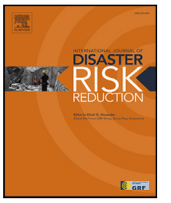
International Journal of Disaster Risk Reduction 97 (2023) 104067

Contents lists available at ScienceDirect

International Journal of Disaster Risk Reduction

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



Efficient offloading in disaster-affected areas using unmanned aerial vehicle-assisted mobile edge computing: A gravitational search algorithm-based approach

Santanu Ghosh, Pratyay Kuila *

Department of Computer Science & Engineering, National Institute of Technology Sikkim, Ravangla, South Sikkim 737139, India

ARTICLE INFO

Keywords:
Edge computing (EC)
Task offloading
Unmanned aerial vehicles (UAV)
Delay
Energy
Gravitational search algorithm (GSA)

ABSTRACT

The collection and processing of real-time data from a disaster-affected area is challenging. Unmanned aerial vehicles (UAVs) can efficiently gather the data and then transfer it to the edge servers (ESs) to timely initiate the rescue process. Consideration of energy and delay in a UAV-assisted edge network is very important, as both the UAVs and the smart mobile devices (SMDs) in the network have energy constraints and low processing capacity. Offloading is a promising technique to preserve the precious energy of the SMDs. In this research, gravitational search algorithm (GSA)-based offloading is presented for UAV-assisted mobile edge computing (MEC)-enabled disaster-affected areas. The problem is first mathematically formulated and shown to be computationally hard. Efficient encoding of agents (solution vectors) is given for the offloading problem. Fitness function is designed by considering the energy, delay, and load balancing of the ESs. The proposed GSA is executed by considering multiple disaster scenarios, and its performance is compared with other evolutionary algorithms (EAs) like the genetic algorithm (GA), particle swarm optimization (PSO), and fireworks algorithm (FWA). It has been observed that the GSA outperforms the other EAs in almost all the considered experiment scenarios. GSA claims a 30%–40% improvement for delay, 3%–5% for energy consumption, and more than 40% for load balancing. Statistical and convergence analyses are also conducted. The convergence of the GSA is found to be faster than that of the other EAs.

1. Introduction

Natural calamities and disasters are always detrimental to the welfare of animals on the planet Earth. Sometimes mass disasters excessively affect the environment, communication, road connectivity, and economics of the established living society [1,2]. When disaster strikes, rapid and effective responses are crucial and important to save the victimized population and infrastructure. It is very challenging and extremely difficult to accurately collect and process the data in an emergency situation during a disaster [3]. Generally, the accessibility to disaster-affected areas is destroyed after comprehensive damages due to a disaster like an earthquake, hurricane, or flood. In this crucial circumstance, the collection of local information and knowing the status of the various affected areas are hardly feasible for humanitarian actors (HAs) or social service organizations (SSOs) [4,5]. Even reaching out to a particular region becomes infeasible for unmanned ground vehicles (UGVs) due to the damages in the region. In most cases, some humans are trapped in the affected areas and are unable to communicate using mobile phones due to the destruction of the mobile network infrastructure. It has been observed that some cities have a tendency to catch fire after a disaster, especially overcrowded cities [6].

* Corresponding author.
E-mail addresses: santanu1109ghosh@gmail.com (S. Ghosh), pratyay_kuila@yahoo.com (P. Kuila).

https://doi.org/10.1016/j.ijdrr.2023.104067
Received 31 January 2023; Received in revised form 17 September 2023; Accepted 15 October 2023
Available online 16 October 2023

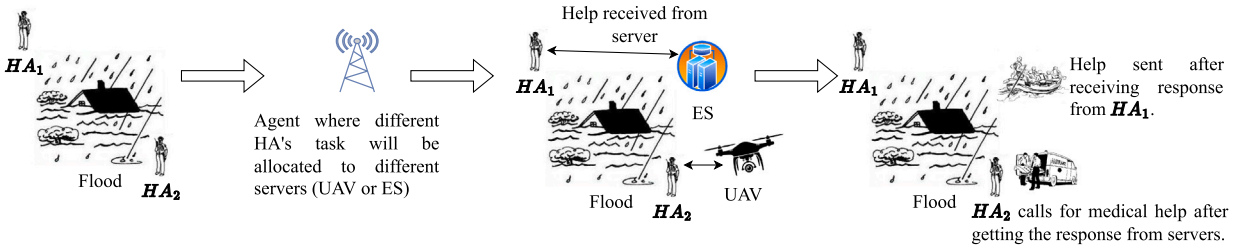


Fig. 1. UAV-assisted MEC in a disaster-affected area.

Unfortunately, it is not feasible to immediately know the exact location of the fire and start the damage control operation due to the lack of accessibility and information-sharing options.

A convolutional neural network (CNN)-based model is proposed in [6] for a fire detection system for post-disaster applications focusing on the city of İstanbul. The detection of fire, which requires the processing of some compute-intensive tasks, is not possible for local smart mobile devices (SMDs) with their limited processing capability. With the advancement of communication technology, unmanned aerial vehicles (UAVs) are found to be very suitably applicable due to their free movement in disaster-affected areas [7]. The UAVs with computation capability can be used to process the compute-intensive tasks of the SMDs. Some Internet of Things (IoT) or SMDs may be deployed well before in disaster-prone areas. The mobile phones of the locally trapped humans can also be used as a source of local data. The UAVs can efficiently collect the data from such SMDs and then pass the important data to the nearest station to timely recognize the level of damage and accordingly initiate the rescue process, as shown in Fig. 1. Moreover, the UAVs can also be used as network service providers. Thus, the deployment of UAVs with computation capabilities is a promising solution to enhance reachability and connectivity in disaster-affected areas.

Because of their limited power sources, IoT devices have energy constraints [8]. The IoT or SMDs need to frequently execute compute-intensive tasks and thereby drain their precious energy [9]. For more efficient task processing, the mobile edge computing (MEC) system provides us an effective solution by enabling the SMDs to offload the resource-intensive tasks to the edge servers (ESs) and UAVs [10]. Nevertheless, despite the energy efficiency of the SMDs, task offloading not only increases the load on the network but also imposes the inevitable and unbearable latency during task processing. The SMDs can directly communicate with the nearest ES or UAV for faster processing of tasks, but it may drain their precious energy due to the long-haul communication [11]. Furthermore, such lengthy communication may occasionally increase communication delay [12,13]. Therefore, an intelligent decision should be taken to trade-off between energy efficiency and tolerable delay in the systems [14,15]. Moreover, balancing the task-waiting load of the servers is also important [16]. The deployment of UAV-assisted MEC is a promising solution in disaster-affected areas. However, there are two critical challenges to utilizing the abilities of UAV-assisted MEC in disaster management: efficient task allocation to the processing units (PUs) or servers, and management of computational tasks and resource optimization. With the limited number of UAVs and ESs, the tasks from the SMDs are to be distributed efficiently such that the overall delay and energy consumption of the UAVs, ESs, and SMDs are minimized. This research seeks to contribute to task offloading in the field of disaster management. Such contributions may improve response capabilities, minimize the impact of disasters, and save lives.

In a multi-UAV-aided MEC network, the full offloading problem is non-deterministic polynomial complete (NP-complete). Therefore, it is not feasible to design an algorithm to provide the optimal solution in polynomial time. Note that it is not desirable to look towards the exponential algorithm for the optimal solution due to its higher computational time to make a decision. Evolutionary algorithms (EAs) like the gravitational search algorithm (GSA), genetic algorithm (GA), particle swarm optimization (PSO), etc. have attracted researchers for their ability to provide feasible solutions for complex NP-complete problems in polynomial time. GSA [17] is one of the well-applied EAs for solving various complex real-life problems [18–20]. GSA is enabled with extensive exploration and exploitation capabilities for the non-convex solution space. The offloading problem in a multi-UAV-assisted MEC for disaster applications is multi-objective in nature. The wide diversification and intensification of GSA make it attractive to be employed for the multi-objective offloading problem.

In this paper, GSA is employed in UAV-assisted MEC-enabled disaster-affected areas for energy, delay, and load balancing-aware full offloading problems (EDLAOP). The author's contributions are as follows:

- GSA is efficiently employed for full offloading in multi-UAV-assisted MEC-enabled systems. Considering the disaster-affected applications, delays in offloading, energy consumption, and load balancing of the UAVs are taken care of.
- The solution vectors or agents in GSA are efficiently encoded for the multi-UAV and multi-edge server systems. The decoding of the agents is also given. An agent provides a complete solution to the full offloading problem in multi-UAV-assisted MEC-enabled systems.

- The fitness is efficiently crafted using three parameters. Minimization of energy consumption, delay, and load balancing of the processing units (ESs and UAVs) are considered. Computational, transmission, receiving, and waiting delays are considered. Similarly, the computational, transmission, and receiving energy of a task are considered. Moreover, the propulsion energy of UAVs is also taken into account. The propulsion energy consists of the flying and hovering energy of the UAVs.
- Extensive simulation is conducted in several disaster scenarios with different experiment setups. The performance of the GSA is compared with other EAs, e.g., GA, PSO, and FWA. The statistical test, analysis of variance (ANOVA), is conducted. Moreover, the alternative average convergence rate (AACR) is measured for the GSA.

The remainder of the article is structured as follows: A literature review is shown in Section 2. The Section 3 provides the system models with problem formulation. The proposed work is discussed with illustrations in Section 4. The simulation of the proposed algorithm, along with performance evaluation and statistical analysis are done in Section 5. Section 6 summarizes the work with future directions.

2. Literature review

Research has been done using technology to provide humanitarian aid. Technology helps in providing aid faster. According to Lozano and Tien [21], the use of various technologies for comprehensive damage assessments after a disaster like an earthquake or a hurricane has rapidly increased. A fire detection system for post-disaster applications using a deep learning model is proposed in [6]. Saif et al. [22] show how UAVs can be deployed during and after disasters. Integration of blockchain technology with the Internet of Things (IoT) and the Internet of Everything (IoE) is suggested for inclusion in the disaster management framework by Javadpour et al. [23]. Few applications using artificial intelligence (AI)-based techniques, particularly for identifying and routing during disasters, are provided by Farazmehr et al. [24]. The use of technology becomes essential in the current generation for providing humanitarian aid during or after disasters.

Very few research has been done on task offloading for providing humanitarian aid during and after disasters. Shi et al. [25] have proposed a partial offloading scheme using UAV clusters. Here, the tasks are partially offloaded to a UAV, and then the UAVs collaboratively process the partially offloaded tasks. The authors have employed deep reinforcement learning (DRL) to reduce the overall energy consumption. However, the authors have ignored total delay, which is one of the most essential and crucial parameters for disaster management. This is because providing timely services in the disaster-affected areas is essential for humanitarian aid. Note that the energy consumption due to transmission, receiving of the tasks, and inter-cluster communication is overlooked [25].

Wang et al. [26] have proposed offloading for post-disaster rescue using UAVs without considering energy consumption. Here, delay is minimized by considering battery and computing resource limitations. Chaudhry and Rishiwal [27] have proposed fuzzy-based offloading for disaster management in urban and rural areas. Here, the authors have considered energy consumption and load balancing. The important parameter, delay is ignored as in [25]. A dynamic task offloading system for disaster management systems based on network status is suggested in [28]. The research has reduced the overall reaction time but ignored the limited energy resources.

Although task offloading has a great impact in disaster scenarios to provide faster humanitarian aid, as discussed in Section 1, the same is not significantly addressed. There are several important parameters like delay, energy consumption, and load balancing that are yet to be addressed simultaneously [25–28]. There are a few offloading algorithms using MEC or UAV-assisted MEC as also surveyed below.

2.1. Task offloading using MECs

Li et al. [29] have presented partial offloading with resource allocation for mobile edge servers (MESs) based on genetic algorithm (GA). The chromosome is encoded with a length of $3n$ for n number of user devices. Accordingly, the crossover and mutation operations are modified for the encoded chromosomes. Li et al. [30] propose an artificial fish swarm algorithm (AFSA)-based scheduling of tasks for wirelessly powered MEC. The main aim of this work is to reduce delay by considering the constraints of wireless power transfer (WPT) energy. In [16], authors have first formulated the offloading problem in the form of multi-objective mixed integer linear programming. Then ant colony optimization (ACO) is employed for the same. The main objectives are the minimization of energy consumption while maximizing the number of offloaded tasks.

Zhang et al. [31] suggest an effective technique for offloading. The workload offloading and power control strategies for user equipment (UE) were jointly tuned to increase the number of UEs. The authors have used dynamic programming (DP) to solve the problem. Yan et al. [32] discussed a two-user MEC network where all the wireless devices (WDs) had a set of tasks to complete. The objectives are to decrease the energy consumption of the WDs and execution delays by considering resource allocation. The authors have used Gibbs sampling technique for the same. Pham et al. [33] discuss an optimization system for offloading tasks and allocating resources by adapting a matching algorithm. By simultaneously optimizing individual calculation decisions, user transmission capacity, and computing resources, the system specifically aimed to decrease the overall computation overhead. As per Zhang et al. [34], a mechanism was formed for energy-efficient offloading in MEC-enabled 5G heterogeneous networks. A mobile device has the option of doing operations locally on the device or forwarding them to the MEC servers for remote computation. You and Tang [35] have minimized energy and delay while offloading tasks using PSO in an industrial IoTs environment.

Table 1
Summary of various parameters in the literature.

Reference	Energy	Delay	Load-Balancing	Methodologies
[16]	√	X	√	ACO
[31]	√	√	X	DP
[29]	√	X	X	GA
[32]	√	√	X	Heuristics
[33]	√	√	X	Heuristics
[38]	√	X	X	Heuristics
[39]	√	√	X	EPEC
[36]	√	√	X	FWA
[30]	√	√	X	AFSA
[40]	√	√	X	LSTM
[41]	√	√	X	Heuristics
[37]	√	√	X	PSO
[42]	√	X	X	Heuristics
[35]	√	√	X	PSO
This paper	√	√	√	GSA

Table 2
Summary of various delays used in the literature.

Reference	Waiting	Receiving	Transmission	Computation
[31]	X	X	X	√
[29]	X	X	√	√
[32]	X	√	√	√
[33]	X	√	√	√
[38]	X	X	√	√
[39]	X	X	√	√
[36]	X	X	√	√
[43]	X	X	X	√
[35]	X	X	√	√
This paper	√	√	√	√

2.2. Task offloading using UAV-assisted MEC

In this subsection, task offloading using UAV-assisted MEC, fog, and clouds are discussed. Fog computing with UAVs was discussed by Li et al. [36]. To lower the overall latency for UAV-based fog computing networks, the fireworks algorithm (FWA) for scheduling and multiple task offloading is developed. Zhang et al. [37] suggest shifting home automation application processing workloads to neighboring cloudlets. A heuristic algorithm based on PSO was used to schedule mobile services and offered a method for energy-efficient offloading for home automation applications.

Research by Zhang et al. [38] asserts that by utilizing Lagrangian duality and Successive Convex Approximation (SCA), it is possible to lower energy usage by scheduling the computation bits, time slots, allocating transmission power, and designing an efficient trajectory for the UAV. The objective of the work in [43] is to minimize task completion time using a joint optimization algorithm. The authors claim to decrease completion time by effectively alternating optimization procedures for both offloading modes. By combining the optimization of terminal device scheduling, slot size, resource distribution, and UAV trajectory, both the partial offloading and binary offloading models were taken into account. It was decided to use a non-LoS channel paradigm for UAV-ground communication. To solve the problems, the SCA, Karush–Kuhn–Tucker (KKT) conditions, and penalized technique are alternated among optimization methods. Liu et al. [39] discuss two sets of UAVs: one is a high-altitude platform UAV (HAP-UAV), and the other is a low-altitude platform UAV (LAP-UAV). HAP-UAV is allocated as the MEC server. The stochastic equilibrium problem of equilibrium programs with equilibrium constraints (EPEC) models is proposed to maximize profit for HAP-UAV and minimize LAP-UAV's cost. Here, offloading costs are taken into account. The uplink and downlink delays are minimized in the research by Liu et al. [44]. Here, the authors have proposed two novel multi-stage resource allocation algorithms. Wu et al. [40] combine the UAV location optimization method with the LSTM-based task prediction algorithm to offer an energy-efficient optimization approach based on a three-layer offloading technique. In an effort to decrease latency and consumption of energy, the research in [41] implements agent-based task offloading in UAV-assisted MEC. Du et al. [42] focus on resource allocation and computation offloading in hybrid cloud and fog computing systems using two heuristics. It presents a suboptimal, low-complexity approach to handle the joint optimization as a mixed-integer non-linear programming (MINLP) problem.

A summary of the considered parameters in the literature is given in Table 1. The novelties of the proposed work in contrast to the existing research are given below:

- To the best of our knowledge, this is the first research work on UAV-assisted task offloading for MEC systems considering all the parameters as mentioned in Tables 1, 2, and 3 for disaster-affected applications.
- In this research, we have considered most of the relevant and essential parameters of energy consumption and delay. These parameters are overlooked in the existing research works [25–28] and accordingly, their mathematical models are designed. Our proposed model has considered all the mentioned parameters as per Tables 2 and 3.

Table 3
Summary of various energy consumption in the literature.

Reference	Propulsion	Receiving	Transmission	Computation
[16]	X	X	√	√
[38]	√	X	√	√
[37]	X	X	√	√
[30]	X	X	√	√
[40]	√	X	√	√
[41]	√	X	√	√
[42]	X	X	√	√
[34]	X	X	√	√
[35]	X	X	√	√
This paper	√	√	√	√

Table 4
Notation and meanings.

Notation	Meaning
π_j	jth task
τ^c_{hover}	Hovering delay of u_c
C_b	Computation capacity of p_b in cycles/bit
e^c_{fly}	Flying energy of u_c
e^c_{hover}	Hovering energy of u_c
e_{prop}	Propulsion energy of UAV
E_{total}	Total energy
f_b	CPU frequency of p_b
n	Number of task
p_b	bth PU
P_{Size}	Population size
s_j	jth SMD
T_{total}	Total delay
$ti^{bj}_{rec}, e^{bj}_{rec}$	Receiving delay and energy from p_b to s_j
$ti^{jb}_{comp}, e^{jb}_{comp}$	Computation delay and energy of π_j in p_b
$ti^{jb}_{tran}, e^{jb}_{tran}$	Transmission delay and energy of π_j to p_b
ti^{j}_{wait}	Waiting delay of π_j
ti^{b}_{wait}	Waiting delay of p_b
ti^{j}_{off}	Offloading delay of task π_j
$Load$	Load of all the PUs
u_c	cth UAV

- Different types of evolutionary algorithms (EAs) were compared with the GSA algorithm based on the proposed novel fitness function.

3. System model and problem formulation

It is assumed there is a disaster-affected area where smart mobile devices (SMDs) or IoT nodes are deployed to track and monitor the effects of the disaster. Some edge servers (ESs) are located within the proximity range of the SMDs. Moreover, UAVs are also deployed for faster collection of data from the SMDs and communication to the ESs. The application scenario is assumed to have three computation layers. The first layer is the SMDs or IoT layer, which consists of n number of SMDs, $S = \{s_1, s_2, \ldots, s_n\}$. The next layer is the UAV layer, which comprises w number of UAVs: $U = \{u_1, u_2, \ldots, u_w\}$. The third layer is the edge layer with t number of ESs, $E = \{e_1, e_2, \ldots, e_t\}$. Therefore, we have a total $t + w = q$ number of processing units (PUs), as UAVs and ESs, to process the application requests received from the SMDs. The set of q number of PUs is represented as $P = \{p_1, p_2, \ldots, p_q\}$. It is assumed that each SMD generates a task. Therefore, there are n number of tasks, $Ts = \{\pi_1, \pi_2, \ldots, \pi_n\}$ that are generated by the n number of SMDs.

It is assumed that the SMDs and UAVs are moving around the affected scenarios and the ESs are static at their position. As shown in Fig. 2, an intelligent agent (IA) controls and decides the efficient communication between the SMDs and all the UAVs and ESs. The IA acts as a decision maker regarding sending the tasks from the SMDs to either the UAVs or the ESs. Here, it is assumed that all the servers and SMDs are within the communication range of each other. The UAVs are flying at a fixed height and each SMD has a single task request at a time. Table 4 is referred for different notations. Various models are described in the following subsections.

3.1. Delay model

For disaster applications, delay is one of the important parameters to be considered. There are several reasons behind the delay such as: transmission, computation, receiving, and waiting/queuing delays. All such delays are discussed below.

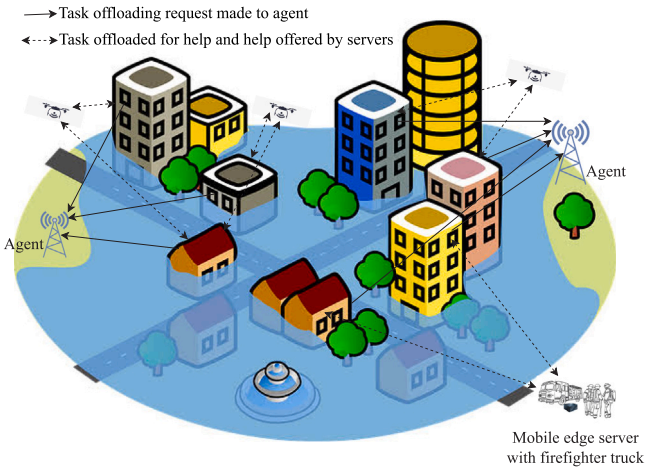


Fig. 2. System model.

- Transmission Delay: It refers to the delay during transmission of data from SMD to servers. Let us assume that an SMD s_j transmits L_{in} bits of data to the PU p_b by using a channel with data transmission rate R_{jb} bits per second (bps). Therefore, the transmission delay can be mathematically represented as [31,38]:

$$ti_{tran}^{jb} = \frac{L_{in}}{R_{jb}} \tag{1}$$

  where, $R_{jb} = B_0 \times \log_2(1 + SNC)$, B_0 is the bandwidth of the channel. The SNC value can be computed as in [31,38].
- Receiving Delay: Similar to the transmission delay, the receiving delay can be computed as follows for L_{out} bits of received data.

$$ti_{rec}^{bj} = \frac{L_{out}}{R_{jb}} \tag{2}$$

- Computation Delay: A computation delay of π_j on the p_b is the duration of processing L_{in} bits of data by the p_b. It can be computed as [31]:

$$ti_{comp}^{jb} = \frac{L_{in} \times C_b}{f_b} \tag{3}$$

  where C_b is required CPU cycles (CPU cycles/bit) to process one bit. f_b is the computation capacity of the p_b, i.e., number of CPU cycles per second.
- Waiting Delay: Let the p_b has already l number of tasks in its job queue to execute. All these tasks are offloaded from other SMDs. Meanwhile, a new task π_j is offloaded to p_b. Therefore, the π_j needs to wait until all the previous tasks are completely executed in p_b. Such waiting time can be estimated as:

$$ti_{wait}^{j} = \sum_{r=1}^{l} ti_{comp}^{rb} \tag{4}$$

Let us define a Boolean variable to denote the offloading decision as follow:

$$\alpha_{jb} = \begin{cases} 1, & \text{if } \pi_j \text{ is executed at } p_b \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

Now, the corresponding offloading delay for π_j while offloaded to p_b can be estimated as:

$$ti_{off}^{j} = \alpha_{jb}\{ti_{tran}^{jb} + ti_{wait}^{j} + ti_{comp}^{jb} + ti_{rec}^{bj}\} \tag{6}$$

Therefore, the total delay to offload all n number of tasks can be expressed as:

$$T_{total} = \sum_{j=1}^{n} ti_{off}^{j} \tag{7}$$

3.2. Energy model

Energy is consumed due to: transmission, computation, receiving, and the propulsion of UAVs.

- Transmission Energy: This is consumed energy during transmission of data from SMD to the corresponding PU [31,38]. It can be estimated as follows.

$$e^{jb}_{tran} = P_{jb} \times ti^{jb}_{tran} \tag{8}$$

where P_{jb} is the required power per unit of time to transmit a task from s_j to p_b.

- Receiving Energy: Similar to the transmission energy, the receiving energy can be calculated as follows.

$$e^{bj}_{rec} = P_{bj} \times ti^{bj}_{rec} \tag{9}$$

- Computation Energy: It is the required energy to compute or execute the offloaded task π_j by the p_b. It can be estimated as [45]:

$$e^{jb}_{comp} = k \times f_b^3 \times ti^{jb}_{comp} \tag{10}$$

where, k is the constant. Therefore, the total energy usage for offloading π_j to the p_b is the summation of the energy consumed due to transmission (e^{jb}_{tran}), computation (e^{jb}_{comp}) and receiving (e^{bj}_{rec}). Therefore, it can be expressed as:

$$e^{j}_{off} = \alpha_{jb}\{e^{jb}_{tran} + e^{jb}_{comp} + e^{bj}_{rec}\} \tag{11}$$

- Propulsion Energy: It is the energy used by a UAV during flying and hovering. During hovering, the UAV consumes energy due to the rotation of the rotary blade and induced power to hover at a fixed height. Let the UAV u_c hovers for τ^c_{hover} unit of time. P_0 and P_1 are the energy consumed per unit time due to the rotation of the rotary blade and induction. Therefore, the hovering energy (e^c_{hover}) of u_c can be computed as follow:

$$e^c_{hover} = (P_0 + P_1) \times \tau^c_{hover} \tag{12}$$

The hovering time, τ^c_{hover} is equivalent to the total amount of time spent in sending, waiting, computing, and receiving of all the tasks by the UAV as shown below.

$$\tau^c_{hover} = \sum_{v=1}^{y} [ti^{vc}_{tran} + ti^{c}_{wait} + ti^{c}_{comp} + ti^{cv}_{rec}] \tag{13}$$

Three factors are responsible for the flying energy consumption: the blade profile, induced, and parasite part [38,46]. While the induced part is connected to induction power, the blade profile component is connected to the power used to rotate rotary blades. The parasite component is connected to the fuselage drag ratio (d_r), rotor disc area (G), air density (ρ), and solidity of the rotor (S). UAV's vertical movement is considered in the fourth part which is associated with power per unit time, P_2. The total flying energy can be formulated as per the following Eq. (14).

$$e^c_{fly} = \tau\Big[P_0\Big(1 + \frac{3\|V_{U,xy}\|^2}{V^2_{tip}}\Big) + \frac{1}{2} d_r S \rho G \|V_{U,xy}\|^3 + P_1 \sqrt{\Big(\sqrt{1 + \frac{\|V_{U,xy}\|^4}{4V_0^2}} - \frac{\|V_{U,xy}\|^2}{2V_0^2}\Big)} + P_2\|V_{U,z}\|\Big] \tag{14}$$

where, V_{tip}, V_0 are rotor blade tip speed and mean rotor induced velocity in hover. $V_{U,xy}, V_{U,z}$ are horizontal and vertical velocities of UAV. τ specifies the flying time for u_c. Therefore, the propulsion energy of the u_c is calculated as:

$$e^c_{prop} = e^c_{fly} + e^c_{hover} \tag{15}$$

The total propulsion energy of all the UAVs is calculated as $e_{prop} = \sum_{c=1}^{w} e^c_{prop}$.

Note that in some applications, all the tasks may be offloaded to the nearest ES. Therefore, none of the UAVs are in use. Let us define a Boolean variable to denote such circumstances as follows:

$$\lambda_c = \begin{cases} 1, & \text{if } \exists \pi_j \text{ that is offloaded to UAV } u_c \\ 0, & \text{otherwise} \end{cases} \tag{16}$$

Therefore, total energy consumption can be estimated as follows:

$$E_{total} = \sum_{j=1}^{n} e^j_{off} + \sum_{c=1}^{w} \lambda_c . e^c_{prop} \tag{17}$$

3.3. Load balancing

A PU's load represents the sum of all tasks' waiting times with respect to the PU. Assume, the p_b is offloaded with l number of tasks. Therefore, the load of p_b is the total waiting time of all l number of tasks on p_b and it can be estimated as:

$$load^b = \sum_{r=1}^{l} ti^r_{wait} \tag{18}$$

Let $\mu = \frac{1}{q}\sum_{b=1}^{q} load^b$ be the average load of the PUs. Then the load of the system is expressed as:

$$Load = \sum_{b=1}^{q}(load^b - \mu)^2 \tag{19}$$

3.4. Problem formulation

Let us state the energy, delay, and load balancing aware full offloading problem (EDLAOP) as, "Given, n number of tasks, $\{\pi_1, \pi_2, \ldots, \pi_n\}$ those are generated by the n number of SMDs and $q = t + w$ number of processing units (PUs) including t number of ESs and w number of UAVs, the problem is to fully offload all the n number of tasks to the q number PUs by minimizing the total energy, delay, and load balancing of the PUs. It is assumed that an SMD generates a task only". The problem is mathematically formulated as follows.

Objective 1: Minimize total delay (T_{total}) (20)

Objective 2: Minimize total energy (E_{total}) (21)

Objective 3: Minimize total load ($Load$) (22)

Subject to,

$$\Sigma_{b=1}^{q}\alpha_{jb} = 1, \forall j, 1 \le j \le n \tag{23}$$

$$\alpha_{jb} \in \{0,1\}, \forall j, 1 \le j \le n, \forall b, 1 \le b \le q \tag{24}$$

$$\lambda_c \in \{0,1\}, \forall c, 1 \le c \le w \tag{25}$$

$$c = b - t, \forall b > t \tag{26}$$

$$\lambda_c = \lceil \frac{1}{n}\Sigma_{j=1}^{n}\alpha_{jb} \rceil, \forall b > t \tag{27}$$

$$\Sigma_{c=1}^{w}\lambda_c \le \Sigma_{j=1}^{n}\Sigma_{b=1}^{q}\alpha_{jb} \tag{28}$$

$$q = t + w \tag{29}$$

The Eqs. (20)–(22) denote the objectives of the assumed offloading problem (EDLAOP). T_{total}, E_{total} and $Load$ can be computed by the Eqs. (7), (17) and (19) respectively. In this formulation, n, q, t, and w are constant and denote the number of tasks, PUs, ESs, and UAVs respectively as mentioned in Section 3 and problem statement. The symbols, b, j and c are used to denote the specific PU number (p_b), task number (π_j) and UAV number (u_c). Now, the constraints (23)–(24) restrict that a task can be fully offloaded to one PU only. When a task (π_j) is offloaded to a PU (p_b), then Boolean variable α_{jb} becomes 1. As per constraint (23) and (24), it cannot be offloaded to another PU. This is because, $\Sigma_{b=1}^{q}\alpha_{jb} = 1$. The constraint (25) denotes whether a UAV is used during the operation or not. It says if the Boolean variable $\lambda_c = 1$, then the UAV u_c is utilized. The constraint (26) indicates that initial $[1...t]$ PUs are considered as ESs and remaining $[(t+1)...q]$ PUs are considered as UAVs. The constraints (27)–(28) ensure cohesive relation between two Boolean variables, λ_c and α_{jb}. As mentioned before, the constraint (29) indicates total q PUs is the combination of t number of ESs and w number of UAVs.

There are a few constants and variables that are used for the calculation of T_{total}, E_{total} and $Load$ as per the Eqs. (7), (17) and (19) respectively. The constant parameters to compute delay, energy, and load are $L_{in}, L_{out}, B_0, P_{jb}, P_{bj}, k, P_0, P_1, P_2, V_{U,xy}, V_0, V_{U,z}$ and V_{tip} as discussed in Sections 3.1–3.3. The corresponding values of these parameters are given in the Table 9. The parameters, SNC, C_b, and f_b are dependent on the processing unit (PU), and their respective values are also given in Table 9. Here, the variable parameters are the two Boolean variables, α_{jb} and λ_c as discussed by the Eqs. (5) and (16) respectively. These two variables influence the energy and delay calculation as per the Eqs. (6), (11), and (17). Here, our objective is to offload the tasks to the right PUs such that the total delay (T_{total}), energy (E_{total}), and load ($Load$) gets minimized. By the Boolean variables λ_c and α_{jb}, the offloading decision is made.

Remark 3.1. It is noteworthy that the aforementioned problem is both a mixed-integer non-linear programming (MINLP) problem and a multi-objective one. The objective functions are non-linear and the variables, α_{jb} and λ_c are constrained to be integer.

Remark 3.2. As seen in Table 1, very few research works considered load balancing. Moreover, the existing models did not use all the delay and energy parameters as shown in Tables 2, and 3. Therefore, their mathematical models did not consider load balancing and all the delay and energy parameters in contrast to the proposed mathematical model. The proposed model has considered all the delay and energy components by the objective variables T_{total} and E_{total} as given in the Eqs. (6), (7), (11), (15) and (17). Note that the considered energy and delay parameters have a huge impact on disaster-affected applications as discussed in Section 1.

Theorem 1. *The EDLAOP is NP-Complete.*

Proof. The bin packing problem (BPP) is known to be NP-complete. Consider the energy, delay, and load balancing aware offloading problem (EDLAOP) which is similar to BPP. As a result, the EDLAOP is a generalization of the BPP or BPP $\le_p$ EDLAOP. Here, $\le_p$ refers to polynomial-time reduction. Therefore, all the problems in NP are polynomial-time reducible to EDLAOP, i.e., $\forall A \in$ NP, $A \le_p$ EDLAOP. Additionally, EDLAOP $\in$ NP since it can be verified in polynomial time. Thus, it can be said that EDLAOP is NP-complete. □

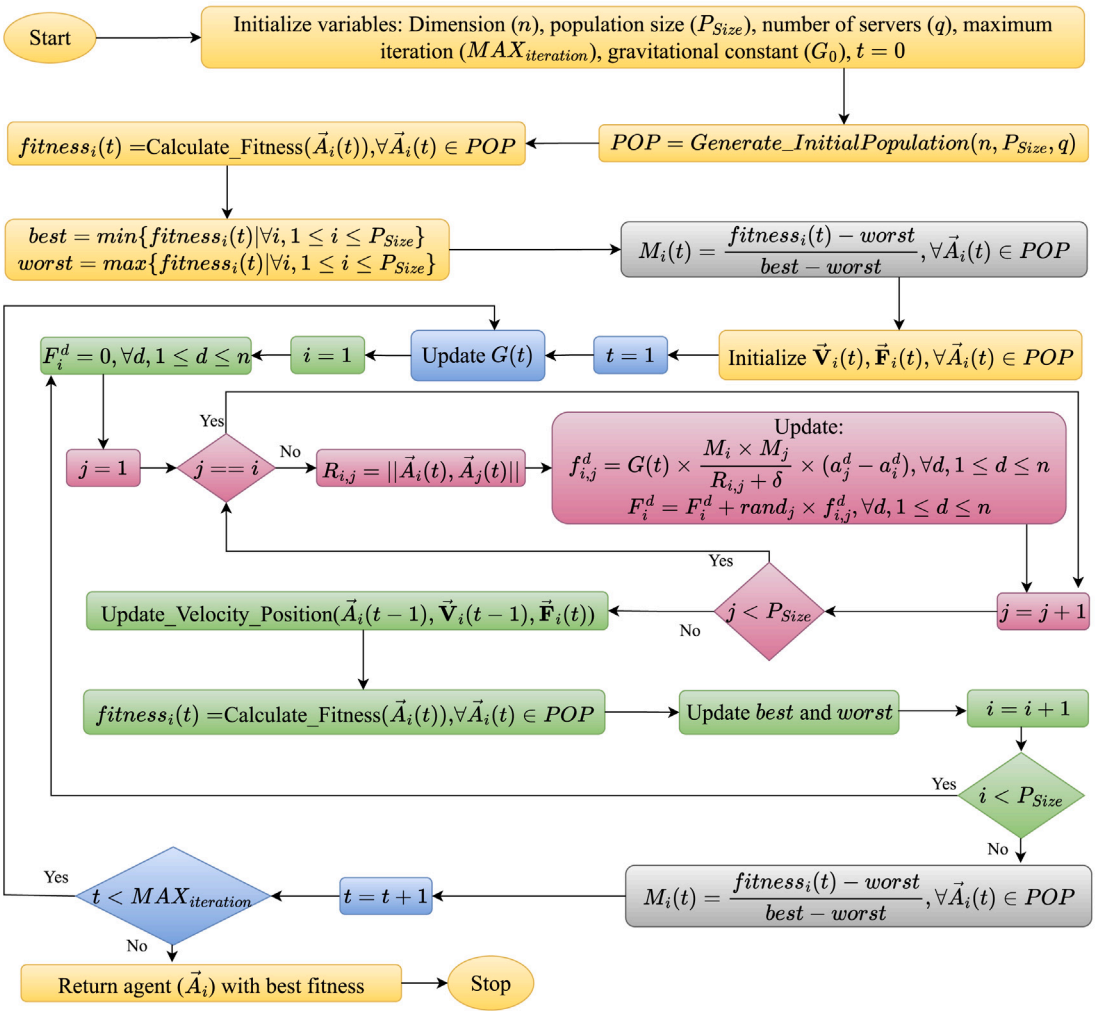


Fig. 3. GSA flowchart.

4. Proposed work

In this paper, a GSA-based novel energy, delay, and load balancing-aware full offloading technique is proposed for UAV-assisted MEC-enabled disaster-affected areas. The proposed work is discussed in the following subsections.

4.1. An overview of GSA

GSA is a meta-heuristic optimization technique. It is built on the concept of the Newton's law of gravity. According to the law, each object in the universe attracts every other objects in the cosmos. The force of gravity acting between two objects is directly proportional to their masses and inversely proportional to the distance among the objects. Like other EAs, GSA is also a population-based optimization method. An agent is used to represent a possible solution to the problem. The flow of GSA is given in Fig. 3, and the corresponding phases are discussed in the following subsections based on the considered offloading problem.

4.2. Initialization of agent and generation of initial population

In GSA, a solution vector is denoted as an agent. An agent can be represented as, $\vec{A}_i = \{a_i^1, a_i^2, \ldots, a_i^n\}$, where, n denotes the dimension and a_i^d is dth component of $\vec{A}_i$. Here, the agent's dimension is kept the same as the number of tasks, i.e., n. Each component (a_i^j) of $\vec{A}_i$ can be initialized as:

$$a_i^j = random(0, 1) \times q \tag{30}$$

Table 5
A population for Illustration 4.1.

Agent ($\vec{A}_i$)	a_i^1	a_i^2	a_i^3	a_i^4	a_i^5
$\vec{A}_1$	1.665	0.626	1.607	0.069	2.037
$\vec{A}_2$	1.134	0.626	1.607	2.302	0.421
$\vec{A}_3$	2.123	3.478	2.035	0.365	1.433

Table 6
Decoding of the agent $\vec{A}_3$ from Table 5.

Agent ($\vec{A}_i$)	a_i^1	a_i^2	a_i^3	a_i^4	a_i^5
$\vec{A}_3$	2.123	3.478	2.035	0.365	1.433
Offloaded PU	p_3	p_4	p_3	p_1	p_2

where q denotes the number of servers or PUs. Therefore, the range of each component of the agents is $(0, q)$. A population is a collection of randomly produced P_{Size} number of agents. Initially, a population is generated as per the Algorithm 1.

Illustration 4.1. Let us consider a system with 4 servers or PUs and 5 tasks. Therefore, the dimension of the agent is $n = 5$. Let us create a population of size three, i.e., $P_{Size} = 3$. The population can be created using the Algorithm 1. A population of three agents is shown in the Table 5.

Algorithm 1 : Generate_InitialPopulation(n, P_{Size}, q)

Input: (1) n: Dimension of the agents.
(2) P_{Size}: Population size.
(3) Number of servers or PUs, q.
Output: Initial population, POP.

```
Initialize POP = { }
Initialize A_i = {a_i^1, a_i^2, ..., a_i^n}, ∀i, 1 ≤ i ≤ P_Size.
for i = 1 to P_Size
    for d = 1 to n
        a_i^d = random(0, 1) × q.
    end for
    POP = POP ∪ A_i
end for
return POP.
```

Lemma 4.1. *The time complexity of Generate_InitialPopulation(n, P_{Size}, q) is $\Theta(P_{Size} \times n)$.*

Proof. An agent of dimension n can be generated in $\Theta(n)$ time. Therefore, a population of size P_{Size} can be generated in $\Theta(P_{Size} \times n)$ time as per the Algorithm 1. □

4.3. Agent decoding

The offloading problem should always have a full solution generated by an agent. Here, each agent ($\vec{A}_i$) can be decoded as the assignment of all the tasks to PUs. The task π_j can be assigned to the $\lceil a_i^j \rceil^{th}$ server or PU. Here, the system is considered with $q = t + w$ PUs with t number of ESs and w number of UAVs. Therefore, initial $[1, \ldots, t]$ values are considered as ES and the next $[t + 1, \ldots, q]$ are considered as UAV.

Illustration 4.2. Let us consider the agent $\vec{A}_3$, which is generated as discussed in the Illustration 4.1 and Table 5. It can be observed that the 2nd component of $\vec{A}_3$ is 3.478. Therefore, the 2nd task, π_2 can be assigned to the $\lceil 3.478 \rceil^{th} = 4^{th}$ PU, i.e., p_4. Similarly, all the task assignments can be decoded and the complete decoding of $\vec{A}_3$ is shown in Table 6.

4.4. Computation of fitness

The effectiveness of the solution is reflected by the fitness value. Let us consider an agent, $\vec{A}_i = \{a_i^1, a_i^2, \ldots, a_i^n\}$ which is first decoded as discussed in Section 4.3. As per the decoding, the task π_j is offloaded to the PU p_b such that $b = \lceil a_i^j \rceil$. Based on the decoding, the fitness of $\vec{A}_i$ is calculated by considering the following objectives:

- *Objective 1:* The first objective is to reduce the total delay as shown in Eq. (31).

$$\text{Minimize } T_{total} = \sum_{\forall a_i^j \in \vec{A}_i} \alpha_{jb} \left\{ ti_{tran}^{jb} + ti_{wait}^{j} + ti_{comp}^{jb} + ti_{rec}^{bj} \right\} \tag{31}$$

- *Objective 2:* The second objective is to reduce the total energy as shown in Eq. (32).

$$\text{Minimize } E_{total} = \sum_{\forall a_i^j \in \vec{A}_i} \alpha_{jb} \left\{ e_{tran}^{jb} + e_{comp}^{jb} + e_{rec}^{bj} \right\} + \sum_{c=1}^{w} \lambda_c \left\{ e_{fly}^{c} + e_{hover}^{c} \right\} \tag{32}$$

- *Objective 3:* The third objective is to minimize the load of the PUs as shown in Eq. (33).

$$\text{Minimize } Load = \sum_{b=1}^{q} (load^b - \mu)^2 \tag{33}$$

Now, the weight sum approach (WSA) [47] is used to merge the three objectives as follows.

$$\text{Minimize } Fitness = \omega_1 \times T_{total} + \omega_2 \times E_{total} + \omega_3 \times Load \tag{34}$$

where, ω_1, ω_2 and ω_3 are the weight values of the corresponding objectives such that, $\sum \omega_i = 1$ and $\forall \omega_i, 0 \leq \omega_i \leq 1$. The best solution vector is the agent with the lowest fitness value. The fitness computation is given by the Algorithm 2.

Algorithm 2 : Calculate_Fitness($\vec{A}_i(t)$)

Input: An agent, $\vec{A}_i(t) = \{a_i^1(t), a_i^2(t), ..., a_i^n(t)\}$ and $q = t + w$.
Output: Fitness value for $\vec{A}_i(t)$.

1: Initialize ω_1, ω_2 and ω_3.
2: Initialize $T_{total} = 0, E_{task} = 0, E_{UAV} = 0, Load = 0$.
3: Initialize $load(b) = 0.0, \forall b, 1 \leq b \leq q$.
4: Initialize $FinishTime(b) = 0.0, \forall b, 1 \leq b \leq q$.
5: **for** each $a_i^j(t) \in \vec{A}_i(t)$
6: $b = \lceil a_i^j(t) \rceil$ // Decoding as per the Section 4.3.
7: $T_{total} = T_{total} + \{ti_{tran}^{jb} + ti_{wait}^{j} + ti_{comp}^{jb} + ti_{rec}^{bj}\}$ // Update delay.
8: $E_{task} = E_{task} + \{e_{tran}^{jb} + e_{comp}^{jb} + e_{rec}^{bj}\}$
9: **if** $b \geq t + 1$ **then** // Offloaded to UAV.
10: $E_{UAV} = E_{UAV} + \{e_{fly}^{c} + e_{hover}^{c}\}$ // Update UAV energy consumption.
11: **end if**
12: $load(b) = load(b) + FinishTime(b)$
13: $FinishTime(b) = FinishTime(b) + ti_{comp}^{jb}$
14: **end for**
15: $\mu = \frac{1}{q} \sum_{b=1}^{q} load(b)$ // Calculate average load.
16: $Load = \sum_{b=1}^{q} (load(b) - \mu)^2$ // Calculate Load by Eq. (19).
17: $E_{total} = E_{task} + E_{UAV}$ // Update total energy consumption.
18: $Fitness = \omega_1 \times T_{total} + \omega_2 \times E_{total} + \omega_3 \times Load$.
19: **return** $Fitness$

Lemma 4.2. *The time complexity of Calculate_Fitness($\vec{A}_i(t)$) is $\Theta(n)$.*

Proof. The delay and energy with respect to an agent of dimension n can be computed in $\Theta(n)$ time. For the calculation of load, it takes $\Theta(q)$ time. As $n > q$, the total complexity of Calculate_Fitness($\vec{A}_i(t)$) is $\Theta(n)$ as per the Algorithm 2. □

4.5. Computation of mass

Mass ($M_i(t)$) of an agent $\vec{A}_i(t)$ depends on the fitness value of the agent. The better is fitness, the higher the mass. The mass can be calculated as per Eq. (35).

$$M_i(t) = \frac{fitness_i(t) - worst}{best - worst} \tag{35}$$

where *best* and *worst* are the best and worst fitness values over the population. For the considered minimization problem, *best* and *worst* are computed as per the following equations.

$$best = min\{fitness_i(t) | \forall i, 1 \leq i \leq P_{Size}\} \tag{36}$$

$$worst = max\{fitness_i(t) | \forall i, 1 \leq i \leq P_{Size}\} \tag{37}$$

Algorithm 3 : Calculate_Force($\vec{A}_i(t)$)

Input: An agent, $\vec{A}_i(t) = \{a_i^1(t), a_i^2(t), ..., a_i^n(t)\}$

Output: Overall force on $\vec{A}_i$: $\vec{\mathbf{F}}_i(t+1) = \{F_i^1(t+1), ..., F_i^d(t+1), ..., F_i^n(t+1)\}$

```
for d = 1 to n
    F_i^d(t+1) = 0
    for j = 1 to P_Size
        if j ≠ i then
            R_{i,j}(t) = ||A_i(t), A_j(t)||
            f_{i,j}^d(t+1) = G(t) × (M_i(t) × M_j(t)) / (R_{i,j}(t) + δ) × (a_j^d(t) − a_i^d(t))
            F_i^d(t+1) = F_i^d(t+1) + rand_j × f_{i,j}^d(t+1)
        end if
    end for
end for
return F⃗_i(t+1)
```

4.6. Applied force

According to the Newton's Law of Gravity, each object attracts every other objects in the cosmos based on their mass and distance. By mimicking this concept, each agent $\vec{A}_i(t)$ faces the gravitational force from all other agents $\vec{A}_j(t)$ of the population. Such force is directly proportional to the product of the mass of the agents and inversely proportional to the Euclidean distance among the agents. Let, $f_{i,j}^d(t)$ denote the gravitational force among $\vec{A}_i(t)$ and $\vec{A}_j(t)$ towards dth component of the agents. $f_{i,j}^d(t)$ can be calculated as follows.

$$f_{i,j}^d(t+1) = G(t) \times \frac{M_i(t) \times M_j(t)}{R_{i,j} + \delta} \times (a_j^d(t) - a_i^d(t)) \tag{38}$$

where, $R_{i,j}(t) = \|\vec{A}_i(t), \vec{A}_j(t)\|$ refers to the Euclidean distance among $\vec{A}_i(t)$ and $\vec{A}_j(t)$. δ refers to a small positive constant to restrict the denominator that should not be zero in the case when the $R_{i,j}(t) = 0$. Therefore, the total force towards the dth component of $\vec{A}_i(t)$ from the rest of the agents of the population can be calculated as per Eq. (39).

$$F_i^d(t+1) = \sum_{j=1, j \neq i}^{P_{Size}} rand_j \times f_{i,j}^d(t+1) \tag{39}$$

where $rand_j$ is a random number between 0 and 1. The force is calculated by the Algorithm 3. In each iteration t, the $G(t)$ is updated using Eq. (40).

$$G(t) = G(0) \times e^{\left(\frac{-\alpha \times t}{MAX_{ite}}\right)} \tag{40}$$

where, $G(0)$ is the initial value, MAX_{ite} is total number of iteration and α is descending coefficient or gravitational scaling factor.

Lemma 4.3. *The time complexity of Calculate_Force($\vec{A}_i(t)$) is $\Theta(n \times P_{Size})$.*

Proof. To measure the total force on dth component, the gravitational forces from all other agents of the population need to be measured. Therefore, to measure the complete force on all the n components of the agent it takes $\Theta(n \times P_{Size})$ time as per the Algorithm 3. □

4.7. Calculation of acceleration, velocity and position

The position of all the agents is updated in each iteration for the whole population. It is expected that the new position will be comparably better than the previous position in terms of the fitness value. The new position is updated by the velocity of the agents in the current iteration and the velocity is updated by the acceleration. Let $ac_i^d(t)$ denote the acceleration of the dth component of $\vec{A}_i$ at tth iteration. It is calculated as per Eq. (41).

$$ac_i^d(t) = \frac{F_i^d(t)}{M_i(t)} \tag{41}$$

Algorithm 4 : Update_Velocity_Position($\vec{A}_i(t-1), \vec{\mathbf{V}}_i(t-1), \vec{\mathbf{F}}_i(t)$)

Input: An agent, $\vec{A}_i(t-1)$ with velocity $\vec{\mathbf{V}}_i(t-1)$ and updated force, $\vec{\mathbf{F}}_i(t)$
Output: Updated velocity, $\vec{\mathbf{V}}_i(t)$ and position, $\vec{A}_i(t)$.

for $d = 1$ to n
  $ac_i^d(t) = \dfrac{F_i^d(t)}{M_i(t)}$
  $v_i^d(t) = v_i^d(t-1) + ac_i^d(t)$
  $a_i^d(t) = a_i^d(t-1) + v_i^d(t)$
end for
return $\vec{A}_i(t), \vec{\mathbf{V}}_i(t)$

After calculating the acceleration, the velocity is updated as per Eq. (42).

$$v_i^d(t+1) = v_i^d(t) + ac_i^d(t+1) \tag{42}$$

After updating the velocity, the position is also updated using Eq. (43).

$$a_i^d(t+1) = a_i^d(t) + v_i^d(t+1) \tag{43}$$

The calculation of acceleration, velocity, and updating of position is given by the Algorithm 4. The pseudo-code of GSA is shown in the Algorithm 5 and the flowchart is given in Fig. 3.

Algorithm 5 : GSA(n, P_{Size}, q)

Input: (1) n: Dimension of the agents.
(2) P_{Size}: Population size.
(3) Number of servers or PUs, q.
Output: Best agent of the population.

Initialize $t = 0$.
POP=Generate_InitialPopulation(n, P_{Size}, q). // Algorithm 1.
$fitness_i(t)$=Calculate_Fitness($\vec{A}_i(t)$), $\forall \vec{A}_i(t) \in POP$. // Algorithm 2.
$best = min\{fitness_i(t) | \forall i, 1 \le i \le P_{Size}\}$. // Minimization problem.
$worst = max\{fitness_i(t) | \forall i, 1 \le i \le P_{Size}\}$.
$M_i(t) = \dfrac{fitness_i(t) - worst}{best - worst}, \forall \vec{A}_i(t) \in POP$.
Initialize $\vec{\mathbf{V}}_i(t) = \{v_i^1(t), v_i^2(t), ..., v_i^n(t)\}, \forall \vec{A}_i(t) \in POP$ // Initially, $v_i^d(t) = 0, \forall d, 1 \le d \le n$.
Initialize $\vec{\mathbf{F}}_i(t) = \{F_i^1(t), F_i^2(t), ..., F_i^n(t)\}, \forall \vec{A}_i(t) \in POP$ // Initially, $F_i^d(t) = 0, \forall d, 1 \le d \le n$.
for $t = 1$ to MAX_{ite}
  Update $G(t)$. // Using equation (40).
  for $i = 1$ to P_{Size}
    $\vec{\mathbf{F}}_i(t)$=Calculate_Force($\vec{A}_i(t-1)$) // Algorithm 3.
    $\{\vec{A}_i(t), \vec{\mathbf{V}}_i(t)\}$=Update_Velocity_Position($\vec{A}_i(t-1), \vec{\mathbf{V}}_i(t-1), \vec{\mathbf{F}}_i(t)$) // Algorithm 4.
    $fitness_i(t)$=Calculate_Fitness($\vec{A}_i(t)$). // Algorithm 2.
    if $best \ge fitness_i(t)$ **then** // For minimization problem.
      $best = fitness_i(t)$
    end if
    if $worst \le fitness_i(t)$ **then** // For minimization problem.
      $worst = fitness_i(t)$
    end if
  end for
  $M_i(t) = \dfrac{fitness_i(t) - worst}{best - worst}, \forall \vec{A}_i(t) \in POP$.
end for
$BestAgent = \text{Argmin}\{fitness_i(t) | \forall \vec{A}_i(t) \in POP\}$. // Minimization problem.
Return $BestAgent$.

Remark 4.1. After getting the updated position from Algorithm 4, there needs to be a check-bound operation. This is because the updated position $a_i^d(t+1)$ might get less than 0 or bigger than q which is not desirable. The check-bound operation fixes this. If the updated position $a_i^d(t+1) > q$, then $a_i^d(t+1)$ is set to q, which means the task π_d is assigned to the PU p_q. If the updated position $a_i^d(t+1) < 0$, then $a_i^d(t+1)$ is set to 1, which means the π_d is assigned to p_1.

Table 7
Check bound of agent $\vec{A}_3$ after running Algorithm 4.

Agent ($\vec{A}_i$)	a_i^1	a_i^2	a_i^3	a_i^4	a_i^5
$\vec{A}_3$	12.123	1.012	−33.478	2.033	1.733
Allocated PU	**4.000**	1.012	**1.000**	2.033	1.733

Table 8
Complexity of different phases in GSA.

Module	Complexity	Frequency	Total complexity
Generate_InitialPopulation(n, P_{Size}, q)	$\Theta(P_{Size} \times n)$	1	$\Theta(P_{Size} \times n)$
Calculate_Fitness($\vec{A}_i(t)$)	$\Theta(n)$	$\Theta(P_{Size} \times MAX_{ite})$	$\Theta(P_{Size} \times MAX_{ite} \times n)$
Calculate_Force($\vec{A}_i(t)$)	$\Theta(n \times P_{Size})$	$\Theta(P_{Size} \times MAX_{ite})$	$\Theta(P_{Size}^2 \times MAX_{ite} \times n)$
Update_Velocity_Position($\vec{A}_i(t-1), \vec{V}_i(t-1), \vec{F}_i(t)$)	$\Theta(n)$	$\Theta(P_{Size} \times MAX_{ite})$	$\Theta(P_{Size} \times MAX_{ite} \times n)$

Illustration 4.3. Let us consider the agent $\vec{A}_3$, whose position is updated using Algorithm 4. Let the updated position for $\vec{A}_3$ be $\{12.123, 1.012, -33.478, 2.033, 1.733\}$. It can be seen that a_3^1 is bigger than the number of available servers. The value of a_3^1 gets updated as 4 after the check-bound operation. The updated position for a_3^3 is −33.478. The check-bound operation changes the position of a_3^3 to 1. This means the tasks π_1 and π_3 are assigned to the 4th and 1st PU respectively. Similarly, all the task assignments can be checked-bound as shown in Table 7.

Lemma 4.4. *The time complexity of Update_Velocity_Position($\vec{A}_i(t-1), \vec{\mathbf{V}}_i(t-1), \vec{\mathbf{F}}_i(t)$) is $\Theta(n)$.*

Proof. The velocity and position of an agent of size n can be computed in $\Theta(n)$ times as per Algorithm 4. □

Lemma 4.5. *The worst case time complexity of GSA(n, P_{Size}, q) is $\Theta(P_{Size}^2 \times MAX_{ite} \times n)$.*

Proof. The time complexity of Generate_InitialPopulation(n, P_{Size}, q) is $\Theta(P_{Size} \times n)$ (refer to Lemma 4.1). Time complexity of Calculate_Fitness($\vec{A}_i(t)$) is $\Theta(n)$ (refer to Lemma 4.2). Time complexity of Calculate_Force($\vec{A}_i(t)$) is $\Theta(n \times P_{Size})$ (refer to Lemma 4.3). Time complexity of Update_Velocity_Position($\vec{A}_i(t-1), \vec{\mathbf{V}}_i(t-1), \vec{\mathbf{F}}_i(t)$) is $\Theta(n)$ (refer to Lemma 4.4). The Generate_InitialPopulation (n, P_{Size}, q) is revoked only once at line 2 of Algorithm 5. The rest of the phases are iterated for $\Theta(P_{Size} \times MAX_{ite})$ times in line 9–23. Therefore, the total time complexity of the GSA is $\Theta(P_{Size}^2 \times MAX_{ite} \times n)$. The overall analysis is given in Table 8. □

Remark 4.2. Diversification and intensification of the GSA are maintained by tuning the following parameters: The gravitational constant (G) has a huge role in making a balance between the diversification (or exploration) and intensification (or exploitation) of the search space. A large value of G encourages more gravitational forces between the agents and thereby produces higher acceleration. Higher acceleration helps to explore the solution space. Contrary to this, the smaller value of G helps better exploitation of the search space through lower acceleration. In the GSA, the value of G is dynamically updated as per the Eq. (40). This is to note that the value of G is tuned by the descending coefficient (α) and the maximum iteration number (refer to Eq. (40)). In the beginning, a larger G encourages better exploration, and then the smaller G helps in exploitation. Moreover, the random generation of the agents also helps in the exploration. A larger population size influences better exploration, and a larger number of iterations ensures both exploration and exploitation.

Moreover, the computation of the force (Eq. (38)) depends on the mass and distances between the agents. The mass is proportional to the fitness of the agent. Therefore, better solutions or heavy mass agents generate more forces towards the nearest agents. Thus, the exploitation of the search spaces is ensured.

Kindly note that, though a larger value of population size and iteration influence both exploration and exploitation, they also increase computational complexity. Therefore, we have to make a trade-off between the selection of population size, iteration number, and computation complexity.

5. Simulation results

5.1. System specifications and simulation set up

The proposed work is simulated on a computer with Ubuntu 20.04.3 LTS operating system, Intel (R) Xeon (R) Gold 6226R CPU, 2.90 GHz, and 1 TB of RAM. Python 3.9.13 is used as the programming language.

The proposed GSA is compared with the genetic algorithm (GA), particle swarm optimization (PSO), and fireworks algorithm (FWA). The termination condition, MAX_{ite} for all the algorithms is taken as 200 iterations. The population size (P_{Size}) is taken as 10 for GSA, PSO, and GA and 30 for FWA. The tournament selection is used for GA. All the algorithms are executed with the fitness function $Calculate\_Fitness(\vec{A}_i(t))$, Algorithm 2. The crossover rate for GA is taken at 0.1, while the mutation rate is 0.05. In PSO, the values of inertia weight, and the social and cognitive constants are taken as 0.7968, 2.1, and 2.1 respectively. Conventional

Table 9
Notation and values.

Notation	Values	Notation	Values
B_0	1 000 000 Hz [29]	d_r	0.3 [38]
τ	3600 s	v_{tip}	120 m/s [38]
k	10^{-27} [38]	v_0	4.03 [38]
n	4–200	S	0.05 [38]
P_{Size}	10–30	ρ	1.225 [38]
$MAX_{iteration}$	100	C_{UAV}, C_{EC}	1000 cycles/byte [29,38]
α	20	L_{in}, L_{out}	400 MB
G_0	100	f_{UAV}	4-6 GHz [38]
P_{bj}, P_{jb}	0.1–0.3 W	f_{EC}	2 GHz [38]
P_0	158.76 W [38]	P_1	88.63 W [38]
P_2	11.46 W	$\omega_1, \omega_2, \omega_3$	0.3, 0.4, 0.3

Table 10
Offloading 10 tasks in 4 PUs with 3 ESs, $\{p_1, p_2, p_3\}$ and 1 UAV, $\{p_4\}$ using GSA.

Task:	π_1	π_2	π_3	π_4	π_5	π_6	π_7	π_8	π_9	π_{10}	Fitness
Allocated PUs	p_1	p_2	p_4	p_1	p_2	p_2	p_2	p_4	p_3	p_3	2 272 147.06

Table 11
Task offloading among $\{p_0, p_1, p_2\}$ ES and $\{p_3\}$ UAV PU, using GSA for scenario-1.

Total task	π_1	π_2	π_3	π_4	π_5	π_6	π_7	π_8	π_9	π_{10}	π_{11}	π_{12}	π_{13}	π_{14}	π_{15}	π_{16}	π_{17}	π_{18}	π_{19}	π_{20}
20	p_1	p_1	p_4	p_2	p_4	p_2	p_3	p_3	p_4	p_2	p_4	p_1	p_3	p_1	p_2	p_3	p_3	p_1	p_3	p_2
19	p_1	p_1	p_2	p_2	p_1	p_0	p_2	p_3	p_3	p_2	p_3	p_1	p_0	p_1	p_0	p_2	p_0	p_3	p_0	
18	p_2	p_2	p_0	p_0	p_1	p_0	p_3	p_3	p_1	p_1	p_2	p_3	p_0	p_1	p_1	p_2	p_3	p_0		
17	p_0	p_1	p_3	p_0	p_3	p_1	p_2	p_0	p_0	p_2	p_1	p_3	p_1	p_2	p_2	p_0	p_3			
16	p_1	p_0	p_2	p_1	p_1	p_2	p_0	p_2	p_0	p_2	p_0	p_3	p_1	p_0	p_3	p_3				
15	p_0	p_1	p_0	p_3	p_0	p_1	p_2	p_1	p_2	p_1	p_2	p_3	p_2	p_0	p_3					
14	p_2	p_0	p_3	p_1	p_3	p_2	p_1	p_1	p_1	p_3	p_0	p_0	p_2	p_0						
13	p_3	p_2	p_0	p_0	p_0	p_3	p_3	p_2	p_0	p_1	p_1	p_2	p_1							
12	p_2	p_0	p_3	p_2	p_0	p_1	p_3	p_3	p_0	p_1	p_2	p_1								
11	p_2	p_3	p_2	p_2	p_1	p_1	p_0	p_0	p_3	p_0	p_1									
10	p_0	p_1	p_3	p_0	p_1	p_0	p_1	p_3	p_2	p_2										
9	p_0	p_0	p_3	p_1	p_0	p_3	p_2	p_2	p_1											
8	p_0	p_3	p_1	p_2	p_3	p_2	p_0	p_1												
7	p_1	p_0	p_0	p_2	p_2	p_3	p_1													
6	p_0	p_1	p_3	p_0	p_1	p_2														
5	p_0	p_0	p_3	p_1	p_2															
4	p_3	p_2	p_1	p_0																

Table 12
Comparison in terms of delay for scenario-1, experiment-2.

Number of tasks	50	100	150	200
GSA	201 149.09	1 565 566.61	5 259 885.10	12 448 553.51
PSO	298 051.30	2 301 779.48	7 718 358.61	12 460 066.23
GA	355 014.88	2 551 446.03	8 291 582.28	19 184 585.99
FWA	435 568.65	9 055 950.45	13 213 254.15	159 697 749.01

FWA [36] is used with tournament selection and Gaussian mutation. The explosion radius and number of explosion sparks were taken as 8 and 30 respectively. Table 9 shows the considered values of the variables for the simulation.

5.2. Simulation scenarios

Let us consider a disaster that occurred and affected a densely populated urban area, resulting in widespread infrastructure damage and limited access to critical resources. The affected area includes multiple disaster hot-spots with varying degrees of damage, making it challenging to coordinate humanitarian aid. The SMDs were previously deployed in this disaster-prone area. There are some remote edge servers (ESs) and rotary-wing UAVs that are also deployed and connected with the IoT nodes. A base station (BS) is also installed by the government as the intelligent agent (IA). Let us assume a scenario for the illustration. There are 10 task requests from 10 SMDs, $Ts = \{\pi_1, \pi_2, \ldots, \pi_{10}\}$ those are required to be offloaded at 4 PUs as 3 ESs, $\{p_1, p_2, p_3\}$ and a UAV, p_4. The IA executes the GSA-based offloading algorithm to take the offloading decision as shown in Table 10 along with the fitness value. Similarly, two scenarios are considered for extensive simulation, as discussed below.

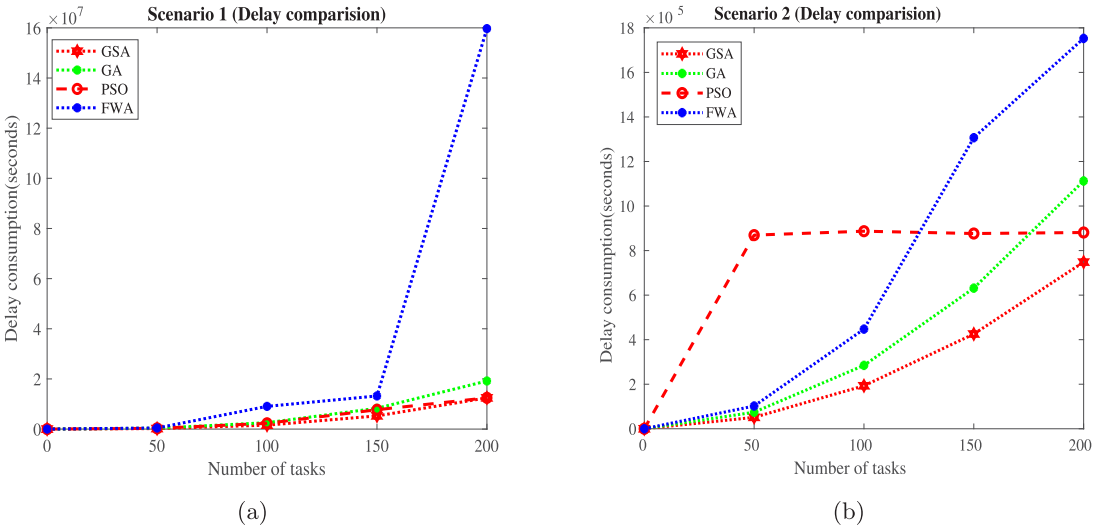


Fig. 4. Comparison in terms of delay, (a) Scenario-1 and (b) Scenario-2.

Table 13
Comparison in terms of energy for scenario-1, experiment-2.

Number of tasks	50	100	150	200
GSA	951 780.05	957 161.05	962 539.49	967 920.50
PSO	967 920.50	967 920.50	967 920.50	967 920.50
GA	4 589 078.57	7 367 328.47	10 143 093.43	12 920 839.71
FWA	957 022.93	967 646.81	978 268.13	988 892.02

Table 14
Comparison in terms of load balance for scenario-1, experiment-2.

Number of tasks	50	100	150	200
GSA	1621.61	1090.77	5853.27	21 087.11
PSO	5579.09	9545.89	11 827.75	17 228.73
GA	2713.56	7972.55	14 045.34	15 197.13
FWA	25 763.51	187 876.12	473 923.06	1 157 878.77

5.2.1. Scenario - 1

A scenario is considered with 4 PUs as 3 ESs, $\{p_0, p_1, p_2\}$, and a rotary-wing UAV (p_3). All the PUs are assumed to have equal computation capacity. Now, three experiments are conducted on this scenario, as follows:

Experiment-1: Here, 4–20 tasks are taken. Table 11 shows the complete offloading of the tasks to different PUs. It can be observed that the tasks are scheduled almost equally for all the PUs, as the PUs are considered to have equal computation capability.

Experiment-2: In this experiment, the number of tasks varied between 50, 100, 150, and 200. Subsequently, the different fitness values (delay, energy, load balance, and overall fitness) are noted for GSA, GA, PSO, and FWA. The algorithms are run for 100 iterations, and the population size is taken as 50. Figs. 4(a), 5(a), 6(a) and 7(a) demonstrate the outcomes of the experiment by considering the fitness value on delay, energy, load balance, and total fitness, respectively. The Tables 12, 13, 14 and 15 also present the values for the same. It can be observed that the proposed GSA performs better than the GA, PSO, and FWA.

5.2.2. Scenario - 2

The scenario is considered with five PUs having two ESs, $\{p_5, p_4\}$ and four UAVs, $\{p_3, p_2, p_1, p_0\}$. Here, a heterogeneous system is considered. It is assumed that p_0 has maximum computation capability and the remaining PUs have lesser capability, with p_5 having the minimum computation capacity.

Experiment-1: Similar to scenario-1, 4–20 tasks are offloaded to different PUs. Table 16 shows the complete offloading of the tasks to the PUs. It can be observed that most of the tasks are offloaded to p_0 as it has maximum computation capability, as assumed in the scenario-2. This is in contrast to scenario-1 where the tasks are almost equally distributed over the PUs.

Experiment-2: The number of tasks and other parameters related to EAs are taken the same as scenario-1. Figs. 4(b), 5(b), 6(b), and 7(b) show the performance in terms of delay, energy, load balance, and overall fitness. It can be observed that the performance in scenario-2 is better for all the algorithms than in scenario-1. This is due to the availability of more resources as PUs in scenario-2. It can also be noticed in the case of PSO that the delay, energy, and overall fitness are found to be high as most of the tasks are offloaded to very few PUs. The Tables 17, 18, 19 and 20 also depict the same.

Table 15
Comparison in terms of overall fitness for scenario-1, experiment-2.

Number of tasks	50	100	150	200
GSA	441 324.65	852 861.66	1 964 611.18	4 122 944.99
PSO	2 275 807.54	4 134 199.56	4 127 118.62	4 134 806.02
GA	2 428 245.83	3 033 174.27	4 250 121.48	5 514 668.99
FWA	526 725.27	2 264 181.75	17 196 372.11	34 239 488.42

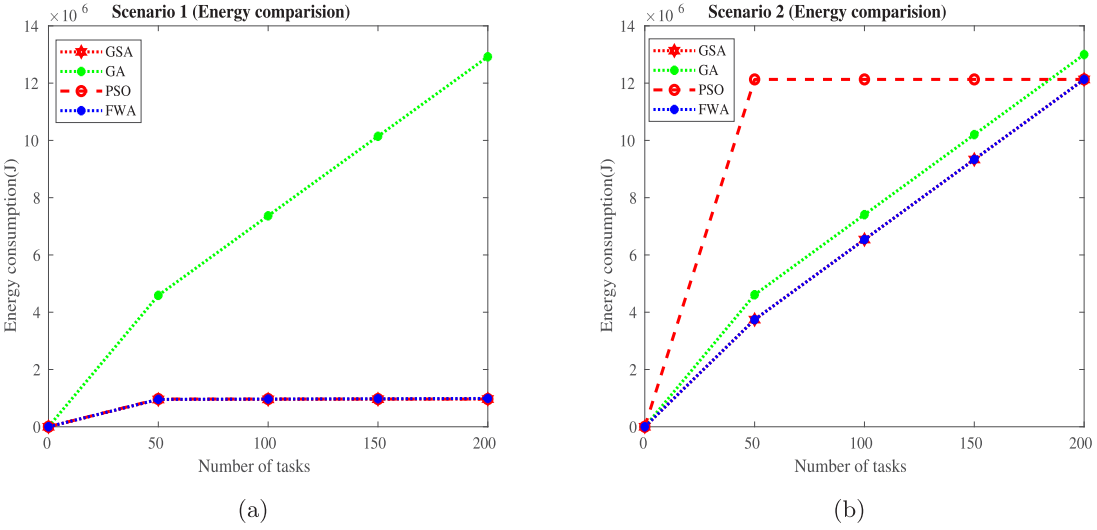


Fig. 5. Comparison in terms of energy (a) Scenario-1. (b) Scenario-2.

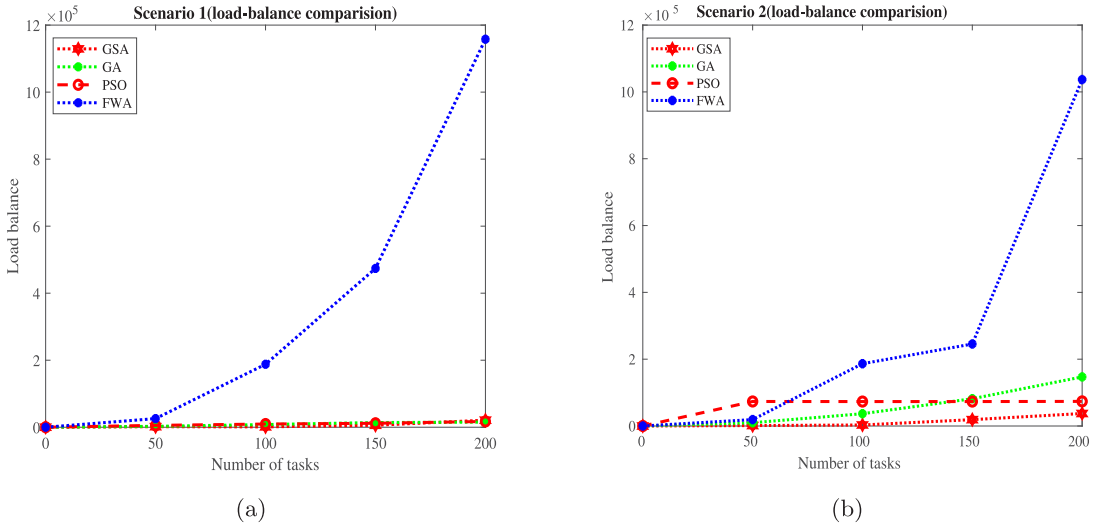


Fig. 6. Comparison in terms of load balance, (a) Scenario-1 and (b) Scenario-2.

Experiment-3: Here, the performance is observed for a smaller population size of 30 with 100 tasks for both the scenarios. Tables 21 and 22 along with Figs. 8(a)–(b), and 9(a)–(b) show that the GSA performs comparably better even for smaller population size.

5.2.3. Scenario - 3

Here a large scenario is considered with fifty PUs, which consist of forty ESs and ten UAVs. The ESs and UAVs are considered with, $C_b = 1000$, i.e., $C_{UAV} = C_{ES} = 1000$, $f_{EC} = [1 - 2]$ GHz and $f_{UAV} = 0.5$ GHz. Tables 23–26 shows various comparisons with respect to overall fitness, energy, delay, and load balance.

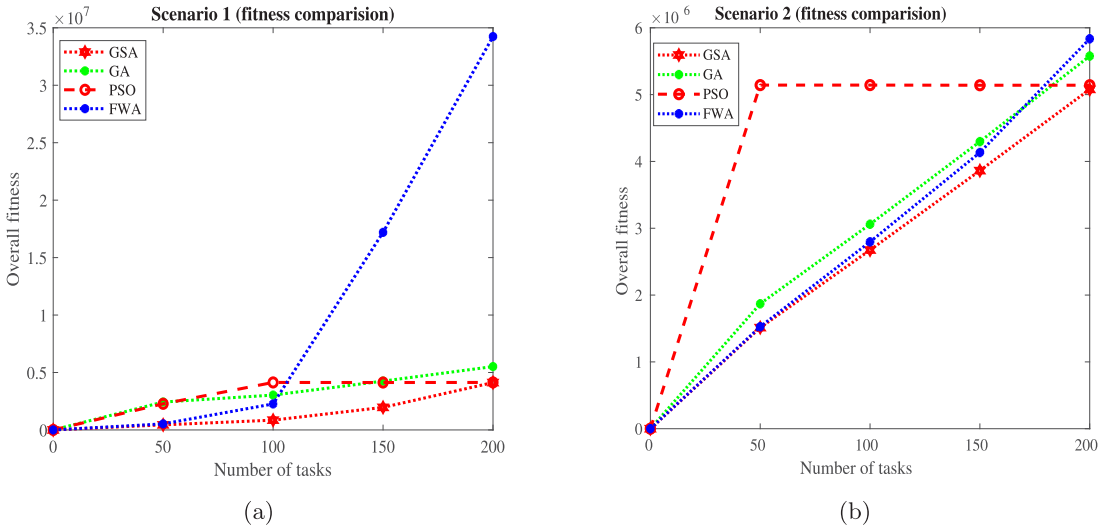


Fig. 7. Comparison in terms of overall fitness, (a) Scenario-1 and (b) Scenario-2.

Table 16
Task offloading among $\{p_5, p_4\}$ ES and $\{p_3, p_2, p_1, p_0\}$ UAV PU, using GSA for scenario-2.

Total task	π_1	π_2	π_3	π_4	π_5	π_6	π_7	π_8	π_9	π_{10}	π_{11}	π_{12}	π_{13}	π_{14}	π_{15}	π_{16}	π_{17}	π_{18}	π_{19}	π_{20}
20	p_4	p_6	p_6	p_3	p_1	p_1	p_0	p_0	p_1	p_1	p_1	p_0	p_0	p_0	p_0	p_0	p_0	p_1	p_0	p_3
19	p_0	p_0	p_1	p_0	p_0	p_1	p_0	p_0	p_1	p_0	p_0	p_3	p_1	p_0	p_3	p_0	p_1	p_0	p_1	
18	p_0	p_0	p_1	p_0	p_4	p_0	p_0	p_0	p_4	p_3	p_0	p_0	p_0	p_5	p_0	p_0	p_0	p_5		
17	p_0	p_0	p_3	p_3	p_0	p_0	p_4	p_0	p_0	p_0	p_0	p_3	p_0	p_0	p_3	p_0	p_3			
16	p_0	p_1	p_0	p_5	p_3	p_0	p_3	p_0	p_0	p_1	p_0	p_0	p_5	p_0	p_3	p_3				
15	p_0	p_5	p_0	p_0	p_0	p_0	p_0	p_0	p_5	p_0	p_0	p_5	p_0	p_0	p_0					
14	p_3	p_0	p_0	p_0	p_0	p_0	p_1	p_1	p_3	p_0	p_3	p_0	p_0	p_0						
13	p_0	p_0	p_0	p_0	p_0	p_3	p_3	p_0	p_3	p_0	p_0	p_0	p_0							
12	p_1	p_0	p_3	p_3	p_0	p_0	p_0	p_0	p_0	p_0	p_0	p_0								
11	p_0	p_0	p_3	p_3	p_3	p_3	p_0	p_0	p_0	p_0	p_0									
10	p_0	p_0	p_0	p_0	p_0	p_3	p_0	p_0	p_3	p_0										
9	p_0	p_0	p_0	p_0	p_0	p_0	p_0	p_0	p_0											
8	p_0	p_0	p_0	p_0	p_0	p_0	p_0	p_0												
7	p_0	p_0	p_0	p_0	p_0	p_0	p_0													
6	p_0	p_0	p_0	p_0	p_0	p_0														
5	p_0	p_0	p_0	p_0	p_0															
4	p_0	p_0	p_0	p_5																

Table 17
Comparison in terms of delay for scenario-2, experiment-2.

Number of tasks	50	100	150	200
GSA	51 268.18	193 001.47	425 400.07	748 267.37
PSO	869 411.24	887 016.47	876 581.94	881 148.00
GA	74 320.59	284 551.67	631 345.82	1 112 517.63
FWA	102 153.24	447 420.22	1 306 734.26	1 752 689.67

Table 18
Comparison in terms of energy for scenario-2, experiment-2.

Number of tasks	50	100	150	200
GSA	3 741 919.48	6 537 634.68	9 333 062.06	12 129 090.00
PSO	12 129 892.02	12 129 924.00	12 129 862.30	12 129 967.33
GA	4 610 753.02	7 406 971.43	10 202 453.50	12 998 397.94
FWA	3 742 393.01	6 538 529.76	9 333 913.89	12 129 965.87

5.2.4. Performance in optimization problem solver

PuLP is an open-source mathematical modeling library in Python. The outcome of GSA is compared with the results using PuLP. Here, 5–20 tasks are taken with 2 PUs. The performance of GSA and the PuLP is observed to be similar for smaller data sizes in terms of the number of tasks and PUs, as shown in Table 27. However, such an optimizer fails to provide results for larger data sizes

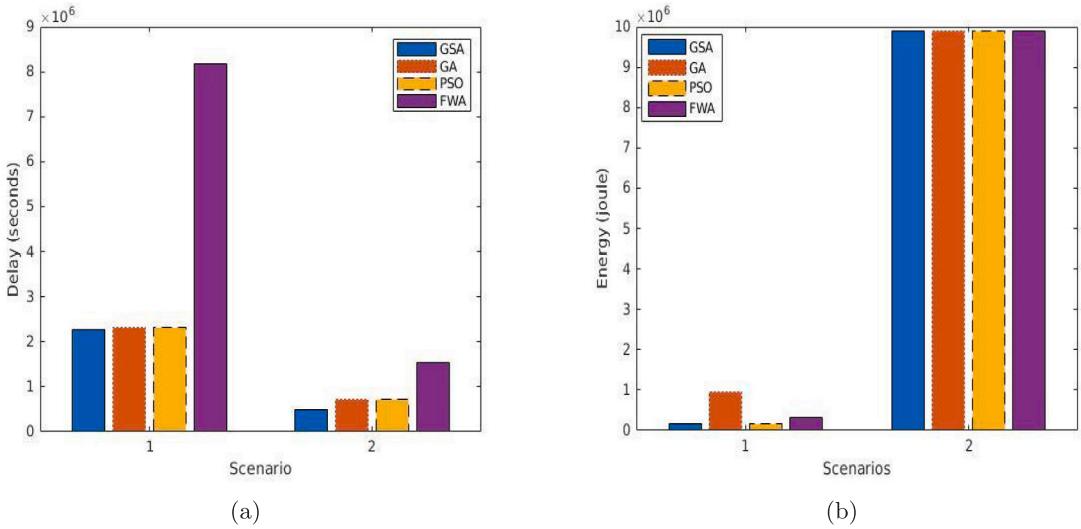


Fig. 8. Comparison in both the scenarios in terms of (a) delay and (b) energy.

Table 19
Comparison in terms of load balance for scenario-2, experiment-2.

Number of tasks	50	100	150	200
GSA	1324.90	3054.05	19 035.54	37 380.76
PSO	73 405.73	73 171.36	73 171.46	73 645.93
GA	9797.13	36 791.12	81 701.51	146 918.58
FWA	19 156.52	186 478.19	245 452.15	1 036 983.34

Table 20
Comparison in terms of overall fitness for scenario-2, experiment-2.

Number of tasks	50	100	150	200
GSA	1 512 715.80	2 673 927.78	3 862 382.33	5 078 282.51
PSO	5 142 277.85	5 142 566.24	5 140 212.62	5 140 686.23
GA	1 869 495.63	3 060 248.02	4 296 477.43	5 576 982.89
FWA	1 526 350.96	2 796 504.16	4 133 544.48	5 835 653.51

Table 21
Comparison for delay and energy in scenario-1 and 2, experiment-3.

	Delay		Energy	
	Scenario-1	Scenario-2	Scenario-1	Scenario-2
GSA	2 274 832.07	493 105.51	967 646.81	9 891 969.82
PSO	2 313 291.05	719 487.82	967 646.81	9 892 993.45
GA	2 313 291.05	714 712.83	967 646.81	9 893 161.81
FWA	8 172 368.16	1 543 182.03	967 646.81	9 893 104.16

Table 22
Comparison for load balance and overall fitness in scenario-1 and 2, experiment-3.

	Load balance		Overall fitness	
	Scenario-1	Scenario-2	Scenario-1	Scenario-2
GSA	10 085.03	29 391.26	1 070 202.44	4 109 406.06
PSO	13 721.75	93 571.66	1 075 704.43	4 199 804.40
GA	15 741.80	94 688.88	2 323 482.20	4 199 515.81
FWA	140 527.94	563 807.07	2 572 233.23	4 517 980.53

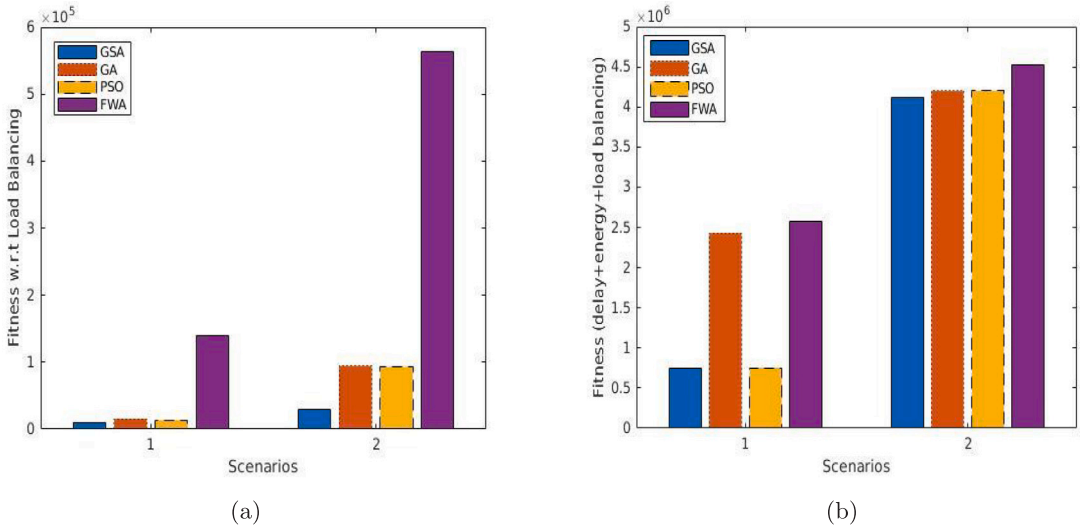


Fig. 9. Comparison in both the scenarios in terms of (a) load balance and (b) overall fitness.

Table 23
Comparison in terms of overall fitness for scenario-3.

Number of tasks	350	400	450	500
GSA	2 302 241.44	2 313 245.01	2 315 631.87	2 322 514.88
PSO	2 305 890.71	2 325 923.24	2 322 827.73	2 333 521.38
GA	2 374 342.42	2 407 522.98	2 490 946.30	2 521 083.64
FWA	3 376 734.27	3 340 618.72	5 185 069.30	5 623 233.12

Table 24
Comparison in terms of energy for scenario-3.

Number of tasks	350	400	450	500
GSA	147 757.25	168 866.89	189 973.97	211 083.61
PSO	166 212.18	191 935.56	217 236.95	240 863.17
GA	169 567.63	193 613.28	221 011.82	240 863.17
FWA	151 951.55	173 061.19	194 168.27	215 277.92

Table 25
Comparison in terms of delay for scenario-3.

Number of tasks	350	400	450	500
GSA	68 383.71	77 554.62	87 371.64	98 338.96
PSO	78 203.72	89 382.32	100 548.61	111 725.34
GA	78 211.54	89 390.00	100 560.21	111 738.93
FWA	78 183.37	89 359.74	100 523.31	111 699.68

Table 26
Comparison in terms of load balance for scenario-3.

Number of tasks	350	400	450	500
GSA	24 428.79	35 684.17	13 797.55	38 004.64
PSO	65 165.59	53 710.64	102 922.99	179 690.20
GA	79 718.79	82 740.42	73 785.46	38 514.66
FWA	1 166 729.55	1 970 276.60	2 371 251.98	12 919 413.16

Table 27
Fitness comparison between PuLP and GSA.

Number of tasks	5	10	15	20
GSA	1 961 359.68	2 108 244.56	2 255 254.79	2 402 360.75
PuLP	1 961 460.66	2 108 407.55	2 255 712.50	2 402 946.95

Table 28
Comparison with respect to energy consumption.

Number of tasks	20	19	18	17	16
Proposed GSA	243.08	230.92	218.76	206.61	194.45
PSO [35]	243.08	230.92	218.76	206.61	194.45

Table 29
Comparison with respect to delay.

Number of tasks	20	19	18	17	16
Proposed GSA	3278.91	3100.10	3185.24	2758.62	2862.59
PSO [35]	3952.32	3741.80	3617.14	3283.00	3085.04

Table 30
Comparison with respect to load balance.

Number of tasks	20	19	18	17	16
Proposed GSA	629.43	717.08	427.81	262.10	266.63
PSO [35]	903.70	909.98	593.27	496.62	365.52

Table 31
Result of ANOVA test for scenario-1, overall fitness model.

Source of variations	Sum of square	Degree of freedom	Mean Square (MS)	F-static	p-value	F-critical
Between groups	9.54×10^{13}	3	3.18×10^{13}	10.1983	0.0005378	3.24
Within groups	4.99×10^{13}	16	3.11×10^{12}			
Total	1.45×10^{14}	19	7.65×10^{12}			

Table 32
Fitness with respect to iterations.

Iterations:	1	50	100	150	200	250	300
GSA	16 803.50	10 661.55	10 661.51	10 661.49	10 661.44	10 661.44	10 661.44
GA	10 847.94	10 847.74	10 661.61	10 661.62	10 661.60	10 661.54	10 661.54
PSO	11 923.78	11 363.19	11 146.13	11 020.69	10 877.75	10 847.90	10 804.05

in terms of 100 tasks, etc. This is because the considered problem, EDLAOP is NP-complete, and such an optimizer uses exponential algorithms for such problems and thereby fails for large input sizes. In contrast, the GSA is a polynomial-time algorithm that can efficiently provide feasible and approximate solutions even for large data sizes.

5.2.5. Comparison with another task offloading model

Here, the usability of our proposed model and GSA is shown using a numerical example and compared with the solution received by the research done by You and Tang [35]. The PSO [35] algorithm, along with the corresponding model, has been executed. The best solution vectors generated by PSO [35] and the proposed GSA are compared. There are four MEC servers where tasks would be offloaded from 20–16 SMDs. The performances of the solution vectors are compared, and the comparison data is shown in Tables 28, 29, and 30. It is observed that the energy consumption in both the algorithms is the same. However, the proposed GSA is found to be better in terms of the other parameters: delay and load balance.

5.3. Analysis of Variance (ANOVA)

ANOVA is used for examining the significance of the outcome [20,48,49]. The research evaluates GSA, PSO, GA, and FWA by using the null hypothesis (H_0) and alternative hypothesis (H_1). H_0 is rejected if P value is less than α level, which is 0.05. Along with this, H_0 is rejected if F-static (F_{stat}) is greater than F-critical (F_{cric}). Here, H_0 refers to the fact that the mean of PSO, GSA, GA, and FWA are equal. H_1 refers they are not the same. For the ANOVA test, 10 samples were taken for GSA, PSO, GA, and FWA for 20 tasks. There are two scenarios, and for each scenario, there are four parameters (delay, energy, load balance, and overall fitness). Here, the analysis is shown only for overall fitness in scenario-1. The result of the ANOVA is shown in Table 31.

Here, H_0 is rejected as the p-value $< \alpha$. The averages of certain groups are thought not to be equal. Among the averages of certain groups, the difference is noticed. It reflects that there is little possibility of a type 1 mistake (rejecting a correct H_0). The smaller the p-value the stronger it support H_1. The F-statistic is 10.1983, which is not in the 95% region of acceptance: $[-\infty : 3.2389]$.

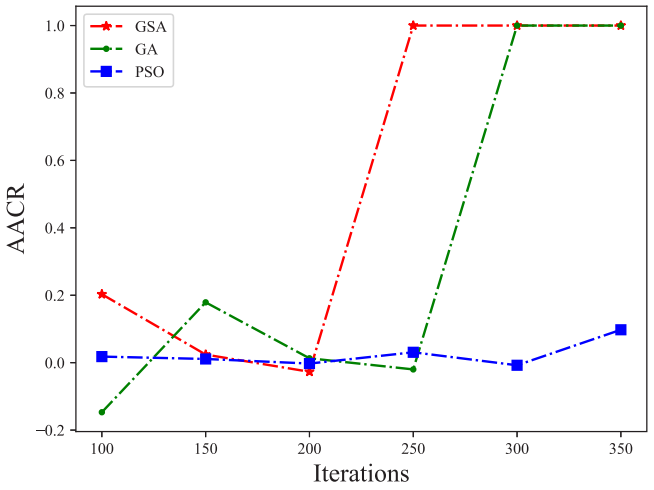


Fig. 10. AACR value.

Table 33
The fitness values for PCC test.

#Tasks	Fitness (X)	Energy (y_1)	Delay (y_2)	Load (y_3)
40	2 295 928.08	5 686 907.09	57 028.97	13 521.84
60	2 330 634.26	5 691 155.37	143 596.24	36 977.45
80	2 382 987.73	5 695 404.82	274 216.48	75 202.88
100	2 434 206.44	5 699 654.88	113 270.21	113 270.21

Table 34
Components to calculate PCC for fitness and energy ($CorCoe(X, y_1)$).

$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})^2$	$(y_i - \bar{y})^2$
−65 011.04	−6373.45	4 226 436 297.04	40 620 864.90
−30 304.86	−2125.16	918 384 994.19	4 516 347.52
22 048.60	2124.28	486 140 872.20	4 512 565.51
73 267.31	6374.33	5 368 099 080.97	40 632 210.43
$\sum(x_i - \bar{x})(y_i - \bar{y}) =$	992 615 822.03	$\sqrt{\sum(x_i - \bar{x})^2(y_i - \bar{y})^2} =$	996 502 443.30

5.4. Pearson correlation coefficient

The degree and direction of the association between two variables are measured by the Pearson correlation coefficient (PCC) [50]. The correlation coefficient, $CorCoe(x_i, y_i)$ between two variables x_i and x_i is calculated using Eq. (44).

$$CorCoe(x_i, y_i) = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2(y_i - \bar{y})^2}} \tag{44}$$

where, $\bar{x}$ and $\bar{y}$ are the means of the variables x_i and y_i respectively. The $CorCoe(x_i, y_i)$ is one of the measurements to test whether two variables x_i and y_i are significantly co-related or not ($-1 \leq CorCoe(x_i, y_i) \leq +1$). The $CorCoe(x_i, y_i) = 0$ signifies there is no co-relation between the variables.

Here, we have collected different outputs of the proposed GSA by changing various input parameters. Now, the co-relations between the final fitness value (X) with respect to the other objective parameters: energy (y_1), delay (y_2) and load balancing (y_3) are measured. Table 33 shows the collected output of the algorithm by varying the number of tasks and PUs. At first, the co-relation ($CorCoe(X, y_1)$) between the overall fitness value (X) and energy (y_1) is calculated using Table 34. It is observed that the $CorCoe(X, y_1)$ is 0.996. This is a high positive correlation between fitness and energy. Similarly, the co-relations with respect to delay and load are also calculated, and the values are observed as $CorCoe(X, y_2) = 0.375$ and $CorCoe(X, y_2) = 0.99$. Therefore, all the energy, delay, and load are positively co-related to fitness.

5.5. Convergence of GSA

Alternative average convergence rate (AACR) [51] is calculated as per Eq. (45). For our research, τ_0 refers to 50th iteration, t refers to current iteration, $f_{t+\tau_0}$ refers to the fitness value in $(t + \tau_0)$ iteration. Similarly, f_t, $f_{t-\tau_0}$ refers to fitness at t and $(t - \tau_0)$

iteration respectively.

$$AACR = 1 - \left| \frac{f_{t+\tau_0} - f_t}{f_t - f_{t-\tau_0}} \right|^{\frac{1}{\tau_0}}, \text{if } t \geq \tau_0 \tag{45}$$

Here, 20 tasks are used in scenario 1. Table 32 shows the fitness value up to 300 iterations. The AACR is computed by considering the fitness values for the iteration numbers 1, 50, 100, 150, 200, 250, 300, and 350 respectively. It can be noticed from Fig. 10 that GSA has reached convergence around 200 iterations, unlike other techniques. The inherent scalability characteristics of GSA make it suitable for solving large-scale problems. GSA has the competence to work with large population size and adapt the search process to a wide range of problem dimensions. Moreover, the convergence property enables it to scale up and tackle a wide range of problems.

6. Conclusion

The paper proposes an approach for task offloading based on GSA for multi-UAV-assisted MEC-enabled disaster applications. The proposed work considers the challenges of minimizing delay, load balance, and energy in offloading. The agents are efficiently encoded and decoded to solve the offloading problem. The fitness function is derived by considering the delay, energy, and load balancing of the PUs. The simulation of various scenarios establishes the efficacy and wide applicability of the proposed work.

Assume a large disaster-affected region with massive damages to roads, infrastructure, and network connectivity, e.g., an earthquake or flood-affected area. It is not feasible for the human rescue team to immediately spread over the region and start the rescue operation. Some IoT devices or smart mobile devices (SMDs) were previously deployed in the disaster-prone region. The SMDs may collect and send information about the disaster, e.g., catching of fire, leakages of toxic gas, etc. Moreover, the local mobile phones of the individuals who are trapped may also be used as a source of local data. All these data require further processing by the local UAV or edge server (ES) to identify and initiate the rescue operation. Note that the SMDs are not equipped with sufficient computation and energy to frequently process such data, so offloading of such tasks is necessary. After processing the data, the UAVs/ESs first identify and confirm the level of damage and accordingly request the rescue operation at the rescue station. The same responses are shared with the trapped mobile phones or the deployed SMDs to avoid panic situations. The proposed model and algorithm can efficiently take the offloading decision so that humanitarian aid can be provided faster.

However, there are some managerial implications in the deployment of the proposed GSA-based offloading. Many parameters of UAV and ES are to be reset properly before the execution of the algorithm. Moreover, the GSA parameters, weight values, population size, etc. are also to be tuned. Technical expertise may also be required to handle the flying UAVs. Future studies might focus on the deployment of the proposed mechanism in real life. Moreover, further efforts may be given to implementing adaptive mechanisms, taking into account dynamic changes in the catastrophe scenario, or investigating hybrid approaches combining various optimization techniques.

Declaration of competing interest

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

Data availability

No data was used for the research described in the article.

References

[1] A. Khan, S. Gupta, S.K. Gupta, Multi-hazard disaster studies: Monitoring, detection, recovery, and management, based on emerging technologies and optimal techniques, Int. J. Disaster Risk Reduct. 47 (2020) 101642.
[2] H. Jayanthi, G.J. Husak, C. Funk, T. Magadzire, A. Adoum, J.P. Verdin, A probabilistic approach to assess agricultural drought risk to maize in Southern Africa and millet in Western Sahel using satellite estimated rainfall, Int. J. Disaster Risk Reduct. 10 (2014) 490–502.
[3] S.M.S.M. Daud, M.Y.P.M. Yusof, C.C. Heo, L.S. Khoo, M.K.C. Singh, M.S. Mahmood, H. Nawawi, Applications of drone in disaster management: A scoping review, Sci. Justice 62 (1) (2022) 30–42.
[4] K. Nero, K. Orru, T.-O. Nævestad, A. Olson, M. Schobert, P. Windsheimer, J. Keränen, P. Jukarainen, J. Kajganovic, Care organisations role as intermediaries between the authorities and the marginalised in crisis management, Int. J. Disaster Risk Reduct. 86 (2023) 103516.
[5] A. Maghsoudi, R. Harpring, W.D. Piotrowicz, D. Kedziora, Digital technologies for cash and voucher assistance in disasters: A cross-case analysis of benefits and risks, Int. J. Disaster Risk Reduct. (2023) 103827.
[6] T. Kustu, A. Taskin, Deep learning and stereo vision based detection of post-earthquake fire geolocation for smart cities within the scope of disaster management: İstanbul case, Int. J. Disaster Risk Reduct. (2023) 103906.
[7] J. Shahmoradi, E. Talebi, P. Roghanchi, M. Hassanalian, A comprehensive review of applications of drone technology in the mining industry, Drones 4 (3) (2020) 34.
[8] F. Farahbakhsh, A. Shahidinejad, M. Ghobaei-Arani, Multiuser context-aware computation offloading in mobile edge computing based on Bayesian learning automata, Trans. Emerg. Telecommun. Technol. 32 (1) (2021) e4127.
[9] Z. Yang, J. Hou, M. Shikh-Bahaei, Energy efficient resource allocation for mobile-edge computation networks with NOMA, in: 2018 IEEE Globecom Workshops (GC Wkshps), IEEE, 2018, pp. 1–7.
[10] L. Lin, X. Liao, H. Jin, P. Li, Computation offloading toward edge computing, Proc. IEEE 107 (8) (2019) 1584–1607.

[11] W. Ye, J. Luo, F. Shan, W. Wu, M. Yang, Offspeeding: Optimal energy-efficient flight speed scheduling for UAV-assisted edge computing, Comput. Netw. 183 (2020) 107577.
[12] X. Hu, K.-K. Wong, K. Yang, Z. Zheng, Task and bandwidth allocation for UAV-assisted mobile edge computing with trajectory design, in: 2019 IEEE Global Communications Conference (GLOBECOM), IEEE, 2019, pp. 1–6.
[13] M. Li, N. Cheng, J. Gao, Y. Wang, L. Zhao, X. Shen, Energy-efficient UAV-assisted mobile edge computing: Resource allocation and trajectory optimization, IEEE Trans. Veh. Technol. 69 (3) (2020) 3424–3438.
[14] F. Zhou, R.Q. Hu, Z. Li, Y. Wang, Mobile edge computing in unmanned aerial vehicle networks, IEEE Wirel. Commun. 27 (1) (2020) 140–146.
[15] Y. Wang, Z.-Y. Ru, K. Wang, P.-Q. Huang, Joint deployment and task scheduling optimization for large-scale mobile users in multi-UAV-enabled mobile edge computing, IEEE Trans. Cybern. 50 (9) (2019) 3984–3997.
[16] Z. Wang, P. Li, S. Shen, K. Yang, Task offloading scheduling in mobile edge computing networks, Procedia Comput. Sci. 184 (2021) 322–329.
[17] E. Rashedi, H. Nezamabadi-Pour, S. Saryazdi, GSA: a gravitational search algorithm, Inform. Sci. 179 (13) (2009) 2232–2248.
[18] P.K. Ram, P. Kuila, GSA-based approach for gene selection from microarray gene expression data, Mach. Learn. Algorithms Appl. (2021) 159–174.
[19] A.S. Thakur, T. Biswas, P. Kuila, Gravitational search algorithm based task scheduling for multi-processor systems, in: 2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN), IEEE, 2018, pp. 253–257.
[20] T. Biswas, P. Kuila, A.K. Ray, M. Sarkar, Gravitational search algorithm based novel workflow scheduling for heterogeneous computing systems, Simul. Model. Pract. Theory 96 (2019) 101932.
[21] J.-M. Lozano, I. Tien, Data collection tools for post-disaster damage assessment of building and lifeline infrastructure systems, Int. J. Disaster Risk Reduct. (2023) 103819.
[22] A. Saif, K. Dimyati, K.A. Noordin, N.A. Mosali, G. Deepak, S.H. Alsamhi, Skyward bound: Empowering disaster resilience with multi-UAV-assisted B5G networks for enhanced connectivity and energy efficiency, Internet of Things (2023) 100885.
[23] A. Javadpour, F.S. AliPour, A.K. Sangaiah, W. Zhang, F. Ja'far, A. Singh, An IoE blockchain-based network knowledge management model for resilient disaster frameworks, J. Innov. Knowl. 8 (3) (2023) 100400.
[24] S. Farazmehr, Y. Wu, Locating and deploying essential goods and equipment in disasters using AI-enabled approaches: A systematic literature review, Prog. Disaster Sci. (2023) 100292.
[25] M. Shi, X. Zhang, J. Chen, H. Cheng, UAV cluster-assisted task offloading for emergent disaster scenarios, Appl. Sci. 13 (8) (2023) 4724.
[26] Y. Wang, W. Chen, T.H. Luan, Z. Su, Q. Xu, R. Li, N. Chen, Task offloading for post-disaster rescue in unmanned aerial vehicles networks, IEEE/ACM Trans. Netw. 30 (4) (2022) 1525–1539.
[27] R. Chaudhry, V. Rishiwal, An efficient task allocation with fuzzy reptile search algorithm for disaster management in urban and rural area, Sustain. Comput.: Inform. Syst. 39 (2023) 100893.
[28] H. Satake, Y. Kobayashi, R. Tani, H. Shigeno, Dynamic task offload system adapting to the state of network resources in mobile edge computing, in: 2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), IEEE, 2020, pp. 1–6.
[29] Z. Li, Q. Zhu, Genetic algorithm-based optimization of offloading and resource allocation in mobile-edge computing, Information 11 (2) (2020) 83.
[30] T. Li, F. Yang, D. Zhang, L. Zhai, Computation scheduling of multi-access edge networks based on the artificial fish swarm algorithm, IEEE Access 9 (2021) 74674–74683.
[31] B. Zhang, G. Zhang, W. Sun, K. Yang, Task offloading with power control for mobile edge computing using reinforcement learning-based Markov decision process, Mob. Inf. Syst. (2020).
[32] J. Yan, S. Bi, Y.J. Zhang, M. Tao, Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency, IEEE Trans. Wireless Commun. 19 (1) (2019) 235–250.
[33] Q.-V. Pham, T. Leanh, N.H. Tran, B.J. Park, C.S. Hong, Decentralized computation offloading and resource allocation for mobile-edge computing: A matching game approach, IEEE Access 6 (2018) 75868–75885.
[34] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, Y. Zhang, Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks, IEEE Access 4 (2016) 5896–5907.
[35] Q. You, B. Tang, Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things, J. Cloud Comput. 10 (2021) 1–11.
[36] X. Li, L. Zhou, Y. Sun, B. Ulziinyam, Multi-task offloading scheme for UAV-enabled fog computing networks, EURASIP J. Wireless Commun. Networking 2020 (1) (2020) 1–16.
[37] J. Zhang, Z. Zhou, S. Li, L. Gan, X. Zhang, L. Qi, X. Xu, W. Dou, Hybrid computation offloading for smart home automation in mobile cloud computing, Pers. Ubiquitous Comput. 22 (1) (2018) 121–134.
[38] T. Zhang, Y. Xu, J. Loo, D. Yang, L. Xiao, Joint computation and communication design for UAV-assisted mobile edge computing in IoT, IEEE Trans. Ind. Inform. 16 (8) (2019) 5505–5516.
[39] J. Liu, L. Li, F. Yang, X. Liu, X. Li, X. Tang, Z. Han, Minimization of offloading delay for two-tier UAV with mobile edge computing, in: 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC), IEEE, 2019, pp. 1534–1538.
[40] G. Wu, Y. Miao, Y. Zhang, A. Barnawi, Energy efficient for UAV-enabled mobile edge computing networks: Intelligent task prediction and offloading, Comput. Commun. 150 (2020) 556–562.
[41] R. Wang, Y. Cao, A. Noor, T.A. Alamoudi, R. Nour, Agent-enabled task offloading in UAV-aided mobile edge computing, Comput. Commun. 149 (2020) 324–331.
[42] J. Du, L. Zhao, J. Feng, X. Chu, Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee, IEEE Trans. Commun. 66 (4) (2017) 1594–1608.
[43] Y. Xu, T. Zhang, J. Loo, D. Yang, L. Xiao, Completion time minimization for UAV-assisted mobile-edge computing systems, IEEE Trans. Veh. Technol. 70 (11) (2021) 12253–12259.
[44] S. Liu, T. Yang, Delay aware scheduling in UAV-enabled OFDMA mobile edge computing system, IET Commun. 14 (18) (2020) 3203–3211.
[45] J. Zhang, L. Zhou, Q. Tang, E.C.-H. Ngai, X. Hu, H. Zhao, J. Wei, Stochastic computation offloading and trajectory scheduling for UAV-assisted mobile edge computing, IEEE Internet Things J. 6 (2) (2018) 3688–3699.
[46] Y. Zeng, J. Xu, R. Zhang, Energy minimization for wireless communication with rotary-wing UAV, IEEE Trans. Wireless Commun. 18 (4) (2019) 2329–2345.
[47] P. Kuila, P.K. Jana, Energy efficient clustering and routing algorithms for wireless sensor networks: Particle swarm optimization approach, Eng. Appl. Artif. Intell. 33 (2014) 127–140.
[48] P.K. Ram, P. Kuila, GAAE: a novel genetic algorithm based on autoencoder with ensemble classifiers for imbalanced healthcare data, J. Supercomput. (2022) 1–32.
[49] S. Harizan, P. Kuila, A novel NSGA-II for coverage and connectivity aware sensor node scheduling in industrial wireless sensor networks, Digit. Signal Process. 105 (2020) 102753.
[50] A. Sysoev, Sensitivity analysis of mathematical models, Computation 11 (8) (2023) 159.
[51] Y. Chen, J. He, Average convergence rate of evolutionary algorithms in continuous optimization, Inform. Sci. 562 (2021) 200–219.