Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.050380

ARTICLE

Enhanced Hybrid Equilibrium Strategy in Fog-Cloud Computing Networks with Optimal Task Scheduling

Muchang Rao and Hang Qin*

School of Computer Science, Yangtze University, Jingzhou, 434000, China

*Corresponding Author: Hang Qin. Email: hangqin100@hotmail.com

Received: 05 February 2024 Accepted: 02 April 2024 Published: 15 May 2024

ABSTRACT

More devices in the Intelligent Internet of Things (AIoT) result in an increased number of tasks that require low latency and real-time responsiveness, leading to an increased demand for computational resources. Cloud computing's low-latency performance issues in AIoT scenarios have led researchers to explore fog computing as a complementary extension. However, the effective allocation of resources for task execution within fog environments, characterized by limitations and heterogeneity in computational resources, remains a formidable challenge. To tackle this challenge, in this study, we integrate fog computing and cloud computing. We begin by establishing a fog-cloud environment framework, followed by the formulation of a mathematical model for task scheduling. Lastly, we introduce an enhanced hybrid Equilibrium Optimizer (EHEO) tailored for AIoT task scheduling. The overarching objective is to decrease both the makespan and energy consumption of the fog-cloud system while accounting for task deadlines. The proposed EHEO method undergoes a thorough evaluation against multiple benchmark algorithms, encompassing metrics like makespan, total energy consumption, success rate, and average waiting time. Comprehensive experimental results unequivocally demonstrate the superior performance of EHEO across all assessed metrics. Notably, in the most favorable conditions, EHEO significantly diminishes both the makespan and energy consumption by approximately 50% and 35.5%, respectively, compared to the second-best performing approach, which affirms its efficacy in advancing the efficiency of AIoT task scheduling within fog-cloud networks.

KEYWORDS

Artificial intelligence of things; fog computing; task scheduling; equilibrium optimizer; differential evaluation algorithm; local search

1 Introduction

As the Internet of Things (IoT) and artificial intelligence (AI) technologies advance, the conventional IoT landscape has transitioned into the realm of the AIoT [1]. This evolution represents a distinctive trajectory in the future of IoT [2]. AIoT technologies have found extensive applications across diverse industries, including intelligent transportation [3], smart homes [4,5], smart cites [6,7], etc. AIoT contains a large number of applications. The large number of tasks generated by these applications are mostly delay-sensitive tasks, which put forward higher requirements for real-time

computing [8,9]. Owing to the extensive data generated by diverse applications, the demand for computing power has significantly increased [10]. The computing and storage resources of the AIOT device itself cannot meet this challenge, necessitating the transmission of data to the cloud for processing [11]. Yet, long-distance communication incurs heightened latency [12,13], which is deemed unfit for numerous AIoT applications. Once the latency is high, it will have unimaginable consequences, such as the Internet of Vehicles, smart medical care, etc., that require real-time response [14,15].

To address specific constraints of cloud computing and better cater to the diverse application scenarios of the IoT, fog computing has surfaced as a timely solution. As an extension of the cloud computing concept, Cisco formally introduced fog computing as a new computing model in 2012 [16]. Serving as an intermediary layer between cloud computing and terminal devices brings computing resources in proximity to the devices and is more suitable for processing delay-sensitive tasks [17]. Fog nodes have certain computing resources and storage capabilities, which extend the advantages of cloud computing to network edge devices. In contrast to AIoT equipment, fog computing possesses superior computing resources and enhanced computational capabilities. Distinguishing itself from cloud computing, fog computing is characterized by reduced latency [18]. To fully leverage the unique advantages of both fog computing and cloud computing, a burgeoning computational paradigm involves integrating these two approaches to process tasks [19]. Indeed, computational resources are inherently limited and diverse in nature, mirroring the heterogeneous characteristics of AIoT tasks. With varying latency tolerances and computational resource demands, allocating suitable nodes for each task within fog-cloud environments poses a significant challenge in resource management. One of the primary concerns in computational resource management is the reduction of system latency, which directly impacts the quality of service for users [20–22]. Additionally, the utilization of resources entails a substantial energy cost, making the enhancement of energy efficiency another critical challenge in resource management [23–25]. Hence, there is an immediate necessity for an efficient scheduling strategy that can judiciously allocate computational resources for AIoT tasks, striking a harmonious balance between the competing objectives of minimizing system latency and energy consumption [26].

Task scheduling poses a formidable challenge due to its NP complexity, prompting numerous studies to propose various approaches to tackle this issue. However, existing methodologies are not without limitations. Traditional classical task scheduling methods often underperform in this domain, while machine learning techniques, particularly deep learning, offer promising results but require substantial hardware resources and entail long execution times. Meta-heuristic algorithms, characterized by their simplicity and high efficiency [27], represent a widely adopted approach to task scheduling. They can provide optimal scheduling solutions within a reasonable timeframe [28,29]. However, most existing research in this area focuses solely on optimizing task scheduling for either the fog or cloud layer [30–32], failing to address the simultaneous requirements of low-latency and high computational demands of AIoT tasks. Additionally, certain methodologies only account for the scheduling effects of small-scale tasks [33–37], neglecting to comprehensively evaluate the methods' applicability across various scenarios. Moreover, some studies only consider single-objective optimization, concentrating solely on reducing system latency or energy consumption [38–41]. This approach may not ensure high-quality user services and low energy costs simultaneously.

Equilibrium Optimizer (EO) is a physics-inspired metaheuristic algorithm recently introduced by Faramarzi et al. [42], and has demonstrated its excellent performance in multiple fields, such as image segmentation, engineering technology, photovoltaic models, economic dispatch, etc. [43]. In comparison to many other metaheuristic algorithms, EO demonstrates superior exploration and exploitation capabilities, leading to faster convergence. This inherent efficiency enables EO to generate task scheduling results within a notably shorter timeframe. In contrast to genetic algorithm (GA),

it possesses the benefits of having fewer parameters and ease of implementation. EO has better exploration and exploitation capabilities, and particles with concentration are treated as search agents. While classic particle swarm optimization (PSO) relies on a single optimal solution to update particles, the algorithm is prone to getting trapped in local optima [44]. In contrast, EO constructs a balanced pool of four optimal solutions and their average solution. The particle concentration is revised through the random selection of a candidate solution from the equilibrium pool. Each candidate solution is assigned an equal probability of being chosen, thus alleviating the risk of getting trapped in local optima throughout the optimization process.

Therefore, based on the aforementioned challenges and the current shortcomings in research, this paper proposes an improved metaheuristic algorithm based on EO to address the AIoT task scheduling problem in fog-cloud environments while considering the optimization of makespan and energy consumption. Our principal contributions are threefold:

1) We formulate a model for a heterogeneous fog-cloud system, taking into account a comprehensive array of factors such as diverse computational resources, network constraints, and varying task characteristics. This model is designed to intricately weigh the interplay between energy consumption and completion time, effectively characterizing the task scheduling paradigm as a challenging bi-objective optimization problem.
2) We introduce an improved algorithm, EHEO, derived from the EO, demonstrating enhanced equilibrium between exploration and exploitation. This modification effectively addresses task scheduling complexities in the fog-cloud environment. To the best of our understanding, this marks the inaugural application of EO in task scheduling within a fog-cloud system.
3) We conduct extensive simulation experiments, revealing that our proposed algorithm surpasses alternative methods by markedly reducing maximum completion time, and energy consumption, enhancing completion rates, and minimizing average waiting times. These results affirm the efficacy of the EHEO algorithm in advancing the optimization of fog-cloud system task scheduling.

The following sections of this paper are organized as follows. Section 2 furnishes a summary of the research related to task scheduling in cloud and fog computing environments. Section 3 constructs the fog-cloud system, mathematically models the AIoT task scheduling problem in fog-cloud systems, and defines the objective functions for task scheduling. In Section 4, we elaborate on the specific details of EHEO. Section 5 compares the performance of EHEO with other algorithms on various task scheduling metrics. Section 6 provides a summary of the study and delves into promising avenues for future research.

2 Related Work

Scholars have delved deeply into the task scheduling challenge within fog-cloud networks, devising numerous methods to contend with it. We choose three main categories of methods for elucidation: Traditional classical algorithms, metaheuristic algorithms, and machine learning techniques. The related work within these categories is systematically summarized and compared in Table 1.

Table 1: Comparison of related work

Works	System	Strategy	latency	Energy	Deadline	Average waiting time
[30]	Fog	Priority-aware semi-greedy	✓	✓	✓	×
[31]	Fog	Greedy-heuristic	✓	✓	✓	×
[32]	Fog	Distributed greedy algorithm	✓	×	×	×
[33]	Fog-cloud	Evolutionary algorithm	✓	×	×	×
[34]	Fog	MPA	✓	✓	×	×
[35]	Fog	AMO	✓	✓	×	✓
[36]	Fog-cloud	NSGA-II	×	✓	✓	×
[37]	Fog-cloud	GWO	✓	✓	×	×
[38]	Fog-cloud	Laxity and ant colony system algorithm	×	✓	✓	×
[39]	Fog	GWO	✓	×	✓	×
[40]	Fog-cloud	Evolutionary algorithm	✓	×	×	×
[41]	Fog-cloud	NSGA-II	✓	×	×	×
[45]	Fog-cloud	GA	×	×	✓	×
[46]	Fog-cloud	Deep learning	✓	×	×	×
[47]	Fog	Deep learning	×	✓	✓	×
[48]	Fog	Deep learning	×	×	✓	×
[49]	Fog	Deep learning	✓	×	×	×
This work	Fog-cloud	EHEO	✓	✓	✓	✓

Classical algorithms are a type of method that has been applied to task scheduling relatively early on. Azizi et al. [30] tackled the optimization of latency and energy consumption in scheduling IoT tasks within heterogeneous fog networks. They introduced two new methods based on semi-greedy strategies. This study emphasized task deadlines and total energy consumption. Misra et al. [31] introduced a task offloading scheme for software-defined networks, focusing on minimizing latency and energy consumption while taking into account network performance. They utilized a greedy heuristic to address this issue. Task deadlines were not considered in this approach. Maray et al. [32] used the Markov decision process to reduce the latency of edge servers but did not consider the system energy consumption.

Metaheuristic algorithms are currently the most widely used methods for tackling task scheduling-related problems. Nguyen et al. [33] presented an evolutionary algorithm-based approach to optimize execution time and costs in fog-cloud environments, but this study did not consider factors like energy consumption and deadlines. Abdel-Basset et al. [34] proposed a refined marine predator algorithm (MPA) to enhance service quality in fog computation task scheduling. However, their primary emphasis was on optimizing energy consumption and completion time within the fog layer. Ghanavati et al. [35] proposed the Ant Mating Optimization (AMO) to reduce completion time and energy consumption in fog computing platforms. Mousavi et al. [36] proposed a directed non-dominated sorting genetic algorithm to decrease server energy consumption and overall response time.

Saif et al. [37] presented a Multi-Objective Grey Wolf Optimization algorithm (MGWO) aiming to reduce latency and energy consumption in fog-cloud systems. However, their study did not consider task deadlines and average waiting time. Xu et al. [38] applied a strategy based on laxity and ant colony systems, aiming to decrease energy consumption while satisfying mixed deadlines. However, they did not consider an important metric, maximum completion time. Dabiri et al. [39] employed both Grey Wolf Optimization (GWO) and Grasshopper Optimization algorithm (GOA) to address task scheduling in the fog-cloud collaborative environment, optimizing violation time and system energy consumption, while also considering the maximum completion time. Hosseinioun et al. [40] applied a hybrid Invasive Weed Optimization and Cultivation Algorithm (IWO-CA) evolutionary approach to reduce the energy consumption of fog nodes. However, they did not take into account completion time and deadlines. Ali et al. [41] proposed a Discrete Non-dominated Sorting Genetic Algorithm II (DNSGA-II) for fog-cloud task scheduling, aiming to balance completion time and cost. Nonetheless, this method did not account for energy consumption and deadlines. Aburukba et al. [45] employed a genetic algorithm to maximize task completion rate within deadlines, but without considering completion time and system energy consumption.

Alelaiwi [46] adopted a deep learning-based approach to reduce system response time. Sellami et al. [47] introduced a deep reinforcement learning (DRL) method for task scheduling in the edge network, aiming to enhance energy efficiency and reduce network latency. Sheng et al. [48] utilized a policy-based DRL algorithm to enhance scheduling success rate and satisfaction. Fan et al. [49] devised an actor-critic reinforcement learning strategy within an iterative learning environment to meet Quality of Service (QoS) constraints while minimizing task latency.

3 System and Problem Formulation

In this part, we initially establish the foundational structure of the fog-cloud system and introduce its components. We outline the fundamental task-scheduling process within this system and subsequently employ a mathematical model to articulate the task scheduling problem specific to the fog-cloud system. The mathematical symbols utilized in this study are presented in Table 2.

Table 2: Main notation definition

Notation	Description
T	Set of AIoT tasks, where $\|T\| = n$
R	Set of computing nodes, where $\|R\| = m$
k	Index of AIoT tasks
l	Index of computing nodes
T_k	AIoT task
S_k^{in}	Input size of T_k
S_k^{ld}	Load size of T_k
D_k	Deadline for T_k
S_k^{out}	Output size of T_k
R_l	Computing node
P_l	Processing rate of R_l
$X_{k,l}$	Decision variable set to 1 if T_k is allocated to R_l
$CT_{k,l}$	Computation time of T_k on R_l

(Continued)

Table 2 (continued)

Notation	Description
RT_k	Response time of T_k
TT_k	Transmission time of T_k
$WT_{k,l}$	Waiting time of T_k on R_l
ET_l	Execution time for R_l to complete all tasks
MK	Makespan of the system
PT	Propagation delay between device and computing node
SR	Success rate of completing tasks by the deadline
E_l	Energy consumption of R_l
E_{tot}	Total system energy consumption
α_l	Energy consumption of R_l while active
β_l	Energy consumption of R_l while idle
r	Transmission rate

3.1 System Architecture

From Fig. 1, it can be observed that the fog-cloud system consists of the following components: The AIoT device layer, the task scheduler, the fog computing layer, and the cloud computing center. The AIoT layer comprises various intelligent terminal devices, which generate a significant number of task requests. The task scheduler is typically deployed on the intelligent gateway, responsible for assigning received tasks to computing nodes for processing. The fog computing layer consists of a multitude of nodes, including routers, gateways, physical servers, and more, each possessing varied computing, storage, and communication capabilities. These nodes are situated at the edge of the network, typically near the intelligent terminal devices, with negligible network latency. The cloud computing center is a collection of virtual machines that have great computing power and large storage capacities. It is equipped with powerful processing capabilities. However, as it is often located far away from the terminal devices, data transmission may suffer from some network latency, which could hinder real-time computational demands. Therefore, the cloud computing center is better suited for managing large-scale tasks or those with lower real-time requirements.

The task scheduler is equipped to gather detailed information about all computing nodes, tasks, network bandwidth, and more. Once the task request originating from the smart terminal device is transmitted to the task scheduler through the gateway, the scheduling algorithm (EHEO) generates the final scheduling outcome based on the acquired resource and task details, in conjunction with the imposed constraints. Consequently, the task is dispatched to the appropriate fog or cloud node for processing. As previously discussed, each task exhibits varying latency tolerance and computational resource requirements, underscoring the pivotal role of the EHEO scheduling algorithm in balancing latency and energy consumption during the task scheduling process. Upon completion of the assigned task by the fog or cloud node, the computation results are relayed back to the end device. In this study, we make the assumption that all tasks are mutually independent, and each node can only process one task at a time, with each task being allocated to a single node.

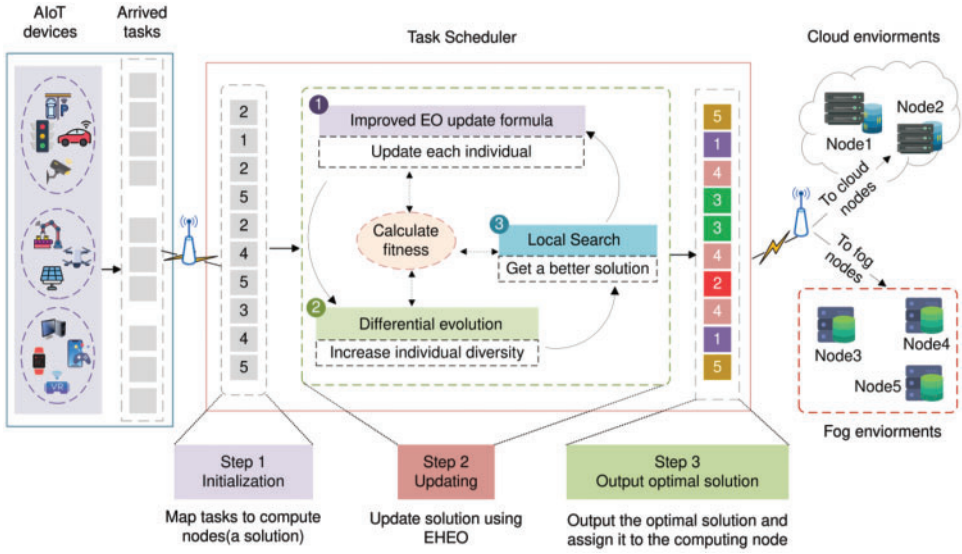


Figure 1: Fog-cloud system architecture

3.2 Problem Formulation

In a fog-cloud system, task scheduling pertains to the allocation of tasks generated by AIoT devices to computing nodes according to certain scheduling policy, which may be allocated to cloud nodes or fog nodes for execution. In this study, our goal is to satisfy the task requests of end devices as much as possible and minimize the makespan of the system and the energy consumption. To simplify the problem, we make the following provisions: All AIoT tasks are mutually independent; each node can only process one task at the same time, and each task can be allocated to just one node.

Consider a fog-cloud system with n tasks $T = \{T_1, T_2, \ldots, T_n\}$, where each task T_k can be represented as a quadruple: $T_k = \langle S_k^{in}, S_k^{out}, S_k^{ld}, D_k \rangle$, S_k^{in} and S_k^{out} represent the input and output data sizes of the task, respectively, measured in kilobytes (KB), influencing the transfer time of the task. S_k^{ld} represents the task workload size in Million instructions per second (MIPS), and D_k represents the task deadline in seconds. The fog-cloud system has m computing nodes $R = \{R_1, R_2, \ldots, R_m\}$, including fog computing nodes and cloud computing nodes, where each node has a processing rate of P_l in MIPS. The association between tasks and nodes can be expressed using an A matrix of size $n \times m$, and the value of decision variable $X_{k,l}$ reflects whether task T_k is assigned to resource R_l.

$$X_{k,l} = \begin{cases} 1, & \text{if } T_k \text{ allocated to } R_l \\ 0, & \text{else} \end{cases}, \forall k \in T, \forall l \in R. \tag{1}$$

3.2.1 Makespan

The start execution time of all computing nodes is the moment 0. The computing nodes process tasks in the order of their sequential arrival. The makespan, defined as the longest execution time among all nodes:

$$MK = \max_{\forall l \in R}(ET_l), \tag{2}$$

where ET_l represents the time required for R_l to complete the allocated tasks, and its computation is carried out as follows:

$$ET_l = \sum_{\forall k \in T} CT_{k,l} \times X_{k,l}, \forall l \in R, \tag{3}$$

where $CT_{k,l}$ can be expressed as:

$$CT_{k,l} = \sum_{\forall l \in R} \frac{S_k^{ld}}{P_l} \times X_{k,l}, \forall k \in T. \tag{4}$$

Fig. 2 shows the execution of assigning 10 tasks to 3 computing nodes. Both nodes 1 and 2 are assigned 3 tasks, and node 3 is assigned 4 tasks. The length of the rectangular box represents the execution time of each task, it is evident that node 1 requires the most time to execute all tasks, so the makespan is 2.32 s.

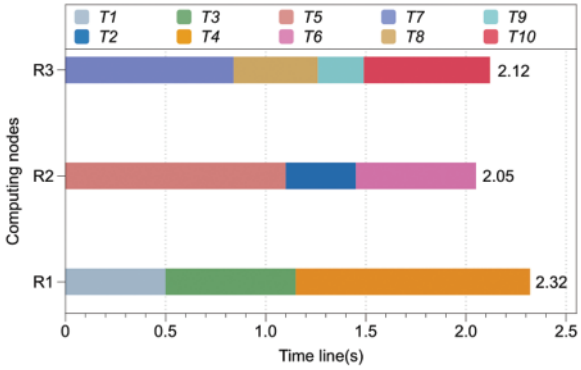


Figure 2: Schematic diagram of makespan

3.2.2 Response Time

For T_k, RT_k is the total time experienced from the time the AIoT device sends out to the time the result of the calculation is returned. It consists of four components: (a) Transmission time TT_k, comprises both the uplink time for the task to reach the node from the end device and the downlink time required for the task result to be returned. (b) Propagation delay PT between AIoT devices and computing nodes. We set the PT between AIoT devices and cloud nodes to 200 ms, while the PT between AIoT devices and fog nodes is negligible. (c) Calculation time $CT_{k,l}$ of T_k on R_l. (d) Waiting time $WT_{k,l}$ before T_k begins execution on R_l.

$$RT_k = TT_k + 2 \times PT + CT_{k,l} + WT_{k,l}. \tag{5}$$

The task's transmission time is determined by the subsequent equation:

$$TT_k = \frac{S_k^{in} + S_k^{out}}{r}, \tag{6}$$

where r is the network transmission rate, we set the transmission rate of T_k to the fog node to be 3 M/s and the transmission rate to the cloud node to be 10 M/s.

All tasks are executed in the order of arrival. The waiting time is the gap between when the task starts executing and when it arrives at the node. The waiting time of T_k on R_l is determined as follows:

$$WT_{k,l} = \sum_{\forall k \in T} \sum_{\forall l \in R} CT_{k,l} \times X_{k,l}. \tag{7}$$

If the aggregate count of tasks fulfilled by the deadline is μ, the success rate is:

$$SR = \frac{\mu}{n} \times 100\%. \tag{8}$$

3.2.3 Energy Consumption

In this study, we exclusively focus on the energy consumption of fog and cloud nodes, while the energy consumption of network transmission will be disregarded. Moreover, the energy usage of a node in the idle state is 60% [34] of its active state consumption. The idle time of a node is equivalent to maskspan minus the node's active time. The expression for calculating the energy consumption of R_l is as follows:

$$E_l = (ET_l \times \alpha_l + (MK - ET_l) \times \beta_l) \times P_l, \tag{9}$$

$$\alpha_l = 10^{-8} \times (P_l)^2, \tag{10}$$

$$\beta_l = 0.6\alpha_l. \tag{11}$$

Here, α_l represents the energy consumption of R_l in the active state, and β_l represents the energy consumption of R_l in the idle state. The overall energy consumption of all nodes is determined as follows:

$$E_{tot} = \sum_{l=1}^{m} E_l. \tag{12}$$

3.2.4 Fitness Function

In the realm of task scheduling, optimization algorithms employ a fitness function to assess the excellence of solutions and make decisions guided by the corresponding fitness values. The aim is to identify the best solution for the distribution of tasks. In this research, the primary objective is to achieve a simultaneous reduction in both the makespan and energy consumption. Therefore, we utilize a fitness function that integrates both makespan and energy consumption metrics to assess the quality of the obtained results. We employ a linear weighted combination approach to formulate the objective function:

$$Minimize : \omega \times MK + (1 - \omega) \times E_{tot},$$

Subject to

$$\sum_{k=1}^{n} X_{k,l} = 1, \forall k \in R,$$

$$RT_k \leq D_k, \forall k \in T, \forall l \in R, \tag{13}$$

where ω is the weighting factor of the objective function, when $\omega > 0.5$, it indicates that the emphasis should be placed on optimizing the MK; when $\omega = 0.5$, it signifies equal priority for both; and when $\omega < 0.5$, it indicates that the focus should be on minimizing the total energy consumption. In this study, we set $\omega = 0.5$, signifying that MK and total energy consumption are equally important in this problem. Constraint conditions ensure that a task can only be assigned to one computing node, and

each task receives a response before its deadline. Soft deadlines are employed in this study, meaning that if the task response time exceeds the deadline, the task will continue to be executed.

4 Task Scheduling Algorithm

Compared to other metaheuristic algorithms, EO demonstrates high effectiveness in solving optimization problems. However, in specific cases, EO may still encounter challenges such as falling into local optima and low accuracy in specific cases. To address these limitations, we have made improvements to the EO, resulting in the EHEO. EHEO builds upon the EO and integrates scheduling strategies from Differential Evolution (DE) and local search algorithm. In this section, we present a comprehensive overview of the EHEO.

4.1 Equilibrium Optimizer

The Equilibrium Optimizer is an innovative metaheuristic algorithm inspired by controlled volume dynamic mass balancing. The algorithm's foundation lies in the mass balance equation, which encapsulates the physical phenomena of mass inflow, outflow, and generation within a control volume. Each concentration of particles corresponds to a solution to the optimization problem, and these concentrations are defined by the mass balance equation from physics. EO aims to find an equilibrium state, where particle concentrations represent optimal solutions. The optimization process of EO can be categorized into three distinct stages.

4.1.1 Initialization

Similar to other metaheuristic optimization algorithms, EO generates uniformly randomly distributed particles in the solution space. The population is initialized using the following equation:

$$C_i = Lb + rand_i \left(Ub - Lb\right), i = 1, 2, \ldots, N, \tag{14}$$

where C_i is the initial concentration vector of each particle, Ub and Lb denote the upper and lower bounds of the solution-seeking space, respectively, $rand_i$ denotes a randomly generated vector within the range $[0, 1]$, and N represents the number of particles in the population. After initializing the population, each particle is sorted by fitness in preparation for the construction of equilibrium candidates.

4.1.2 Constructing Equilibrium Pool

EO tries to find the equilibrium state of the system and approaches the global optimal solution of the problem when the equilibrium state is reached. EO starts the optimization without information about the equilibrium state but provides a search pattern for the particles based on the candidate solutions in the equilibrium state pool. The candidate solution is composed of five particles consisting of the current four best-adapted particles and their average values:

$$C_{eq,pool} = \left[C_{eq1}, C_{eq2}, C_{eq3}, C_{eq4}, C_{ave}\right]. \tag{15}$$

EO updates the concentration of particles by randomly selecting a candidate solution from $C_{eq,pool}$ with the same probability in each iteration. This strategy helps mitigate the risk of EO falling into a local optimum to some extent.

4.1.3 Updating the Concentration

The role of the polynomial term F is designed to harmonize the algorithm's exploration and exploitation capabilities, thereby enhancing its overall performance. This can be formulated as follows:

$$F = e^{-\lambda(t-t_0)}. \tag{16}$$

The vector λ is a randomly generated vector in [0, 1], and t is a function of the iteration number *Iter*, with a decreasing trend as the iteration number increases:

$$t = \left(1 - \frac{Iter}{Max\_iter}\right)^{\left(a_2 - \frac{Iter}{Max\_iter}\right)}, \tag{17}$$

where *Iter* and *Max_iter* denote the current and maximum number of iterations, respectively, and the constant term a_2 is used to regulate the exploitation capability. To improve the exploration and exploitation capability of the EO, the t_0 term is added:

$$t_0 = \frac{1}{\lambda} \ln\left(-a_1 \text{sign}(r - 0.5)\left[1 - e^{-\lambda t}\right]\right) + t, \tag{18}$$

where a_1 is a constant term affecting the exploration ability. A larger value of a_1 leads to enhanced exploration and reduced exploitation; while a higher a_2 strengthens exploitation but diminishes exploration. To further improve the exploitation capability, another factor, the generation rate G, is introduced and is expressed as follows:

$$G = G_0 F, \tag{19}$$

where G_0 is the initial generation rate, which is formulated as follows:

$$G_0 = GCP\left(C_{eq} - \lambda C\right), \tag{20}$$

$$GCP = \begin{cases} 0.5r_1 & r_2 \geq GP \\ 0 & r_2 < GP \end{cases}, \tag{21}$$

where C is the current particle concentration, r_1 and r_2 denote two randomly chosen numbers from the interval [0, 1], *GCP* is the generation rate control parameter which determines the number of particles whose concentration is updated using the *GCP*, and *GP* is the generation probability. Ultimately, the update equation of EO is shown as follows:

$$C = C_{eq} + (C - C_{eq})F + \frac{G}{\lambda V}(1 - F). \tag{22}$$

4.2 Improved Update Formula

The exponential factor F is an important coefficient that affects the update of particle concentration. It can be seen from the equation that F be 0, and when F is 0, G is also 0. At this point, the particle concentration will no longer be updated. To avoid the situation where the EO algorithm prematurely encounters update stagnation, we have made the following improvements to the update formula in this article:

$$C = \begin{cases} C_{eq} + \left(C - C_{eq}\right)F + \frac{G}{\lambda V}\left(1 - F\right), r_2 \geq GP \\ C_{eq} + \left(C - C_{rand}\right)r_3 + \left(C_{worst} - C\right)r_4, else \end{cases}, \tag{23}$$

where C_{rand} is a particle arbitrarily chosen from the population other than C_{eq1}, C_{eq2}, C_{eq3}, C_{eq4} and C_{ave}, C_{worst} is the one with the worst adaptation, and r_3, r_4 are random numbers between [0, 1]. The new update formula no longer relies only on the particles in the equilibrium pool for updating, but also considers random particles and particles with the worst fitness, which helps prevent the algorithm from being trapped in local optima to some extent and speeds up the global convergence.

4.3 Differential Evolution (DE)

The individuals in the EO update their positions based on the candidate solutions in the equilibrium pool, and the candidate solutions in the equilibrium pool play an absolute guiding role in the position update of the whole population, which easily leads the EO to get stuck in a suboptimal state. To prevent the reduction in diversity when the population iterates to a specific region, we employ the variation, crossover, and selection operations from the DE [50]. These operations help maintain population diversity and prevent the EO from prematurely converging to a local optimum, thereby enhancing its ability to find the optimal solution.

4.3.1 Mutation Operator

The DE algorithm has several mutation methods, and in this paper, we use the DE/rand/1 variant. DE generates mutation vectors V_i for each vector X_i in the following manner:

$$V_i^t = X_{r1}^t + F\left(X_{r2}^t - X_{r3}^t\right), \tag{24}$$

where X_{r1}^t, X_{r2}^t and X_{r3}^t are three individuals chosen randomly from the current population, F is a scaling factor between [0, 1], and $r1 \neq r2 \neq r3$.

4.3.2 Crossover Operator

The crossover operation generates trial vectors $U_{i,j}^t$ from the target $X_{i,j}^t$ and variant vectors $V_{i,j}^t$, aiming to enhance population diversity:

$$U_{i,j}^{t+1} = \begin{cases} V_{i,j}^t, & if\ rand\,(0,1) \leq CR\ or\ j = randn\,(i) \\ X_{i,j}^t, & else \end{cases} \quad j = 1, 2, \ldots n, \tag{25}$$

where CR denotes the crossover probability, $rand\,(0, 1)$ denotes a random number between [0, 1], and $randn(i)$ is a random integer between $[1, 2, \cdots, N]$, which serves to guarantee that at least one dimension of the information comes from the mutant individual.

4.3.3 Selection Operator

The greedy strategy was used to compare the parent with its offspring to select new individuals:

$$X_i^{t+1} = \begin{cases} U_i^{t+1}, & if\ f\left(U_i^{t+1}\right) < f\left(X_i^t\right) \\ X_i^t, & else \end{cases}, \tag{26}$$

where f is the fitness function, and the better individual is selected as the individual of generation $t+1$ by comparing the fitness values of U_i^{t+1} and X_i^t.

4.4 Local Search Algorithm

In pursuit of an enhanced solution, we have refined the local search algorithm (LSA) introduced in literature [51], and the pseudocode of LSA is shown in Algorithm 1. It can better balance resource

load and reduce makespan. During each iteration, the LSA is applied to the optimal solution. It starts by calculating the average execution time Et_{ave} across all nodes, divides the nodes into two groups according to the size relationship with the Et_{ave}, the node list smaller than the Et_{ave} is R_d, and the node list larger than the Et_{ave} is R_u. Then, 20% of the nodes are randomly selected from R_d and R_u respectively to form R_d^{sub} and R_u^{sub} (if the number of nodes in R_d^{sub} and R_u^{sub} is different, the smaller one is used as the final number of adjustable nodes). Then, a task is randomly selected from the task list corresponding to each node in R_u^{sub} to form the *adjust_list*, which will be assigned to the nodes in R_d^{sub}. Afterwards, the *adjust_list* is sorted in ascending order of task length, and the nodes in R_d^{sub} are sorted in ascending order of processing capability. Finally, the sorted tasks are assigned to the sorted nodes one by one to form a new task allocation scheme, which is the new solution. If the fitness value of the LSA solution is lower than that of the current optimal solution, the LSA solution replaces the current optimal solution. Otherwise, the solution remains unchanged. The principle of the LSA is shown in Fig. 3.

Algorithm 1: Local search algorithm

Input: The best solution obtained in each iteration *best_solution*.
Output: *new_best_solution*.

```
new_bset_solution = best_solution;
Calculate the execution time of each node using Eq. (3);
Calculate Et_ave;
Construct R_d and R_u based on the size relationship with Et_ave;
Randomly select nodes from R_d and R_u by 20% each to form R_d^sub and R_u^sub;
Randomly select a task from the task list corresponding to each node in R_u^sub to
    construct adjust_list;
Sort adjust_list according to their task length from smallest to largest;
Sort the nodes in R_d^sub in ascending order according to their processing capacity;
Assign the sorted tasks to the nodes in order;
if (f(new_bset_solution) < f(bset_solution));
    best_solution = new_bset_solution.
end if
```

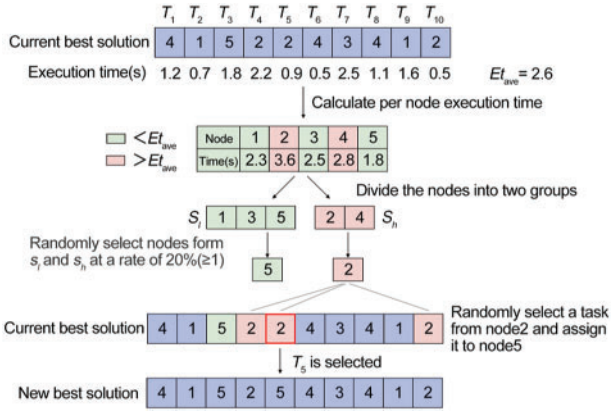


Figure 3: Schematic diagram of local search algorithm

4.5 The Proposed Task Scheduler

To enhance the task scheduling performance of EO in fog-cloud networks, we integrate various strategies as mentioned above. EO serves as the main body of the scheduling algorithm to generate initial solutions, and the modified update formula to some extent avoids algorithm stagnation. Subsequently, the differential evolution algorithm is applied to preserve population diversity. Finally, local search algorithm is employed on the generated optimal solution to further refine its quality. In EHEO, each particle represents a scheduling scheme, and the fitness function is used to assess the quality of each individual. The solution is constantly updated before the algorithm terminates, and the ultimate solution is the optimal task scheduling scheme given by the algorithm. The EHEO task scheduling process is shown in Fig. 4.

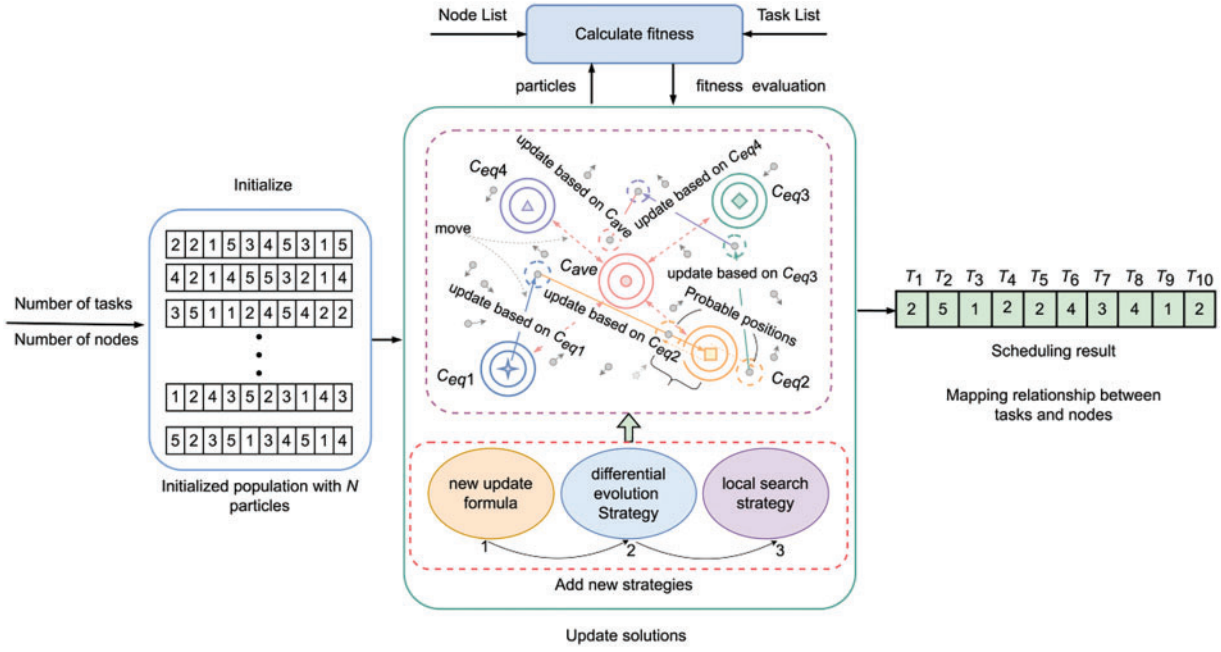


Figure 4: Task scheduling with EHEO

4.5.1 Initial Phase

As task scheduling is discrete while EO algorithm is designed for continuous optimization problems, real values need to be converted into virtual machine numbers during initialization. The following formula is used to initialize the population:

$$C_{ij} = floor\left(Lb_{ij} + rand\left(Ub_{ij} - Lb_{ij}\right)\right), j = 1, 2, \ldots, n, \tag{27}$$

where *rand* is a random number between [0, 1], *Lb* is set to 1 as the lower bound, and *Ub* is the upper bound with a value of *m* denoting computing nodes. The dimension of the particles is determined by the total tasks, denoted as *n*, and the value of each dimension is the number of the computing node. The *floor* function is utilized to discretize the actual values.

For instance, consider a scenario with 5 tasks and 3 computing nodes, and let us say we have a solution $C_i = [2.1, 1.6, 2.4, 3.5, 1.2]$, after applying the *floor* function, it becomes a discrete solution

$C_i = [2, 1, 2, 3, 1]$, which means the first and third tasks are allocated to node 2, the second and fifth tasks are assigned to node 1, and the fourth task is designated to node 3.

4.5.2 Update Stage

At this stage, the fitness value of each particle is obtained through Eq. (13). An equilibrium pool is formed by selecting the best 4 particles based on fitness values and including particles obtained from the average of these top 4 particles. Eq. (23) is used to update each particle. The updated formula is not exclusively dependent on particles within the equilibrium pool; instead, it incorporates a probability of updating the current particle based on information from other particles, thus solving the problem of EO algorithm getting stuck. After updating all individuals, DE is utilized to enhance the variety within the population and prevent getting trapped in local optima. Finally, the LSA is used to further enhance the optimal solution. Upon reaching the termination condition for iterations, the algorithm will return the optimal individual, representing the best solution for task scheduling. The pseudocode of the EHEO is presented hereafter.

Algorithm 2: EHEO task scheduler

Input: Population size N, maximum iterations Max_iter, number of tasks n, number of nodes m
Output: $T \rightarrow R$.

Initialize the populations of particles $C_i (i = 1, 2, \ldots, N)$;
Assign a high fitness value to each candidate in the equilibrium pool;
Set parameters $a_1 = 2$, $a_2 = 1$, $GP = 0.5$;
while $(Iter < Max\_iter)$
  for each i particle **do**
    Calculate the fitness value of particle $i, f(C_i)$;
    if $(f(C_i) < f(C_{eq1}))$
      update C_{eq1} with C_i and $f(C_{eq1})$ with $f(C_i)$;
    else if $(f(C_i) > f(C_{eq1}) \,\&\, f(C_i) < f(C_{eq2}))$
      update C_{eq2} with C_i and $f(C_{eq2})$ with $f(C_i)$;
    else if $(f(C_i) > f(C_{eq1}) \,\&\, f(C_i) > f(C_{eq2}) \,\&\, f(C_i) < f(C_{eq3}))$
      update C_{eq3} with C_i and $f(C_{eq3})$ with $f(C_i)$;
    else if $(f(C_i) > f(C_{eq1}) \,\&\, f(C_i) > f(C_{eq2}) \,\&\, f(C_i) > f(C_{eq3}) \,\&\, f(C_i) < f(C_{eq4}))$
      update C_{eq4} with C_i and $f(C_{eq4})$ with $f(C_i)$;
    end if
  end for
  Calculate $C_{ave} = (C_{eq1} + C_{eq2} + C_{eq3} + C_{eq4})/4$;
  Construct Equilibrium pool $C_{ceq,pool} = [C_{eq1}, C_{eq2}, C_{eq3}, C_{eq4}, C_{ave}]$;
  Accomplish the memory saving $(Iter > 1)$;
  update t using Eq. (4);
  for each i particle **do**
    Update positions using Eq. (23);
  end for
  Apply DE;
  Apply LSA;
  $Iter = Iter + 1$.
End while

4.5.3 Algorithm Complexity Analysis

Analyzing the performance of an algorithm often involves assessing its complexity. The time complexity of EHEO is influenced by the algorithm's structure as a whole, including EO, DE, LSA, *Max_iter*, the population size N and the dimensionality of the task scheduling problem d. The time complexity of EO is $O(N \times d \times Max\_iter)$, DE is $O((N \times d + N \times d) \times Max\_iter)$, and LSA is $O(d \times Max\_iter)$, the time complexity of EHEO is:

$$O(EHEO) = O(EO) + O(DE) + O(LSA). \tag{28}$$

According to the feature that the time complexity needs to be expressed by only one term of maximum magnitude, the EHEO algorithm time complexity can be simplified to $O(N \times d \times Max\_iter)$, therefore, the complexity of our proposed EHEO is the same as that of the original EO.

5 Numerical Results

We use simulation experiments to evaluate the performance of EHEO on the problem of scheduling tasks containing dual goals. Two sets of experiments are conducted to assess and analyze the performance of EHEO with the original EO, PSO [52], and Whale Optimization Algorithm (WOA) [53] on different metrics. These metrics include: 1) Makespan, a crucial measure of system latency and operational efficiency, often prioritized in optimization studies. 2) Energy consumption, a critical consideration impacting device battery life and costs. 3) Success rate, indicating the efficiency of resource management and system stability. 4) Average waiting time, reflecting the quality of user service. Additionally, we explore the convergence performance of each algorithm, providing insights into their efficiency. The language used for the simulation experiments is Python, and they are performed on a computer equipped with 2 GHz Intel Core i5, 8 GB RAM, and a macOS system.

5.1 Dataset Description and Parameters Setting

To thoroughly assess the efficacy of the EHEO, we have established two datasets for experimental purposes. The first set includes different numbers of tasks, namely 100, 200, 300, 400, 500, 600, 700, 800, 900, and 1000, with 60 fog nodes and 2 cloud computing nodes. The second set maintains a constant task count of 500, with the number of fog nodes varying from 30, 40, 50, 60, 70, 80, 90, to 100, and 2 cloud computing nodes. All tasks are heterogeneous, with attribute values generated based on established practices [30,40,54]. Specifically, task loads range from 500 to 2000 MIPS, input file sizes vary between 1 and 100 KB, output file sizes also fall within the 1 to 100 KB range, and task deadlines span from 2000 to 4500 ms.

To simplify the study, we standardize the network transmission rate from the end device to the fog node at 3M/s and the transmission rate to the cloud node at 10 M/s [54]. The processing rate is randomly generated between [4000–8000] MIPS for cloud nodes and [500–4000] MIPS for fog nodes [30]. To ensure the reliability of the experimental results, each selected algorithm is independently run for 20 rounds. In each round, the maximum number of iterations is set to 100, and the population size is set to 20. These parameter configurations are determined through extensive experimentation and reference to existing research [8,21,53], and all experimental results are averaged.

The specific parameters of the simulation experiment are presented in Table 3. The parameter values of each algorithm are selected from the suggested values of the original paper, and the detailed parameter settings can be found in Table 4.

Table 3: Simulation experiment parameters

Parameter	Value
Number of fog nodes	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
Number of cloud nodes	2
Number of AIoT tasks	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
The processing rate of fog nodes	[500–4000] MIPS
The processing rate of cloud nodes	[4000–8000] MIPS
Task length	[500–2000] MIPS
Input file size of the AIoT task	[1–1000] KB
Output file size of the AIoT task	[1–100] KB
Task deadline	[2000–4500] ms
Number of runs	20
Maximum iterations	100
Population size	20

Table 4: Algorithm parameters

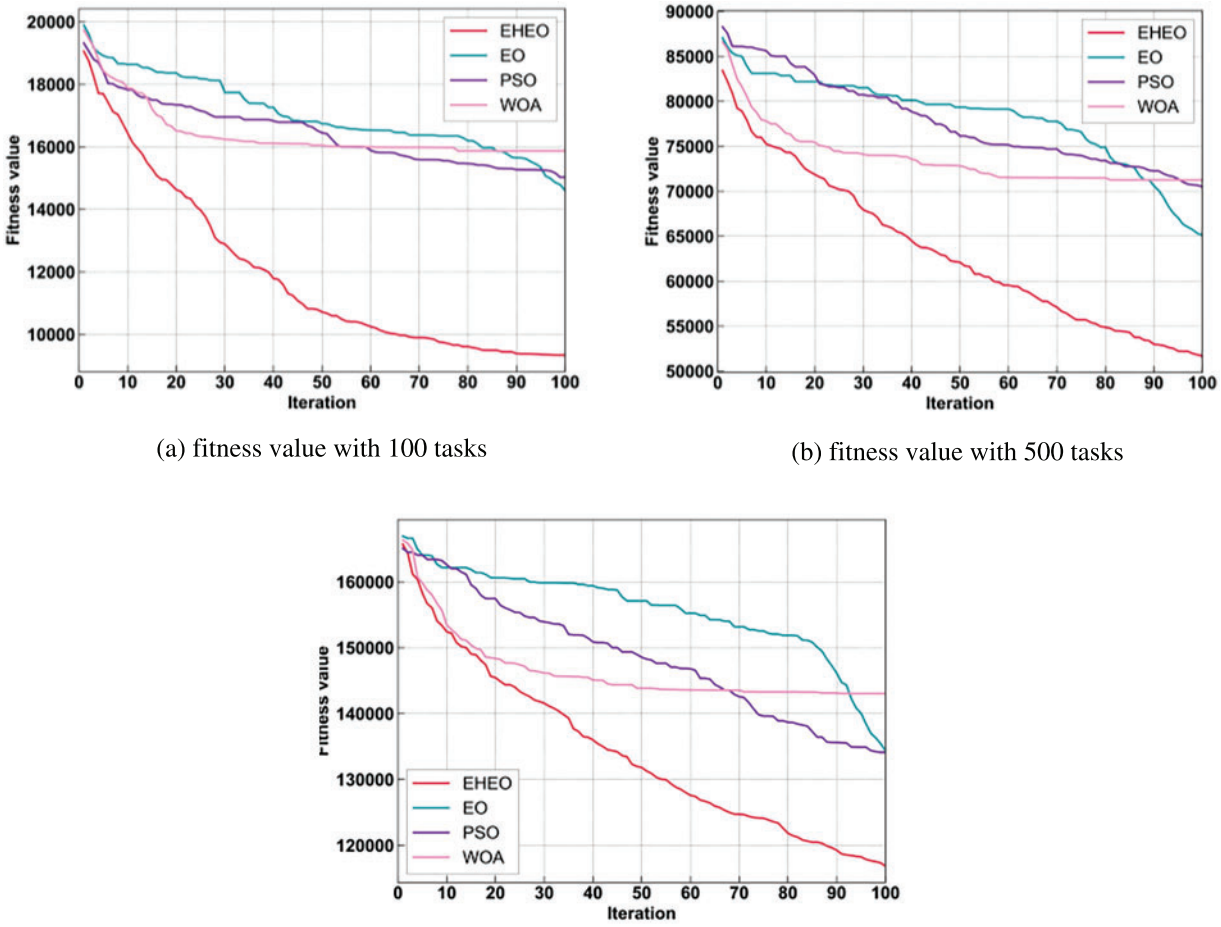
Algorithm	Parameter	value
EHEO	a_1	2
	a_2	1
	GP	0.5
	F	0.8
	CR	0.9
EO	a_1	2
	a_2	1
	GP	0.5
PSO	C_1	2
	C_2	2
	w	0.9 to 0.4
WOA	b	1
	a	2 to 0
	l	[−1, 1]

5.2 Results Analysis

5.2.1 Comparison with Varying Numbers of Tasks

In this scenario, we discuss the performance of EHEO, EO, PSO, and WOA under varying task quantities.

Fig. 5 shows the best fitness convergence curves for EHEO, EO, PSO, and WOA. Here we compare the variation in fitness values for task counts of 100, 500, and 1000. It is obvious that EHEO has the smallest fitness value at different task numbers and the fastest convergence speed, while WOA converges more rapidly in the early phase but tends to get stuck in a local optimum in the subsequent phase, and EO and PSO converge more slowly all the time, with some acceleration only in the late iteration. This is mainly because the new update formula can accelerate the convergence speed of EHEO. DE contributes to increasing population diversity during the iteration process, thereby reducing the likelihood of EHEO falling into a local optimum. Additionally, LSA effectively balances the resource load situation, further enhancing the quality of the optimal solution and accelerating the convergence of the algorithm. This underscores the effectiveness of our enhancements to the EO.



(a) fitness value with 100 tasks

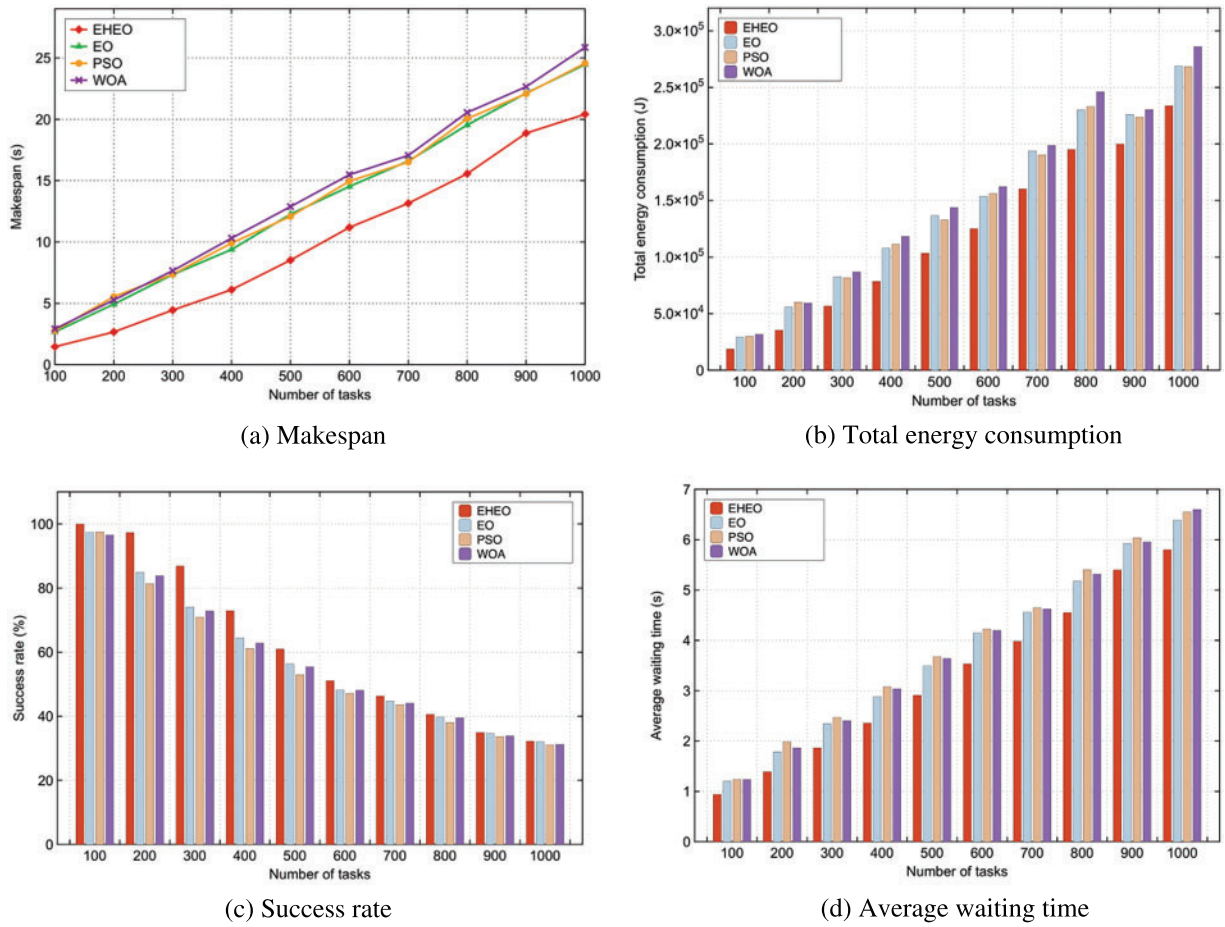
(b) fitness value with 500 tasks

(c) fitness value with 1000 tasks

Figure 5: The convergence curves with different numbers of tasks

Fig. 6a illustrates a comparison of the makespan among EHEO, EO, PSO, and WOA. A smaller makespan indicates better performance, which means that the system can complete all tasks in a shorter time. The figure unmistakably illustrates that with an increase in the number of tasks, the makespan also rises, indicating an increased load on each node. In the experiment, EHEO always maintains the smallest makespan and outperforms EO, PSO, and WOA. The performance of EO and

PSO is similar, and WOA performs the worst. Additionally, the makespan of all algorithms generally increases linearly, indicating that EHEO and other algorithms have good stability. EHEO can achieve a shorter makespan and good stability mainly because it combines the advantages of the differential evolution algorithm with the original EO. This combination accelerates the convergence rate, and the integrated local search algorithm enhances the quality of solutions, yielding a smaller makespan compared to other algorithms. Furthermore, we can see that the original EO algorithm performs similarly to the PSO and WOA algorithms in terms of makespan, without any significant advantages. However, our proposed EHEO algorithm can significantly reduce makespan, which indicates that our improvement to the original EO algorithm is effective.



(a) Makespan

(b) Total energy consumption

(c) Success rate

(d) Average waiting time

Figure 6: Simulation results with different numbers of tasks

Fig. 6b illustrates the energy consumption of various algorithms with an increasing number of tasks. The overall energy consumption of the system increases in parallel with the growing number of tasks. This escalation is attributed to the heightened total processing time for tasks, resulting in elevated energy consumption. According to the experimental results, EHEO consistently maintains the smallest energy consumption. With a low number of tasks, the disparities in total completion time are negligible, resulting in comparable total energy consumption between EHEO and other algorithms. However, as the task quantity grows, the disparity in total completion time becomes more pronounced,

and the EHEO exhibits markedly lower energy consumption compared to the other algorithms. This underscores the enhanced energy efficiency of the EHEO algorithm in multi-task scenarios.

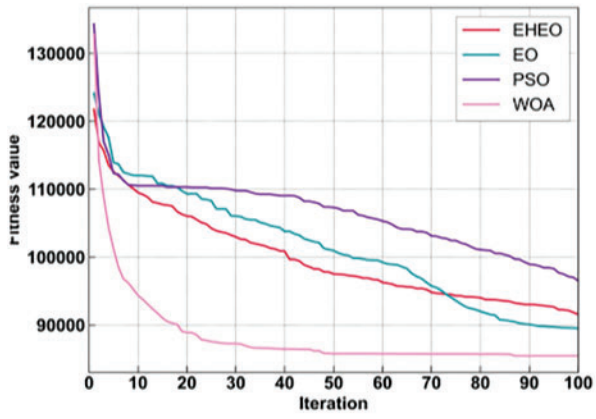
The proportion of tasks completed before the deadline is also an important metric to consider, as shown in Fig. 6c, which displays the performance of the EHEO, EO, PSO, and WOA in meeting task deadlines. According to the experimental results, EHEO consistently achieves the highest proportion of tasks completed within the deadline, indicating that it can more effectively utilize resources by prioritizing shorter deadline tasks to appropriate computing nodes. However, as the task quantity grows, the gap between EHEO and other algorithms in completing tasks within the deadline gradually decreases. This is primarily attributed to the fixed number of computational nodes, which imposes a limitation on the number of tasks achievable within the deadline. The number of tasks increases much faster than the number that can be completed within the deadline, so this phenomenon is consistent with normal trends.

Fig. 6d compares the average waiting time of these methods. According to the experimental results, EHEO consistently exhibits the shortest average waiting time, showcasing superior performance in this regard for tasks of varying sizes. Specifically, EHEO exhibits significantly lower average waiting times compared to other algorithms when the number of tasks is large. This suggests that EHEO performs better in managing large-scale task scheduling. This is mainly because when the search dimension is small, the search results of all algorithms are relatively close. When the search dimension increases, EHEO can quickly find the optimal solution in the shortest possible time.

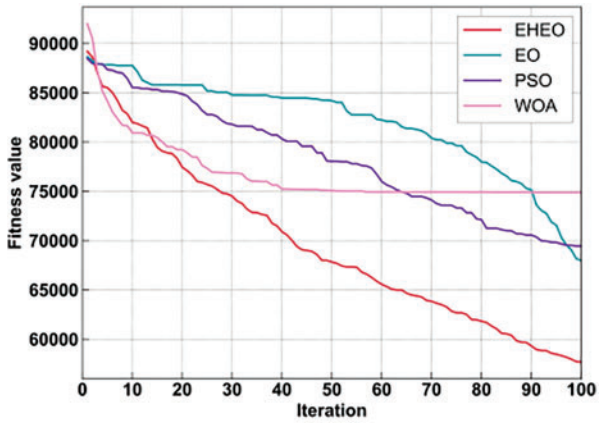
5.2.2 Comparison with Varying Numbers of Fog Nodes

In this section, we will delve into the performance analysis of the EHEO under the variation of the number of fog nodes.

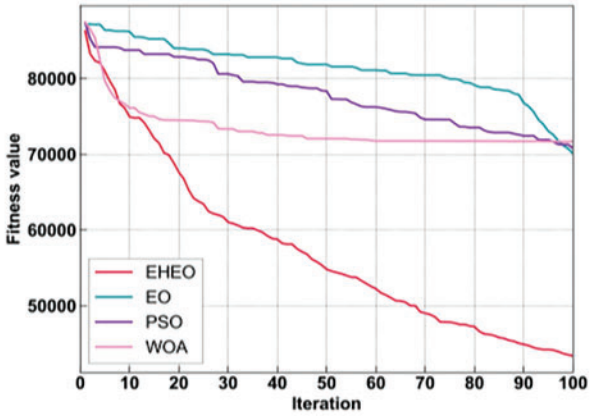
We compare the convergence of fitness among different algorithms for fog node numbers of 10, 50, and 100. Fig. 7 indicates that WOA exhibits the most favorable convergence performance at 10 fog nodes. EHEO shows convergence performance similar to EO, while PSO performs the least effectively. As the quantity of fog nodes grows, EHEO exhibits the fastest convergence and achieves the smallest fitness value. EO maintains a slower convergence speed, and while the convergence accelerates later, the final result is inferior to EHEO. PSO and WOA, on the other hand, lag in terms of performance. When the quantity of fog nodes is limited, EHEO does not exhibit as robust convergence as WOA, possibly due to WOA being more effective in smaller search spaces. Nonetheless, as the search space expands, WOA converges swiftly in the initial phase but plateaus in the later stages, indicative of its intrinsic balance between exploration and exploitation. Examining the convergence curves reveals that the conventional EO converges slowly and is susceptible to local optima, highlighting the algorithm's limitations. In contrast, the EHEO effectively addresses these issues and improves overall performance.



(a) fitness value with 10 nodes



(b) fitness value with 50 nodes



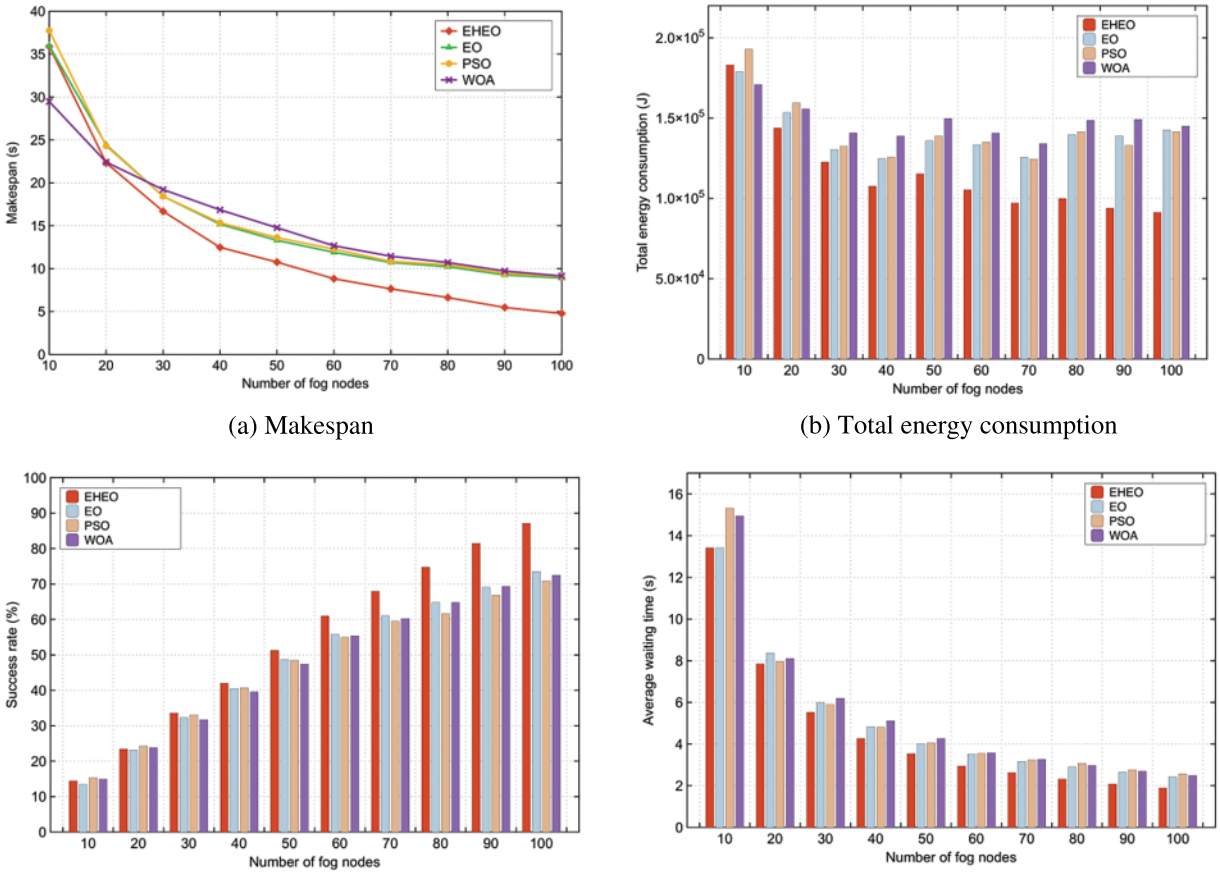
(c) fitness value with 100 nodes

Figure 7: The convergence curves with different numbers of fog nodes

In Fig. 8a, the makespan variation of EHEO, EO, PSO, and WOA is depicted as the number of fog nodes increases. It can be observed that the makespan of all algorithms shows a decreasing trend. When there are 10 nodes, WOA exhibits the most favorable performance, and the makespan of EHEO is close to the values of EO and PSO. However, when the number of nodes increases, EHEO experiences a significant reduction in makespan, and it always has the smallest makespan among these algorithms. Even with over 70 fog nodes, EHEO consistently achieves a substantial reduction in makespan, demonstrating its robust performance. In contrast, EO, PSO, and WOA show only marginal decreases in makespan, emphasizing the efficiency of EHEO in optimizing task scheduling. This can be attributed to the improvement strategy employed by EHEO, which improves the balance of exploration and exploitation. This strategy allows EHEO to efficiently search the global space, speeding up convergence and shortening the optimization search process.

Fig. 8b illustrates the overall energy consumption of the system employing various algorithms. Despite occasional minor fluctuations in the energy consumption of EHEO with an increasing number of fog nodes, the overall trend is a decrease. Additionally, as the quantity of fog nodes rises, the energy consumption of EHEO shows a considerable reduction compared to EO, PSO, and WOA. And the energy consumption of EO, PSO and WOA basically does not decrease after the fog node increases to

30, and even slightly increases the trend. The experimental results demonstrate that EHEO possesses superior load balancing capabilities, considering resource load conditions and effectively balancing the dual constraints of makespan and energy consumption. In comparison to EO, PSO, and WOA, EHEO consistently maintains the lowest makespan and energy consumption in most scenarios. On the flip side, EO, PSO, and WOA encounter challenges in effectively balancing system energy consumption while reducing makespan, and their resource allocation is not sufficiently balanced.



(a) Makespan

(b) Total energy consumption

(c) Success rate

(d) Average waiting time

Figure 8: Simulation results with different numbers of fog nodes

Fig. 8c shows the success rate of tasks before their deadlines as the number of fog nodes varies. It is evident that when the number of fog nodes is limited (10–30 nodes), EHEO performs similarly to EO, PSO, and WOA. As the fog node count increases further, the success rate of EHEO significantly improves and surpasses that of EO, PSO, and WOA. When there are 100 fog nodes, the success rate of EHEO exceeds 85%, which is about 15 percentage points higher than EO.

Fig. 8d displays the average waiting time of the four algorithms in the context of task scheduling. Similar to the trend of makespan, as the fog node count increases, EHEO consistently outperforms the other algorithms, although the average waiting time for all algorithms is gradually decreasing. This indicates that EHEO can always provide better scheduling solutions, thereby utilizing computing resources more effectively and completing tasks in the shortest possible time.

6 Concluding Remarks

In this paper, we introduce an enhanced hybrid Equilibrium Optimizer algorithm, denoted as EHEO, tailored to tackle task scheduling challenges within the fog-cloud environment of the AIoT. The primary aim of EHEO is to concurrently minimize the makespan and energy consumption while adhering to task deadlines. Through the incorporation of differential evolution and local search algorithms, coupled with modifications to the EO update formula, EHEO is engineered to enhance its overall performance. To substantiate the efficacy of our proposed approach, we conduct extensive experiments. The results reveal that our algorithm outperforms other methods across all metrics. It not only meets the real-time requirements of AIoT tasks but also reduces the operational costs of the fog-cloud system. This will provide effective solutions and approaches for various AIoT applications in the fog-cloud network, such as smart transportation and smart healthcare.

While our current research has provided a promising solution for AIoT task scheduling within fog-cloud systems, there are still significant areas for further exploration. Specifically, we have yet to account for the mobility of end devices and fog nodes, a factor we plan to address in subsequent phases of our research program. In addition, we envisage integrating existing algorithms with machine learning techniques to achieve more nuanced task partitioning and arrangement within the AIoT. Furthermore, we plan to consider task dependencies and dynamic characteristics to adapt our approach to a broader spectrum of AIoT applications, thereby further augmenting the efficiency of task scheduling.

Acknowledgement: We extend our sincere gratitude to all members for their steadfast support of our work. Furthermore, our thanks go to the editors and reviewers whose diligent efforts have greatly contributed to the improvement of this manuscript.

Funding Statement: This work was supported in part by the Hubei Natural Science and Research Project under Grant 2020418, in part by the 2021 Light of Taihu Science and Technology Project, and in part by the 2022 Wuxi Science and Technology Innovation and Entrepreneurship Program.

Author Contributions: Study conception and design: M. Rao and H. Qin; data collection: M. Rao; analysis and interpretation of results: M. Rao and H. Qin; draft manuscript preparation: M. Rao and H. Qin. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are available from the corresponding author, H. Qin, upon reasonable request.

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

References

[1] A. Ghosh, D. Chakraborty, and A. Law, "Artificial intelligence in Internet of things," *CAAI Trans. Intell. Technol.*, vol. 3, no. 4, pp. 208–218, 2018. doi: 10.1049/trit.2018.1008.

[2] L. Jia, Z. Zhou, F. Xu, and H. Jin, "Cost-efficient continuous edge learning for artificial intelligence of things," *IEEE Internet Things J.*, vol. 9, no. 10, pp. 7325–7337, May 2022. doi: 10.1109/JIOT.2021.3104089.

[3] X. Yang, Y. Xu, L. Kuang, Z. Wang, H. Gao and X. Wang, "An information fusion approach to intelligent traffic signal control using the joint methods of multiagent reinforcement learning and artificial intelligence of things," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 7, pp. 9335–9345, Jul. 2022. doi: 10.1109/TITS.2021.3105426.

[4] B. Dong, Q. Shi, Y. Yang, F. Wen, Z. Zhang and C. Lee, "Technology evolution from self-powered sensors to AIoT enabled smart homes," *Nano Energy*, vol. 79, pp. 105414, Jan. 2021. doi: 10.1016/j.nanoen.2020.105414.

[5] B. L. Risteska Stojkoska, and K. V. Trivodaliev, "A review of internet of things for smart home: Challenges and solutions," *J. Clean. Prod.*, vol. 140, pp. 1454–1464, Jan. 2017. doi: 10.1016/j.jclepro.2016.10.006.

[6] S. Singh, P. K. Sharma, B. Yoon, M. Shojafar, G. H. Cho, and I. H. Ra, "Convergence of blockchain and artificial intelligence in IoT network for the sustainable smart city," *Sustain. Cities Soc.*, vol. 63, no. 10, pp. 102364, Dec. 2020. doi: 10.1016/j.scs.2020.102364.

[7] Z. Chen, C. B. Sivaparthipan, and B. Muthu, "IoT based smart and intelligent smart city energy optimization," *Sustain. Energy Technol. Assess.*, vol. 49, no. 12, pp. 101724, Feb. 2022. doi: 10.1016/j.seta.2021.101724.

[8] S. Ghanavati, J. Abawajy, and D. Izadi, "Automata-based dynamic fault tolerant task scheduling approach in fog computing," *IEEE Trans. Emerg. Top. Comput.*, vol. 10, no. 1, pp. 488–499, Jan. 2022. doi: 10.1109/TETC.2020.3033672.

[9] C. Chakraborty, K. Mishra, S. K. Majhi, and H. K. Bhuyan, "Intelligent latency-aware tasks prioritization and offloading strategy in distributed fog-cloud of things," *IEEE Trans. Ind. Inform.*, vol. 19, no. 2, pp. 2099–2106, Feb. 2023. doi: 10.1109/TII.2022.3173899.

[10] Y. A. Qadri, A. Nauman, Y. B. Zikria, A. V. Vasilakos, and S. W. Kim, "The future of healthcare internet of things: A survey of emerging technologies," *IEEE Commun. Surv. Tutor.*, vol. 22, no. 2, pp. 1121–1167, 2020. doi: 10.1109/COMST.2020.2973314.

[11] J. Kang *et al.*, "Communication-efficient and cross-chain empowered federated learning for artificial intelligence of things," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 5, pp. 2966–2977, Sep. 2022. doi: 10.1109/TNSE.2022.3178970.

[12] L. Kong *et al.*, "Edge-computing-driven internet of things: A survey," *ACM Comput. Surv.*, vol. 55, no. 8, pp. 1–41, Aug. 2023. doi: 10.1145/3555308.

[13] S. Chen *et al.*, "Internet of things based smart grids supported by intelligent edge computing," *IEEE Access*, vol. 7, pp. 74089–74102, 2019. doi: 10.1109/ACCESS.2019.2920488.

[14] N. Tariq *et al.*, "The security of big data in fog-enabled iot applications including blockchain: A survey," *Sensors*, vol. 19, no. 8, pp. 1788, 2019. doi: 10.3390/s19081788.

[15] H. Yang, A. Alphones, W. -D. Zhong, C. Chen, and X. Xie, "Learning-based energy-efficient resource management by heterogeneous RF/VLC for ultra-reliable low-latency industrial IoT networks," *IEEE Trans. Ind. Inform.*, vol. 16, no. 8, pp. 5565–5576, Aug. 2020. doi: 10.1109/TII.2019.2933867.

[16] A. Jain and P. Singhal, "Fog computing: Driving force behind the emergence of edge computing," in *2016 Int. Conf. Syst. Model. Adv. Res. Trends (SMART)*, Moradabad, India, IEEE, 2016, pp. 294–297. doi: 10.1109/SYSMART.2016.7894538.

[17] M. Al-khafajiy, T. Baker, H. Al-Libawy, Z. Maamar, M. Aloqaily and Y. Jararweh, "Improving fog computing performance via *Fog*-2-*Fog* collaboration," *Future Gener. Comput. Syst.*, vol. 100, no. 6, pp. 266–280, Nov. 2019. doi: 10.1016/j.future.2019.05.015.

[18] Y. L. Jiang, Y. S. Chen, S. W. Yang, and C. H. Wu, "Energy-efficient task offloading for time-sensitive applications in fog computing," *IEEE Syst. J.*, vol. 13, no. 3, pp. 2930–2941, Sep. 2019. doi: 10.1109/JSYST.2018.2877850.

[19] J. Ren, G. Yu, Y. He, and G. Y. Li, "Collaborative cloud and edge computing for latency minimization," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 5031–5044, May 2019. doi: 10.1109/TVT.2019.2904244.

[20] S. Khan, I. Ali Shah, N. Tairan, H. Shah, and M. Faisal Nadeem, "Optimal resource allocation in fog computing for healthcare applications," *Comput. Mater. Contin.*, vol. 71, no. 3, pp. 6147–6163, 2022. doi: 10.32604/cmc.2022.023234.

[21] M. Abd Elaziz, L. Abualigah, and I. Attiya, "Advanced optimization technique for scheduling IoT tasks in cloud-fog computing environments," *Future Gener. Comput. Syst.*, vol. 124, no. 9, pp. 142–154, Nov. 2021. doi: 10.1016/j.future.2021.05.026.

[22] Z. Liu, X. Yang, Y. Yang, K. Wang, and G. Mao, "DATS: Dispersive stable task scheduling in heterogeneous fog networks," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3423–3436, Apr. 2019. doi: 10.1109/JIOT.2018.2884720.

[23] Z. He, Y. Zhang, B. Tak, and L. Peng, "Green fog planning for optimal internet-of-thing task scheduling," *IEEE Access*, vol. 8, pp. 1224–1234, 2020. doi: 10.1109/ACCESS.2019.2961952.

[24] M. Abdel-Basset, R. Mohamed, R. K. Chakrabortty, and M. J. Ryan, "IEGA: An improved elitism-based genetic algorithm for task scheduling problem in fog computing," *Int. J. Intell. Syst.*, vol. 36, no. 9, pp. 4592–4631, Sep. 2021. doi: 10.1002/int.22470.

[25] A. Ali Salamai, A. Abdulrahman Ageeli, and E S. M. El-kenawy, "A new task scheduling scheme based on genetic algorithm for edge computing," *Comput. Mater. Contin.*, vol. 71, no. 1, pp. 843–854, 2022. doi: 10.32604/cmc.2022.017504.

[26] H. B. Mahajan, A. Badarla, and A. A. Junnarkar, "CL-IoT: Cross-layer Internet of Things protocol for intelligent manufacturing of smart farming," *J. Ambient Intell. Humaniz. Comput.*, vol. 12, no. 7, pp. 7777–7791, Jul. 2021. doi: 10.1007/s12652-020-02502-0.

[27] G. Hu, Y. Guo, G. Wei, and L. Abualigah, "Genghis Khan shark optimizer: A novel nature-inspired algorithm for engineering optimization," *Adv. Eng. Inform.*, vol. 58, no. 8, pp. 102210, Oct. 2023. doi: 10.1016/j.aei.2023.102210.

[28] G. Taheri, A. Khonsari, R. Entezari-Maleki, and L. Sousa, "A hybrid algorithm for task scheduling on heterogeneous multiprocessor embedded systems," *Appl. Soft Comput.*, vol. 91, no. 3, pp. 106202, Jun. 2020. doi: 10.1016/j.asoc.2020.106202.

[29] J. P. B. Mapetu, Z. Chen, and L. Kong, "Low-time complexity and low-cost binary particle swarm optimization algorithm for task scheduling and load balancing in cloud computing," *Appl. Intell.*, vol. 49, no. 9, pp. 3308–3330, Sep. 2019. doi: 10.1007/s10489-019-01448-x.

[30] S. Azizi, M. Shojafar, J. Abawajy, and R. Buyya, "Deadline-aware and energy-efficient IoT task scheduling in fog computing systems: A semi-greedy approach," *J. Netw. Comput. Appl.*, vol. 201, no. 10, pp. 103333, May 2022. doi: 10.1016/j.jnca.2022.103333.

[31] S. Misra and N. Saha, "*Detour:* Dynamic task offloading in software-defined fog for iot applications," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1159–1166, May 2019. doi: 10.1109/JSAC.2019.2906793.

[32] M. Maray, E. Mustafa, J. Shuja, and M. Bilal, "Dependent task offloading with deadline-aware scheduling in mobile edge networks," *Int. Things*, vol. 23, no. 1, pp. 100868, Oct. 2023. doi: 10.1016/j.iot.2023.100868.

[33] B. M. Nguyen, H. Thi Thanh Binh, T. The Anh, D. Bao Son, "Evolutionary algorithms to optimize task scheduling problem for the IoT based bag-of-tasks application in cloud-fog computing environment," *Appl. Sci.*, vol. 9, no. 9, pp. 1730, Apr. 2019. doi: 10.3390/app9091730.

[34] M. Abdel-Basset, R. Mohamed, M. Elhoseny, A. K. Bashir, A. Jolfaei, and N. Kumar, "Energy-aware marine predators algorithm for task scheduling in IoT-based fog computing applications," *IEEE Trans. Ind. Inform.*, vol. 17, no. 7, pp. 5068–5076, Jul. 2021. doi: 10.1109/TII.2020.3001067.

[35] S. Ghanavati, J. Abawajy, and D. Izadi, "An energy aware task scheduling model using ant-mating optimization in fog computing environment," *IEEE Trans. Serv. Comput.*, vol. 15, no. 4, pp. 2007–2017, Jul. 2022. doi: 10.1109/TSC.2020.3028575.

[36] S. Mousavi, S. E. Mood, A. Souri, and M. M. Javidi, "Directed search: A new operator in NSGA-II for task scheduling in IoT based on cloud-fog computing," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 2144–2157, Apr. 2023. doi: 10.1109/TCC.2022.3188926.

[37] F. A. Saif, R. Latip, Z. M. Hanapi, and K. Shafinah, "Multi-objective grey wolf optimizer algorithm for task scheduling in cloud-fog computing," *IEEE Access*, vol. 11, pp. 20635–20646, 2023. doi: 10.1109/ACCESS.2023.3241240.

[38] J. Xu, Z. Hao, R. Zhang, and X. Sun, "A method based on the combination of laxity and ant colony system for cloud-fog task scheduling," *IEEE Access*, vol. 7, pp. 116218–116226, 2019. doi: 10.1109/ACCESS.2019.2936116.

[39] S. Dabiri, S. Azizi, and A. Abdollahpouri, "Optimizing deadline violation time and energy consumption of IoT jobs in fog–cloud computing," *Neural Comput. Appl.*, vol. 34, no. 23, pp. 21157–21173, Dec. 2022. doi: 10.1007/s00521-022-07596-5.

[40] P. Hosseinioun, M. Kheirabadi, S. R. Kamel Tabbakh, and R. Ghaemi, "A new energy-aware tasks scheduling approach in fog computing using hybrid meta-heuristic algorithm," *J. Parallel Distrib. Comput.*, vol. 143, pp. 88–96, Sep. 2020. doi: 10.1016/j.jpdc.2020.04.008.

[41] I. M. Ali, K. M. Sallam, N. Moustafa, R. Chakraborty, M. Ryan, and K. K. R. Choo, "An automated task scheduling model using non-dominated sorting genetic algorithm ii for fog-cloud systems," *IEEE Trans. Cloud Comput.*, vol. 10, no. 4, pp. 2294–2308, Oct. 2022. doi: 10.1109/TCC.2020.3032386.

[42] A. Faramarzi, M. Heidarinejad, B. Stephens, and S. Mirjalili, "Equilibrium optimizer: A novel optimization algorithm," *Knowl.-Based Syst.*, vol. 191, pp. 105190, Mar. 2020. doi: 10.1016/j.knosys.2019.105190.

[43] R. Rai and K. G. Dhal, "Recent developments in equilibrium optimizer algorithm: Its variants and applications," *Arch. Comput. Methods Eng.*, vol. 30, no. 6, pp. 3791–3844, Jul. 2023. doi: 10.1007/s11831-023-09923-y.

[44] M. Ghasemi, M. Zare, A. Zahedi, M. A. Akbari, S. Mirjalili, and L. Abualigah, "Geyser inspired algorithm: A new geological-inspired meta-heuristic for real-parameter and constrained engineering optimization," *J. Bionic. Eng.*, vol. 21, no. 1, pp. 374–408, Jan. 2024. doi: 10.1007/s42235-023-00437-8.

[45] R. O. Aburukba, T. Landolsi, and D. Omer, "A heuristic scheduling approach for fog-cloud computing environment with stationary IoT devices," *J. Netw. Comput. Appl.*, vol. 180, no. 1, pp. 102994, Apr. 2021. doi: 10.1016/j.jnca.2021.102994.

[46] A. Alelaiwi, "An efficient method of computation offloading in an edge cloud platform," *J. Parallel Distrib. Comput.*, vol. 127, no. 8, pp. 58–64, May 2019. doi: 10.1016/j.jpdc.2019.01.003.

[47] B. Sellami, A. Hakiri, S. B. Yahia, and P. Berthou, "Energy-aware task scheduling and offloading using deep reinforcement learning in SDN-enabled IoT network," *Comput. Netw.*, vol. 210, no. 3, pp. 108957, Jun. 2022. doi: 10.1016/j.comnet.2022.108957.

[48] S. Sheng, P. Chen, Z. Chen, L. Wu, and Y. Yao, "Deep reinforcement learning-based task scheduling in iot edge computing," *Sensors*, vol. 21, no. 5, pp. 1666, Feb. 2021. doi: 10.3390/s21051666.

[49] Q. Fan, J. Bai, H. Zhang, Y. Yi, and L. Liu, "Delay-aware resource allocation in fog-assisted iot networks through reinforcement learning," 2020. Accessed: Jan. 25, 2024. [Online]. Available: http://arxiv.org/abs/2005.04097

[50] S. Das, S. S. Mullick, and P. N. Suganthan, "Recent advances in differential evolution–An updated survey," *Swarm Evol. Comput.*, vol. 27, no. 3, pp. 1–30, Apr. 2016. doi: 10.1016/j.swevo.2016.01.004.

[51] M. Abdel-Basset, D. El-Shahat, M. Elhoseny, and H. Song, "Energy-aware metaheuristic algorithm for industrial-internet-of-things task scheduling problems in fog computing applications," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 12638–12649, Aug. 2021. doi: 10.1109/JIOT.2020.3012617.

[52] N. Dordaie and N. J. Navimipour, "A hybrid particle swarm optimization and hill climbing algorithm for task scheduling in the cloud environments," *ICT Express*, vol. 4, no. 4, pp. 199–202, Dec. 2018. doi: 10.1016/j.icte.2017.08.001.

[53] X. Chen *et al.*, "A WOA-based optimization approach for task scheduling in cloud computing systems," *IEEE Syst. J.*, vol. 14, no. 3, pp. 3117–3128, Sep. 2020. doi: 10.1109/JSYST.2019.2960088.

[54] Z. Yin *et al.*, "A multi-objective task scheduling strategy for intelligent production line based on cloud-fog computing," *Sensors*, vol. 22, no. 4, pp. 1555, Feb. 2022. doi: 10.3390/s22041555.