High-Confidence Computing 5 (2025) 100317

Contents lists available at ScienceDirect

High-Confidence Computing

journal homepage: www.sciencedirect.com/journal/high-confidence-computing

Research Article

Dynamic OBL-driven whale optimization algorithm for independent tasks offloading in fog computing

Zulfiqar Ali Khan *, Izzatdin Abdul Aziz

Department of Computer and Information Sciences, Universiti Teknologi PETRONAS, Seri Iskandar 32610, Perak, Malaysia

ARTICLE INFO

Article history:
Received 18 December 2024
Revised 17 February 2025
Accepted 27 February 2025
Available online 18 March 2025

Keywords:
Fog computing
Task offloading
Whale Optimization Algorithm (WOA)
Opposition-Based Learning (OBL)

ABSTRACT

Cloud computing has been the core infrastructure for providing services to the offloaded workloads from IoT devices. However, for time-sensitive tasks, reducing end-to-end delay is a major concern. With advancements in the IoT industry, the computation requirements of incoming tasks at the cloud are escalating, resulting in compromised quality of service. Fog computing emerged to alleviate such issues. However, the resources at the fog layer are limited and require efficient usage. The Whale Optimization Algorithm is a promising meta-heuristic algorithm extensively used to solve various optimization problems. However, being an exploitation-driven technique, its exploration potential is limited, resulting in reduced solution diversity, local optima, and poor convergence. To address these issues, this study proposes a dynamic opposition learning approach to enhance the Whale Optimization Algorithm to offload independent tasks. Opposition-Based Learning (OBL) has been extensively used to improve the exploration capability of the Whale Optimization Algorithm. However, it is computationally expensive and requires efficient utilization of appropriate OBL strategies to fully realize its advantages. Therefore, our proposed algorithm employs three OBL strategies at different stages to minimize end-to-end delay and improve load balancing during task offloading. First, basic OBL and quasi-OBL are employed during population initialization. Then, the proposed dynamic partial-opposition method enhances search space exploration using an information-based triggering mechanism that tracks the status of each agent. The results illustrate significant performance improvements by the proposed algorithm compared to SACO, PSOGA, IPSO, and oppoCWOA using the NASA Ames iPSC and HPC2N workload datasets.

1. Introduction

Over the past decade, the rapid growth of the Internet of Things (IoT) and the developments in digital technologies have transformed various sectors, including smart homes, transportation, agriculture, and manufacturing industries. The IoT applica tions produce massive volumes of data that require both processing and storage. Cloud computing has been managing the needs of such devices by offering virtualized services on a pay-per-use policy. However, in recent years, IoT-based systems have evolved drastically, widening the gap between cloud computing and the needs of modern IoT applications. Many applications, such as virtual reality, gaming, and smart cities have strict requirements for response time and need real-time processing [1]. However, the long geographical distances between cloud servers and end users, limited bandwidth, and the resource-constrained nature of IoT devices result in longer delays, higher costs, and reduced Quality of Service (QoS).

Fog computing emerged to alleviate the above issues, complementing the cloud infrastructure by providing services at the network edge. Fog nodes allow the generated workloads to be executed in close proximity instead of being transmitted to distant cloud servers. This reduces delays, costs, and energy consumption of IoT devices while improving QoS [2]. Fog computing not only executes delay-intolerant workloads at the network edge but also reduces network traffic [3]. However, fog layer has limited resources and requires efficient usage to provide better QoS. The fog layer plays a crucial role in handling delay-sensitive tasks that need to be processed within a short span of time [4], while cloud computing remains the backbone for data storage and analytics.

The aim of task offloading is to execute workloads by utilizing distributed fog and cloud resources, thereby providing better QoS. In a three-layer architecture as shown in Fig. 1, IoT/edge devices offload their workload to gateway devices at the network edge. These gateway devices execute such requests and return the results to the originating devices, offering minimum delays. However, if the number of requests is large or the tasks are computation-intensive, the gateway devices may not be able to handle them, necessitating offloading to other fog nodes. Furthermore, if the fog layer lacks the necessary resources to execute the

* Corresponding author.
E-mail address: zulfiqar_22005381@utp.edu.my (Z.A. Khan).

https://doi.org/10.1016/j.hcc.2025.100317

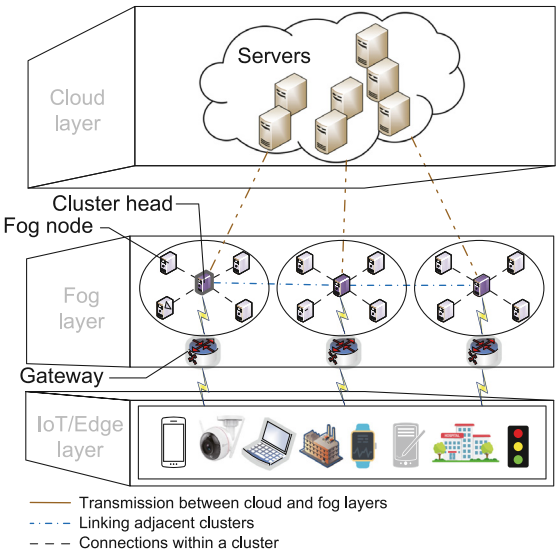

Fig. 1. Three-layered task offloading architecture.

workload, it will be offloaded to cloud servers. Ideally, a nearby fog node should be the offloading destination to minimize delay, but such nodes are susceptible to being overloaded. Therefore, maintaining an appropriate load on all fog nodes is critical for better system efficiency.

Various heuristic based task offloading algorithms have been developed, but they are feasible for a limited number of tasks and resources. Task offloading is NP-hard, where the number of computations increases exponentially with the number of inputs [5]. Therefore, on a large scale, heuristic searching techniques struggle to provide solution in polynomial time. Meta-heuristic algorithms, on the other hand, have proven useful by providing near-optimal solutions in a reasonable amount of time.

In our recent review studies related to fog computing, such as [6] on task scheduling and [7] on task offloading, it has been observed that a relatively new meta-heuristic algorithm, the Whale Optimization Algorithm (WOA), has received notable attention from researchers. However, being an exploitation-driven algorithm, the limited exploration ability of WOA results in reduced solution diversity, local optima, and poor convergence.

Opposition-Based Learning (OBL) is a useful method used to improve the solution diversity in WOA. There are broadly three OBL strategies: basic OBL, quasi-OBL, and partial-OBL. Basic OBL involves creating opposite clones of agents, while quasi-OBL generates nearly opposite clones. Partial-OBL, on the other hand, clones agents by making changes in some dimensions instead of the entire solution. In , numerous studies utilizing OBL in WOA are discussed. However, they have their advantages and disadvantages. The two major shortcomings in these studies that hinder the productivity of OBL are: the application of inappropriate OBL strategies at different stages in WOA and the jumping rate-based triggering mechanism. The jumping rate is often based on the probability of a random number or an adaptive weight rather than the information of agents. Due to these limitations, the complete potential of OBL in WOA remains untapped. To address these issues, this paper proposes the Dynamic Opposition-based learning Whale Optimization Algorithm (DOWOA), offering enhanced solution diversity. It also enables the proposed algorithm to escape local optima and prevent premature convergence. The contributions of this study are:

(1) Formulating task offloading problem in a heterogeneous fog-cloud environment with multiple clusters of fog nodes.
(2) Generating the initial population utilizing OBL and quasi-OBL to achieve better solution diversity.
(3) Proposing a dynamic partial opposition procedure to enhance global search using an information-based triggering mechanism. It minimizes end-to-end delay and improves load balancing among heterogeneous nodes.
(4) Conducting numerous simulations to evaluate the proposed algorithm against SACO [8], PSOGA [9], IPSO [10], and oppoCWOA [11] using the NASA Ames iPSC/860 and HPC2N workload instances.

The rest of the paper is organized as follows: Section 2 presents the related studies for task offloading in fog computing. Section 3 describes the proposed system model and formulation of task offloading problem. Section 4 highlights the basic Whale Optimization Algorithm, while introduces OBL and its applications for enhancing WOA. Section 6 details the proposed DOWOA. The workload datasets are illustrated in Section 7, while Section 8 presents the experimental environment. Section 9 discusses the simulations results. The conclusion and future research direction are outlined in Section 10.

2. Related studies on task offloading

The following are some of the related studies on task offloading in fog computing.

[12] presented a micro-services-based framework for a two-tier architecture. It performed efficient task offloading by using task sequencing, resource matching, and allocating appropriate nodes to minimize delay and cost. However, the study considered semi-heterogeneous nodes. Similarly, [13] optimized the number of tasks offloaded to remote nodes by minimizing delay, local processing time, and waiting time with the help of a task and data offloading algorithm. For minimizing response time and communication cost, [5] proposed a roulette-based Ant Colony Optimization (ACO) algorithm and a discrete Particle Swarm Optimization (PSO) to effectively balance the workload on fog devices. A mutation operation was utilized in the PSO to improve local search while maintaining solution diversity. On the other hand, the ACO utilized the roulette wheel selection to pick a route after calculating the probability for each path.

In [14], an improved Contract Net Protocol and Beetle Antennae Search (BAS) minimized the energy consumption and delay for a distributed fog network. However, the delay caused by the division of tasks into subtasks and the reception of the results from different nodes at various times was not discussed. In [15], a fog-cloud architecture using queuing models was presented to minimize cost, energy consumption, and delay. A stochastic gradient descent algorithm optimized the given objectives by offloading computation-intensive tasks to appropriate nodes. In [16], a dynamic task offloading algorithm was proposed to provide bandwidth guarantees on an Software-Defined Networking (SDN) network. It minimized delay and response time, and enhanced system throughput by optimal assignment of nodes considering the computational capabilities of nodes and available network bandwidth. However, being a two-tier architecture, cloud nodes were not considered. In [17], an evolutionary game was formulated to address task offloading problem using Maynard replicator dynamics. The study minimized both energy consumption and execution time of workload through the energy game and time game approaches, respectively.

A blockchain-enabled edge system was proposed to enhance the reliability and efficiency of edge devices [18]. A stacked sorting mechanism was adopted to sort and rank tasks for better resource allocation on edge nodes. Moreover, a Zipf distribution predicted hot data. However, a limited number (100) of tasks was

used during simulations. [9] presented a task offloading technique to optimize both energy consumption and task completion time. This was achieved by selecting a target fog node and assigning relevant resources. The problem was presented as a mixed-integer nonlinear problem and solved using the hybrid GA-PSO algorithm. However, the proposed algorithm utilized three different populations, followed by picking the best agents from them, which made the algorithm computationally expensive. A five-point crossover operation using the roulette wheel selection and a large-scale mutation operation were also employed to escape local optima. In [19] the levy flight was added to the Moth Flame Optimization, resulting in LMFO to minimize delay and energy consumption by 22%–29% compared to state-of-the-art algorithms.

In [8], the novel Smart Ant Colony Optimization (SACO) was proposed to minimize computation and communication latencies. In another study [10], an Improved Integer-PSO was presented to reduce the overall running time of workloads. The primitive PSO was transformed into Integer-PSO, with the addition of an adaptive inertia weight. In contrast to traversing techniques, the study emphasized the reduction in running time by 90%.

Artificial intelligence has been extensively used in vehicular applications, but it requires high computational power, which is lacking in vehicles and roadside equipment. This undermines the development of intelligent applications in vehicular networking. To address this challenge, a genetic algorithm was optimized to outsource the computations of sigmoid function to other vehicles and roadside units [20]. In addition, homomorphic encryption was applied in sub-task computations to improve data security in multi-party computations. However, only Particle Swarm Optimization (PSO) was used as a benchmark algorithm.

In [21], the Stackelberg game was employed to model the interaction between end users and fog nodes. An Energy and Delay-Aware Task Offloading (EDATO) algorithm was proposed to minimize delay and energy consumption. EDATO selected fog nodes considering the priority specifications of clients. The selection was made by the cloud server after receiving information from both the client and fog nodes. Similarly, in [22], the problems of dynamism and fog's heterogeneous limited resource were addressed by proposing TDATO, a double auction-based framework. It provided incentives to fog nodes, encouraging them to execute as many tasks as possible. In addition, two algorithms (DMS and DPS) were proposed for ensuring security and tolerance for latency by allocating the tasks to the best resources, thus increasing resource utilization. However, due to the incentive-based mechanism, faster fog nodes always operate with higher loads and vice versa.

[23] proposed the Binary Linear Weight Jaya (BLWJAYA) algorithm to offload tasks to appropriate fog nodes and virtual machines in the cloud. The algorithm used fuzzy logic to select the target destination for minimizing latency and maximizing resource utilization. It also reduced energy consumption while improving load balancing. In [24], an optimal task offloading algorithm (OEeRA) improved response time, energy efficiency, and fault-tolerance in an IoT-Fog network. It allocated resources with minimum cost and resolved faults by assigning fog nodes and resource blocks. The concept of updating resource blocks was utilized to improve fault-tolerance.

In healthcare, storing patients' hot data in edge nodes minimizes response time. However, current schemes do not guarantee the necessary requirements for hot data storage on edge nodes. Therefore, detecting and deleting redundant data under privacy constraints becomes a challenge. To address this problem, a redundant data deletion scheme was proposed, utilizing Content Extraction Signature (CES) and blockchain. This scheme reads ciphertext to evaluate hot data requirements after redundant data deletion [25]. However, the study did not discuss the blockchain's high computational requirements on resource-constrained edge nodes. Table 1 provides an overview of the related studies.

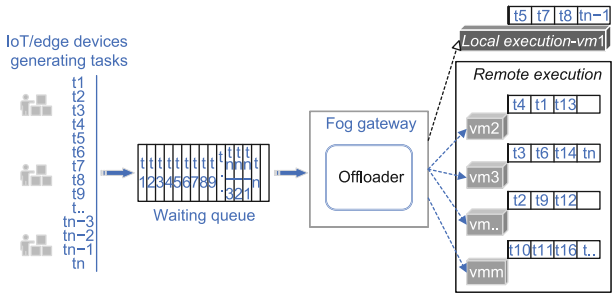

Fig. 2. Task offloading process.

3. Proposed system model and task offloading formulation

In a three-layer architecture, as depicted in Fig. 1, IoT/edge devices constitute the first layer, while fog nodes are at the second layer. The third layer is the cloud layer, possessing huge number of resources. There are several clusters of fog nodes, each with a Cluster Head (CH), responsible for scheduling, offloading, and resource management activities. A CH is also connected to two adjacent clusters, working in collaboration on the left and right. However, the communication between cluster heads is restricted within one-hop communication [26]. The CH maintains a table for all fog nodes in a cluster. The entries in this table are updated as soon as the status of fog nodes changes. It also maintains the records of fog nodes in adjacent clusters. A gateway is a fog node, which is more powerful than IoT gadgets. It receives the offloaded tasks from layer-1, executes them, and returns the results accordingly. However, computation-intensive tasks that a gateway device cannot execute are offloaded to fog/cloud nodes.

Our task offloading algorithm runs on a gateway device, receiving various kinds of workloads from layer-1. It is assumed that each offloaded task is non-preemptive and independent of other tasks. Moreover, every task has a specific number of instructions, file size, and size of output generated after its execution. Task execution on a gateway device is treated as local execution, while running a task on fog/cloud nodes is considered remote execution, as depicted in Fig. 2.

Eq. (1) computes the execution time of task i on fog/cloud node j.

$$ExT^{j}_{t_i} = \frac{t_i\ MI}{j\ MIPS} \tag{1}$$

3.1. Delay

A task incurs various types of delays upon offloading [19]. On a gateway device, these include task execution delay and queuing delay, as multiple tasks are offloaded to a gateway device. The queuing delay on any node can be computed using Eq. (2), which shows the execution time of all offloaded tasks on node j prior to the arrival of t_i.

$$QD^{j}_{t_i} = \sum_{o=1}^{a-1} \frac{SizeOft_o}{j\ MIPS} \tag{2}$$

The delay for an offloaded task on a remote node further involves transmission delay (TD) and propagation delay (PD), which can be expressed using Eqs. (3) and (4), respectively.

$$TD^{j}_{t_i} = \frac{SizeOfFile\&Output_i}{bw_j} \tag{3}$$

$$PD^{j}_{t_i} = \frac{DistanceBetweenNodes}{PS} \tag{4}$$

Table 1
Overview of related work.

Ref.	Year	Strength (s)	Weakness (es)	Architecture			Dataset	Tools
				2-tier	3-tier	4-tier		
[12]	2020	A mobility-based framework to handle delay sensitive applications with minimum cost.	Semi-heterogeneous setup.	✓	×	×	Random	GenyMotion
[13]	2020	Semi Definite Relaxation and Separable Semi Definite Programming minimizing delay, transmission delay, local processing time, and waiting time on each fog node.	Homogeneous fog nodes.	✓	×	×	Random	MATLAB
[5]	2020	Roulette-based ACO algorithm and a discrete PSO algorithm balancing the workload on fog devices. Utilizing a mutation operation to improve local search while maintaining solution diversity.	While providing minimum response time, the corresponding effect on energy consumption was not highlighted.	×	✓	×	Random	MATLAB
[14]	2020	An algorithm based on improved Contract Net Protocol and Beetle Antennae Search (BAS) minimizing energy usage and delay.	Did not highlight the delay due to the division of tasks into subtasks and managing the reception of their computation results at different timings.	✓	×	×	Random	–
[15]	2021	Minimizing energy consumption, delay, and cost of remote execution using queuing models and the Stochastic Gradient Descent algorithm (SGD).	Homogeneous fog nodes.	×	✓	×	Random	MATLAB
[16]	2021	Selecting optimal offloading nodes by filtering and ranking candidate nodes based on the computing resources and network conditions.	Offloading to cloud nodes was not considered.	✓	×	×	–	Mininet, Ryu framework
[17]	2021	Formulating task offloading as an evolutionary game and solving it via Maynard replicator dynamics.	Transmission overhead not considered.	×	×	✓	Synthetic	MATLAB
[9]	2021	Presenting task offloading as a mixed-integer nonlinear problem followed by applying the hybrid GA-PSO algorithm.	Computationally expensive.	✓	×	×	Random	–
[18]	2021	A blockchain-enabled edge system enhancing the reliability and efficiency of edge devices. A stacked sorting mechanism was adopted to sort and rank tasks to improve resource allocation on edge nodes.	Only 100 tasks were used in the simulations.	×	✓	×	–	–
[19]	2022	Using the levy flight and the Moth Flame Optimization algorithm to reduce energy consumption during task offloading.	No cooperation among fog nodes.	×	✓	×	–	MATLAB
[8]	2022	A smart ACO scheduler offloading tasks to a fog network for improving the computation and communication latencies and QoS.	It did not discuss the improvement in latency at the cost of higher energy consumption.	×	✓	×	–	MATLAB
[10]	2023	Offloading tasks utilizing the Improved Integer-PSO by reducing overall running time.	Only 11 tasks used during experiments.	×	✓	×	IoT-Compute dataset	–
[21]	2023	Energy and Delay-Aware Task Offloading algorithm selected fog nodes keeping in view the priority specifications of clients. The selection was made by the cloud server after the reception of information from both the client and fog nodes.	The selection of nodes by the cloud server introduced delays.	×	✓	×	–	–
[20]	2023	An optimized genetic algorithm outsourced computationally intensive tasks to other vehicles and roadside units. In addition, homomorphic encryption was applied in sub-task computations to improve data security in multi-party computations.	Only Particle Swarm Optimization (PSO) was used as a benchmark algorithm.	×	✓	×	–	–
[22]	2024	Executing maximum number of tasks based on providing incentives to fog nodes. It ensured security and tolerance for latency by allocating tasks to the best resources, thus increasing resource utilization.	Faster fog nodes remained overloaded and vice versa.	×	✓	×	Random	–
[23]	2024	The Binary Linear Weight Jaya (BLWJAYA) algorithm offloading tasks using fuzzy logic.	A semi-heterogeneous system.	×	✓	×	HCSP, NASA	iFogSim
[24]	2024	OEeRA used to allocate resources with minimum cost and resolved faults by utilizing the concept of resource blocks.	Did not consider failure of fog nodes.	×	✓	×	–	MATLAB
[25]	2024	A redundant data deletion scheme utilized the Content Extraction Signature (CES) and blockchain to evaluate hot data requirements after deleting redundant data.	Did not discuss the blockchain's high computational requirements on resource-constrained edge nodes.	×	✓	×	–	–

✓ *indicates "True".*
× *indicates "False".*
– *indicates "Not indicated".*

Here, the euclidean distance formula is employed to find the distance between two nodes. bW stands for bandwidth, while PS is the propagation speed of a network. Thus, the delay for an offloaded task on a remote node can be expressed by Eq. (5).

$$D_{t_i} = ExT^j_{t_i} + QD^j_{t_i} + TD^j_{t_i} + PD^j_{t_i} \tag{5}$$

In this study, the propagation delay is assumed to be negligible. The average delay for all tasks can be computed using Eq. (6).

$$Avg\_Delay = \frac{\sum_{i=1}^{n} D_{t_i}}{n} \tag{6}$$

The execution time of all offloaded tasks on a node can be expressed using Eq. (7).

$$ExT^j_{t_{all}} = \frac{\sum_{i=1}^{k/l/m} t_i\ MI}{j\ MIPS} \tag{7}$$

Table 2
Description of all notations.

Notation	Description
i	Task index
j	Node index
n	Number of tasks
r	Number of nodes
o	Any offloaded task on a node
a	All offloaded tasks on a node
k	All tasks running on a gateway device
l	All tasks running on a fog node
m	All tasks running on a cloud node
MI	Million(s) of instructions
$MIPS$	Million(s) of instructions per second
$ExT_{t_i}^{j}$	Running duration of task i on node j
$QD_{t_i}^{j}$	Queuing delay of task i on node j
$TD_{t_i}^{j}$	Transmission delay of task i on node j
$PD_{t_i}^{j}$	Propagation delay of task i on node j
D_{t_i}	Total delay for task i
$ExT_{t_{all}}^{all}$	Execution time of all offloaded tasks on all nodes
LB_{avg}	Average load on each node

3.2. Load balancing

A balanced load in a network improves efficiency and end-to-end delay. The capacity of a node can be determined by considering the number of CPU cores, MIPS of each core, and bandwidth using Eq. (8) [23].

$$Cf_i = (No.OfCores\ x\ j\ MIPS) + bW_j \tag{8}$$

The capacity of all nodes can be computed using Eq. (9).

$$Cf_{all} = \sum_{j=1}^{r} Cf_j \tag{9}$$

The load of all offloaded tasks on a single node is already computed by Eq. (7). Now, the load of all nodes can be computed using Eq. (10).

$$ExT_{t_{all}}^{all} = \sum_{j=1}^{r} ExT_{all}^{j} \tag{10}$$

The ratio of $ExT_{t_{all}}^{j}$ to Cf_j determines the load balance for each node using Eq. (11). The average load balancing value for all nodes can be computed using Eq. (12).

$$LB_j = \frac{ExT_{all}^{j}}{Cf_j} \tag{11}$$

$$LB_{avg} = \frac{\sum_{j=1}^{r} LB_j}{r} \tag{12}$$

A lower value for LB_{avg} is desirable, as it indicates a balanced workload on fog nodes, avoiding both overloading and underloading. If a task executes locally, it results in lower end-to-end delay. However, if more tasks are executed locally, the queuing delay increases. Conversely, offloading tasks to remote nodes decreases the load on a local node, reducing queuing delays but results in greater transmission delays. Therefore, there needs to be a balanced distribution of tasks among offloading destinations. Ideally delay-intolerant tasks should be executed on local nodes or near-by fog nodes, while computation-intensive tasks should be offloaded to remote cloud nodes. Table 2 provides the description of all notations.

4. Whale optimization algorithm

The Whale Optimization Algorithm (WOA) [27] is inspired by the hunting strategy of a group of humpback whales using the shrinking encircling maneuver and bubble-net feeding/hunting /attacking. It starts with a random population of whales, followed by updating their positioning in the search space. The best whale is identified based on certain criteria, while other whales are guided towards it using random search, encircling prey (fish/krill), and bubble-net feeding.

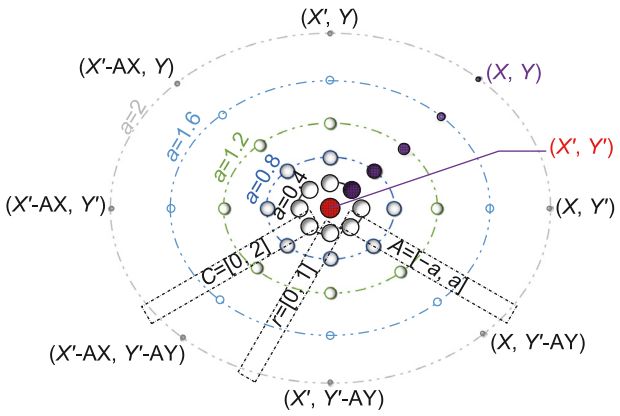

Fig. 3. Shrinking encircling mechanism.

Random search: A whale's random searching promotes the exploration of solution space. After selecting a random whale, other whales update their positions in relation to this random agent, as expressed by Eqs. (13)–(14).

$$D = |C * randPos - X_t| \tag{13}$$

$$X_{t+1} = randPos - A * D \tag{14}$$

D represents a whale's distance X_t at time t from the random whale, while X_{t+1} represents the updated whale location in relation to the random whale. A and C are coefficient vectors that drive the convergence behavior of WOA, computed using Eqs. (15) and (16).

$$A = 2 * a * r - a \tag{15}$$

$$C = 2 * r \tag{16}$$

Here, the value of a linearly decreases from 2 to 0, so the values of A fall in the range of [-a, a]. r is a random number in the range of [0, 1]. A higher a value promotes exploration, while a lower value encourages exploitation.

Shrinking encircling: After the prey has been identified, all whales begin to encircle it. Since the prey's location is not known in advance, WOA assumes that the current best whale represents the target prey or is near it. In shrinking encircling maneuver as displayed in Fig. 3, the circumference of this circular movement gradually reduces, thus bringing the whales closer to the prey, promoting exploitation. Eqs. (17) and (18) represent this behavior.

$$D = |C * bestWhalePos - X_t| \tag{17}$$

$$X_{t+1} = bestWhalePos - A * D \tag{18}$$

Here, D represents the distance of a whale from the best whale at iteration t, while X_{t+1} is the updated whale location at iteration $t + 1$.

Bubble-net feeding: In addition to shrinking encircling, whales also update their positions in a spiral-shaped path relative to the best whale, as shown in Fig. 4. This behavior is called bubble-net feeding, and it also promotes exploitation. This mechanism is expressed using Eqs. (19) and (20).

$$D' = |bestWhalePos - X_t| \tag{19}$$

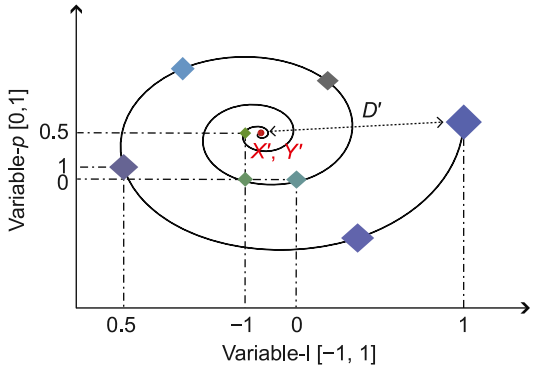

Fig. 4. Spiral position update during bubble-net feeding.

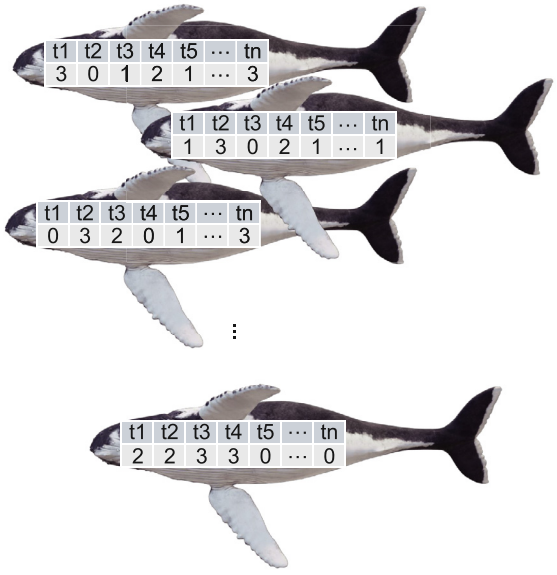

Fig. 5. Individual whales as solutions in WOA.

t_1	t_2	t_3	t_4	⋯	t_{n-2}	t_{n-1}	t_n
2	5	4	3	⋯	8	1	9

Fig. 6. Mapping of tasks to nodes.

$$X_{t+1} = D' * e^{bl} * \cos(2 * \pi * l) + bestWhalePos \tag{20}$$

The whale's position in Eq. (20) is computed using a spiral-shaped path by employing a constant $b = 1$, which defines the shape of the spiral. l is another random number ranging between $[-1, 1]$. p is a random probability in the range of [0, 1], used to switch among random search, shrinking encircling, and bubble-net feeding.

A population comprises numerous whales (agents). Each whale represents a solution which is a mapping of tasks to nodes, as shown in Fig. 5. As the number of tasks are usually more than nodes, there is a many-to-one mapping between the tasks and nodes. For example, Fig. 6 shows the mapping of n tasks on 10 fog nodes.

4.1. Limitations of WOA

The major limitation of WOA is its poor exploration mechanism. The variables of p and A allow random search in the search space approximately 25% of the time, resulting in poor convergence, local optima, and reduced diversity. However, simply increasing this percentage also deteriorates the effectiveness of shrinking encircling and bubble-net feeding. An alternative approach needs to be developed to enhance the exploration potential without compromising the existing exploitation-focused mechanisms.

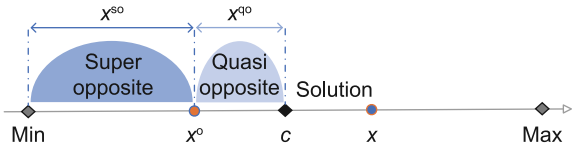

Fig. 7. OBL, super OBL, and quasi-OBL.

5. Opposition-based learning (OBL)

Tizhoosh proposed opposition-based learning [28] to explore search spaces by incorporating a population and its opposite simultaneously. Tizhoosh claimed that the opposite of a random number has a higher probability of being closer to the solution than the random number itself.

Suppose x is a solution in the range of [min, max]. The opposite solution of x can be computed as:

$$x^o = min + max - x \tag{21}$$

The center of the solution space is considered a reference point, defined as c.

$$c = \frac{min + max}{2} \tag{22}$$

According to [29], OBL has two types: super opposition-based learning and quasi-opposition-based learning. The super opposite and quasi-opposite solutions for x can be computed as

$$x^{so} = rand[min, min + max - x]; \ if \ x > c \tag{23}$$

$$x^{so} = rand[min, max] - x; \ if \ x = c \tag{24}$$

$$x^{so} = rand[min + max - x, max]; \ if \ x < c \tag{25}$$

$$x^{qo} = rand[min + max - x, c]; \ if \ x > c \tag{26}$$

$$x^{qo} = \emptyset; \ if \ x = c \tag{27}$$

$$x^{qo} = rand[c, min + max - x]; \ if \ x < c \tag{28}$$

The center point of the search space is critical to determine [29] super-opposite and quasi-opposite solutions as shown in Fig. 7. When OBL is applied to a specific segment of a solution instead of an entire agent, is called partial-OBL [11].

Opposition-based learning and WOA

The studies [30–35], and [36] utilized both chaotic maps and OBL strategies. However, the application of OBL varied. [34,37–39], and [35] only employed OBL in initial population generation. This helped to generate a better initial population. The studies [31,40–43], and [44] only employed OBL for every agent after population initialization. On the other hand, [45] employed OBL in both population initialization and generation jumping processes. [30,33,46,47], and [48] also applied OBL in both initial population generation and later on for every agent in various ways. In all studies, OBL was either computed in every iteration or triggered based on a jumping rate. This rate was either based

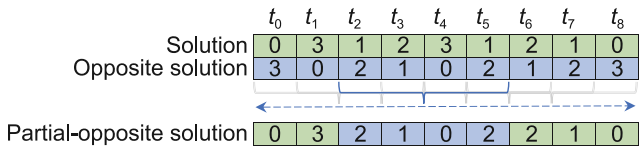

Fig. 8. Dynamic generation of a partial-opposite solution.

on the probability of a random number or an adaptive weight. These five studies have made effective use of OBL. However, the computation of OBL in every iteration and the jumping rate based triggering mechanism pose a bottleneck. [32] executed OBL if the fitness of entire population did not improve for a specific ratio of iterations. However, it transformed all agents into opposite clones instead of individual agents, leading to premature convergence. Moreover, while using OBL, half of the agents serve as regular agents, while the remaining half constitutes opposite, partial-opposite, or quasi-opposite population, which require additional computations. The frequent changes in population due to executing OBL strategies require repeated computation of fitness value, making OBL-based WOAs computation-intensive.

6. Proposed dynamic OBL-driven whale optimization algorithm

Unlike basic OBL, which modifies all dimensions to opposite values, partial-OBL alters some dimensions based on requirements. In this paper, a dynamic partial-OBL strategy is proposed to convert a candidate solution into partial-opposite solution by selecting an estimated one-third of the items from an opposite solution. The selection of the items from the opposite solution is dynamic, each time selecting a different chunk of indices. The dynamic selection helps avoid getting stuck in local optimal regions.

Two points, A and B, are identified in a set of tasks using Eqs. (29) and (30).

$$Point_A = \frac{No.OfTasks}{3} - 1 \tag{29}$$

$$Point_B = No.OfTasks - \frac{No.OfTasks}{3} \tag{30}$$

To construct a partial-opposite solution, the range of indices for copying items from an opposite solution is calculated using Eq. (31).

$$range = point_B - point_A \tag{31}$$

The first crossover point is randomly generated using Eq. (32), while the second crossover point is computed by adding the value of the range to the first crossover point using Eq. (33).

$$CrossOverPoint_1 = rand(No.OfTasks - range) \tag{32}$$

$$CrossOverPoint_2 = CrossOverPoint_1 + range \tag{33}$$

For a set of nine tasks and four nodes, Fig. 8 displays a sample solution (mapping of offloaded tasks to nodes) and the procedure for generating a dynamic partial-opposite solution.

In the previous section, different OBL strategies are utilized to enhance the exploration potential of WOA, thus improving the solution diversity. However, our proposed DOWOA utilizes three OBL variants and a new triggering mechanism instead of jumping rate to make efficient use of OBL. The salient features of our proposed algorithm are:

(1) During population initialization, OBL and quasi-OBL are used alongside random initialization to create every three sets of agents. However, instead of generating complete OBL and quasi-OBL populations, only the required agents are initialized and merged into a single initial population. It ensures extensive coverage of the solution space.
(2) After initialization, dynamic partial-OBL is employed for every agent provided its fitness does not improve for a predefined waiting threshold, instead of using a jumping rate.
(3) The algorithm keeps only one population of agents instead of multiple populations, which is crucial to minimize the number of computations.
(4) Some elite agents are also preserved from undergoing any type of change. This proves useful, particularly when the number of promising solutions is limited in a search space.

These four features allow the proposed DOWOA to increase solution diversity, prevent premature convergence, and escape local optima while remaining lightweight. In contrast to jumping rate, the proposed triggering mechanism avoids the frequent computation of partial-OBL. The time complexity of DOWOA is $O(T(W * D) + F * W)$, where T is the number of iterations, W is the number of agents, D is the dimension of the problem, and F is the cost of fitness function evaluation.

The worst time complexity (O) of traditional WOA, oppoCWOA, and our proposed DOWOA remain the same because the addition of OBL takes constant time complexity. Both oppoCWOA and DOWOA utilize partial-OBL operations. However, there are two major differences in DOWOA. First, the agents' initialization employs simple OBL and quasi-OBL. These constant time operations replace random initialization and execute once. Furthermore, the dynamic partial-OBL in DOWOA executes less frequently as opposed to the partial-OBL routine in oppoCWOA, which executes in every iteration, leading to prolonged execution time. In any task offloading scenario including large-scale deployments, the additional computational overhead of dynamic partial-OBL is minimal. Improving optimization performance of WOA by OBL at the expense of additional constant-time complexity in oppoCWOA was considered acceptable. However, DOWOA's information-based triggering mechanism reduces this complexity in addition to enhancing the optimization performance. Algorithm 1 presents the pseudocode of the proposed algorithm.

Algorithm 1 takes two inputs: a set of tasks and another set of nodes. The nodes constitute a gateway device, fog nodes, and cloud nodes. The output is a mapping of tasks to nodes. At line 1, population initialization is performed using random initialization, OBL, and quasi-OBL for every three agents. Line 2 initializes the basic parameters. In line 3, an array *wait* is initialized with zeros, with an entry for every agent. *waitThreshold* is initialized with 10, indicating a maximum threshold for not improving an agent's fitness. It determines the triggering of computing dynamic partial-opposite of an agent. In line 4, the fitness of all whales is calculated while the best agent and elites are determined in line 5. *NoOfWhales*/*waitThreshold* specifies the ratio of elites in the population.

The main loop starts in line 6 followed by another loop for every agent in line 7. A third for loop starts in line 9, working on every task. However, it only executes for non-elite whales. Another set of variables is initialized with zeros at line 10. A new whale position is calculated (line 13) by tracking a random whale if p is less than 0.5 and the absolute value of A is greater than or equal to 1. The shrinking encircling is conducted (line 15) if p is less than 0.5 and $|A|$ is less than 1. If p is greater than or equal to 0.5, the whale's position is updated through a spiral-shaped path in line 18. If the new whale's position is outside the solution

Algorithm 1 Dynamic OBL-driven whale optimization algorithm (DOWOA).

Input: Tasks: T, gateway/fog/cloud nodes: *Nodes*
Output: *Map*(T, *Nodes*)

1: Initialize whales' population using random initialization, Eq. (21) for OBL, and Eqs. (26)-(28) for quasi-OBL.
2: Initialize a, A, C, p, l, b
3: Initialize an array $wait$ and $waitThreshold$
4: Compute fitness of all whales
5: Determine the best whale as *gBW* and a set of elites containing $NoOfWhales/waitThreshold$ agents.
6: **while** $t \leq tMax$ **do**
7: **for each** whale **do**
8: **if** !*elite* **then**
9: **for each** task **do**
10: Initialize local variables including D^r, D', and D
11: **if** $p < 0.5$ **then**
12: **if** $|A| \geq 1$ **then**
13: Update positioning using Eq. (13) and Eq. (14)
14: **else if** $|A| < 1$ **then**
15: Compute positioning using Eq. (17) and Eq. (18)
16: **end if**
17: **else if** $p \geq 0.5$ **then**
18: Calculate whale position using Eq. (19) and Eq. (20)
19: **end if**
20: **if** whale positioning is beyond the search space **then**
21: Compute a random whale position
22: **end if**
23: update population
24: **end for**
25: **end if**
26: **if** $wait == waitThreshold$ & !*elite* **then**
27: Calculate opposite solution using Eq. (21)
28: Compute dynamic partial-opposite solution using Eqs. (29)-(33)
29: update population
30: reset $wait$ to zero
31: **end if**
32: Calculate delay using Eq. (6)
33: Compute fitness
34: $wait++$
35: **if** *fitness* > *PrevFitness* **then**
36: Update *fitness*
37: $wait--$
38: **if** *fitness* > *gBFitness* **then**
39: *gBFitness* ← *fitness*
40: Update global best solution
41: **end if**
42: **end if**
43: Update a, A, C, p, l
44: Update *elites*
45: **end for**
46: **end while**

space, a random position is generated (line 21). The population is updated in line 23.

The dynamic partial-OBL is executed if the corresponding waiting index for an agent reaches the waiting threshold (lines 26 to 31). The population is updated and the waiting index is reset to zero. Lines 32–34 calculate the fitness of the current agent and increment the corresponding value in the $wait$ array. If the new fitness is better than the previous, the agent's fitness is updated, followed by decrementing its $wait$ value in lines 35–37. Furthermore, if the fitness is better than the global best fitness, the current fitness is declared the global best fitness, and the relevant agent is saved as final solution in lines 38–40. The basic parameters and the elite whales are updated in lines 43–44.

Considering the WOA's limitations, as highlighted in Section 4.1, the key improvement DOWOA offers over traditional WOA and OBL-based WOAs is the enhanced exploration potential. It is achieved in two steps. First, by using basic OBL and quasi-OBL in population initialization, ensuring extensive coverage of the solution space. Second, when the fitness of an agent does not improve until the waiting threshold due to limited exploration of the solution space, the dynamic OBL is executed. This operation is only triggered when the traditional WOA is unable to explore the solution space further and fails to improve an agent's fitness. Our proposed algorithm first relies on the traditional WOA's operations and when these fail to improve the fitness, the proposed dynamic routine is executed. This creates a balance between exploration and exploitation, providing optimal solution quality. The information-based triggering mechanism is unique compared to benchmark algorithms (SACO, PSOGA, IPSO, oppoCWOA). It significantly improves the solution quality, thereby reducing end-to-end delay and improving load balancing on fog nodes. It ensures an optimal balance between local and global searching, escaping local optimal regions.

Table 3
Range of instructions, which are in MI [Min, Max].

Instances	NASA	HPC2N
100	[2, 318592]	[1, 2879488]
200	[2, 337536]	[1, 3455952]
300	[2, 1227648]	[2, 2765488]
400	[2, 1224064]	[1, 2885616]
500	[1, 1255936]	[1, 4146816]
600	[1, 1224192]	[1, 4035584]

Table 4
Additional workload attributes of NASA and HPC2N.

Instances	File size	Output size	Memory required
100	[20, 100] MB	[5, 100] MB	[50, 200] MB
200	[20, 100] MB	[5, 100] MB	[50, 200] MB
300	[20, 100] MB	[5, 100] MB	[50, 200] MB
400	[20, 100] MB	[5, 100] MB	[50, 200] MB
500	[20, 100] MB	[5, 100] MB	[50, 200] MB
600	[20, 100] MB	[5, 100] MB	[50, 200] MB

Table 5
Data center configuration.

Parameter	Cloud	Adjacent fog clusters	Fog cluster	Gateway	Unit
No. of VMs	5	10	16	1	VM
CPU	522–720	207–930	147–995	728	MIPS
Bandwidth	1024	1024	1024	1024	MB/s

7. Workloads

The workloads of NASA Ames IPSC/860 and HPC2N are used to evaluate the performance of all algorithms. For calculating delay, the data about file size and the size of generated output by each task are required. However this data is made available by [49] with the NASA workload only. The same random data generated from a uniform distribution is also appended to HPC2N in this study. The range of instructions for all instances are given in Table 3, while the ranges for the additional fields are provided in Table 4.

8. Experimental environment

All simulations are conducted in iFogSim [50], running on Intel(R) Core(TM) i7-4790-3.60 GHz with 8 GB of RAM. The data center comprises 32 heterogeneous virtual machines (VMs) according to the specifications by Braun et al. [51]. The VMs are segregated in three layers as depicted in Table 5 and Fig. 1.

9. Results and discussion

Figs. 9 and 10 depict the convergence behavior of all algorithms (using 30 agents) by running 300 instructions from the

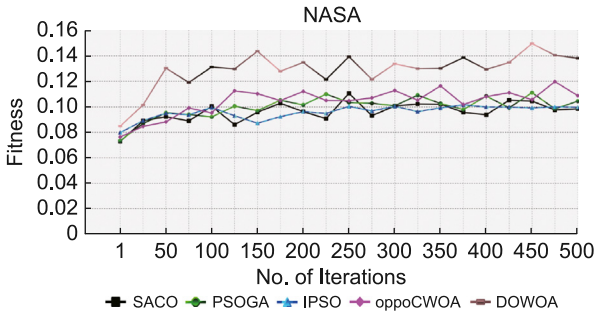

Fig. 9. NASA convergence graph for 300 instructions.

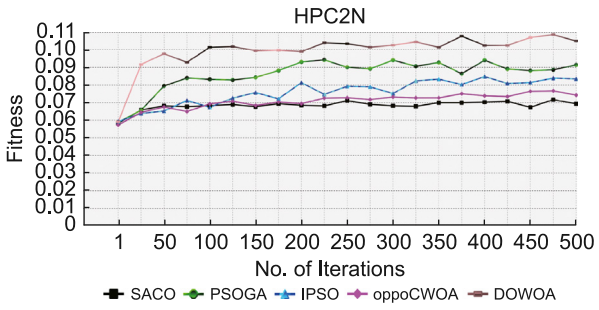

Fig. 10. HPC2N convergence graph for 300 instructions.

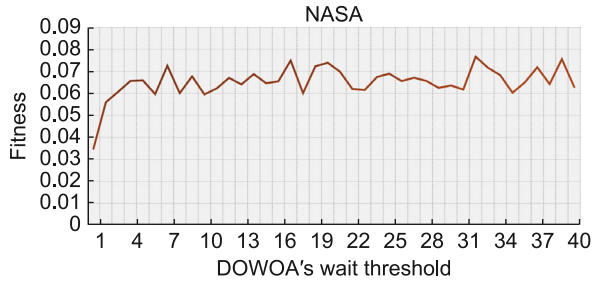

Fig. 11. Performance of DOWOA for various values of wait threshold.

NASA and HPC2N datasets, respectively. On the NASA dataset, the minimum difference in fitness value among all algorithms occurs at 75 and 275 iterations. Similarly, the variation in fitness value is minimal at 75 iterations on the HPC2N. It is observed that all algorithms provide near-optimal performance until 250 iterations, with minor improvements onwards. It is evident that DOWOA provides the highest fitness values on both datasets. The different convergence patterns by all algorithms are primarily attributed to the different nature of tasks in the datasets, as given in Table 3.

Figs. 11 and 12 show the performance variation of DOWOA with an increasing weight threshold on both datasets. As the fitness figures on both workloads vary due to the different characteristics of tasks, the wait threshold between 5 and 12 provides reasonably optimal performance. After the wait threshold of 10, the performance fluctuates, particularly on HPC2N.

For combinatorial optimization problems, the wait threshold of 10 provides optimal results. However, for other optimization problems, this value might require fine-tuning. The number of times the dynamic partial-OBL executes depends on the state of individual agents that varies for every agent because the population is initialized with OBL and quasi-OBL for enhanced diversity. For some agents, the partial-OBL may be triggered a few times, while for others, the same routine will be executing many times.

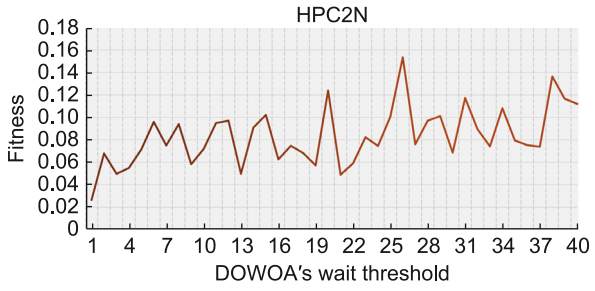

Fig. 12. Performance of DOWOA for various values of wait threshold.

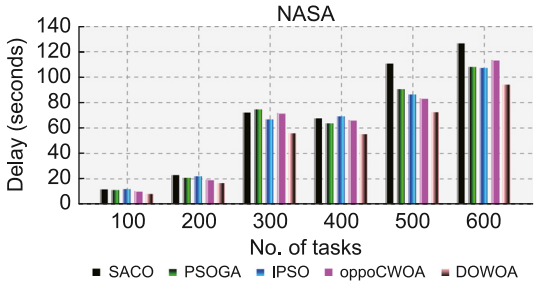

Fig. 13. Delay for various number of tasks.

9.1. Comparative analysis of DOWOA with other algorithms

Delay: It is one of the primary concerns in task offloading especially for time-sensitive applications [52]. The end-to-end delay comprises execution delay, queuing delay, and communication delay. The execution delay and queuing delay are longer in local execution, while they are the least in remote execution, particularly in the cloud. The queuing delays are longer for resource-constrained nodes compared to nodes with higher computation capability [19]. Therefore, there needs to be an optimal offloading destination for every task to minimize end-to-end delay.

In Fig. 13, the proposed DOWOA demonstrates minimum end-to-end delay across all the NASA instances. The next promising figures for average minimum delay are shown by IPSO, PSOGA, and oppoCWOA for different instances. IPSO capitalizes on its better performance with the help of an adaptive weighting strategy, while PSOGA maintains three populations simultaneously. The partial-OBL strategy in oppoCWOA is triggered only when the random probability of a jumping rate is less than 0.7, unlike the triggering mechanism employed in DOWOA, which tracks the status of every agent. The information-based triggering only executes the dynamic partial-OBL if an agent's fitness does not improve before the end of a waiting threshold. In other words, when the gradual changes employed by DOWOA fail to improve an agent's fitness, the dynamic partial-opposite routine is executed to break the stagnation.

Like the results on the NASA instances, the delay minimization on the HPC2N follows a similar pattern, as displayed in Fig. 14. Despite the similar trend, there is a considerable increase in delays due to the computation-intensive nature of HPC2N tasks. Among all the techniques, the delay optimization of oppoCWOA is affected the most compared to the NASA instances. SACO displays inferior optimization characteristics on both workloads.

Considering the performance of the proposed DOWOA in minimizing delay, the algorithm shows progressive improvements as the number of tasks increases, as illustrated in Figs. 13 and 14, demonstrating better scalability.

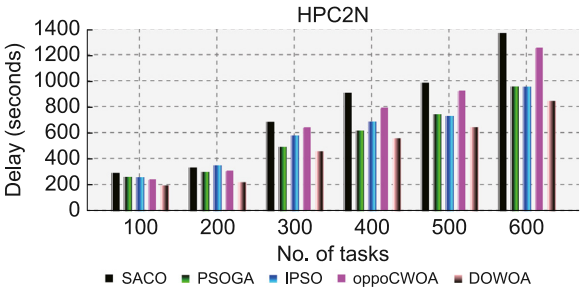

Fig. 14. Delay for various number of tasks.

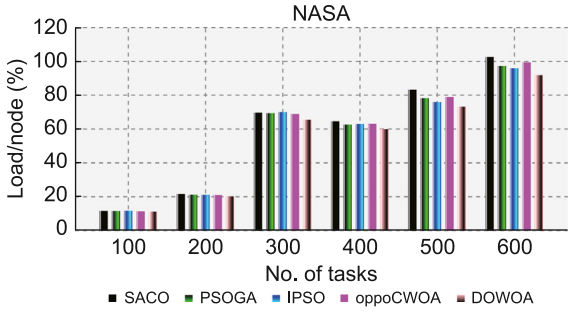

Fig. 15. Load distribution for various number of tasks.

Load balancing: Achieving equal load on all resources over a network is decisive for various performance improvements, such as lowering costs, higher resource utilization, and better system efficiency. On the NASA workload, the average load balancing rate for 100 instructions is 11.1% for DOWOA in comparison to 11.3% exhibited by oppoCWOA as depicted in Fig. 15. However, with the increasing number of tasks, DOWOA provides minimum average load per node, followed by IPSO, among all algorithms.

On HPC2N, DOWOA also provides considerably lower values for average load of tasks on each node for various instances, as illustrated in Fig. 16. When some nodes receive more offloaded tasks than others, the average load balancing numbers surge. Better load balancing is only achieved if tasks are offloaded onto all machines according to their computation capability, rather than solely on the number of tasks. Since all the nodes in our simulation setup are heterogeneous, DOWOA minimizes end-to-end delay by placing a commensurate load on each node according to its computational capacity, which translates into lower queuing delays. The resource utilization of heterogeneous fog nodes, based on their capacity, prevents both underloading and overloading. It is observed that with only 100 instruction on both the NASA and HPC2N instances, there is a nominal difference among the load balancing rates of algorithms. However, this difference becomes more pronounced as the number of tasks increases.

In fog-cloud computing, efficient load balancing facilitates dynamic scaling and seamless expansion of resources. The need for scaling may arise due to an increase in demand for resources or the provisioning of additional resources as part of fault-tolerance procedures. Similarly, improved load balancing reduces the likelihood of system imbalance when resources are shrinking. Furthermore, optimum load balancing in a fog-cloud framework with multiple clusters [2,11,23,26] of fog devices enables seamless infrastructure expansion. Energy efficiency is also a critical consideration for fog nodes with limited resources. However, the scope of this study does not take energy consumption into consideration.

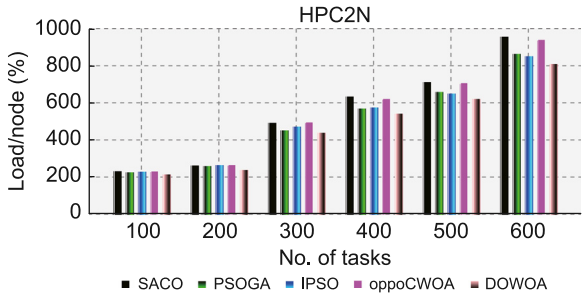

Fig. 16. Load distribution for various number of tasks.

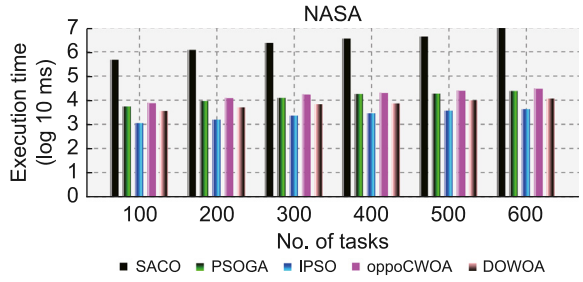

Fig. 17. Execution time for various number of tasks.

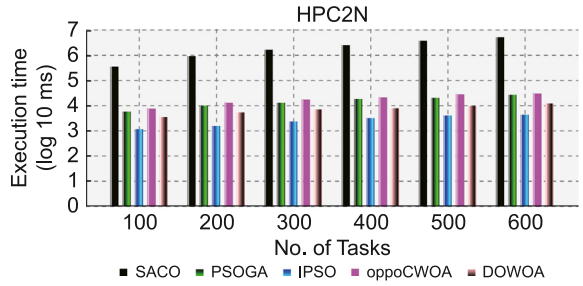

Fig. 18. Execution time for various number of tasks.

Execution time: Due to the significant difference in execution times between SACO and other algorithms, a logarithmic scale is used on the *y*-axis in both Figs. 17 and 18.

Although there is variation in the running times of the algorithms on the NASA and HPC2N workloads, the logarithm scale on the *y*-axis makes this difference less apparent.

SACO has the highest execution time due to the complexity of local pheromones updates. The local pheromones for all tasks are updated in every iteration, leading to prolonged running time. PSOGA managing three different populations requires a larger number of computations, leading to extended running time. The hybridization strategy in PSOGA is effective in providing better-quality solutions. However, it is still inferior to DOWOA. Our proposed enhancements can be applied to any meta-heuristic algorithm. However, they extend the execution time. IPSO shows superior running time as it uses an adaptive weight that does not significantly increase the execution time of the PSO, as shown in Figs. 17 and 18. DOWOA achieves reduced running time compared to oppoCWOA due to the limited computation of partial-opposite solutions.

In oppoCWOA, the random execution of partial-OBL leads to reduced solution diversity and premature convergence, degrading solution quality and increasing execution time. Furthermore, the maintenance of multiple populations and repeated fitness computations in PSOGA and oppoCWOA also contribute to their consistently longer execution times.

Table 6
Results on the NASA instances.

Instance	Algorithm	Fitness	Delay	Load/node	Execution time
100	SACO	0.10	11.10	11.52	452 998.92
	PSOGA	0.10	10.73	11.50	5230.68
	IPSO	0.10	11.38	11.62	**1070.24**
	oppoCWOA	0.11	10.00	11.37	7450.64
	DOWOA	**0.16**	**7.61**	**11.08**	3385.16
200	SACO	0.09	22.51	21.54	1151025.72
	PSOGA	0.10	20.46	21.07	8914.80
	IPSO	0.09	21.44	21.04	**1485.00**
	oppoCWOA	0.11	18.85	20.97	11 967.36
	DOWOA	**0.13**	**16.22**	**20.09**	4759.20
300	SACO	0.09	71.49	69.50	2271392.64
	PSOGA	0.09	74.15	69.34	11 884.84
	IPSO	0.10	66.27	69.87	**2178.76**
	oppoCWOA	0.10	71.23	68.94	16 802.28
	DOWOA	**0.12**	**55.23**	**65.39**	6615.60
400	SACO	0.09	67.15	64.52	3441557.28
	PSOGA	0.10	63.16	62.51	17 247.32
	IPSO	0.09	68.57	62.90	**2770.84**
	oppoCWOA	0.09	65.74	63.11	19 823.96
	DOWOA	**0.11**	**54.50**	**59.75**	7076.44
500	SACO	0.06	110.12	83.18	4084533.40
	PSOGA	0.08	90.06	78.14	17 901.04
	IPSO	0.09	85.80	75.89	**3509.96**
	oppoCWOA	0.09	82.86	79.03	24 158.24
	DOWOA	**0.10**	**71.80**	**73.13**	9599.68
600	SACO	0.07	125.92	102.48	9473798.48
	PSOGA	0.09	107.55	97.11	22 900.80
	IPSO	0.09	106.71	95.86	**4106.08**
	oppoCWOA	0.08	112.98	99.48	29 757.84
	DOWOA	**0.10**	**93.58**	**91.77**	11 163.88

Table 7
Results on the HPC2N instances.

Instance	Algorithm	Fitness	Delay	Load/node	Execution time
100	SACO	0.07	293.77	235.84	340 865.00
	PSOGA	0.08	263.10	229.42	5576.64
	IPSO	0.08	260.51	233.03	**1148.24**
	oppoCWOA	0.09	240.50	230.54	7736.00
	DOWOA	**0.11**	**197.58**	**217.02**	3475.12
200	SACO	0.07	336.21	265.45	898 467.52
	PSOGA	0.08	301.72	262.29	9812.08
	IPSO	0.07	351.48	267.73	**1525.12**
	oppoCWOA	0.08	307.15	264.21	13 023.08
	DOWOA	**0.11**	**222.99**	**241.11**	5208.16
300	SACO	0.06	688.25	495.96	1595804.92
	PSOGA	0.09	494.91	455.57	12 568.76
	IPSO	0.08	582.17	475.96	**2307.28**
	oppoCWOA	0.07	640.43	494.62	17 272.24
	DOWOA	**0.10**	**460.85**	**441.66**	6934.08
400	SACO	0.06	912.16	638.46	2432577.48
	PSOGA	0.09	620.81	572.06	17 808.24
	IPSO	0.08	690.61	578.28	**3116.00**
	oppoCWOA	0.07	790.98	620.41	21 435.72
	DOWOA	**0.10**	**561.42**	**545.99**	7680.88
500	SACO	0.06	988.36	715.52	3656618.76
	PSOGA	0.08	745.75	661.72	19 903.56
	IPSO	0.09	732.00	653.94	**3895.68**
	oppoCWOA	0.07	922.34	705.58	28 125.08
	DOWOA	**0.10**	**645.82**	**624.26**	9544.80
600	SACO	0.06	1371.09	960.10	5028566.24
	PSOGA	0.09	959.48	867.09	26 044.08
	IPSO	0.08	957.82	855.39	**4227.48**
	oppoCWOA	0.06	1254.77	939.60	30 420.00
	DOWOA	**0.10**	**849.13**	**813.69**	11 713.88

If a workload consists of delay-sensitive tasks, DOWOA is the best algorithm for offloading such workloads, as it minimizes delay while maintaining load balancing under various load conditions. However, if limited resources and the running time of task offloading algorithm are bottlenecks in addition to delay-insensitive tasks, where meeting deadlines is not stringent, IPSO appears to be a better choice. Nonetheless, DOWOA provides superior solution quality while requiring fewer iterations, as depicted in Figs. 9 and 10. Both DOWOA and IPSO demonstrate better scalability to minimize delay and reduce execution time, respectively. The proposed DOWOA enhances system performance by tracking each agent's status in the search space rather than relying solely on traditional random operations.

Tables 6 and 7 detail the average values of all algorithms for the NASA and HPC2N workloads, respectively. These values are calculated by running each algorithm 25 times.

9.2. Statistical assessment

The standard deviation, Friedman test, and Wilcoxon signed-rank test are conducted to illustrate the statistical significance of the obtained results.

Standard deviation represents the variation in observed values, showing the range of spread spectrum. In other words, it shows the consistency in observations. Tables 8 and 9 provide the figures of standard deviation for various metrics on the NASA and HPC2N instances, respectively.

In heterogeneous fog environments residing between cloud and edge devices, it is a challenge to make efficient use of each node having limited resources. In the simulation results, DOWOA shows the most consistent figures (exhibiting minimum standard deviation) for both delay and load balancing, illustrating better reliability. On the other hand, IPSO and PSOGA exhibit the most consistent fitness figures on the NASA and HPC2N workloads, respectively.

Table 8
Values of standard deviation on NASA workload.

Metric	SACO	PSOGA	IPSO	oppoCWOA	DOWOA
Fitness	0.013	0.008	**0.006**	0.012	0.021
Delay	45.712	38.365	36.908	39.193	**32.742**
Load/node	35.421	33.464	32.910	34.205	**31.453**

Table 9
Values of standard deviation on HPC2N workload.

Metric	SACO	PSOGA	IPSO	oppoCWOA	DOWOA
Fitness	0.005	**0.003**	0.007	0.009	0.008
Delay	412.480	267.120	257.254	383.190	**251.400**
Load/node	277.838	243.887	237.164	271.075	**229.873**

The Friedman test is a non-parametric test used to detect any difference between the means of three or more groups of observations [53]. It involves the relative and average ranking of observations to calculate the test statistic, also called chi-square. The chi-square values of different parameters are provided in Table 10. Our proposed DOWOA received the first rank in all observations of fitness, delay, and average load/node on both the workload datasets. If the test statistic's value is greater than the critical value at a particular degrees of freedom (df), the null hypothesis is rejected, showing a significant difference among the means of three or more groups of data. As all the chi-square values are greater than the critical values at 4 df, there is a significant difference among the means of observations for SACO, PSOGA, IPSO, oppoCWOA, and DOWOA.

Another useful non-parametric test is the Wilcoxon signed-rank test [54]. It compares two pairs of observations through the Wilcoxon statistic. First, the difference between observations is computed, followed by ranking the corresponding absolute values. These absolute differences are ranked and relevant signs (positive or negative) are assigned. The positive and negative ranks are separately added and the minimum value is selected as the Wilcoxon statistic. If the test statistic is less than or equal

Table 10
Values of Friedman test statistic.

Metric	NASA	HPC2N
Fitness	**16.13**	**18.93**
Delay	**16.13**	**18.53**
Load/node	**16.53**	**19.86**

Critical values at 4 df: 7.6 for $\alpha = 5\%$, 10.2 for $\alpha = 1\%$.

Table 11
Values of Wilcoxon test statistic.

Metric	SACO-DOWOA	PSOGA-DOWOA	IPSO-DOWOA	oppoCWOA-DOWOA
Fitness	**0**	**0**	**0**	**0**
Delay	**0**	**0**	**0**	**0**
Load/node	**0**	**0**	**0**	**0**

Critical value: 0 for $n = 6$.

to a critical value, the null hypothesis is rejected; otherwise, the alternative hypothesis is rejected. Table 11 provides the values of the Wilcoxon statistic for DOWOA against all other algorithms. All values of the test statistic are equal to the critical value for six data points. Hence, the null hypothesis is rejected in favor of the alternative hypothesis. It indicates that the median difference between each pair of algorithms is not zero. Hence, there is a significant difference between each pair of algorithms.

10. Conclusion

The emergence of fog computing has brought several advancements to the cloud computing paradigm, primarily attributed to service provisioning near the network edge. Nevertheless, the vulnerability of fog computing, resulting from the limited number and capacity of resources for offloaded tasks needs further investigation. To effectively utilize the heterogeneous resources in a three-layer infrastructure with multiple fog clusters, this study proposes the dynamic OBL-driven Whale Optimization Algorithm (DOWOA) to offload independent tasks. Opposition-Based Learning (OBL) is computationally expensive due to the maintenance of multiple populations and a random probability-based triggering mechanism. Moreover, OBL-based WOAs often undermine its strength due to the improper use of OBL strategies. In response to these challenges, first the proposed DOWOA uses basic OBL and quasi-OBL during population initialization. Then, the proposed dynamic partial-opposite routine is executed using an information-based triggering mechanism, enhancing the exploration potential of the Whale Optimization Algorithm (WOA). Along with improving solution diversity by the dynamic routine, it also reduces the execution time by more than 50% compared to the widely used jumping rate-based WOAs. The DOWOA provides significant performance improvements to minimize end-to-end delay and improve load balancing compared to SACO, PSOGA, IPSO, and oppoCWOA. In future, we aim to test DOWOA in a real time fog testbed (EmuFog) with dynamic task arrivals to further assess its reliability in terms of fault-tolerance and energy efficiency optimization.

CRediT authorship contribution statement

Zulfiqar Ali Khan: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Izzatdin Abdul Aziz:** Supervision, Resources, Project administration, Funding acquisition.

Declaration of competing interest

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

Acknowledgment

This research work is supported and funded by 'Data Analytics and Visualization Development System for Subsurface Co2 Storage and Fluid Production', Cost centre (015MD0-166), under the Center for research in Data Science (CerDaS), Universiti Teknologi PETRONAS, Malaysia.

References

[1] L. Sun, G. Xue, R. Yu, TAFS: A truthful auction for IoT application offloading in fog computing networks, IEEE Internet Things J. 10 (4) (2022) 3252–3263.
[2] S.K. Srichandan, S.K. Majhi, S. Jena, K. Mishra, R. Bhat, A secure and distributed placement for quality of service-aware IoT requests in fog-cloud of things: A novel joint algorithmic approach, IEEE Access (2024).
[3] A. Najafizadeh, A. Salajegheh, A.M. Rahmani, A. Sahafi, Multi-objective task scheduling in cloud-fog computing using goal programming approach, Clust. Comput. 25 (1) (2022) 141–165.
[4] L.T. Oliveira, L.F. Bittencourt, T.A. Genez, E. de Lara, M.L. Peixoto, Enhancing modular application placement in a hierarchical fog computing: A latency and communication cost-sensitive approach, Comput. Commun. 216 (2024) 95–111.
[5] M.K. Hussein, M.H. Mousa, Efficient task offloading for IoT-based applications in fog computing using ant colony optimization, IEEE Access 8 (2020) 37191–37201.
[6] Z.A. Khan, I.A. Aziz, N.A.B. Osman, et al., A review on task scheduling techniques in cloud and fog computing: Taxonomy, tools, open issues, challenges, and future directions, IEEE Access (2023).
[7] Z.A. Khan, I.A. Aziz, N.A.B. Osman, A review on task offloading using meta-heuristic algorithms on fog computing, in: 2023 IEEE 21st Student Conference on Research and Development (SCOReD), IEEE, 2023, pp. 469–475.
[8] A. Kishor, C. Chakarbarty, Task offloading in fog computing for using smart ant colony optimization, Wirel. Pers. Commun. 127 (2) (2022) 1683–1704.
[9] O.-K. Shahryari, H. Pedram, V. Khajehvand, M.D. TakhtFooladi, Energy and task completion time trade-off for task offloading in fog-enabled IoT networks, Pervasive Mob. Comput. 74 (2021) 101395.
[10] J. He, W. Bai, Computation offloading and task scheduling based on improved integer particle swarm optimization in fog computing, in: 2023 3rd International Conference on Neural Networks, Information and Communication Engineering, NNICE, IEEE, 2023, pp. 633–638.
[11] Z. Movahedi, B. Defude, A.M. Hosseininia, An efficient population-based multi-objective task scheduling approach in fog computing systems, J. Cloud Comput. 10 (1) (2021) 53.
[12] X. Zhao, C. Huang, Microservice based computational offloading framework and cost efficient task scheduling algorithm in heterogeneous fog cloud network, IEEE Access 8 (2020) 56680–56694.
[13] M. Mukherjee, S. Kumar, C.X. Mavromoustakis, G. Mastorakis, R. Matam, V. Kumar, Q. Zhang, Latency-driven parallel task data offloading in fog computing networks for industrial applications, IEEE Trans. Ind. Informatics 16 (9) (2019) 6050–6058.
[14] X. Li, Z. Zang, F. Shen, Y. Sun, Task offloading scheme based on improved contract net protocol and beetle antennae search algorithm in fog computing networks, Mob. Networks Appl. 25 (6) (2020) 2517–2526.
[15] F. Sufyan, A. Banerjee, Computation offloading for smart devices in fog-cloud queuing system, IETE J. Res. 69 (3) (2023) 1509–1521.
[16] L.-A. Phan, D.-T. Nguyen, M. Lee, D.-H. Park, T. Kim, Dynamic fog-to-fog offloading in SDN-based fog computing systems, Future Gener. Comput. Syst. 117 (2021) 486–497.
[17] H. Mahini, A.M. Rahmani, S.M. Mousavirad, An evolutionary game approach to IoT task offloading in fog-cloud computing, J. Supercomput. 77 (2021) 5398–5425.
[18] L. Zhang, Y. Zou, W. Wang, Z. Jin, Y. Su, H. Chen, Resource allocation and trust computing for blockchain-enabled edge computing system, Comput. Secur. 105 (2021) 102249.
[19] P. Singh, R. Singh, Energy-efficient delay-aware task offloading in fog-cloud computing system for IoT sensor applications, J. Netw. Syst. Manage. 30 (1) (2022) 14.
[20] D. Liu, F. Sun, W. Wang, K. Dev, Distributed computation offloading with low latency for artificial intelligence in vehicular networking, IEEE Commun. Stand. Mag. 7 (1) (2023) 74–80.

[21] C. Lv, F. Shen, F. Yan, L. Cao, C. Wang, Y. Zhang, Task offloading for fog-based meta networks: An energy and delay aware mechanism, in: 2023 IEEE International Conference on Metaverse Computing, Networking and Applications (MetaCom), IEEE, 2023, pp. 370–377.
[22] B. Mikavica, A. Kostic-Ljubisavljevic, A truthful double auction framework for security-driven and deadline-aware task offloading in fog-cloud environment, Comput. Commun. 217 (2024) 183–199.
[23] A. Mahapatra, S.K. Majhi, K. Mishra, R. Pradhan, D.C. Rao, S.K. Panda, An energy-aware task offloading and load balancing for latency-sensitive IoT applications in the fog-cloud continuum, IEEE Access (2024).
[24] B. Premalatha, P. Prakasam, Optimal energy-efficient resource allocation and fault tolerance scheme for task offloading in IoT-FoG computing networks, Comput. Netw. 238 (2024) 110080.
[25] Z. Lejun, P. Minghui, S. Shen, W. Weizheng, J. Zilong, S. Yansen, C. Huiling, G. Ran, S. Gataullin, Redundant data detection and deletion to meet privacy protection requirements in blockchain-based edge computing environment, China Commun. 21 (3) (2024) 149–159.
[26] C. Chakraborty, K. Mishra, S.K. Majhi, H.K. Bhuyan, Intelligent latency-aware tasks prioritization and offloading strategy in distributed Fog-Cloud of Things, IEEE Trans. Ind. Informatics 19 (2) (2022) 2099–2106.
[27] S. Mirjalili, A. Lewis, The whale optimization algorithm, Adv. Eng. Softw. 95 (2016) 51–67.
[28] H.R. Tizhoosh, Opposition-based learning: a new scheme for machine intelligence, in: International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06), 1, IEEE, 2005, pp. 695–701.
[29] H.R. Tizhoosh, M. Ventresca, S. Rahnamayan, Opposition-based computing, Oppositional Concepts Comput. Intell. (2008) 11–28.
[30] M. Kumar, A. Chaparala, OBC-WOA: opposition-based chaotic whale optimization algorithm for energy efficient clustering in wireless sensor network, Intelligence 250 (1) (2019).
[31] H. Chen, W. Li, X. Yang, A whale optimization algorithm with chaos mechanism based on quasi-opposition for global optimization problems, Expert Syst. Appl. 158 (2020) 113612.
[32] X. Shi, Z. Yin, L. Wang, H. Liang, Z. Wang, Solar cell parameter identification based on opposition-based chaotic whale optimization algorithm, in: 2022 IEEE 5th International Electrical and Energy Conference, CIEEC, IEEE, 2022, pp. 500–505.
[33] S. Mukherjee, P.K. Roy, Chaotic-opposition whale optimization algorithm based load flow analysis of small-scale, median and broad critical power systems, in: 2022 IEEE International Power and Renewable Energy Conference, IPRECON, IEEE, 2022, pp. 1–6.
[34] M. Li, G. Xu, Q. Lai, J. Chen, A chaotic strategy-based quadratic opposition-based learning adaptive variable-speed whale optimization algorithm, Math. Comput. Simulation 193 (2022) 71–99.
[35] C. Paul, P.K. Roy, V. Mukherjee, Chaotic-quasi-opposition based whale optimization technique applied to multi-objective complementary scheduling of grid connected hydro-thermal–wind–solar-electric vehicle system, Optim. Control. Appl. Methods (2024).
[36] S. Mukherjee, P.K. Roy, Load flow solution for radial distribution networks using chaotic opposition based whale optimization algorithm, in: International Conference on Computational Intelligence in Communications and Business Analytics, Springer, 2023, pp. 78–92.
[37] Y. Li, W.-g. Li, Y.-t. Zhao, A. Liu, Opposition-based multi-objective whale optimization algorithm with multi-leader guiding, Soft Comput. 25 (24) (2021) 15131–15161.
[38] S. Dey, P.K. Roy, A. Sarkar, Adaptive IIR model identification using chaotic opposition-based whale optimization algorithm, J. Electr. Syst. Inf. Technol. 10 (1) (2023) 33.
[39] B.C. Rao, S. Kumhar, Design and performance analysis of opposition based whale optimization algorithm tuned power system stabilizer for multimachine stability, I- Manager' s J. Power Syst. Eng. 10 (4) (2023) 15.
[40] H.S. Alamri, K.Z. Zamli, M.F.A. Razak, A. Firdaus, Solving 0/1 knapsack problem using opposition-based whale optimization algorithm (OWOA), in: Proceedings of the 2019 8th International Conference on Software and Computer Applications, 2019, pp. 135–139.
[41] Z. Qiang, F. Qiaoping, H. Xingjun, L. Jun, Parameter estimation of muskingum model based on whale optimization algorithm with elite opposition-based learning, in: IOP Conference Series: Materials Science and Engineering, 780, (2) IOP Publishing, 2020, 022013.
[42] M. Raja, S. Dhanasekaran, V. Vasudevan, Opposition based joint grey wolf-whale optimization algorithm based attribute based encryption in secure wireless communication, Wirel. Pers. Commun. (2022) 1–21.
[43] M. Wu, D. Yang, T. Liu, Whale optimization algorithm with opposition-based learning strategy for solving flexible job shop scheduling problem, in: ITM Web of Conferences, vol. 45, EDP Sciences, 2022, p. 01033.
[44] Y. Lu, C. Yi, J. Li, W. Li, An enhanced opposition-based golden-Sine whale optimization algorithm, in: International Conference on Cognitive Computing, Springer, 2023, pp. 60–74.
[45] D. Cao, Y. Xu, Z. Yang, H. Dong, X. Li, An enhanced whale optimization algorithm with improved dynamic opposite learning and adaptive inertia weight strategy, Complex & Intell. Syst. 9 (1) (2023) 767–795.
[46] W.L. Wang, W.K. Li, Z. Wang, L. Li, Opposition-based multi-objective whale optimization algorithm with global grid ranking, Neurocomputing 341 (2019) 41–59.
[47] X. Liang, Z. Zhang, A whale optimization algorithm with convergence and exploitability enhancement and its application, Math. Probl. Eng. 2022 (1) (2022) 2904625.
[48] C. Paul, P.K. Roy, V. Mukherjee, Optimal solution for hydro–thermal–wind–solar scheduling using opposition-based whale optimization algorithm, Soft Comput. 28 (7) (2024) 6003–6037.
[49] Z.A. Khan, I.A. Aziz, Ripple-induced whale optimization algorithm for independent tasks scheduling on fog computing, IEEE Access (2024).
[50] H. Gupta, A. Vahid Dastjerdi, S.K. Ghosh, R. Buyya, IFogSim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments, Software: Pr. Exp. 47 (9) (2017) 1275–1296.
[51] T.D. Braun, H.J. Siegel, N. Beck, L.L. Bölöni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, et al., A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel Distrib. Comput. 61 (6) (2001) 810–837.
[52] J. Lin, H. Rao, S. Liang, Y. Zhao, Q. Ren, G. Jia, Aphto: a task offloading strategy for autonomous driving under mobile edge, J. Supercomput. (2024) 1–33.
[53] A. Demircioglu, The effect of data resampling methods in radiomics, Sci. Rep. 2024 14 (1) (2024) 1–11, http://dx.doi.org/10.1038/s41598-024-53491-5.
[54] M.D. Riina, C. Stambaugh, N. Stambaugh, K.E. Huber, Continuous variable analyses: t-test, Mann–Whitney, Wilcoxin rank, in: Translational Radiation Oncology, Elsevier, 2023, pp. 153–163.