apply the classical GA operators. Fig. 8 shows a comparison of the proposed optimization and a GA. The population size is 800 particles (or individuals for GA). Fig. 8 shows that the proposed GPU-based PSO has a higher convergence rate than the GA.

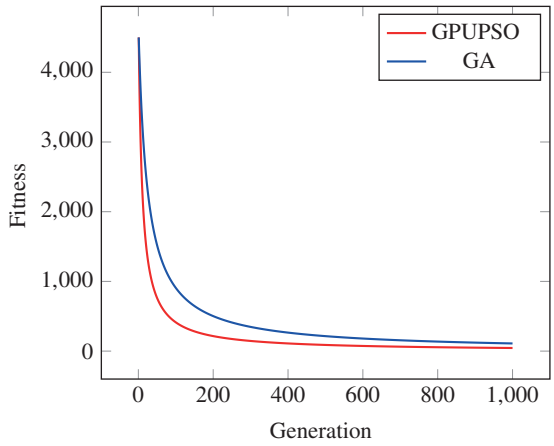


Figure 8: A comparison between the proposed GPU-based PSO (red) and a GA (blue). The GPU-based PSO converges faster to the optimal value than the GA

Table 4: The timing (in msec) of the proposed approach for different numbers of wireless devices (positioned randomly)

#*D*	CPU		GPU
	VD	Part	PSO
50	8	10	516
100	8.8	14	1350
150	9.5	19	2020
200	10	26	2784
250	10.4	29	3406

The analysis of the computational time shows that the GPU architecture speeds up calculation, as presented in Table 4. The columns from left to right are the number of wireless devices, the Voronoi construction time, the partitioning time, and the optimization time. The number of iterations and swarm size are set to 1000 and 800, respectively. The timings presented in Table 4 show that the optimization process incurs the main bulk of the computational cost of the proposed approach. In addition, when the number of wireless devices increases, the dimensionality of the problem also increases, and the optimization step is the most strongly affected portion

of the proposed approach. However, due to the high performance of the GPU architecture, the performance difference is negligible.

6 Conclusion

This paper proposed a UCMEC system to improve the performance of offloading tasks from mobile devices with the goal of minimizing the task delay and the energy consumption of the system. The proposed system partitions the ground devices into regions where the UAV can hover over each region to process the offloaded tasks. A VD is used for the partitioning process. The UAV trajectory over the regions is optimized using a GPU-based PSO. The performance of the proposed system was validated by comparison with other algorithms in the literature.

The limitation of the proposed offloading process is that it is valid only for stationary devices. In other words, it does not take into account the dynamic positioning of the ground sensors. In addition, the performance of the PSO in defining the optimal clustering and the shortest path of the UAV is very sensitive to the intrinsic parameters of the PSO, namely, the inertia, cognitive, and social parameters. In fact, this is a general drawback for all metaheuristic approaches. Our future work will consider the mobility of ground devices as well as the adaptive setting of the algorithm parameters.

Funding Statement: This work was funded by the University of Jeddah, Saudi Arabia, under Grant No. (UJ-20-102-DR). The authors, therefore, acknowledge the technical and financial support by the University.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Hussein, M. K., Mousa, M. H. (2020). Efficient task offloading for IoT-based applications in fog computing using ant colony optimization. *IEEE Access, 8,* 37191–37201. DOI 10.1109/Access.6287639.
2. Jiang, E., Wang, L., Wang, J. (2021). Decomposition-based multi-objective optimization for energy-aware distributed hybrid flow shop scheduling with multiprocessor tasks. *Tsinghua Science and Technology, 26(5),* 646–663. DOI 10.26599/TST.2021.9010007.
3. Xu, X., Li, H., Xu, W., Liu, Z., Yao, L. et al. (2022). Artificial intelligence for edge service optimization in internet of vehicles: A survey. *Tsinghua Science and Technology, 27(2),* 270–287. DOI 10.26599/TST.2020.9010025.
4. Wang, S., Zhang, X., Zhang, Y., Wang, L., Yang, J. et al. (2017). A survey on mobile edge networks: Convergence of computing, caching and communications. *IEEE Access, 5,* 6757–6779. DOI 10.1109/ACCESS.2017.2685434.
5. Mabrouki, J., Azrour, M., Dhiba, D., Farhaoui, Y., Hajjaji, S. E. (2021). IoT-Based data logger for weather monitoring using arduino-based wireless sensor networks with remote graphical application and alerts. *Big Data Mining and Analytics, 4(1),* 25–32. DOI 10.26599/BDMA.2020.9020018.
6. Malek, Y. N., Najib, M., Bakhouya, M., Essaaidi, M. (2021). Multivariate deep learning approach for electric vehicle speed forecasting. *Big Data Mining and Analytics, 4(1),* 56–64. DOI 10.26599/BDMA.2020.9020027.
7. Mohamed, N., Al-Jaroodi, J., Jawhar, I., Noura, H., Mahmoud, S. (2017). UAVFog: A UAV-based fog computing for internet of things. *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, San Francisco, CA, USA, IEEE. https://ieeexplore.ieee.org/document/8397657/.

8. Cheng, N., Xu, W., Shi, W., Zhou, Y., Lu, N. et al. (2018). Air-ground integrated mobile edge networks: Architecture, challenges, and opportunities. *IEEE Communications Magazine, 56(8),* 26–32. DOI 10.1109/MCOM.35.
9. Liu, Y., Song, Z., Xu, X., Rafique, W., Zhang, X. et al. (2021). Bidirectional GRU networks-based next POI category prediction for healthcare. *International Journal of Intelligent Systems*. DOI 10.1002/int.22710.
10. Yuan, L., He, Q., Chen, F., Zhang, J., Qi, L. et al. (2022). CSEdge: Enabling collaborative edge storage for multi-access edge computing based on blockchain. *IEEE Transactions on Parallel and Distributed Systems, 33(8),* 1873–1887. DOI 10.1109/TPDS.2021.3131680.
11. Xu, X., Liu, X., Yin, X., Wang, S., Qi, Q. et al. (2020). Privacy-aware offloading for training tasks of generative adversarial network in edge computing. *Information Sciences, 532,* 1–15. DOI 10.1016/j.ins.2020.04.026.
12. Bejaoui, A., Park, K. H., Alouini, M. S. (2020). A QoS-oriented trajectory optimization in swarming unmanned-aerial-vehicles communications. *IEEE Wireless Communications Letters, 9(6),* 791–794. DOI 10.1109/LWC.5962382.
13. Zhu, B. J., Hou, Z. X., Lu, Y. F., Shan, S. Q. (2015). The direction zone of engineless UAVs in dynamic soaring. *Computer Modeling in Engineering and Sciences, 105(6),* 467–490. DOI 10.3970/cmes.2015.105.467.
14. Zhu, B. J., Hou, Z. X., Wang, X. Z., Chen, Q. Y. (2015). Long endurance and long distance trajectory optimization for engineless UAV by dynamic soaring. *Computer Modeling in Engineering & Sciences, 106(5),* 357–377. DOI 10.3970/cmes.2015.106.357.
15. Liu, D. N., Hou, Z. X., Guo, Z., Yang, X. X., Gao, X. Z. (2016). Permissible wind conditions for optimal dynamic soaring with a small unmanned aerial vehicle. *Computer Modeling in Engineering & Sciences, 111(6),* 531–565. DOI 10.3970/cmes.2016.111.531.
16. Aggarwal, S., Kumar, N. (2020). Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges. *Computer Communications, 149,* 270–299. DOI 10.1016/j.comcom.2019.10.014.
17. Zhang, J., Chen, T., Zhong, S., Wang, J., Zhang, W. et al. (2019). Aeronautical AdHoc networking for the internet-above-the-clouds. *Proceedings of the IEEE, 107(5),* 868–911. DOI 10.1109/PROC.5.
18. Qi, X., Li, B., Chu, Z., Huang, K., Chen, H. et al. (2019). Secrecy energy efficiency performance in communication networks with mobile sinks. *Physical Communication, 32,* 41–49. DOI 10.1016/j.phycom.2018.06.009.
19. Wang, J., Jiang, C., Wei, Z., Pan, C., Zhang, H. et al. (2019). Joint UAV hovering altitude and power control for space-air-ground IoT networks. *IEEE Internet of Things Journal, 6(2),* 1741–1753. DOI 10.1109/JIoT.6488907.
20. Fu, S., Tang, Y., Zhang, N., Zhao, L., Wu, S. et al. (2020). Joint unmanned aerial vehicle (UAV) deployment and power control for internet of things networks. *IEEE Transactions on Vehicular Technology, 69(4),* 4367–4378. DOI 10.1109/TVT.25.
21. Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K. B. (2017). A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials, 19(4),* 2322–2358. DOI 10.1109/COMST.2017.2745201.
22. Zhou, F., Hu, R. Q., Li, Z., Wang, Y. (2020). Mobile edge computing in unmanned aerial vehicle networks. *IEEE Wireless Communications, 27(1),* 140–146. DOI 10.1109/MWC.7742.
23. Wang, D., Tan, D., Liu, L. (2018). Particle swarm optimization algorithm: An overview. *Soft Computing, 22(2),* 387–408. DOI 10.1007/s00500-016-2474-6.
24. Yu, Y. (2016). Mobile edge computing towards 5G: Vision, recent progress, and open challenges. *China Communications, 13(Supplement 2),* 89–99. DOI 10.1109/CC.6245522.
25. Zakaryia, S. A., Ahmed, S. A., Hussein, M. K. (2021). Evolutionary offloading in an edge environment. *Egyptian Informatics Journal, 22(3),* 257–267. DOI 10.1016/j.eij.2020.09.003.
26. Nguyen, V., Khanh, T. T., van Nam, P., Thu, N. T., Seon Hong, C. et al. (2020). Towards flying mobile edge computing. *2020 International Conference on Information Networking (ICOIN)*, Barcelona, Spain, IEEE. https://ieeexplore.ieee.org/document/9016537/.
27. Hu, Q., Cai, Y., Yu, G., Qin, Z., Zhao, M. et al. (2019). Joint offloading and trajectory design for UAV-enabled mobile edge computing systems. *IEEE Internet of Things Journal, 6(2),* 1879–1892. DOI 10.1109/JIoT.6488907.

28. Wang, R., Cao, Y., Noor, A., Alamoudi, T. A., Nour, R. (2020). Agent-enabled task offloading in UAV-aided mobile edge computing. *Computer Communications, 149,* 324–331. DOI 10.1016/j.comcom.2019.10.021.
29. Tang, Q., Chang, L., Yang, K., Wang, K., Wang, J. et al. (2020). Task number maximization offloading strategy seamlessly adapted to UAV scenario. *Computer Communications, 151,* 19–30. DOI 10.1016/j.comcom.2019.12.018.
30. Yang, L., Yao, H., Wang, J., Jiang, C., Benslimane, A. et al. (2020). Multi-UAV-enabled load-balance mobile-edge computing for IoT networks. *IEEE Internet of Things Journal, 7(8),* 6898–6908. DOI 10.1109/JIoT.6488907.
31. Li, M., Cheng, N., Gao, J., Wang, Y., Zhao, L. et al. (2020). Energy-efficient UAV-assisted mobile edge computing: Resource allocation and trajectory optimization. *IEEE Transactions on Vehicular Technology, 69(3),* 3424–3438. DOI 10.1109/TVT.25.
32. Wang, Y., Ru, Z. Y., Wang, K., Huang, P. Q. (2020). Joint deployment and task scheduling optimization for large-scale mobile users in multi-UAV-enabled mobile edge computing. *IEEE Transactions on Cybernetics, 50(9),* 3984–3997. DOI 10.1109/TCYB.6221036.
33. Chen, J., Chen, S., Luo, S., Wang, Q., Cao, B. et al. (2020). An intelligent task offloading algorithm (iTOA) for UAV edge computing network. *Digital Communications and Networks, 6(4),* 433–443. DOI 10.1016/j.dcan.2020.04.008.
34. Hu, X., Wong, K. K., Yang, K., Zheng, Z. (2019). UAV-Assisted relaying and edge computing: Scheduling and trajectory optimization. *IEEE Transactions on Wireless Communications, 18(10),* 4738–4752. DOI 10.1109/TWC.7693.
35. Li, L., Wen, X., Lu, Z., Jing, W. (2020). An energy efficient design of computation offloading enabled by UAV. *Sensors, 20(12),* 3363. DOI 10.3390/s20123363.
36. Zhan, C., Hu, H., Sui, X., Liu, Z., Niyato, D. (2020). Completion time and energy optimization in the UAV-enabled mobile-edge computing system. *IEEE Internet of Things Journal, 7(8),* 7808–7822. DOI 10.1109/JIoT.6488907.
37. Guo, H., Liu, J. (2020). UAV-Enhanced intelligent offloading for internet of things at the edge. *IEEE Transactions on Industrial Informatics, 16(4),* 2737–2746. DOI 10.1109/TII.9424.
38. Yu, Z., Gong, Y., Gong, S., Guo, Y. (2020). Joint task offloading and resource allocation in UAV-enabled mobile edge computing. *IEEE Internet of Things Journal, 7(4),* 3147–3159. DOI 10.1109/JIoT.6488907.
39. Lan, Y., Wang, X., Wang, C., Wang, D., Li, Q. (2019). Collaborative computation offloading and resource allocation in cache-aided hierarchical edge-cloud systems. *Electronics, 8(12),* 1430. DOI 10.3390/electronics8121430.
40. Wang, L., Wang, K., Pan, C., Xu, W., Aslam, N. et al. (2021). Deep reinforcement learning based dynamic trajectory control for UAV-assisted mobile edge computing. *IEEE Transactions on Mobile Computing,* 1. DOI 10.1109/TMC.7755.
41. He, H., Zhang, S., Zeng, Y., Zhang, R. (2018). Joint altitude and beamwidth optimization for UAV-enabled multiuser communications. *IEEE Communications Letters, 22(2),* 344–347. DOI 10.1109/LCOMM.2017.2772254.
42. Wu, F., Yang, D., Xiao, L., Cuthbert, L. (2019). Energy consumption and completion time tradeoff in rotary-wing UAV enabled WPCN. *IEEE Access, 7,* 79617–79635. DOI 10.1109/Access.6287639.
43. Aurenhammer, F., Klein, R. (2000). Voronoi diagrams**Partially supported by the Deutsche Forschungsgemeinschaft, grant K1 655 2-2. *Handbook of Computational Geometry*, pp. 201–290. North-Holland: Elsevier. https://linkinghub.elsevier.com/retrieve/pii/B9780444825377500061.
44. Guiling, W., Guohong, C., LaPorta, T. (2003). A bidding protocol for deploying mobile sensors. *11th IEEE International Conference on Network Protocols*, USA, http://ieeexplore.ieee.org/document/1249781/.
45. Edla, D. R., Jana, P. K. (2012). A novel clustering algorithm using voronoi diagram. *Seventh International Conference on Digital Information Management (ICDIM 2012)*, Macau, Macao, IEEE. http://ieeexplore.ieee.org/document/6360125/.
46. Mousa, M. H., Hussein, M. K. (2021). High-performance simplification of triangular surfaces using a GPU. *PLoS One, 16(8),* e0255832. DOI 10.1371/journal.pone.0255832.

47. Kennedy, J., Eberhart, R. (1995). Particle swarm optimization. *International Conference on Neural Networks*, vol. 4. Perth, WA, Australia, IEEE. http://ieeexplore.ieee.org/document/488968/.
48. Cui, H., Shu, M., Song, M., Wang, Y. (2017). Parameter selection and performance comparison of particle swarm optimization in sensor networks localization. *Sensors, 17(3),* 487. DOI 10.3390/s17030487.
49. Lalwani, S., Sharma, H., Satapathy, S. C., Deep, K., Bansal, J. C. (2019). A survey on parallel particle swarm optimization algorithms. *Arabian Journal for Science and Engineering, 44(4),* 2899–2923. DOI 10.1007/s13369-018-03713-6.
50. NVIDIA (2021). Compute Unified Device Architecture (CUDA) Programming Guide. https://docs.nvidia.com/cuda/.
51. Elloumi, M., Escrig, B., Dhaou, R., Idoudi, H., Saidane, L. A. (2017). Designing an energy efficient UAV tracking algorithm. *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, Valencia, Spain, IEEE. http://ieeexplore.ieee.org/document/7986274/.
52. Al-Shabi, M. A., Hatamleh, K. S., Asad, A. A. (2013). UAV dynamics model parameters estimation techniques: A comparison study. *2013 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*, Amman, Jordan, IEEE. http://ieeexplore.ieee.org/document/6716436/.
53. Fortune, S. (1987). A sweepline algorithm for voronoi diagrams. *Algorithmica, 2(1–4),* 153–174. DOI 10.1007/BF01840357.
54. Software (2019). A C implementation for creating 2D voronoi diagrams. https://github.com/JCash/voronoi.