Computer Modeling in Engineering & Sciences

Tech Science Press

Doi:10.32604/cmes.2025.059786

ARTICLE

An Adaptive Firefly Algorithm for Dependent Task Scheduling in IoT-Fog Computing

Adil Yousif*

Department of Computer Science, College of Science and Arts-Sharourah, Najran University, Najran Sharourah, 68378, Saudi Arabia

*Corresponding Author: Adil Yousif. Email: ayalfaki@nu.edu.sa

Received: 17 October 2024; Accepted: 20 January 2025; Published: 03 March 2025

ABSTRACT: The Internet of Things (IoT) has emerged as an important future technology. IoT-Fog is a new computing paradigm that processes IoT data on servers close to the source of the data. In IoT-Fog computing, resource allocation and independent task scheduling aim to deliver short response time services demanded by the IoT devices and performed by fog servers. The heterogeneity of the IoT-Fog resources and the huge amount of data that needs to be processed by the IoT-Fog tasks make scheduling fog computing tasks a challenging problem. This study proposes an Adaptive Firefly Algorithm (AFA) for dependent task scheduling in IoT-Fog computing. The proposed AFA is a modified version of the standard Firefly Algorithm (FA), considering the execution times of the submitted tasks, the impact of synchronization requirements, and the communication time between dependent tasks. As IoT-Fog computing depends mainly on distributed fog node servers that receive tasks in a dynamic manner, tackling the communications and synchronization issues between dependent tasks is becoming a challenging problem. The proposed AFA aims to address the dynamic nature of IoT-Fog computing environments. The proposed AFA mechanism considers a dynamic light absorption coefficient to control the decrease in attractiveness over iterations. The proposed AFA mechanism performance was benchmarked against the standard Firefly Algorithm (FA), Puma Optimizer (PO), Genetic Algorithm (GA), and Ant Colony Optimization (ACO) through simulations under light, typical, and heavy workload scenarios. In heavy workloads, the proposed AFA mechanism obtained the shortest average execution time, 968.98 ms compared to 970.96, 1352.87, 1247.28, and 1773.62 of FA, PO, GA, and ACO, respectively. The simulation results demonstrate the proposed AFA's ability to rapidly converge to optimal solutions, emphasizing its adaptability and efficiency in typical and heavy workloads.

KEYWORDS: Fog computing; scheduling; resource management; firefly algorithm; genetic algorithm; ant colony optimization

1 Introduction

The rapid evolution of the Internet of Things (IoT) has been a driving force behind numerous technological advancements, connecting an ever-growing array of devices and sensors [1–3]. An IoT system can be defined as a network of connected devices that interact and communicate with the cloud as well as between the devices themselves. IoT-Fog is a new computing paradigm that processes IoT data on servers close to the source of the data. Resource and task management in fog computing is considered a challenging problem due to the heterogeneity of IoT resources and the huge amount of data that need to be processed by fog servers [4,5]. Fog computing is an efficient computing solution that has been integrated into IoT systems as a complementary to cloud computing [6]. Fog computing aims to enhance the efficiency of IoT systems as the distributed fog servers process the IoT tasks near data sources, which reduces the processing and the

communication latency [7–9]. The architecture of IoT-Fog computing systems consists of three main layers, as shown in Fig. 1.

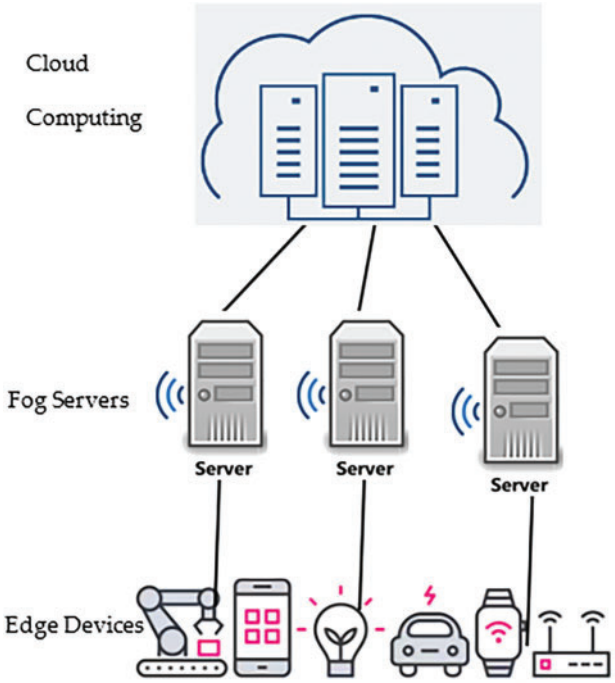

Figure 1: IoT-Fog computing solutions architecture

Fig. 1 describes the IoT-Fog computing architecture. The fog computing technology presents a multi-layered approach to data processing within an Internet of Things (IoT) ecosystem. The first layer of the fog computing architecture is the thing layer. In this layer, various IoT devices such as sensors, actuaries, and mobile devices generate data or perform specific functions based on data they receive from the environment. The middle layer in the IoT solutions architecture is the fog computing layer. This layer consists of fog nodes, which are often distributed closer to the data source. Fog nodes perform preprocessing, analysis, and storage of data to reduce latency, decrease bandwidth usage, and enable quicker decision-making. The topmost layer is the cloud computing layer. The cloud layer contains centralized data centers with high computational and storage capabilities. In the IoT system, the application data that requires intensive computation or long-term storage is sent from the fog nodes to the cloud data centers [10].

Task scheduling in IoT-Fog computing involves distributing and allocating IoT tasks to fog nodes to optimize the fog system's performance. Several task scheduling mechanisms are presented in the literature to schedule IoT-Fog computing tasks [11–13]. Integrating fog computing into IoT networks introduces new complexities, particularly in resource management, such as resource allocation and task scheduling [14,15]. The IoT-Fog tasks are critical in optimizing the performance of fog computing environments, requiring a balance between response time, cost, and resource utilization [12,16].

However, due to the heterogeneity of IoT-Fog resources and the dependencies between tasks, the scheduling process in fog computing is considered a challenging problem. The current task scheduling mechanism in IoT-Fog computing provides good but not optimal solutions [15,17]. Optimization mechanisms are used for task scheduling in IoT-Fog computing. The Firefly Algorithm mimics the behavior of fireflies and is a form of swarm intelligence, and it is adept at solving complex optimization problems [18–20].

The rapid growth of the Internet of Things (IoT) has introduced new challenges in managing fog computing resources effectively. IoT systems produce massive amounts of data that require efficient processing to maintain low latency and high reliability. IoT-Fog computing aims to allocate IoT tasks to devices closer to the data source. The main goals of IoT-Fog computing are to minimize system latency and increase its reliability. In IoT-Fog systems, the heterogeneity of resources and the dynamic nature of tasks produce significant challenges for the IoT-Fog computing task scheduling process. Dependent task scheduling in IoT-Fog computing environments requires an optimal mechanism considering synchronization requirements and communication delays [21–23].

Current task scheduling mechanisms present a good but not optimal scheduling process in IoT-Fog computing environments with synchronization requirements and communication delays. The main problem of scheduling dependent tasks in IoT-Fog computing is finding an optimal mapping between fog tasks and the available resources to optimize the allocation process that considers synchronization requirements and communication delays [24,25].

The main objective of this study is to present an Adaptive Firefly Algorithm (AFA) for dependent task scheduling in IoT-Fog computing. The fitness function of the proposed AFA mechanism considers three important factors: the execution times of tasks, the impact of task synchronization and the communication time between IoT-Fog computing tasks. As IoT-Fog computing depends mainly on distributed fog node servers that dynamically receive tasks, tackling the communications and synchronization issues between dependent tasks is becoming a challenging problem. The proposed AFA aims to address the dynamic nature of IoT-Fog computing environments. The proposed AFA mechanism considers a dynamic light absorption coefficient to control the decrease in attractiveness over iterations. Furthermore, the light absorption coefficient function provides an adaptive control of the exploration–exploitation balance over time in the proposed AFA mechanism. Considering the multilayered architecture of IoT-Fog computing, the proposed AFA mechanism is situated primarily in the Fog layer, where it adaptively schedules the IoT tasks to the fog nodes.

This study presents an Adaptive Firefly Algorithm (AFA) for dependent task scheduling in IoT-Fog computing environments. The proposed AFA uniquely incorporates a dynamic light absorption coefficient that adapts based on workload and iteration progress, enhancing convergence speed and solution accuracy. Furthermore, the proposed AFA mechanism addresses critical task synchronization and communication times issues, which are essential in dynamic fog computing scenarios with interdependent tasks. The simulation results revealed the proposed AFA's superiority under different workloads compared to established algorithms. The proposed AFA provides a significant improvement in scheduling efficiency, making it a valuable contribution to the IoT-Fog computing literature.

Additionally, the adaptive light absorption coefficient in our AFA dynamically balances exploration and exploitation, enhancing convergence speed and solution quality over iterations. This adaptive mechanism is not present in the evolutionary algorithm approach discussed in the cited work.

The method presented in [26] presents an evolutionary algorithm (EA) for clustering and scheduling tasks in IoT edge computing. This method is primarily focusing on improving resource utilization and reducing latency through task clustering. In contrast, the proposed AFA mechanism introduces an Adaptive Firefly Algorithm (AFA) tailored specifically for dependent task scheduling in IoT-Fog computing environments. The proposed AFA addresses task execution times and handle task synchronization and communication delays between dependent tasks, which are important issues in dynamic IoT-Fog networks.

The motivation for this research stems from the need to efficiently manage fog computing nodes, handling task dependencies, and considering the synchronization delays and communication overhead in the scheduling process.

This paper has five sections. Section 2 analyzes the related works. Section 3 demonstrates the proposed Adaptive Firefly Algorithm for task scheduling. Section 4 illustrates the performance evaluation. The conclusion is presented in Section 5.

2 Related Works

The Internet of Things (IoT) has emerged as an essential future technology. However, IoT devices suffer from limitations in computation and storage capacity [27]. Cloud computing is integrated with IoT to process IoT device's data remotely on the internet [28,29]. Regardless of efforts to improve IoT applications with cloud capabilities, IoT device computation issues remain unresolved, as IoT applications generally demand specific features such as mobility, geographical distribution, location awareness, and low latency [30,31]. Fog computing technology was introduced as a solution to enable computing at the edge of the network, offering new services for IoT devices [8,32,33]. In IoT-Fog computing, fog servers are the computing nodes that provide computational resources for IoT applications at the network edge [34]. In IoT-Fog computing, resource management processes such as resource allocation and task scheduling aim to provide short execution times and cost-effective computation processes for IoT devices [35–37]. In this context, enhancing the IoT task scheduling in IoT-Fog computing is crucial for optimizing the performance of IoT environments. The literature reveals a range of mechanisms and methods addressing task scheduling in IoT-Fog computing, focusing on improving the execution times of applications [17,38–40].

The study in [41] presents a bees swarm mechanism for enhancing scheduling in fog computing. The bees swarm mechanism is the trade-off between power consumption and computation latency when scheduling tasks in fog computing. The study in [42] proposed a job scheduling mechanism for IoT-Fog computing based on a modified artificial ecosystem-based optimization to choose a fog computing provider and allocate the IoT jobs on it [42]. Furthermore, the study in [33] has improved the Harris Hawks Optimization (HHO) mechanism and introduced a Discrete Opposition-based mechanism, indicated as DO-HHO, for IoT-Fog computing task scheduling.

A classification study of IoT-Fog computing task scheduling mechanisms on the basis of multiple factors such as throughput, execution times, and fairness of resource allocation is introduced in [43]. The classification process revealed that traditional task scheduling mechanisms have shown certain limitations in handling the dynamic nature of IoT-Fog computing environments [43].

Several studies in the literature examined have focused on adaptable and robust IoT task scheduling mechanisms, such as those offered by swarm intelligence [13,44,45].

Task scheduling mechanisms based on Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO) have been observed for their efficiency in resource management and task scheduling in IoT-Fog computing [46,47]. Furthermore, there is a growing interest in exploring alternative swarm-based mechanisms that could offer even greater efficiency and adaptability in IoT-Fog computing task scheduling [48,49]. Puma Optimizer (PO) is a new metaheuristic mechanism that mimics the intelligence and life of Pumas. PO has considered efficient exploration and exploitation techniques to enhance the optimization process [50]. The study in [51] presented a modified firefly algorithm for addressing task scheduling in cloud-edge environments. A new study in [52] presented task allocation mechanism using the fitness function of two metaheuristic methods: The Krill Herd Algorithm (KHA) and PSO. An artificial

Hummingbird mechanism for task scheduling in fog computing is combined with Opposition-Based Learning (OBL) optimization to optimize the scheduling process [53].

This study explores the application of an Adaptive Firefly Algorithm for task scheduling in fog computing for IoT networks. The standard Firefly Algorithm, characterized by its simplicity and flexibility, mimics fireflies' social and communicative behaviors [54–56]. By adopting the Firefly Algorithm for task scheduling in IoT-Fog computing, this research aims to contribute to the growing body of knowledge in efficient resource management for optimizing the IoT-Fog computing performance [57].

3 The Proposed Adaptive Firefly Algorithm for Task Scheduling in Fog Computing

This study proposes an Adaptive Firefly Algorithm (AFA) for optimizing the independent task scheduling process in IoT-Fog computing. The proposed AFA is a modified version of the standard Firefly Algorithm (FA) considering the execution times of the submitted tasks, the impact of synchronization requirements and the communication time between dependent tasks. As IoT-Fog computing depends mainly on distributed fog node servers that receive tasks in a dynamic manner, tackling the communications and synchronization issues between dependent tasks is becoming a challenging problem. The proposed AFA aims to address the dynamic nature of IoT-Fog computing environments.

3.1 The Proposed Algorithm Architecture

The structure of the proposed AFA mechanism for task scheduling in IoT-Fog computing consists of five phases: initialization, fitness evaluation, attractiveness and movement, iterative optimization, and convergence. The first phase of the proposed AFA mechanism is the initialization phase, in which the proposed AFA mechanism generates an initial random population of fireflies. Each firefly in the initial population represents a schedule of fog computing tasks to resources. The second phase of the proposed AFA mechanism is the fitness evaluation phase, in which each schedule in the firefly's population is evaluated using the objective function of the proposed AFA. The fitness function of the proposed AFA mechanism considers three important factors: the execution times of tasks, the impact of task synchronization and the communication time between IoT-Fog computing tasks. As in the standard Firefly Algorithm, the firefly's fitness represents the firefly's attractiveness or brightness, and the population movements are based on the attractiveness and the distance between the fireflies. In the proposed AFA mechanism, the attractiveness corresponds to the efficiency of the schedule. The attractiveness of a firefly decreases with the distance, which here represents the dissimilarity between population schedules. In the proposed AFA mechanism, each schedule in the population moves towards brighter fireflies, meaning schedules with less fitness value are more likely to be emulated. In the iterative optimization phase, in each iteration, every schedule in the firefly population adjusts its position in response to other fireflies to find a more efficient scheduling position in the solution search space. In the movement equation of the proposed AFA mechanism, the randomness factor α ensures that the proposed AFA mechanism explores different areas of the solution search space. This process enhances the search space convergence and prevents the mechanism from trapping in local optimal solutions. The last phase of the proposed AFA mechanism is the convergence phase. In this phase, the proposed AFA mechanism iteratively enhances the population schedules of each generation, converging towards an optimal or near-optimal solution. The optimal schedule is the one that achieves the best fitness value based on the objective function for all submitted tasks across the fog nodes. The key features of the proposed AFA mechanism are adaptability and efficiency. The proposed AFA mechanism adapts its movement parameters based on the synchronization process and the communication times between dependent IoT-Fog tasks. Furthermore, the proposed AFA mechanism is adaptable to various task scheduling scenarios in IoT-Fog computing and capable of handling dynamic and heterogeneous IoT-Fog environments. By minimizing task

execution times, the proposed AFA mechanism enhances the overall efficiency of the IoT-Fog computing system. The proposed AFA mechanism is suitable for IoT networks utilizing fog computing, where efficient resource utilization and quick response times are crucial. By optimizing task scheduling, the proposed AFA mechanism can significantly improve the performance of the IoT-Fog computing system, leading to more efficient and reliable IoT-Fog applications.

The communication type in the proposed AFA mechanism follows a broker-based approach. In the proposed AFA mechanism, a central broker or scheduler is configured [6]. This central broker is responsible for managing task assignments, coordinating communication between dependent tasks, and synchronizing task execution between the various fog computing nodes [58].

3.2 Mathematical Modelling of the Proposed AFA Mechanism

The following subsections provide an overview of how the proposed AFA mechanism works by describing its objective functions, population movements and convergence process. The process of the proposed AFA mechanism iterates continuously to adjust the population schedules, searching for more optimal solutions.

3.2.1 The Proposed AFA Mechanism Initialization

Let $T = [t_1, t_2, \ldots t_n]$ represent a set of n IoT-Fog computing-dependent tasks that are to be scheduled on $R = [r_1, r_2, \ldots\ldots rm]$ a set of m IoT-Fog server resources with corresponding speeds $S = \{s_1, s_2, \ldots, s_m\}$.

Let $C = \{c_{ij} \mid 1 \leq i, j \leq n\}$ represent the communication time required between tasks t_i and task t_j. This is particularly relevant for dependent tasks where data need to be transferred from one task to another.

Let $S = \{s_{ij} \mid 1 \leq i, j \leq n\}$ represent the synchronization times between tasks t_i and task t_j, capturing the additional time needed to synchronize ta6sks that are dependent on each other.

Let $L = \{l_1, l_2, \ldots, l_n\}$ represent the lengths of n IoT-Fog computing tasks, where li is the length of the task t_i.

Initialize the first random firefly population $Y_0 = [y_1, y_2, \ldots yk]$, where k is the number of fireflies. In the firefly population, y_i is a scheduling vector representing a task-fog server mapping, as each element in y_i corresponds to an IoT-Fog server resource allocated to a fog task.

The initialization phase of the proposed AFA mechanism is considered an important step as it sets the basis for the following exploration and exploitation phases of the AFA mechanism. This phase starts by generating a random initial population of fireflies. Each firefly in the population corresponds to a candidate solution for the task scheduling problem in the IoT-Fog environment. In the proposed AFA mechanism, the position of each firefly in the initial population is generated randomly within the solution search space, ensuring a diverse scheduling candidate at the beginning of the optimization process. This randomness process in the proposed AFA mechanism is constrained by the feasible schedules in the solution search space, considering the task synchronization and the communication time. The randomness process in the proposed AFA mechanism is based on a permutation function $P(r)$. $P(r)$ allocates the IoT-Fog tasks to fog server resources randomly. For each y_i, $P(r)$ generates a random sequence of resource indices for the IoT-Fog task allocation. The initial position of each firefly y_i is determined by applying $P(r)$ for the set of IoT-Fog tasks resulting in a random but feasible IoT-Fog task–resource mapping. The initial allocation for firefly y_i can be expressed as in Eq. (1):

$$y_i = P(r)(T) \tag{1}$$

where $P(r)(T)$ denotes the application of the random permutation function to the set of IoT-Fog tasks T, resulting in a random task–resource mapping that satisfies the IoT-Fog scheduling's constraints.

3.2.2 The AFA Mechanism Objective Function (Fitness Evaluation)

The proposed AFA mechanism calculates the execution time of each schedule in the population using the objective function. The execution time function considers the fog task length, fog resource speed, and the overheads due to communication and synchronization between dependent tasks.

The execution time e_{ij} for task t_i on resource r_j can be calculated by dividing the task length by the resource speed and adding any communication and synchronization overheads. If task t_i is dependent on the task t_k, the communication time c_{ik} and synchronization time s_{ik} need to be included in the execution time calculations. The proposed AFA mechanism calculates the execution time as in Eq. (2):

$$e_{ij} = \frac{l_i}{s_j} + \sum_{k \in D(i)} (c_{ik} + s_{ik}) \tag{2}$$

where $D(i)$ represents the set of tasks that t_i is directly dependent on, and it is assumed that c_{ik} and s_{ik} are zero if there is no direct dependency between tasks t_i and t_k. The proposed AFA calculates the total execution time of a schedule TE_{yi} as the sum of all individual IoT-Fog task execution times, adjusted for any additional wait times due to dependencies and synchronization requirements.

Eq. (2) calculates the execution time (e_{ij}) for task i when assigned to resource j. The execution time is consisting three components $\frac{l_i}{s_j}$, c_{ik} and s_{ik}. $\frac{l_i}{s_j}$ term represents the processing time, where l_i is the length of the task and s_j is the speed of the resource. The communication delay between task iand other dependent tasks, accounting for data transfer times between fog servers. s_{ik} is the synchronization delay, which represents the waiting time required to synchronize with dependent tasks. In real IoT-Fog environments, tasks often depend on each other, requiring both communication and synchronization delays to be considered in the total execution time.

If IoT-Fog computing tasks are executed in parallel, the total execution time is influenced by the task with the longest execution path, including dependencies, as in Eq. (3):

$$TE_{yi} = \max_{\forall t_i \in T} \left(\sum_{k \in D(i)} e_{ik} \right) \tag{3}$$

where $D(i)$ includes task t_i and all its dependencies. This formula calculates the maximum sum of execution times along any dependency path in the schedule. For example, assume a simple case with three tasks (t_1, t_2, t_3) where t_2 depends on t_1, and t_3 depends on t_2, and each task is assigned to a different resource. If the execution times including dependencies are 10, 20, and 15 ms, respectively, the total execution time would be the time taken to complete t_3, which is the last task in this dependency chain.

The objective function of the proposed AFA mechanism can be defined as in Eq. (4):

$$minimize\, F(Y) = \sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij} \left(\frac{l_i}{s_j} + \sum_{k \in D(i)} (c_{ik} + s_{ik}) \right) \tag{4}$$

where

- x_{ij} is a binary variable that equals 1 if task t_i is assigned to resource r_j and 0 otherwise.
- l_i is the length of task t_i.
- s_j is the speed of resource r_j.
- c_{ik} is the communication time required between task t_i and task t_k.

- s_{ik} is the synchronization time required between task t_i and task t_k
- $D(i)$ represents the set of tasks that task t_i is directly dependent on.

The dependency constraints model the dependencies between fog computing tasks to ensure that a task can only start if its dependent tasks have finished. The dependency constraint model defines the start time t_{starti} to represent the starting time of task t_i and the finish time $t_{finishi}$ to represent the finishing time of task t_i calculated as the start time of task t_i plus its execution time as in Eq. (5):

$$t_{finishi} = t_{starti} + e_{ij} \tag{5}$$

where e_{ij} is the execution time of IoT-Fog task t_i on resource r_j.

To ensure that task t_i does not start until all tasks t_k that it depends on have completed, the proposed AFA mechanism impose the following constraints for all tasks t_i and their dependencies t_k in set $D(i)$, which contains the indices of tasks that have be completed before task t_i can start as described in Eq. (6):

$$t_{starti} \geq t_{finishk} \qquad \forall k \in D(i) \tag{6}$$

This constraint ensures that the start time of task t_i is greater than or equal to the finish time of each task t_k it depends on.

Consider an example of fog computing scenario with three tasks t_1, t_2, and t_3, where t_2 depends on t_1 (i.e., t_2 cannot start until t_1 is completed). t_3 depends on both t_1 and t_2. The mathematical Constraints can be defined as:

$t_{start2} \geq t_{finish1}$

$t_{start3} \geq t_{finish2}$

$t_{start3} \geq t_{finish1}$(if t_3 also depends directly on t_1)

If t_1, t_2, and t_3 have execution times based on the resource they are assigned to, the equations also consider these values. For instance, if t_1 is assigned to r_1, t_2 to r_2, and t_3 to r_3, and each task–resource pair has a known execution time e_{ij}, then:

$t_{finish1} = t_{start1} + e_{11}$

$t_{finish2} = t_{start1} + e_{22}$

$t_{finish3} = t_{start1} + e_{33}$

These dependency constraints of modeling are crucial for scheduling dependent tasks in IoT-Fog computing environments, where tasks are interdependent. The scheduling process needs to consider not only resource allocation but also the order and timing of task execution.

3.2.3 Movement and Update of Fireflies of the Proposed AFA Mechanism

The third step of the proposed AFA mechanism is the movement of fireflies in the population towards a more attractive one. The proposed AFA mechanism updates the position of the firefly i towards the more attractive firefly j using Eq. (7) [14]:

$$y_{i+1} = y_i + \beta_0 e^{-\gamma d_{ij}^2}(y_i - y_j) + \alpha \cdot (\text{rand} - 0.5) \tag{7}$$

where β_0 is attractiveness at a distance of 0 (the execution time calculated using Eq. (3)), γ is the light absorption coefficient, d_{ij} is the distance between the schedules y_i and y_j, α is the randomization parameter,

and rand is a random number in [0, 1]. The distance between fireflies in the proposed AFA mechanism represents the dissimilarity between population schedules. The proposed AFA mechanism orders the fog resources in ascending order based on their speeds and calculates the distance between schedules as the difference in their resource allocations. This method transforms the distance between schedules into a sequence comparison task, where the sequence elements are the ordered positions of resources based on their speeds.

Considering the set of resources R, the proposed AFA mechanism orders the resources in ascending order of their speeds to obtain an ordered set $R_{\text{ordered}} = \{r_{o1}, r_{o2}, \ldots, r_{om}\}$, where $s_{o1} \leq s_{o2} \leq \ldots \leq s_{om}$. As defined previously, each schedule y_i in the firefly population represents a sequence of resource allocations to tasks, where each element in the sequence corresponds to the ordinal position of the allocated resource in the ordered set R_{ordered}. For example, if the task t_1 in schedule y_i is allocated to the fastest resource (which is now r_{o1}), the first element of y_i would be 1, representing the first position in the ordered resource list. Given two schedules y_i and y_j, each represented as a sequence of ordinal positions $y_i = [o_{i1}, o_{i2}, \ldots, o_{in}]$ and $y_j = [o_{j1}, o_{j2}, \ldots, o_{jn}]$ (with n tasks), the distance between these schedules based on the difference in resource allocations is calculated as in Eq. (8):

$$d(y_i, y_j) = \sum_{k=1}^{n} | o_{ik} - o_{jk} | \tag{8}$$

where $| o_{ik} - o_{jk} |$ calculates the absolute difference in the ordinal positions of the resources allocated to the task t_k in schedules y_i and y_j. This metric captures the dissimilarity in terms of resource allocation speed preferences between the two schedules. This distance calculation quantifies the differences between two IoT-Fog computing task schedules in terms of their resource speed. For example, consider a fog computing system with four resources $R = \{r_1, r_2, r_3, r_4\}$ with corresponding speeds $S = \{2, 5, 1, 10\}$. After ordering by speed, $R_{\text{ordered}} = \{r_3, r_1, r_2, r_4\}$. Suppose schedule y_i assigns tasks to resources based on some criteria, resulting in the allocation sequence corresponding to their speeds: $y_i = [r_3, r_1, r_2, r_4]$ and schedule y_j has a different allocation as $y_j = [r_1, r_4, r_3, r_2]$, the distance $d(y_i, y_j)$ can be calculated as:

$$d\left(y_i, y_j\right) = | 1 - 2 | + | 2 - 4 | + | 3 - 1 | + | 4 - 3 |$$
$$d\left(y_i, y_j\right) = 1 + 2 + 2 + 1 = 6$$

The calculated distance of 6 represents the cumulative difference in resource allocation positions between schedules y_i and y_j. This example shows that in the two different schedules y_i and y_j some tasks are allocated to resources with more similar speed differences of 1, while other tasks are allocated to resources with more disparate speed differences (differences of 2).

3.2.4 Optimization Process of the Proposed AFA Mechanism

In the proposed AFA mechanism, the optimization phase iterates the scheduling process for a predefined number of generations or until convergence. As defined in the movement phase, the proposed AFA mechanism updates the positions of IoT-Fog schedules based on their relative attractiveness and calculates the new fitness. Based on the objective function, the optimal IoT-Fog computing schedule is the one with the minimum total execution time considering the synchronization and the communication time between dependent tasks. The proposed AFA mechanism considers a dynamic light absorption coefficient γ to control the decrease in attractiveness over iterations. The proposed AFA mechanism defines the light absorption coefficient γ as a function of iteration number iter. The light absorption coefficient function $\gamma(\text{iter})$ provide an adaptive control of the exploration–exploitation balance over time in the proposed AFA mechanism. The

function of $\gamma(\text{iter})$ is described in Eq. (9):

$$\gamma(\text{iter}) = \gamma_0 \cdot e^{-\lambda \cdot \text{iter}} \tag{9}$$

where γ_0 is the initial light absorption coefficient at iter = 0, λ is a decay rate that determines how fast the exploration–exploitation balance changes towards exploitation and iter is the current iteration number. In the proposed AFA mechanism, the optimization process starts with a higher $\gamma(\text{iter})$ making the population schedules move towards distant schedules, resulting in high exploration for the solution search space. This process allows the proposed AFA mechanism to explore more areas of the solution space, searching for potential schedules. As the optimization iterations progress, the light absorption coefficient $\gamma(\text{iter})$ decreases, and the AFA mechanism changes to optimization exploitation. In the exploitation phase, the population schedules are less influenced by far-away schedules. This process makes the proposed AFA mechanism focus more on refining the population schedules found so far instead of searching for new schedules.

3.2.5 Convergence Criterion of the Proposed AFA Mechanism

The last phase of the proposed AFA mechanism is the optimization convergence criterion. In this phase, the proposed AFA mechanism defines two approaches to applying the convergence criterion: a threshold in the change in fitness values and a maximum number of iterations.

The termination criteria for the proposed AFA mechanism optimization process have two forms: a predefined maximum number of iterations and when an optimal or near-optimal solution is found. The proposed AFA mechanism considers a predefined maximum number of iterations of the optimization process as iter $\geq T$ max, where iter the current iteration number of optimizations and T max is the maximum allowed number of iterations. The second termination condition of the proposed AFA mechanism is when the change in the best fitness value over a certain number of optimization iterations is below a predefined threshold, as stated in Eq. (10):

$$\text{Terminate if } \mid f\text{best}(\text{iter}) - f\text{best}(\text{iter} - \Delta\text{iter}) \mid < \varepsilon \tag{10}$$

$f\text{best}(\text{iter})$ is the best fitness value at iteration iter, Δiter is a specified number of previous iterations to compare against and ε is the convergence threshold parameter.

The assumptions of the proposed Adaptive Firefly Algorithm (AFA) are based on practical scenarios in IoT-Fog computing. The proposed mechanism assumes tasks arrive dynamically as in real-world IoT systems where data is generated and processed in real-time. Furthermore, the proposed AFA mechanism considered task dependencies, such as synchronization and communication delays, which are common in distributed fog networks where tasks often rely on each other's output.

3.3 The Proposed Algorithm Pseudo-Code

The provided pseudo-code outlines the Adaptive Firefly Algorithm for Dependent Task Scheduling in IoT-Fog computing. This detailed pseudo-code illustrates each step of the proposed AFA mechanism. The key elements of the proposed AFA mechanism, such as the initialization of parameters and the repetitive process of optimization, play a crucial role in attaining the targeted results in independent task scheduling in IoT-Fog computing (see Algorithm 1):

Algorithm 1: Adaptive firefly algorithm for dependent task scheduling in IoT-Fog computing

1: ***1. Initialize Parameters:***

2: *- Set β_0 (base attractiveness at distance 0)*

(Continued)

Algorithm 1 (continued)

- *Set α (randomization parameter)*
- *Set γ_0 (initial light absorption coefficient)*
- *Set λ (decay rate for γ)*
- *Set $Tmax$ (maximum number of iterations)*
- *Set ε (convergence threshold)*
- *Determine the number of fireflies (k)*

2. Initialize Firefly Population (Y_0):

For *each firefly $i = 1$ to k:*

- *Generate a random schedule $y_i = P(r)(T)$, where $P(r)$ is a permutation function*
- *Calculate initial fitness $f(y_i)$ based on execution times synchronization, and communication times*

3. Optimization Loop:

- $iter = 0$
- ***Repeat until*** *$iter >= Tmax$ or convergence criterion is met:*
- *Update $\gamma(iter)$ using:* $\gamma(\text{iter}) = \gamma_0 \cdot e^{-\lambda \cdot \text{iter}}$
- ***For*** *each firefly i in the population:*
- ***For*** *each firefly $j \neq i$ in the population:*
- *Calculate distance d_{ij} between y_i and y_j:*
- ***For*** *each task $k = 1$ to n:*
- *Order resources R in ascending order based on their speeds to obtain an ordered set* $R_{\text{ordered}} = \{r_{o1}, r_{o2}, \ldots, r_{om}\}$, where $s_{o1} \leq s_{o2} \leq \ldots \leq s_{om}$.

For *each task $k = 1$ to n*

- Calculate the absolute difference in the ordinal positions of resources allocated to task k in y_i and y_j

$$d(y_i, y_j) = \sum_{k=1}^{n} \mid o_{ik} - o_{jk} \mid$$

- ***If*** *$f(y_i) < f(y_j)$ (j is more attractive than i):*
- *Update position of firefly i towards j using:*

$$y_{i+1} = y_i + \beta_0 e^{-\gamma d_{ij}^2}(y_i - y_j) + \alpha \cdot (\text{rand} - 0.5)$$

- *Recalculate fitness $f(y_{i\_new})$*
- *Update the best schedule based on the fitness values of all fireflies*
- $iter = iter + 1$
- *Check for convergence (Termination Criteria):*
- ***If*** *the change in the best fitness value over a certain number of iterations is below ε, terminate optimization*

4. Convergence Phase (Termination Criteria):

-***If*** *iter >= Tmax or |f_best(iter) - –_best(iter-Δiter)| < ε:*

- *Termination is triggered*
- *Identify the optimal schedule based on the best fitness value across all fireflies*
- *Return the optimal schedule and its corresponding fitness value*

5. End Algorithm

4 Performance Evaluation

This study conducted several simulation experiments to evaluate the performance of the proposed Adaptive Firefly Algorithm (AFA) in IoT-Fog computing. In the simulation process, this study considered

three distinct scenarios with varying numbers of jobs and resources to test the performance of the proposed Adaptive Firefly Algorithm (AFA) in different situations.

The parameters for the proposed Adaptive Firefly Algorithm (AFA) were selected based on optimization methods in the literature and fine-tuned through experiments. The main parameters of the proposed mechanism such as population size, light absorption coefficient (γ), and attractiveness (β) were set to balance exploration and exploitation. To ensure experiments fairness, the proposed AFA mechanism was compared against FA, PO, GA, and ACO under different workload scenarios. Each mechanism executed for 100 iterations and the average execution time is calculated.

4.1 Simulation Setup

Server Specifications: The simulations were performed on a DELL server equipped with an Intel(R) Core(TM) i9-10900 CPU, operating at 2.80 GHz with 9 cores and 32 GB of RAM.

Operating System: Ubuntu 20.04 LTS.

Simulation Software: The experiments were conducted using Java version 21, leveraging libraries such as Apache Commons Math for numerical computations to ensure precise execution of the algorithms. To fairly evaluate the proposed Adaptive Firefly Algorithm (AFA) mechanism, the simulation experimentation selected a mix of both well-established and modern optimization algorithms for comparison. These include the standard Firefly Algorithm (FA), Genetic Algorithm (GA), Ant Colony Optimization (ACO), and the recently developed Puma Optimizer (PO). The simulation considered PO specifically because it represents a strong, up-to-date optimization method, making it a valuable benchmark. All the mechanisms were tested under the same simulation conditions, using identical hardware, software environments, and a range of workload scenarios (light, typical, and heavy).

4.1.1 Simulation Metrics

Firefly Algorithm (FA) and Adaptive Firefly Algorithm (AFA)

- Population size: 50 fireflies
- Light absorption coefficient (γ): 1.0
- Attractiveness (β_0): 0.2
- Dynamic adjustment factor for AFA: Details provided in Section 3.2.4 on how γ is adjusted over iterations.

Puma Optimization

- Population size: 50
- Stalking factor r = 0.1
- Ambush factor = 1.0
- Hunt factor = 1.0
- Mutation rate = 0.1

Genetic Algorithm (GA)

- Population size: 50 chromosomes
- Crossover rate: 0.8
- Mutation rate: 0.02

Ant Colony Optimization (ACO)

- Number of ants: 50
- Pheromone evaporation rate (ρ): 0.5

4.1.2 Simulation Scenarios

Workload Characteristics

- Task Characteristics: Tasks varied from 1 KB to 10 MB, representing diverse IoT applications.
- Resource Characteristics: Fog nodes varied in computational capabilities from low-power edge devices to high-performance servers.

Scenarios

- Lightweight Load: 10 tasks distributed among 3 fog nodes.
- Typical Load: 50 tasks distributed among 10 fog nodes.
- Heavy Load: 100 tasks distributed among 20 fog nodes.

Iteration Process

- Each algorithm was run for 100 iterations, with execution times recorded for task allocation, computation, and communication overhead.
- Random Seed: A constant seed (e.g., seed = 42) was used in all simulations to ensure repeatability.

4.2 Experimental Results

This section describes the simulation experiments and the results.

4.2.1 The First Scenario: Lightweight Load

Fig. 2 presents the execution times for the proposed Adaptive Firefly Algorithm (AFA) across 100 iterations under a lightweight load. The results in this scenario reveal a consistent improvement in the execution times of the proposed AFA mechanism. In the first few iterations, the results show that the execution times are stable at approximately 186.37 ms. A significant drop to 155.85 ms is obtained in iteration 9. A gradual improvement in the AFA execution times is achieved in a number of iterations from 10 to 38. Another reduction in the execution times to 122.65 ms is achieved in iteration number 42. Towards the end of the 100 iterations of the simulation process, the proposed AFA mechanism execution times decrease further to 122.65 ms and then to 116.89 ms, with the final iterations stabilizing at approximately 113.19 ms.

Table 1 and Fig. 3 compare the execution times of the proposed Adaptive Firefly Algorithm (AFA) algorithm with the standard Firefly Algorithm (FA), Puma Optimizer (PO), Genetic Algorithm (GA), and Ant Colony Optimization (ACO) across different iterations under a lightweight load. The proposed AFA mechanism for task scheduling in IoT-Fog computing started with 251.31 ms execution time at the first iteration of the optimization process. The execution times of the proposed AFA mechanism continue to improve significantly as iterations progress. In the 50th iteration, the final execution time was reduced significantly to 134.54 ms. The execution times of the remaining scheduling mechanisms in the last iteration were FA, 209.59, PO, 109.44, GA, 247.95, and ACO, 395.93 ms. The ACO mechanism for task scheduling in IoT-Fog computing has the worst execution at 412.01 ms. These results show that the PO mechanism has the shortest execution times in optimizing execution times under lightweight load scenarios.

Table 2 and Fig. 4 compare the average execution times of the proposed AFA, FA, PO, GA, and ACO mechanisms when dealing with a lightweight workload scenario. The proposed PO mechanism demonstrates the best performance with an average execution time of 111.65 ms, while the average execution times of the proposed AFA, FA, GA, and ACO mechanisms were 193.03, 226.20, 261.66, and 397.22 ms, respectively.

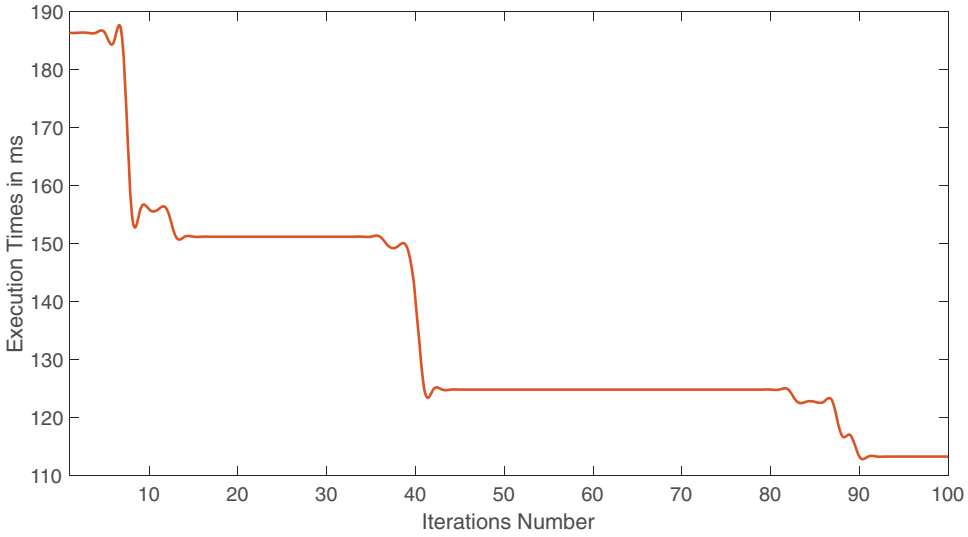

Figure 2: The execution times of 100 iterations of the proposed AFA (Lightweight load)

Table 1: The execution times of the proposed AFA algorithm compared to the firefly algorithm, GA and ACO in ms (Lightweight load)

Iteration no.	AFA	FA	PO	GA	ACO
1	251.31	285.10	125.97	319.42	412.01
10	226.66	232.26	114.35	282.50	395.93
20	226.66	225.42	109.44	258.91	395.93
30	174.23	215.60	109.44	247.95	395.93
40	147.04	214.24	109.44	247.95	395.93
50	134.54	209.59	109.44	247.95	395.93

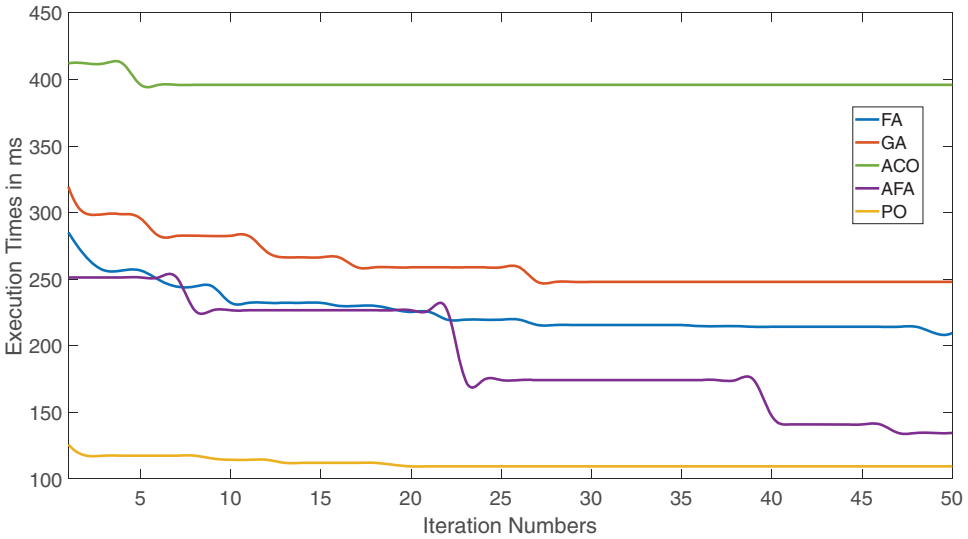

Figure 3: The execution times of the proposed AFA compared to FA, PO, GA and ACO (Lightweight load)

Table 2: The average execution times of the proposed AFA algorithm compared to FA, GA and ACO in ms (Lightweight load)

Algorithm	AFA	FA	PO	GA	ACO
Average execution times	193.03	226.20	111.65	261.66	397.22

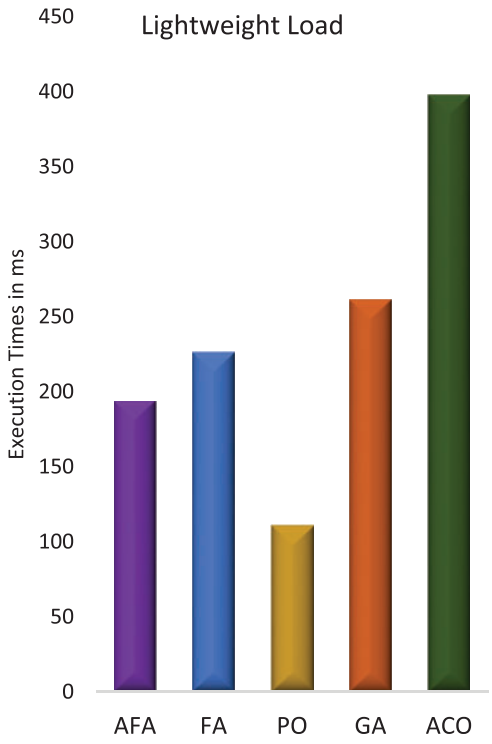

Figure 4: The average execution times of the proposed AFA compared to GA and ACO (Lightweight load)

4.2.2 The Second Scenario: Typical Load

As shown in Fig. 5, under a typical load scenario, the proposed Adaptive Firefly Algorithm (AFA) consistently improves its execution times. In the first iteration, the execution time of the proposed Adaptive Firefly Algorithm (AFA) was 509.79 ms. In the next iterations, the execution times of the proposed AFA mechanism dropped to 443.50 ms. Then, the proposed AFA mechanism optimization process showed a consistent decrease in the execution times until it reached 430.61 ms in the final iteration.

In summary, the proposed AFA mechanism for IoT-Fog scheduling in a typical workload scenario revealed a gradual decrease and stabilization of execution times.

Under a typical load scenario, the proposed AFA mechanism revealed efficient execution times compared to the FA, PO, GA, and ACO mechanisms described in Table 3 and Fig. 6. Throughout the simulation iterations, the four mechanisms improved execution times; however, the proposed AFA mechanism consistently performed the best in optimizing the execution times. In the last iteration of the simulation process, the execution time of the proposed mechanism was 419.73 ms compared to 428.49, 524.63, 565.39, and 877.62 ms for FA, PO, GA and ACO, respectively. These results emphasize the proposed AFA's potential as a highly effective task-scheduling mechanism for IoT-Fog computing in typical workloads.

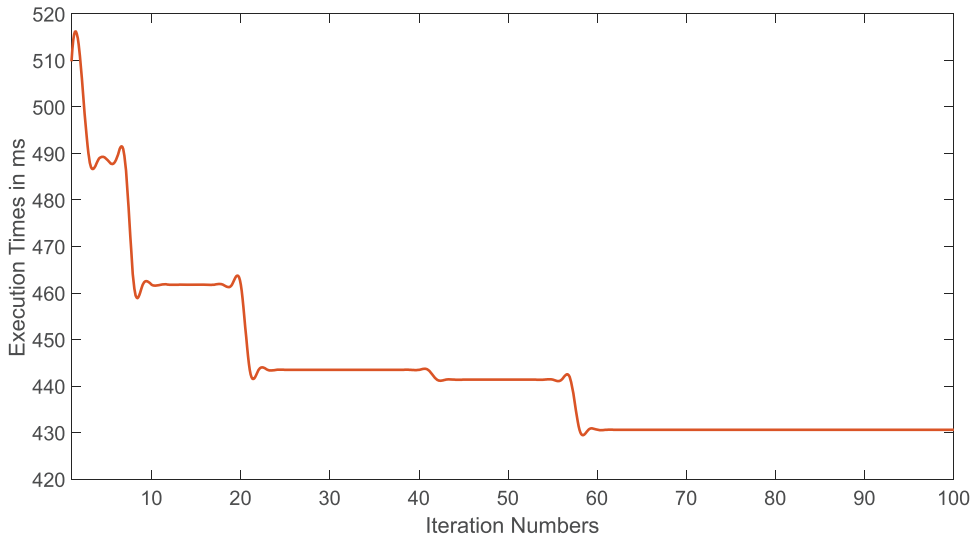

Figure 5: The execution times of 100 iterations of the proposed AFA (Typical load)

Table 3: The execution times of the proposed AFA algorithm compared to FA, PO, GA and ACO in ms (Typical load)

Iteration no.	AFA	FA	PO	GA	ACO
1	509.68	625.21	826.17	668.56	891.37
10	438.20	478.70	643.99	565.39	877.62
20	438.20	464.86	600.03	565.39	877.62
30	438.20	464.02	586.84	565.39	877.62
40	438.20	428.49	537.46	565.39	877.62
50	419.73	428.49	524.63	565.39	877.62

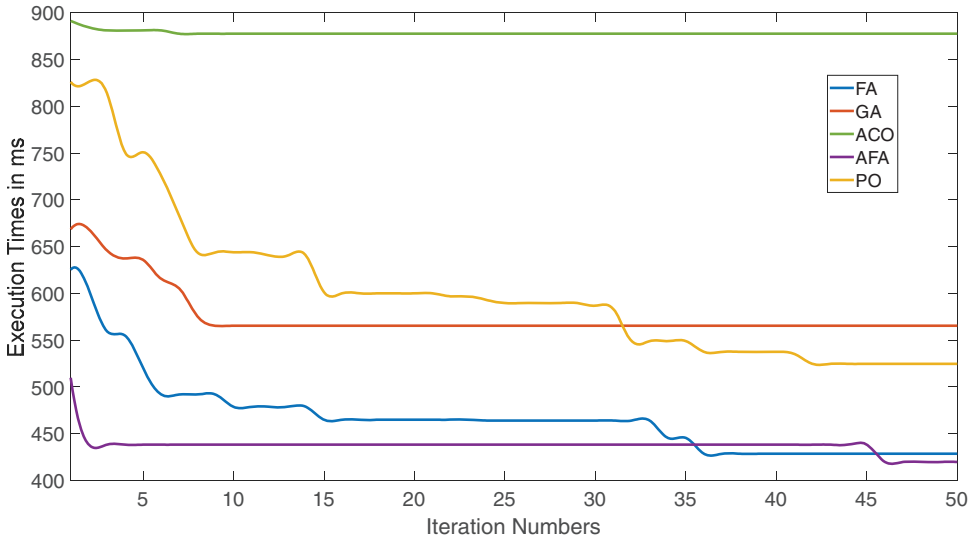

Figure 6: The execution times of the proposed AFA compared to FA, PO, GA and ACO (Typical load)

As shown in Table 4 and Fig. 7, the average execution time for the proposed Adaptive Firefly Algorithm AFA was 437.78 ms, while the average execution times of FA, PO, GA, and ACO were 467.34, 602.41, 575.98, and 878.32, respectively. These results indicate that the proposed algorithm has the shortest execution times on average compared to FA, GA, and ACO in a typical load scenario.

Table 4: The average execution times of the proposed AFA algorithm compared to GA and ACO in ms (Typical load)

Algorithm	AFA	FA	PO	GA	ACO
Average execution times	437.78	467.34	602.41	575.98	878.32

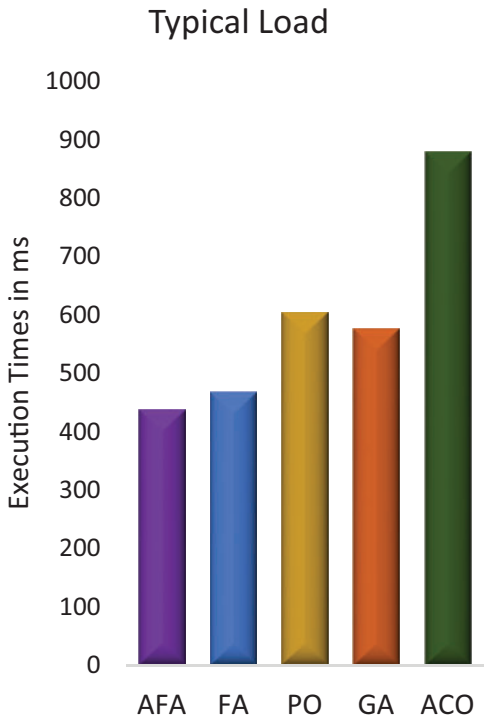

Figure 7: The average execution times of the proposed AFA compared to FA, PO, GA and ACO (Typical load)

4.2.3 The Third Scenario: Heavy Load

In the case of heavy load conditions, the proposed AFA mechanism started with a higher execution time of 1081.42 ms, as visualized in Fig. 8. Then, the execution time dropped to 1015.65 ms, indicating early optimization improvement. The next iteration of the optimization process revealed a gradual reduction in execution times until it stabilized at 935.86 ms for a long number of iterations. Towards the end of the 100 iterations, the proposed AFA mechanism showed a slow reduction in the execution time, ending with 882.90 ms at the last iteration.

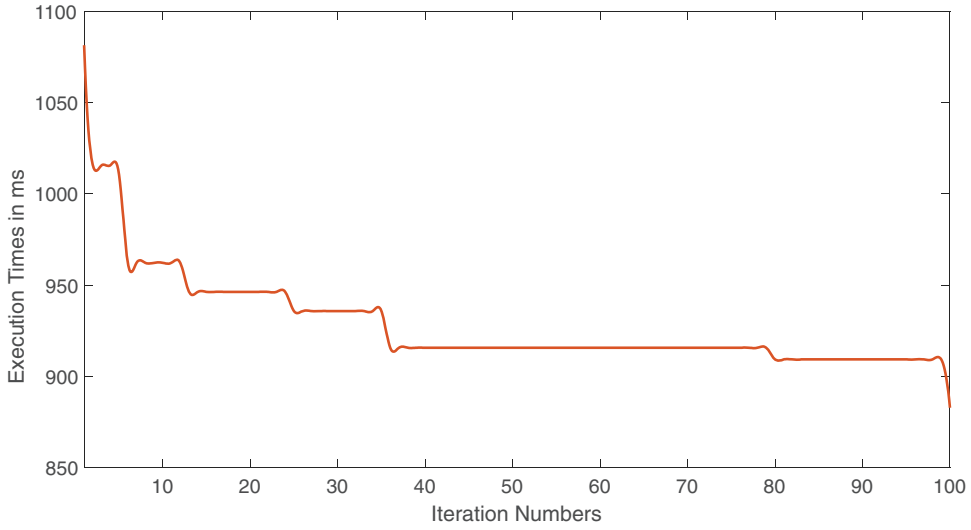

Figure 8: The execution times of 100 iterations of the proposed AFA (Heavy load)

Table 5 and Fig. 9 present the execution times for the proposed AFA mechanism compared to the standard FA, PO, GA, and ACO mechanisms under a heavy load condition. The proposed AFA mechanism begins with an execution time of 1104.90 ms, which is less than the execution times of FA (1227.09 ms), PO (1712.11 ms), GA (1369.14 ms), and ACO (1901.72 ms). The standard FA started at 1227.09 ms, indicating better optimization integration than GA and ACO. As iterations advance, the proposed AFA mechanism for IoT-Fog scheduling significantly improves. The execution times decreased to 864.93 ms by the 40th iteration and remained constant until the end of the simulation. The standard FA's results also show a good performance in the execution times through the optimization iterations by reducing the execution time to 865.36 ms, close to AFA's by the 50th iteration of the simulation. However, GA and ACO obtained longer execution times, ending with 1193.04 and 1763.48 ms, respectively.

In summary, Table 5 and Fig. 9 demonstrate that the proposed AFA mechanism for task scheduling in IoT-Fog computing outperformed FA, GA and ACO mechanisms under heavy load conditions.

Table 5: The execution times of the proposed AFA algorithm compared to FA, PO, GA and ACO in ms (Heavy load)

Iteration no.	AFA	PO	FA	GA	ACO
1	1104.90	1712.11	1227.09	1369.14	1901.72
10	1009.12	1519.46	1065.27	1262.59	1786.88
20	1009.12	1388.62	970.50	1258.96	1763.48
30	1004.61	1299.26	931.92	1227.96	1763.48
40	864.93	1186.56	910.28	1227.96	1763.48
50	864.93	1147.05	865.36	1193.04	1763.48

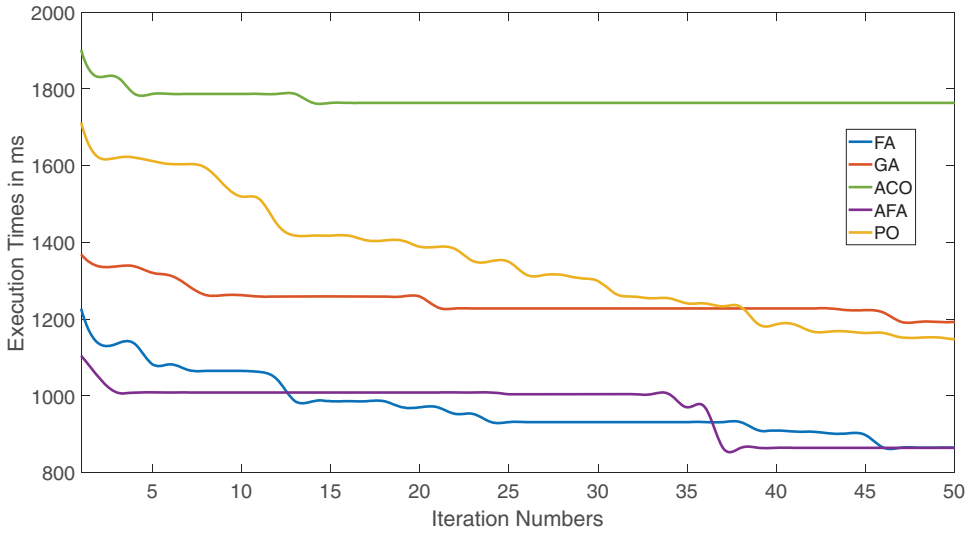

Figure 9: The execution times of the proposed AFA compared to FA, PO, GA and ACO (Heavy load)

Table 6 and Fig. 10 provide the average execution times for the proposed AFA mechanism compared to standard FA, PO, GA, and ACO mechanisms in heavy workload scenarios. The proposed AFA mechanism obtained the shortest average execution time, 968.98 ms. The average execution times of FA, PO, GA, and ACO in heavy workloads were 970.96, 1352.87, 1247.28, and 1773.62, respectively.

Table 6: The average execution times of the proposed AFA algorithm compared to FA, PO, GA and ACO in ms (Heavy load)

Algorithm	AFA	FA	PO	GA	ACO
Average execution times	968.98	970.96	1352.87	1247.28	1773.62

The simulation experiments demonstrate that the proposed Adaptive Firefly Algorithm (AFA) mechanism effectively reduces execution time and handles task dependencies under various workloads (light, typical, and heavy). These results indicate that the proposed AFA mechanism can scale to large-sized fog computing networks with dynamic task arrivals. However, as the number of tasks and fog nodes increases significantly, the proposed AFA mechanism computational overhead may rise due to the iterative nature of the firefly algorithm for the task scheduling optimization process. In terms of robustness, the proposed AFA mechanism adaptively balances the optimization exploration and exploitation process, which allows the proposed mechanism to perform well under different workloads.

4.3 Results and Discussion

This section presents the main results of the proposed adaptive firefly algorithm for dependent task scheduling in IoT-Fog computing.

4.3.1 The Results of the Study

The proposed Adaptive Firefly Algorithm (AFA) performance was evaluated under three workload scenarios—light, typical, and heavy—using execution time as an evaluation metric. The proposed AFA considered the synchronization requirements and communication delays in the execution times calculations.

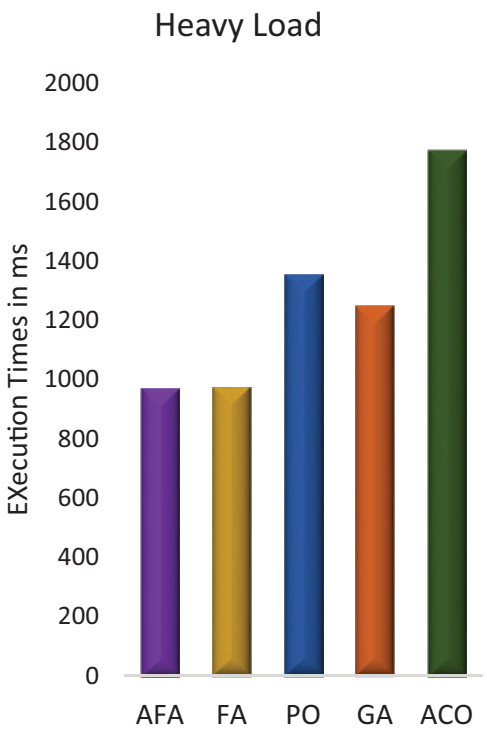

Figure 10: The average execution times of the proposed AFA compared to FA, PO, GA and ACO (Heavy load)

The simulation results demonstrate consistent improvement in execution times across all scenarios, with AFA achieving an average execution time of 968.98 ms under heavy workloads compared to 970.96, 1352.87, 1247.28, and 1773.62 ms for FA, PO, GA, and ACO, respectively.

4.3.2 Statistical Analysis

The experimentation process considered a statistical analysis to confirm that the proposed AFA mechanism achieves faster convergence and more efficient scheduling solutions than other scheduling mechanisms. Under typical workloads, the proposed AFA mechanism reduced execution time by approximately 7% compared to FA and 24% compared to GA. These improvements in execution times are due to the dynamic light absorption coefficient, which balances exploration and exploitation during optimization.

4.3.3 Comparison with Related Work

Current task scheduling mechanisms, such as the Artificial Hummingbird Algorithm, focus on optimizing resource allocation processes and latency reduction of the submitted tasks. However, these mechanisms lack the techniques to adapt dynamically to synchronization and communication delays. The proposed AFA mechanism design addresses these challenges, making it more robust in handling dependent tasks in dynamic IoT-Fog environments.

4.3.4 Practical Implications

The proposed AFA mechanism's robustness and scalability suit large-scale IoT-Fog computing environments with varied workloads. AFA adaptability allows IoT-Fog computing system to perform well in stable

and dynamic environments, providing a practical solution for IoT applications requiring low latency and efficient resource utilization.

5 Conclusions

This study proposed an Adaptive Firefly Algorithm (AFA) for dependent task scheduling in IoT-Fog computing. The performance evaluation of the proposed Adaptive Firefly Algorithm (AFA) under typical and heavy workloads confirms its efficiency in IoT-Fog computing job scheduling. The proposed AFA mechanism consistently outperforms the standard Firefly Algorithm (FA), Puma Optimizer (PO), Genetic Algorithm (GA) and Ant Colony Optimization (ACO) across typical and heavy load scenarios. Puma Optimizer (PO) obtained the shortest execution time under a lightweight load. The proposed AFA's ability to identify and exploit efficient task schedules proves its potential for real-time applications in IoT-Fog networks. The proposed AFA mechanism balanced the scheduling solution search space exploration and exploitation process. This balance significantly enhanced adaptability to task scheduling optimization's proposed AFA mechanism.

However, the proposed AFA mechanism has some limitations. Its computational overhead may be significant for resource-constrained environments such as wireless sensor networks (WSNs). Furthermore, the proposed AFA mechanism's energy consumption is another consideration, as its iterative nature may not be optimal for battery-powered devices.

Future work will address these limitations by optimizing the proposed AFA mechanism for low-power environments. Furthermore, future work may explore the AFA's scalability and robustness in larger-scale fog computing networks and potential combinations with other optimization mechanisms for enhanced performance of IoT-Fog computing.

Acknowledgement: We are thankful to the Deanship of Graduate Studies and Scientific Research at Najran University for funding this work under the Easy Funding Program grant code (NU/EFP/SERC/13/166).

Funding Statement: The authors are thankful to the Deanship of Graduate Studies and Scientific Research at Najran University for funding this work under the Easy Funding Program grant code (NU/EFP/SERC/13/166).

Availability of Data and Materials: The authors confirm that the data supporting the findings of this study are available within the article.

Ethics Approval: Not applicable.

Conflicts of Interest: The author declares no conflicts of interest to report regarding the present study.

References

1. Madakam S, Ramaswamy R, Tripathi S. Internet of Things (IoT): a literature review. J Comput Commun. 2015;3(5):164–73. doi:10.4236/jcc.2015.35021.
2. Gokhale P, Bhat O, Bhat S. Introduction to IoT. Int Adv Res J Sci Eng Technol. 2018;5(1):41–4.
3. Mohamad Noor MB, Hassan WH. Current research on Internet of Things (IoT) security: a survey. Comput Netw. 2019;148:283–94. doi:10.1016/j.comnet.2018.11.025.
4. Ni J, Zhang K, Lin X, Shen X. Securing fog computing for Internet of Things applications: challenges and solutions. IEEE Commun Surv Tutor. 2018;20(1):601–28. doi:10.1109/COMST.2017.2762345.
5. Bittencourt L, Immich R, Sakellariou R, Fonseca N, Madeira E, Curado M, et al. The Internet of Things, fog and cloud continuum: integration and challenges. Internet Things. 2018;3:134–55. doi:10.1016/j.iot.2018.09.005.
6. Morabito R, Cozzolino V, Ding AY, Beijar N, Ott J. Consolidate IoT edge computing with lightweight virtualization. IEEE Netw. 2018;32(1):102–11. doi:10.1109/MNET.2018.1700175.

7. Atlam H, Walters R, Wills G. Fog computing and the Internet of Things: a review. Big Data Cogn Comput. 2018;2(2):10. doi:10.3390/bdcc2020010.
8. Anawar MR, Wang S, Azam Zia M, Jadoon AK, Akram U, Raza S. Fog computing: an overview of big IoT data analytics. Wirel Commun Mob Comput. 2018;2018(1):7157192. doi:10.1155/2018/7157192.
9. Dastjerdi AV, Buyya R. Fog computing: helping the Internet of Things realize its potential. Computer. 2016;49(8):112–6. doi:10.1109/MC.2016.245.
10. Khezri E, Yahya RO, Hassanzadeh H, Mohaidat M, Ahmadi S, Trik M. DLJSF: data-locality aware job scheduling IoT tasks in fog-cloud computing environments. Results Eng. 2024;21:101780. doi:10.1016/j.rineng.2024.101780.
11. Mukherjee M, Guo M, Lloret J, Iqbal R, Zhang Q. Deadline-aware fair scheduling for offloaded tasks in fog computing with inter-fog dependency. IEEE Commun Lett. 2020;24(2):307–11. doi:10.1109/LCOMM.2019.2957741.
12. Alizadeh MR, Khajehvand V, Rahmani AM, Akbari E. Task scheduling approaches in fog computing: a systematic review. Int J Commun Syst. 2020;33(16):e4583. doi:10.1002/dac.4583.
13. Yin C, Fang Q, Li H, Peng Y, Xu X, Tang D. An optimized resource scheduling algorithm based on GA and ACO algorithm in fog computing. J Supercomput. 2024;80(3):4248–85. doi:10.1007/s11227-023-05571-y.
14. Aladwani T. Scheduling IoT healthcare tasks in fog computing based on their importance. Procedia Comput Sci. 2019;163:560–9. doi:10.1016/j.procs.2019.12.138.
15. Matrouk K, Alatoun K. Scheduling algorithms in fog computing: a survey. Int J Networked Distrib Comput. 2021;9(1):59–74. doi:10.2991/ijndc.k.210111.001.
16. Goudarzi M, Palaniswami M, Buyya R. Scheduling IoT applications in edge and fog computing environments: a taxonomy and future directions. ACM Comput Surv. 2023;55(7):1–41. doi:10.1145/3544836.
17. Jamil B, Shojafar M, Ahmed I, Ullah A, Munir K, Ijaz H. A job scheduling algorithm for delay and performance optimization in fog computing. Concurr Comput. 2020;32(7):e5581. doi:10.1002/cpe.5581.
18. Yang X-S, Slowik A. Firefly algorithm. In: Swarm intelligence algorithms. Boca Raton, FL, USA: CRC Press; 2020. p. 163–74.
19. Wu J, Wang YG, Burrage K, Tian YC, Lawson B, Ding Z. An improved firefly algorithm for global continuous optimization problems. Expert Syst Appl. 2020;149:113340. doi:10.1016/j.eswa.2020.113340.
20. Yousif A. An enhanced firefly algorithm for time shared grid task scheduling. Appl Artif Intell. 2021;35(15):1567–86. doi:10.1080/08839514.2021.1987708.
21. Hosseinzadeh M, Azhir E, Lansky J, Mildeova S, Ahmed OH, Malik MH, et al. Task scheduling mechanisms for fog computing: a systematic survey. IEEE Access. 2023;11:50994–1017. doi:10.1109/ACCESS.2023.3277826.
22. Thakur R, Sikka G, Bansal U, Giri J, Mallik S. Deadline-aware and energy efficient IoT task scheduling using fuzzy logic in fog computing. Multimed Tools Appl. 2024. doi: 10.1007/s11042-024-19509-w.
23. Alsadie D. Advancements in heuristic task scheduling for IoT applications in fog-cloud computing: challenges and prospects. PeerJ Comput Sci. 2024;10:e2128. doi:10.7717/peerj-cs.2128.
24. Wadhwa H, Aron R. Optimized task scheduling and preemption for distributed resource management in fog-assisted IoT environment. J Supercomput. 2023;79(2):2212–50. doi:10.1007/s11227-022-04747-2.
25. Ali Ibrahim M, Askar S. An intelligent scheduling strategy in fog computing system based on multi-objective deep reinforcement learning algorithm. IEEE Access. 2023;11:133607–22. doi:10.1109/ACCESS.2023.3337034.
26. Yousif A, Bashir MB, Ali A. An evolutionary algorithm for task clustering and scheduling in IoT edge computing. Mathematics. 2024;12(2):281. doi:10.3390/math12020281.
27. Kumar S, Tiwari P, Zymbler M. Internet of Things is a revolutionary approach for future technology enhancement: a review. J Big Data. 2019;6(1):11. doi:10.1186/s40537-019-0268-2.
28. Wu H, Wolter K, Jiao P, Deng Y, Zhao Y, Xu M. EEDTO: an energy-efficient dynamic task offloading algorithm for blockchain-enabled IoT-edge-cloud orchestrated computing. IEEE Internet Things J. 2021;8(4):2163–76. doi:10.1109/JIOT.2020.3033521.
29. Nassereddine M, Khang A. Applications of Internet of Things (IoT) in smart cities. In: Advanced IoT technologies and applications in the Industry 4.0 digital economy. Boca Raton, FL, USA: CRC Press; 2024. p. 109–36.
30. Alli AA, Alam MM. The fog cloud of things: a survey on concepts, architecture, standards, tools, and applications. Internet Things. 2020;9:100177. doi:10.1016/j.iot.2020.100177.

31. Tavana M, Hajipour V, Oveisi S. IoT-based enterprise resource planning: challenges, open issues, applications, architecture, and future research directions. Internet Things. 2020;11:100262. doi:10.1016/j.iot.2020.100262.
32. Singh J, Singh P, Gill SS. Fog computing: a taxonomy, systematic review, current trends and research challenges. J Parallel Distrib Comput. 2021;157:56–85. doi:10.1016/j.jpdc.2021.06.005.
33. Javaheri D, Gorgin S, Lee JA, Masdari M. An improved discrete Harris hawk optimization algorithm for efficient workflow scheduling in multi-fog computing. Sustain Comput Inform Syst. 2022;36:100787. doi:10.1016/j.suscom.2022.100787.
34. Kaur A, Singh P, Nayyar A. Fog computing: building a road to IoT with fog analytics. In: Tanwar S, editor. Fog data analytics for loT applications: next generation process model with State of the Art Technologies. Singapore: Springer Singapore; 2020. p. 59–78.
35. Ghobaei-Arani M, Souri A, Rahmanian AA. Resource management approaches in fog computing: a comprehensive review. J Grid Comput. 2020;18(1):1–42. doi:10.1007/s10723-019-09491-1.
36. Martinez I, Hafid AS, Jarray A. Design, resource management, and evaluation of fog computing systems: a survey. IEEE Internet Things J. 2021;8(4):2494–516. doi:10.1109/JIOT.2020.3022699.
37. Rashid Dar AB, Ravindran D. Fog computing resource optimization: a review on current scenarios and resource management. Baghdad Sci J. 2019;16(2):419. doi:10.21123/bsj.2019.16.2.0419.
38. Elavarasi R, Silas S. Survey on job scheduling in fog computing. In: 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI); 2019 Apr 23–25; Tirunelveli, India.
39. Yang X, Rahmani N. Task scheduling mechanisms in fog computing: review, trends, and perspectives. Kybernetes. 2021;50(1):22–38. doi:10.1108/K-10-2019-0666.
40. Nazir S, Shafiq S, Iqbal Z, Zeeshan M, Tariq S, Javaid N. Cuckoo optimization algorithm based job scheduling using cloud and fog computing in smart grid. In: Advances in Intelligent Networking and Collaborative Systems: The 10th International Conference on Intelligent Networking and Collaborative Systems (INCoS-2018); Berlin/Heidelberg, Germany: Springer; 2019.
41. Bitam S, Zeadally S, Mellouk A. Fog computing job scheduling optimization based on bees swarm. Enterp Inf Syst. 2018;12(4):373–97. doi:10.1080/17517575.2017.1304579.
42. Abd Elaziz M, Abualigah L, Attiya I. Advanced optimization technique for scheduling IoT tasks in cloud-fog computing environments. Future Gener Comput Syst. 2021;124:142–54. doi:10.1016/j.future.2021.05.026.
43. Razzaq S, Wahid A, Khan F, Amin NU, Shah MA, Akhunzada A, et al. Scheduling algorithms for high-performance computing: an application perspective of fog computing. In: Recent trends and advances in wireless and IoT-enabled networks. Berlin/Heidelberg, Germany: Springer; 2019. p. 107–17.
44. Potu N, Jatoth C, Parvataneni P. Optimizing resource scheduling based on extended particle swarm optimization in fog computing environments. Concurr Comput. 2021;33(23):e6163. doi:10.1002/cpe.6163.
45. Yeh WC, Lai CM, Tseng KC. Fog computing task scheduling optimization based on multi-objective simplified swarm optimization. J Phys: Conf Ser. 2019;1411(1):012007. doi:10.1088/1742-6596/1411/1/012007.
46. Vispute SD, Vashisht P. Energy-efficient task scheduling in fog computing based on particle swarm optimization. SN Comput Sci. 2023;4(4):391. doi:10.1007/s42979-022-01639-3.
47. Gu J, Mo J, Li B, Zhang Y, Wang W. A multi-objective fog computing task scheduling strategy based on ant colony algorithm. In: 2021 IEEE 4th International Conference on Information Systems and Computer Aided Education (ICISCAE); 2021 Sep 24–26; Dalian, China.
48. Mon MM, Khine MA. Scheduling and load balancing in cloud-fog computing using swarm optimization techniques: a survey [dissertation]. Yangon, Myanmar: University of Computer Studies; 2019.
49. Subramoney D, Nyirenda CN. Multi-swarm PSO algorithm for static workflow scheduling in cloud-fog environments. IEEE Access. 2022;10:117199–214. doi:10.1109/ACCESS.2022.3220239.
50. Abdollahzadeh B, Khodadadi N, Barshandeh S, Trojovský P, Gharehchopogh FS, El-kenawy EM, et al. *Puma* optimizer (PO): a novel metaheuristic optimization algorithm and its application in machine learning. Clust Comput. 2024;27(4):5235–83. doi:10.1007/s10586-023-04221-5.
51. Yin L, Sun J, Zhou J, Gu Z, Li K. ECFA: an efficient convergent firefly algorithm for solving task scheduling problems in cloud-edge computing. IEEE Trans Serv Comput. 2023;16(5):3280–93.

52. Javanmardi S, Sakellari G, Shojafar M, Caruso A. Why it does not work? Metaheuristic task allocation approaches in Fog-enabled Internet of Drones. Simul Model Pract Theory. 2024;133:102913. doi:10.1016/j.simpat.2024.102913.
53. Ghafari R, Mansouri N. An efficient task scheduling in fog computing using improved artificial hummingbird algorithm. J Comput Sci. 2023;74:102152. doi:10.1016/j.jocs.2023.102152.
54. Yousif A, Alqhtani SM, Bashir MB, Ali A, Hamza R, Hassan A, et al. Greedy firefly algorithm for optimizing job scheduling in IoT grid computing. Sensors. 2022;22(3):850. doi:10.3390/s22030850.
55. Esa DI, Yousif A. Scheduling jobs on cloud computing using firefly algorithm. Int J Grid Distrib Comput. 2016;9(7):149–58. doi:10.14257/ijgdc.2016.9.7.16.
56. Bacanin N, Zivkovic M, Bezdan T, Venkatachalam K, Abouhawwash M. Modified firefly algorithm for workflow scheduling in cloud-edge environment. Neural Comput Appl. 2022;34(11):9043–68. doi:10.1007/s00521-022-06925-y.
57. Ammari AC, Labidi W, Mnif F, Yuan H, Zhou M, Sarrab M. Firefly algorithm and learning-based geographical task scheduling for operational cost minimization in distributed green data centers. Neurocomputing. 2022;490:146–62. doi:10.1016/j.neucom.2022.01.052.
58. Aljuhani A, Alhubaishy A. Dynamic cloud resource allocation: a broker-based multi-criteria approach for optimal task assignment. Appl Sci. 2024;14(1):302. doi:10.3390/app14010302.