biomimetics

MDPI

Article

Dynamic Random Walk and Dynamic Opposition Learning for Improving Aquila Optimizer: Solving Constrained Engineering Design Problems

Megha Varshney [1], Pravesh Kumar [1], Musrrat Ali [2,*] and Yonis Gulzar [3]

[1] Rajkiya Engineering College (AKTU, Lucknow), Bijnor 246725, India; megha.math21@recb.ac.in (M.V.)
[2] Department of Basic Sciences, Preparatory Year, King Faisal University, Al Ahsa 31982, Saudi Arabia
[3] Department of Management Information Systems, College of Business Administration, King Faisal University, Al Ahsa 31982, Saudi Arabia; ygulzar@kfu.edu.sa
* Correspondence: mkasim@kfu.edu.sa

Abstract: One of the most important tasks in handling real-world global optimization problems is to achieve a balance between exploration and exploitation in any nature-inspired optimization method. As a result, the search agents of an algorithm constantly strive to investigate the unexplored regions of a search space. Aquila Optimizer (AO) is a recent addition to the field of metaheuristics that finds the solution to an optimization problem using the hunting behavior of Aquila. However, in some cases, AO skips the true solutions and is trapped at sub-optimal solutions. These problems lead to premature convergence (stagnation), which is harmful in determining the global optima. Therefore, to solve the above-mentioned problem, the present study aims to establish comparatively better synergy between exploration and exploitation and to escape from local stagnation in AO. In this direction, firstly, the exploration ability of AO is improved by integrating Dynamic Random Walk (DRW), and, secondly, the balance between exploration and exploitation is maintained through Dynamic Oppositional Learning (DOL). Due to its dynamic search space and low complexity, the DOL-inspired DRW technique is more computationally efficient and has higher exploration potential for convergence to the best optimum. This allows the algorithm to be improved even further and prevents premature convergence. The proposed algorithm is named DAO. A well-known set of CEC2017 and CEC2019 benchmark functions as well as three engineering problems are used for the performance evaluation. The superior ability of the proposed DAO is demonstrated by the examination of the numerical data produced and its comparison with existing metaheuristic algorithms.

Keywords: aquila optimizer; dynamic random walk; dynamic opposite learning; engineering design problems

Citation: Varshney, M.; Kumar, P.; Ali, M.; Gulzar, Y. Dynamic Random Walk and Dynamic Opposition Learning for Improving Aquila Optimizer: Solving Constrained Engineering Design Problems. *Biomimetics* **2024**, *9*, 215. https://doi.org/10.3390/biomimetics9040215

Academic Editors: Ameer Hamza Khan, Shuai Li and Danish Hussain

Received: 28 February 2024
Revised: 27 March 2024
Accepted: 2 April 2024
Published: 4 April 2024

1. Introduction

Global optimization is a term used to characterize several scientific and engineering problems that can be resolved using different optimization techniques. These days, the preferred methods for global optimization are metaheuristic algorithms (MAs) since they are protected against local maximum efficacy by their stochastic and dynamic nature [1]. Genetic Evolution [2], Differential Evolution (DE) [3], Particle Swarm Optimization (PSO) [4], Reptile Search Algorithm (RSA) [5], Whale Optimization Algorithm (WOA) [6], Brain Storm Optimization (BSO) [7], Teaching–Learning-Based Optimization (TLBO) [8], etc., are several MAs that have emerged over the past 20 years. One of the better algorithms is the AO method, which Abualigah proposed in 2021 [9], because it is simple to build, has consistent performance, and few configurable parameters. Its strong optimization capabilities have helped with a variety of global optimization problems, including feature selection [10], vehicle route planning [11], and machine scheduling [12].

The No Free Lunch (NFL) theorem [13] was a significant advancement in the field of nature-inspired algorithms. It is impossible to develop a single optimization algorithm that solves every optimization problem, according to the NFL theorem. To put it simply, even if optimization method "A" is ideally suited for a particular set of problems, there is always a subset of problems on which it would perform poorly. As a result, the NFL theorem provides the area of nature-inspired algorithms life and enables academics to either suggest new algorithms or enhance already existing ones. In order to improve existing algorithms, an effective approach for doing so is hybridization—combining the best aspects of multiple algorithms to create a hybridized algorithm. The present study aims to combine the benefits of better exploration and the efficiency of maintaining a balance between exploration and exploitation by improving the AO, DOL, and DRW techniques. This is achieved by drawing inspiration from the advantages of improving the algorithm.

The new NIOA, called Aquila Optimizer, uses the Aquila bird's hunting strategy in an attempt to discover the best solution to an optimization problem. AO is capable of handling a broad range of optimization problems [14]. The first drawback of this algorithm is premature convergence, which happens when the algorithm has a stagnation issue and is unable to explore the whole search space during the process. The second drawback is its low computational efficiency. This provides a poor ideal solution and also prevents the algorithm from searching the whole search space. Aquila Optimizer takes a longer time to converge and to reach the ideal solution than other existing metaheuristic algorithms. Therefore, in the current study, Aquila Optimizer is enhanced so that it can explore the more promising areas that are left in the population's memory. By combining AO with DRW and DOL, suitable harmony between the exploration and exploitation process is formed. The DOL [15] method with its asymmetric and dynamic search space exhibits a great deal of promise. In the meanwhile, the dynamic opposite number, a random candidate, can be computed quickly and easily. This may enhance the algorithm's capacity for exploitation and increase the rate of convergence. The DRW [16] approach focuses on iteratively improving a solution by exploring its closer neighborhood because balancing the search for new promising areas with refining solutions within existing areas is the key to metaheuristics. The following are the paper's contributions:

1. To increase the AO algorithm's computing effectiveness and capacity for local optimal avoidance, a new DRW technique is put forth.
2. To enhance the algorithm's performance and balance between exploration and exploitation, the DOL approach is incorporated into AO for the very first time.
3. The performance of DAO is examined on twenty-nine benchmark functions of CEC 2017, ten benchmark functions of CEC 2019, and then on three engineering design problems, and the results are compared with various algorithms.

The following part of the paper is structured as follows: the fundamental ideas of AO, DOL, and DRW are presented in Section 2. The previous work on AO is explained in Section 3. In Section 4, the proposed DAO algorithm is explained. Section 5 presents the experiments and their findings. Section 6 shows the engineering applications. The study's conclusion is finally presented in Section 7.

2. Algorithm Preliminaries

2.1. Aquila Optimizer

The Aquila bird's hunting strategy served as the inspiration for the Aquila Optimizer (AO) metaheuristic optimization technique [9]. AO mimics the four main prey-hunting strategies, explained as follows:

2.1.1. Expanded Exploration

The expanded exploration x_1 of Aquila Optimizer mimics the high-achieving quickly descending hunting strategy observed in Aquila birds. With this strategy, the bird soars to enormous heights, giving it the opportunity to inspect the whole search area, identify po-

tential prey, and select the ideal hunting place. Equation (1) in [9] provides a mathematical illustration of this strategy.

$$x_1^{(h+1)} = x_{best}^{(h)} \times \left(1 - \frac{h}{H}\right) + \left(x_M^{(h)} - rand \times x_{best}^{(h)}\right) \quad (1)$$

In Equation (1), the maximum number of iterations is represented as H where h denotes the current iteration. The response for the subsequent iteration indicated as $\left(x_1^{(h+1)}\right)$ is found by the first search in the candidate solution population (x_1). The expression $\left(x_{best}^{(h)}\right)$ represents the best outcome achieved so far in the h^{th} iteration. A count of iterations is employed through an equation $\left(1 - \frac{h}{H}\right)$ to modify the search space's depth. Additionally, using Equation (2), where N represents the population size and D is the dimension size, the average value of the locations of connected existing solutions at the h^{th} iteration is determined, represented as $x_M^{(h)}$.

$$x_M^{(h)} = \frac{1}{N}\sum_{i=1}^{N} x_i(h), \text{ for all } i = 1, 2, \ldots, D \quad (2)$$

2.1.2. Narrowed Exploration

In this approach, the Aquila bird hunts; to track prey, they must fly in a contour-like pattern and execute swift gliding strikes inside a small research region. The primary aim of this methodology $\left(x_2^{(h+1)}\right)$, as expressed mathematically in Equation (3), is to identify a solution for the subsequent iterations.

$$x_2^{(h+1)} = x_{best}^{(h)} \times Levy(D) + x_R^{(h)} + (v - u) \times rand \quad (3)$$

In this approach, $Levy(D)$ is the Levy flying distribution for dimension space D. At the h^{th} iteration, the random solution $\left(x_R^{(h)}\right)$ is taken in the range of $[1\ N]$, where N is the population size. The Levy flight distribution is calculated using a fixed constant value of $s = 0.01$ and two randomly selected parameters, u and v, which have values between 0 and 1. The mathematical expression for this computation is provided by Equation (4).

$$Levy(D) = s \times \frac{u \times \sigma}{|v|^{\frac{1}{a}}} \quad (4)$$

Equation (5) finds the value σ, which is obtained using the constant parameter $a = 1.5$.

$$\sigma = \left(\frac{\Gamma(1+a) \times \sin\left(\frac{\pi a}{2}\right)}{\Gamma\left(\frac{(1+a)}{2}\right) \times a \times 2^{\left(\frac{a-1}{2}\right)}}\right) \quad (5)$$

Equations (6) and (7) depict the spiral form inside the search range, denoted by y and x, respectively. Equation (3) specifies this spiral form.

$$y = r_1 + UD_1 \cos\left(-\omega D_1 + \left(\frac{3\pi}{2}\right)\right) \quad (6)$$

$$x = r_1 + UD_1 \sin\left(-\omega D_1 + \left(\frac{3\pi}{2}\right)\right) \quad (7)$$

Variable r_1, over a predefined number of search iterations, takes values between 1 and 20. The constant values of ω and U are 0.005 and 0.00565, respectively. $D_1 \in \mathbb{Z}$ has a range from 1 to the dimension D of the search space.

2.1.3. Expanded Exploitation

During the investigation phase, the Aquila bird meticulously examines the prey area before attacking with a low, slow fall. This strategy, sometimes referred to as expanded exploitation x_3, is represented mathematically in Equation (8).

$$x_3^{(h+1)} = \left(x_{best}^h - x_M^{(h)}\right) \times \theta - rand + ((ub - lb) \times rand + lb) \times \rho \quad (8)$$

$\left(x_3^{(h+1)}\right)$, the result of Equation (8), represents the result for the subsequent iteration. In the h^{th} iteration, $x_{best}(h)$ denotes the current best solution obtained, and $\left(x_M^{(h)}\right)$ denotes the average value of the current solution as determined by Equation (2). Variable "rand" is assigned a random number within the range of (0, 1), while tuning parameters θ and ρ are typically assigned values of 0.1 each. Symbols ub and lb represent the upper and lower bounds, respectively.

2.1.4. Narrowed Exploitation

Aquila birds hunt by taking advantage of their prey's unpredictable ground movement patterns to grab their prey directly. This hunting strategy serves as the basis for the constrained exploitation technique $\left(x_4^{(h)}\right)$ design, which is produced by Equation (9), which also yields the h^{th} iteration of the following solution, denoted as $\left(x_4^{(h+1)}\right)$. Equation (10), which expresses the quality function J, was put out to provide a well-balanced search approach.

$$x_4^{(h+1)} = J \times x_{best}^{(h)} - \left(P_1 \times rand \times x_1^{(h)}\right) - P_2 \times Levy(D) + rand \times P_1 \quad (9)$$

Equations (11) and (12) are used to determine the mobility pattern for the Aquila's prey tracking (P_1) and the trajectory of an attack during an escape, from the beginning to the terminal point (P_2). Both the maximum number of iterations H and the current iteration number h are used in the computations.

$$J(h) = h^{\frac{2 \times rand() - 1}{(1-H)^2}} \quad (10)$$

$$P_1 = 2 \times rand - 1 \quad (11)$$

$$P_2 = 2 \times \left(1 - \frac{h}{H}\right) \quad (12)$$

2.2. Concept of Dynamic Oppositional Learning (DOL)

The objectives of the optimization algorithms are to produce solutions, improve approximated solutions, and look for additional solutions inside the domain. The needs of tackling a complex problem cannot be met by the current solutions. Then, a variety of learning techniques are developed to improve optimization algorithms' performance. Owing to its higher convergence capacity, the opposition-based learning (OBL) technique is the most frequently acknowledged among these learning systems. The following is an introduction to the definition of OBL [15]:

OBL is made up of the real number $x \in \mathbb{R}$ in the interval $x \in [a, b]$. Furthermore, the opposite number, x^{OBL}, is produced.

$$x^{OBL} = a + b - x \quad (13)$$

Regarding a situation with several dimensions, the definition is demonstrated as follows: $x = (x_1, x_2, \ldots, x_D)$ is a point in D dimensional coordinates if and only if $x_1, x_2, \ldots, x_D \in \mathbb{R}$ in the interval $[a_i, b_i]$. As the iteration changes, the associated low

and high bounds of the population are denoted by a_i and b_i, respectively. In the meantime, the definition of the multidimensional opposite point is

$$x_i^{OBL} = a_i + b_i - x_i \tag{14}$$

Even while the OBL method enhances the algorithm's searching capabilities, it still has certain drawbacks, such being premature. Various variations of OBL have been proposed to enhance its performance. To expand the domain known as original notion of quasi-opposite-based learning (QOBL), for example, a quasi-opposite number is employed [17]. In the meantime, a quasi-reflection number is introduced in the interval between the present location and the center position in order to implement a quasi-reflection-based learning (QRBL) method [18].

Phase of Dynamic Opposite Learning: In addition to the OBL variations mentioned above, a novel learning approach called dynamic opposite learning operator (DOL) is used in this work. In order to enhance the TLBO algorithm's performance, Xu et al. originally suggested the DOL method in [15]. When dealing with complex issues, the DOL is included to prevent the algorithm from being too young [19]. Furthermore, in an asymmetric and dynamic search environment, the DOL learning technique is a new variation of the opposition-based learning (OBL) strategy that aids in population learning from the opposite points [20,21].

2.2.1. Dynamic Population Initialization

$x \in [a, b]$ was defined as the initial population in the initialization step. Additionally, x^{OBL} is produced in the opposing domain. $x^{RO}\left(x^{RO} = rand \cdot x^{OBL}, rand \in [0, 1]\right)$ is introduced to replace x^{OBL} in order to expand the searching space and convert the previous symmetric searching space into a dynamic asymmetric domain. The optimizer is then able to prevent prematurity by expanding the searching space. Therefore, in order to enhance the capacity to overcome local optima, a weighting factor w_d is incorporated. This is how the mathematical model is displayed:

$$x^{DOL} = x + w_d \cdot r_1 \cdot \left(r_2 \cdot x^{OBL} - x\right) \tag{15}$$

where $r_2 \in [0, 1]$ is a random parameter. When faced with a multifaceted goal, it manifests as follows:

$$x_{ij}^{DOL} = x_{ij} + w_d \cdot r_1 \cdot \left[r_2 \cdot x_{ij}^{OBL} - x_{ij}\right] \tag{16}$$

where $i = 1, 2, \ldots, N$ is the population size, $j = 1, 2, \ldots, D$ is the dimension of an individual, r_1 and r_2 denote random numbers among $[0, 1]$.

2.2.2. Dynamic Population Jumping Process

In DOL, a jumping rate (J_r) is used to update the population, and a positive weighting factor (w_d) is employed to balance the capabilities of exploration and exploitation. The following is an implementation of the DOL operation procedure, provided that the selection probability is less than J_r.

$$x_{ij}^{DOL} = x_{ij} + w_d \cdot r_1 \cdot \left[r_2 \cdot \left(a_j + b_j - x_{ij}\right) - x_{ij}\right] \tag{17}$$

where a random value x_{ij} is produced as the starting populace; N is the population size; i is the i^{th} solution; x_{ij}^{DOL} is the population created by the DOL technique; j displays the dimension of j^{th}; two random parameters in $[0, 1]$ are called r_1 and r_2; the weighting factor w_d is set to 3; and the jumping rate J_r is set at 1 by conducting sensitivity analysis as in Table 1.

Table 1. The sensitivity analysis of w and J_r.

w	Mean							
	F3		F6		F18		F23	
	J_r = 0.3	J_r = 1	J_r = 0.3	J_r = 1	J_r = 0.3	J_r = 1	J_r = 0.3	J_r = 1
1	6.582×10^3	6.859×10^3	4.251×10^1	4.839×10^1	2.387×10^5	$\mathbf{1.457 \times 10^5}$	4.256×10^2	4.340×10^2
2	6.502×10^3	5.187×10^3	4.216×10^1	3.776×10^1	3.041×10^5	1.567×10^5	4.101×10^2	3.865×10^2
3	6.750×10^3	$\mathbf{4.022 \times 10^3}$	4.469×10^1	$\mathbf{3.675 \times 10^1}$	2.224×10^6	6.025×10^5	3.982×10^2	$\mathbf{3.818 \times 10^2}$
4	8.326×10^3	5.279×10^3	4.476×10^1	3.885×10^1	1.045×10^6	6.649×10^5	4.082×10^2	3.868×10^2
5	8.613×10^3	4.975×10^3	4.468×10^1	3.908×10^1	4.604×10^6	1.106×10^6	4.109×10^2	3.888×10^2
6	9.230×10^3	5.749×10^3	4.886×10^1	4.031×10^1	3.238×10^6	2.014×10^6	4.166×10^2	4.025×10^2
7	9.451×10^3	7.962×10^3	5.108×10^1	4.931×10^1	5.852×10^6	5.806×10^6	4.192×10^2	4.218×10^2
8	1.031×10^4	8.989×10^3	5.228×10^1	4.922×10^1	3.966×10^6	1.155×10^7	4.223×10^2	4.357×10^2
9	9.738×10^3	1.003×10^4	5.263×10^1	5.246×10^1	1.043×10^7	1.255×10^7	4.181×10^2	4.472×10^2
10	1.087×10^4	1.108×10^4	5.446×10^1	5.205×10^1	2.989×10^7	3.067×10^7	4.311×10^2	4.574×10^2

Note: bold is used to indicate the best result.

2.3. Concept of Dynamic Random Walk (DRW)

Dynamic Random Walk (DRW) can be applied to the expanded exploration phase of the Aquila Optimizer metaheuristic algorithm to improve its exploration ability and help it escape local optima by the following equation:

$$x = x_{best} + w \cdot r_3 \cdot (r_4 \cdot rwv - x_{best}) \tag{18}$$

where rwv, random walk vector, is provided by $rwv = r(1, D) - 0.5$. Two random parameters in $[0, 1]$ are called r_3 and r_4. DRW is incorporated into AO to improve its exploration ability. In the early stages of the optimization process, DRW is used to allow the search agents to explore a large search space.

3. Previous Work on AO and DOL

There is always room to enhance an algorithm by increasing and balancing the operators' exploitation and exploration since the NFL theorem opposes the existence of an algorithm that is best suited for all optimization tasks. Plenty of work has been completed in the literature to improve the search efficiency in AO. These improvements include adjusting the algorithm's parameters, including new movement strategies, and merging the algorithm with other optimization methods. The improved versions of AO can handle a large range of difficult real-world optimization problems better than the standard AO. The strategies used in AO are hybridization with NIOAs [22,23], oppositional-based learning [24], chaotic sequence [25], Levy flight-based strategy [26], Gauss map and crisscross operator [27], Niche Thought with Dispersed Chaotic Swarm [28], random learning mechanism and Nelder–Mead Simplex Search [29], wavelet mutation [30], Weighted Adaptive Searching Technique [31], Binay AO [32], and multi-objective AO [33].

DOL strategies are also used in many NIOAs to enhance their performance. First, they were introduced with Teaching–Learning-based Optimization [15], Grey Wolf Optimizer [34], Whale Optimization Algorithm [35], Antlion Optimizer [16], Bald Eagle Search Optimization [36], in the hybrid version of Aquila Optimizer, and Artificial Rabbits Optimization Algorithm [37], and the comprehensive survey with other algorithms can be found in the literature [14].

4. The Proposed DAO Algorithm

Two new features, DOL and DRW, are added to the original AO by the proposed DAO (Dynamic Random Walk and Dynamic Opposition Learning for Improving Aquila Optimizer) algorithm. The aim of DOL population generation is to provide diverse solutions to escape from stagnation, and DOL generation jumping helps in the exploitation ability of the algorithm and accelerates the speed of the algorithm. On the other hand, DRW will help the algorithm to improve its exploration ability. This overall approach will provide the perfect balance between exploration and exploitation and help the algorithm to escape from local optima. Let us examine this improvement working in more detail.

1. **Benefits of using DOL population initialization**

Compared to random initialization, the use of a dynamic opposition population initialization technique in Aquila Optimizer (AO) has various benefits that result in a more diverse solution pool:

(a) Random initialization limitations: Particularly for complex problems, random initialization might produce a population localized in a particular area of the search space, which restricts exploration and raises the possibility of becoming trapped in local optima.
(b) Initialization Based on Dynamic Opposition: For every randomly selected initial point, this method produces an "opposite" solution. With respect to a predetermined reference point (often the centre or limits), the opposing solution is located on the other side of the search area. This forces investigation in many places and produces wider initial dispersion of solutions.

The starting population is more diversified when opposition-based generation and random selection are combined. Because of this diversity, AO is able to investigate various regions of the search field right away. To prevent becoming overly biased in favor of the opposing alternatives, the strategy, nevertheless, maintains a healthy balance by retaining some randomly generated solutions. Overall, we can say that introducing DOL population initialization can help AO in the following ways:

(a) Increased exploration: AO can find promising regions throughout the whole search space by distributing the first solutions more widely.
(b) Decreased chance of local optima: AO is less likely to become stuck in solutions that are only effective in a small area because it starts from a variety of sites.
(c) Faster convergence: When multiple regions are investigated concurrently, a well-distributed population can converge more quickly to the global optimum.

2. **Benefits of using DOL generation jumping:**
    (a) Improved Exploration: Reintroducing exploration in later phases may result in the identification of more effective solutions.
    (b) Escape from Local Optima: AO is nudged away from regions that would not lead to the global optimum by jumping in opposition to underperforming individuals.
    (c) Fine-tuning: By investigating neighboring regions in the opposite direction, the leaps may discover somewhat better choices even if AO converges to a suitable solution.

3. **Benefits of using DRW in place of Aquila's expanded exploration phase:**
    (a) Reduced Complexity: By doing away with the necessity to plan and carry out a specific extended exploration phase, DRW simplifies the algorithm as a whole.
    (b) Effective Exploration: Because of its intrinsic unpredictability, DRW can efficiently explore the search space and perhaps produce outcomes that are comparable to those of Aquila's exploration stage.

In Algorithm 1, DOL Population Initialization and DOL Generation Jumping are used and DRW is used to swap out the expanded exploration of AO. Algorithm 1 illustrates the

phases of this algorithm. In this, the parameter values are taken at their best regarding α, β of AO; weight w_d, jumping rate J_r of DOL; and weight w of DRW for the rest of the paper.

Figure 1 also displays the algorithm DAO flowchart visualization.

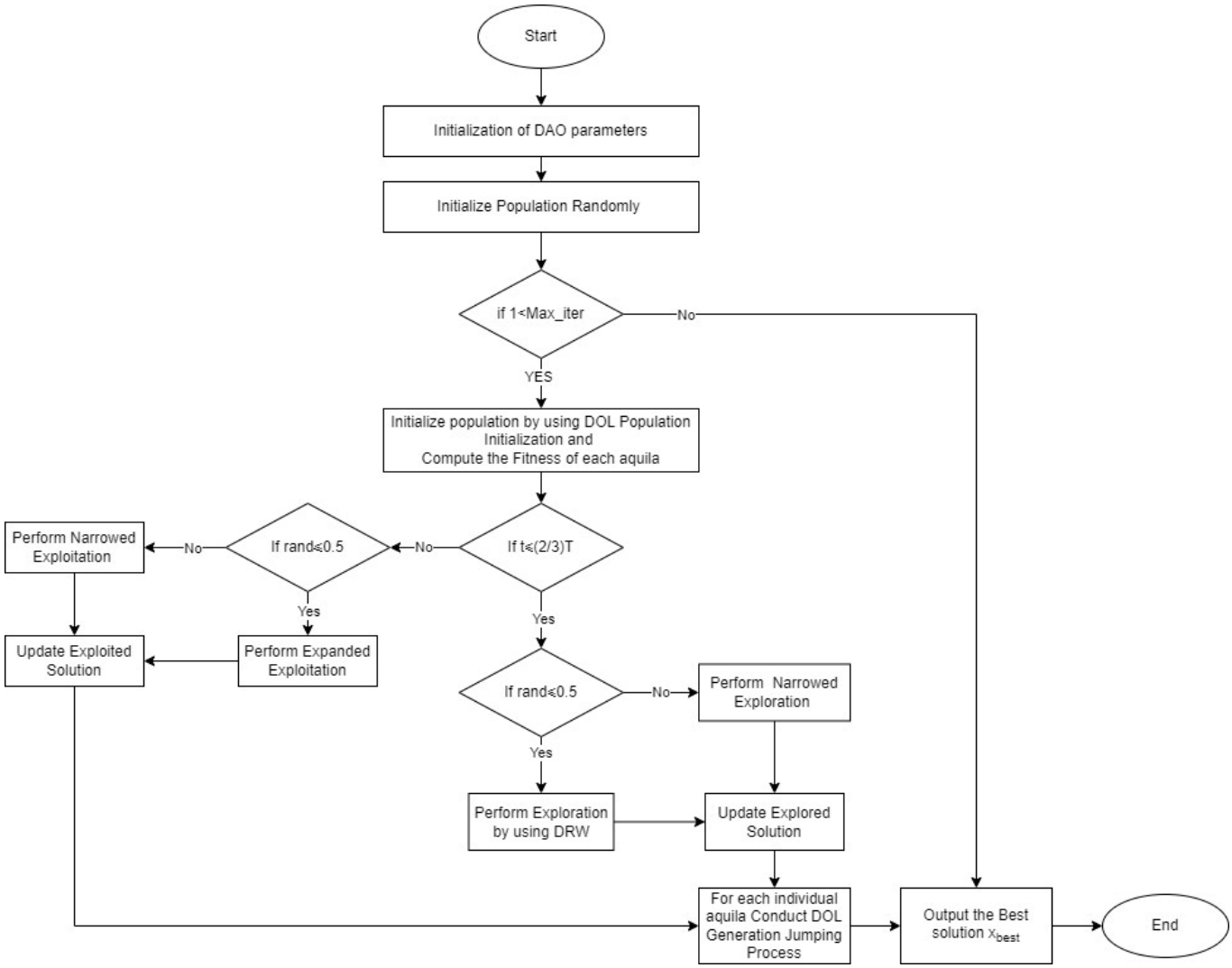

Figure 1. Flowchart of the proposed DAO algorithm.

This section also displays DAO's overall computational complexity. The initialization of the solutions, the computing of the fitness functions, and the updating of the solutions are the three steps that are often taken to ascertain the computational complexity of DAO. Let N represent the total number of solutions, and let $o(N)$ represent the computational complexity of the solutions' initialization processes. The computational complexity of the updating processes for the solutions is $o(N \times D) +$ o$(G \times (N \times D + N \times D))$, where G is the total number of iterations and D is the size of the problem's dimensions. These procedures entail updating the placements of each solution and looking for the best ones. Consequently, the overall computing complexity of the proposed DAO (Dynamic Opposition Learning and Dynamic Random Walk for Improving Aquila Optimizer) is $o(N \times D) + o(G \times (N \times D + N \times D)) = o(ND(1 + 2G))$.

Algorithm 1 DAO Algorithm

Initialize the values of parameters (nPop, nVar, α, β, w, w_d, J_r, Max_iter, etc.)
Establish a random starting position.
Take the counter $t = 1$
While (t < Max_iter), do
  Conduct DOL population initialization using Equation (16)
  Assess the early positions' fitness.
  Verify Boundaries
  For (i = 1: nPop) do
    Update of the existing solution's mean value
    Updated variables include u, v, P_1, P_2, *and* $Levy(D)$
      If $h \leq \left(\frac{2}{3}\right) \times Max\_iter$
        If $rand \leq 0.5$
          Apply DRW using Equation (18)
        Else
          Apply Narrowed Exploration by Equation (3)
        End If
      Else
        If $rand \leq 0.5$
          Apply Expanded Exploitation by Equation (8)
        Else
          Apply Narrowed Exploitation by Equation (9)
        End If
      End If
    Conduct the DOL population jumping process using Equation (17)
    Assess the fitness function.
    Verify boundaries
  End for
t = t + 1
End while
Record best solution (x_{best})

5. Experimental Settings

The algorithms used in the numerical trials include Aquila Optimizer (AO), Modified Aquila Optimizer (MAO) [38], Whale Optimization Algorithm (WOA) [6], Grasshopper Optimization Algorithm (GOA) [39], Reptile Search Algorithm (RSA) [5], and Brain Storm Optimization (BSO) [7]. On a computer with an Intel(R) Core (TM) i7-9750H processor running at 2.60 GHz and 16 GB of RAM, all algorithms were implemented in MATLAB R2021b.

The following five factors are used to assess DAO's (Dynamic Opposition Learning and Dynamic Random Walk for Improving Aquila Optimizer) performance:

1. The optimization errors between the obtained and known real optimal values, average, and standard deviation. Since all objective functions include minimization, the best values—that is, the lowest mean values—are indicated in bold.
2. Non-parametric statistical tests to compare the p-value and the significance level = 0.05 between the compared technique and the suggested algorithm, such as the Wilcoxon rank sum test [40]. For both techniques, there is a significant difference when the p-value is less than 0.05. W/T/L indicates how many wins, ties, and losses the algorithm in question has suffered in contrast to its opponent.
3. The Friedman test is another non-parametric statistical test that is used [41,42]. The average optimization error values are used as test data. The method operates more efficiently with lower Friedman rank values. To make the minimal value stand out, it is bolded.
4. Bonferroni–Dunn's diagram shows the differences in the rankings obtained for each algorithm at dimension 10 by showing the pairwise variances in ranks for each approach at each dimension. Pairwise disparities in rankings are calculated by subtracting the

rank of one algorithm from the rank of another algorithm. In the graphic created by Bonferroni and Dunn, each bar denotes the average pairwise difference in ranks for a certain algorithm at a given dimension. Typically, different algorithms are represented by color-coded bars.

5. A clear visual depiction of the algorithm's accuracy and convergence rate is offered via convergence graphs. If the improved algorithm deviates from the local answer, it explains why.

5.1. Competitive Algorithms Comparison on CEC2017 Benchmark Functions

Five competing algorithms are compared to gauge DAO's efficiency and search performance: MAO (Modified Aquila Optimizer), AO (Aquila Optimizer), RSA (Reptile Search Algorithm), WOA (Whale Optimization Algorithm), and BSO (Brain Storm Optimization). The comparison is made on 29 benchmark functions from IEEE CEC2017 from the literature [43]. The population size (N) was fixed at 50 in each experiment. Maximum iteration is 500 and dimension is 10. The [−100, 100] range was chosen for the search. On each function, each algorithm was executed 30 times.

Parameter Settings: The algorithm's performance depends on the parameter settings, particularly for DAO. In that instance, this part implements the sensitivity analysis of the parameters of DOL. Table 1 contains a detailed explanation of each parameter setting; the mean values are used to compare the results.

The weighting factor w and the jumping rate J_r are set to 1–10 and 0.1–1 in the DAO algorithm, respectively. Here, in Table 1, only $J_r = 0.3$, and 1 is taken because, at other points, the values are not favorable.

Test functions have been chosen for analysis from the literature [43], where F3 and F6 are multimodal functions, F18 is a hybrid function, F23 is a composition function, and, in order to assess performance, the means of the outcomes obtained by DAO are also shown in Table 1. In F3, F6, and F23, respectively, DAO performs better than other settings when $w = 3$ and $J_r = 1$. $w = 3$ and $J_r = 1$ are hence the best parameter settings, and DRW weight $w = 0.5$ is taken from the literature [16]. Table 2 contains the parameter settings of the optimization algorithms used for comparison.

Table 2. Parameter settings of optimization algorithms.

Algorithm	Parameters
DAO	$U = 0.00565, r = 10, \omega = 0.05, \alpha = 0.1, \beta = 0.1, P_1 \in [-1, 1]$, $P_2 = [2, 0], w = 0.5, w_d = 3, J_r = 1$
AO [9]	$U = 0.00565, r = 10, \omega = 0.05, \alpha = 0.1, \beta = 0.1, P_1 \in [-1, 1]$, $P_2 = [2, 0]$
MAO [38]	$U = 0.00565, r = 10, \omega = 0.05, \alpha = 0.1, \beta = 0.1, P_1 \in [-1, 1]$, $P_2 = [2, 0]$
SSA [44]	$v = 0$
WOA [6]	$w_1 = [2, 0], w_2 = [-1, -2], v = 1$
RSA [5]	$a = [2, 0]$
GOA [39]	$l = 1.5,\ f = 0.5$
BSO [7]	$m = 5,\ p_a = 0.2,\ p_b = 0.8,\ p_{b_1} = 0.4,\ p_c = 0.5$

Analysis of IEEE CEC'17 Test Functions

- **Analysis of Unimodal and Multimodal Test Functions**

The mean and standard deviation of algorithms on twenty-nine unimodal, multimodal, and composition functions are displayed in Table 3. The function F1 is unimodal. The results show that, on one unimodal function, DAO outperforms the other algorithms. Moreover, it may be said that the DOL approach, which expands search spaces, has a higher chance of reaching the global optimum for its capacity for exploitation.

Table 3. Mean and standard deviation (STD) obtained from objective function by standard AO, the proposed algorithm DAO, and other metaheuristic algorithms for 10-dimensional CEC 2017 benchmark functions.

Function	DAO	AO	MAO	RSA	WOA	BSO
F1 Mean	$\mathbf{8.388 \times 10^8}$	9.239×10^8	2.159×10^{10}	5.38×10^{10}	9.784×10^8	9.410×10^9
STD	4.709×10^8	6.512×10^6	4.981×10^9	9.29×10^9	7.462×10^6	2.501×10^3
F3 Mean	4.291×10^3	8.809×10^2	2.363×10^5	7.42×10^4	3.663×10^3	$\mathbf{3.001 \times 10^1}$
STD	1.381×10^3	5.485×10^2	4.971×10^4	5.50×10^3	3.232×10^3	1.710×10^2
F4 Mean	$\mathbf{7.619 \times 10^1}$	2.085×10^2	2.527×10^3	1.45×10^4	9.451×10^1	9.495×10^2
STD	4.001×10^1	2.512×10^2	1.304×10^3	4.56×10^3	1.923×10^1	2.001×10^1
F5 Mean	$\mathbf{6.359 \times 10^1}$	7.125×10^1	1.508×10^2	3.89×10^2	8.408×10^1	2.038×10^2
STD	1.584×10^1	1.066×10^1	2.758×10^1	3.30×10^1	2.096×10^1	4.101×10^1
F6 Mean	$\mathbf{3.523 \times 10^1}$	7.745×10^1	9.271×10^1	8.63×10^1	3.627×10^1	5.316×10^1
STD	8.702×10^0	6.053×10^0	1.745×10^1	7.46×10^0	1.012×10^1	6.414×10^0
F7 Mean	8.615×10^1	$\mathbf{5.545 \times 10^1}$	4.585×10^2	6.72×10^2	7.470×10^1	5.110×10^2
STD	2.031×10^1	1.931×10^1	9.379×10^1	6.73×10^1	2.151×10^1	1.011×10^2
F8 Mean	3.211×10^1	$\mathbf{2.408 \times 10^1}$	1.369×10^2	3.11×10^2	4.291×10^1	1.451×10^2
STD	6.033×10^0	6.884×10^0	1.922×10^1	2.80×10^1	1.767×10^1	3.211×10^1
F9 Mean	$\mathbf{2.773 \times 10^2}$	3.135×10^2	4.114×10^3	8.53×10^3	5.919×10^2	3.411×10^3
STD	1.571×10^2	6.321×10^1	1.082×10^3	1.19×10^3	3.820×10^2	6.754×10^2
F10Mean	1.451×10^3	$\mathbf{9.451 \times 10^2}$	2.726×10^3	7.02×10^3	1.181×10^3	4.211×10^3
STD	3.124×10^2	2.686×10^2	2.296×10^2	3.59×10^2	2.751×10^2	6.081×10^2
F11Mean	4.201×10^2	$\mathbf{1.078 \times 10^2}$	2.604×10^4	7.77×10^3	1.417×10^2	1.378×10^2
STD	4.743×10^2	5.818×10^1	2.681×10^4	2.80×10^3	8.465×10^1	4.511×10^1
F12Mean	$\mathbf{5.697 \times 10^6}$	7.862×10^6	2.784×10^9	1.70×10^{10}	7.279×10^6	9.614×10^7
STD	5.285×10^6	3.363×10^6	1.640×10^9	4.36×10^9	5.117×10^6	8.094×10^5
F13Mean	$\mathbf{2.549 \times 10^5}$	2.465×10^5	3.020×10^8	1.18×10^{10}	1.437×10^6	5.216×10^7
STD	6.824×10^5	1.528×10^4	3.011×10^8	4.90×10^9	1.177×10^4	2.340×10^4
F14Mean	$\mathbf{5.424 \times 10^3}$	6.334×10^4	7.503×10^6	3.07×10^6	7.307×10^3	4.170×10^5
STD	8.248×10^3	8.016×10^2	1.063×10^7	3.58×10^6	1.500×10^3	3.152×10^3
F15Mean	$\mathbf{6.293 \times 10^3}$	9.332×10^3	2.148×10^7	6.73×10^8	6.416×10^3	3.112×10^4
STD	3.375×10^3	2.839×10^3	2.908×10^7	5.74×10^8	5.063×10^3	2.122×10^4
F16Mean	$\mathbf{3.248 \times 10^2}$	9.535×10^2	1.178×10^3	3.89×10^3	3.329×10^2	1.504×10^3
STD	1.027×10^2	1.114×10^2	2.349×10^2	6.86×10^2	1.440×10^2	3.314×10^2
F17Mean	$\mathbf{8.911 \times 10^1}$	9.589×10^1	6.631×10^2	5.30×10^3	1.033×10^2	8.120×10^2
STD	2.286×10^1	1.871×10^1	1.916×10^2	6.86×10^3	5.087×10^1	2.401×10^2
F18Mean	2.407×10^5	2.153×10^4	6.274×10^8	3.27×10^7	$\mathbf{1.946 \times 10^4}$	1.120×10^5
STD	3.197×10^5	1.184×10^4	6.430×10^8	3.07×10^7	1.111×10^4	1.001×10^5
F19Mean	3.203×10^4	1.436×10^4	6.471×10^7	$\mathbf{1.32 \times 10^4}$	6.597×10^4	1.301×10^5
STD	4.889×10^4	2.225×10^4	8.754×10^7	1.69×10^9	9.665×10^4	5.361×10^4
F20Mean	$\mathbf{1.701 \times 10^2}$	2.153×10^2	5.419×10^2	8.63×10^2	1.854×10^2	7.219×10^2
STD	5.579×10^1	4.716×10^1	1.330×10^2	1.42×10^2	7.896×10^1	2.015×10^2
F21Mean	$\mathbf{2.299 \times 10^2}$	2.967×10^2	3.375×10^2	6.43×10^2	2.310×10^2	4.004×10^2
STD	5.293×10^1	4.681×10^1	3.148×10^1	4.26×10^1	5.171×10^1	4.051×10^1
F22Mean	$\mathbf{1.758 \times 10^2}$	2.091×10^2	1.798×10^3	5.25×10^3	1.831×10^2	4.001×10^3
STD	5.337×10^1	1.524×10^1	5.866×10^2	1.01×10^3	2.703×10^2	1.701×10^3
F23Mean	$\mathbf{3.843 \times 10^2}$	5.412×10^2	5.423×10^2	1.04×10^3	3.976×10^2	9.991×10^2
STD	2.354×10^1	1.313×10^1	6.781×10^1	1.08×10^2	2.060×10^1	1.013×10^2

Table 3. *Cont.*

Function	DAO	AO	MAO	RSA	WOA	BSO
F24Mean	$\mathbf{3.145 \times 10^2}$	3.437×10^2	5.939×10^2	1.17×10^3	3.870×10^2	1.004×10^3
STD	1.416×10^1	8.266×10^1	7.436×10^1	2.45×10^2	2.521×10^1	9.711×10^1
F25Mean	4.776×10^2	7.949×10^2	1.988×10^3	2.22×10^3	5.651×10^2	$\mathbf{4.101 \times 10^2}$
STD	4.645×10^1	3.036×10^1	7.365×10^2	8.61×10^2	3.538×10^1	9.110×10^0
F26Mean	$\mathbf{6.408 \times 10^2}$	9.175×10^2	2.348×10^3	7.93×10^3	9.465×10^2	5.832×10^3
STD	3.023×10^2	1.623×10^2	3.639×10^2	1.12×10^3	6.068×10^2	1.112×10^3
F27Mean	$\mathbf{4.467 \times 10^2}$	6.041×10^2	7.303×10^2	9.41×10^2	5.379×10^2	1.204×10^3
STD	4.845×10^1	8.332×10^0	1.222×10^2	2.31×10^2	3.300×10^1	2.510×10^2
F28Mean	$\mathbf{4.913 \times 10^2}$	5.965×10^2	1.323×10^3	3.98×10^3	6.153×10^2	5.854×10^2
STD	6.521×10^0	9.938×10^1	2.043×10^2	8.85×10^2	1.794×10^2	5.120×10^1
F29Mean	4.026×10^2	$\mathbf{3.429 \times 10^2}$	1.070×10^3	4.14×10^3	4.614×10^2	1.520×10^3
STD	6.510×10^1	5.123×10^1	2.163×10^2	1.61×10^3	8.636×10^1	3.701×10^2
F30Mean	$\mathbf{3.891 \times 10^4}$	6.647×10^5	1.451×10^8	2.24×10^8	7.597×10^6	5.371×10^5
STD	8.456×10^4	7.482×10^4	1.052×10^8	9.25×10^7	9.042×10^5	3.104×10^5
(W/L/T)	20/9/0	5/24/0	0/29/0	1/28/0	1/28/0	2/27/0
Rank	**1.62**	2.41	4.72	5.62	2.55	4.07
CPU Runtime	3.25×10^4	2.10×10^4	1.29×10^4	5.11×10^4	$\mathbf{4.10 \times 10^3}$	1.29×10^4

Note: bold is used to indicate the best result.

Multimodal functions like F3–F9 are used to confirm DAO's exploring capabilities. The results in Table 3 demonstrate how well DAO performs in comparison to other algorithms, particularly on the F4, F5, F6, and F9 test functions.

- **Analysis of Hybrid and Composition Test Functions**

Hybrid functions are used to evaluate the algorithms by combining unimodal and multimodal functions in order to mimic real-world challenges. It may lead to subpar performance; however, balancing exploitation and exploration capability is important to deal with mixed tasks. Table 3 clearly illustrates the benefits of DAO on F12–F17, F20–F24, F26–F28, and F30, and the composition function indicates that DAO is still able to solve the problem to the same degree as other algorithms. Then, in many real-world scenarios, DAO may effectively balance the rate of convergence and the optimization solution.

The last line of Table 3 shows W/L/T (Win/Loss/Tie), Friedman rank, and CPU runtime. The W/L/T metric shows that DAO performs well on functions with 10 dimensions, outperforming AO, MAO, RSA, WOA, and BSO on 24, 29, 28, 28, and 27 functions, respectively. The Friedman rank of DAO is comparatively less than other MAs, and the CPU runtime of DAO, AO, MAO, RSA, WOA, and BSO is shown in the third-last line of Table 3. The results show that WOA takes much less time than other MAs.

Analysis of Convergence Graph

Figure 2 displays the convergence graphs of the four functions, F4, F9, F13, and F20, where the mean optimizations generated by six algorithms on the IEEE CEC2017 functions with 10 dimensions are displayed. The vertical axis represents the log value of the mean optimizations, while the horizontal axis represents the number of iterations. Figure 2 makes it clear that the convergence speed is fast and that the DAO curves are the lowest. When compared to the original AO in the convergence graphs, DAO can find a better solution, exit local optimization, avoid premature convergence, improve the quality of the solution, and have high optimization efficiency.

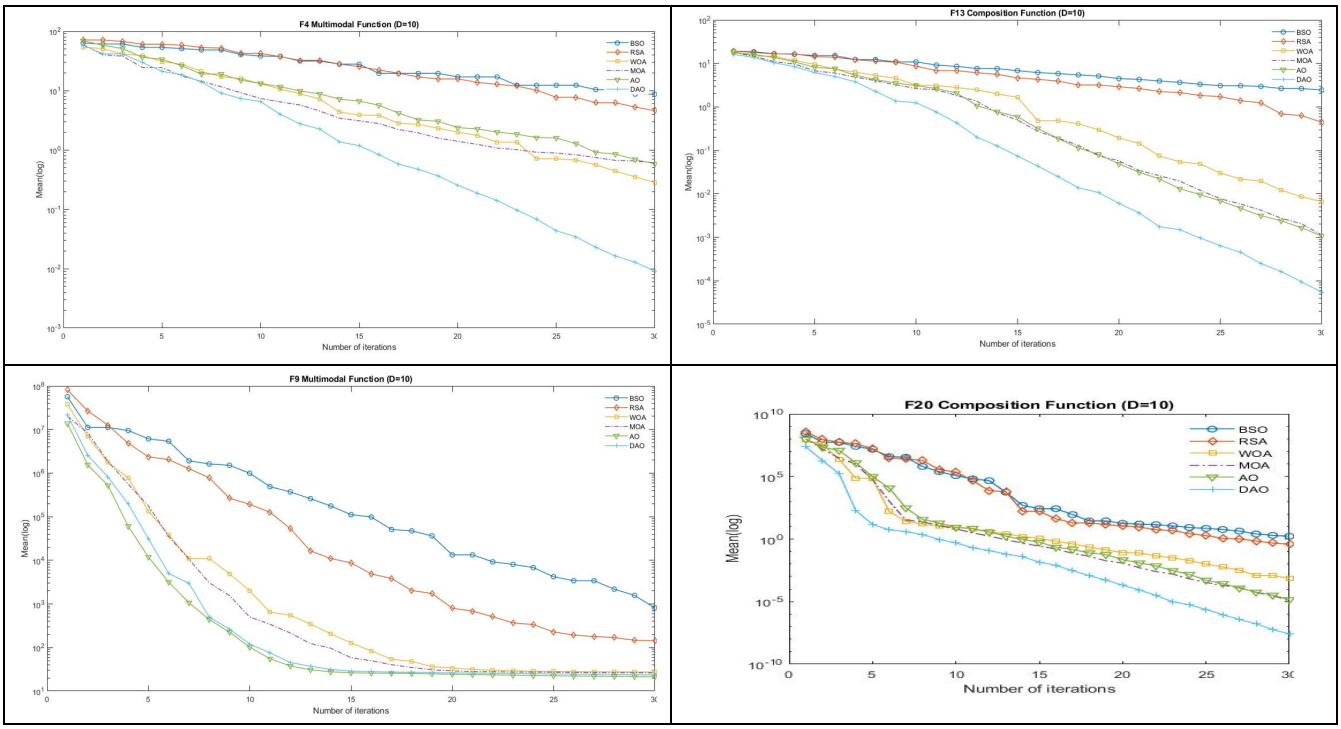

Figure 2. Convergence graphs of F4, F9, F13, and F20 CEC 2017 benchmark function.

Table 4 represents the Wilcoxon rank sum test results. The totals of ranks for positive and negative differences are represented by $\sum R^+$ and $\sum R^-$, respectively. When compared to other algorithms, DAO has a greater positive rank sum. Additionally, in the table, the corresponding z and p values are provided. The significant threshold of difference is $\alpha = 0.05$. This table shows that the performance of DAO is better than other original AO and other metaheuristic algorithms.

Table 4. Summary of non-parametric statistical results by Wilcoxon test and Bonferroni–Dunn test.

Algorithms		$\sum R^+$	$\sum R^-$	z-Value	p-Value	Sign
DAO vs.	AO	21	8	2.022	0.043	=
	MAO	29	0	4.703	0.000	+
	RSA	28	1	4.249	0.000	+
	WOA	24	5	2.757	0.006	=
	BSO	25	4	3.557	0.000	+
CD value at $\alpha = 0.1$		1.1428		CD value at $\alpha = 0.05$		1.2656

The Bonferroni–Dunn's test [45] is used for the DAO algorithm to identify significant differences, and the results are shown in the last line of Table 4. Among all the other algorithms, DAO was found to have the lowest mean rank. The Bonferroni–Dunn graphic in Figure 3 shows the variation in ranks for each method at D = 10. In this figure, a horizontal cut line is drawn, which represents the threshold for the best-performing algorithm, the one with the lowest ranking bar. The height of this cut line is determined by adding the algorithm's ranking. The Bonferroni–Dunn technique computed the equivalent CD for each $\alpha = 0.05$ and $\alpha = 0.1$. Algorithms with a rank bar higher than this line are deemed to perform worse than the control algorithm. As a result, it is evident from the use of the Bonferroni–Dunn technique that AO and WOA are substantially acceptable when compared with DAO.

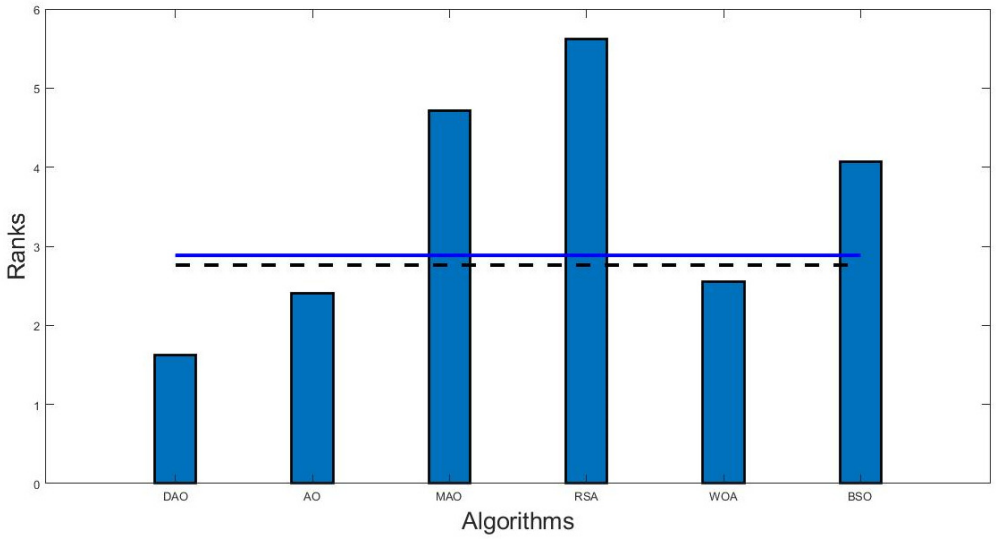

Figure 3. Bonferroni–Dunn bar chart for D = 10. The bar represents the rank of the correspondence algorithm, and horizontal cut lines show the significance level (here, ----- shows sig level at 0.1, and shows significance level at 0.05).

5.2. Competitive Algorithms Comparison on CEC2019 Benchmark Functions

In Table 5, the list of the ten CEC2019 benchmark functions with their dimensions and search ranges is taken from the literature [46].

Table 5. List of 10 benchmark functions of CEC2019 with dimensions and search range.

Func. No.	Functions	Dim	Search Range
F1	Storn's Chebyshev Polynomial Fitting Problem	9	[−8192, 8192]
F2	Inverse Hilbert Matrix Problem	16	[−16,384, 16,384]
F3	Lennard-Jones Minimum Energy Cluster	18	[−4, 4]
F4	Rastrigin's Function	10	[−100, 100]
F5	Griewangk's Function	10	[−100, 100]
F6	Weierstrass Function	10	[−100, 100]
F7	Modified Schwefel's Function	10	[−100, 100]
F8	Expanded Schaffer's F6 Function	10	[−100, 100]
F9	Happy Cat Function	10	[−100, 100]
F10	Ackley Function	10	[−100, 100]

Analysis of IEEE CEC'19 Test Functions

DAO has been implemented on 10 CEC 2019 benchmark functions with 500 iterations and 50 population sizes for 30 independent runs. Its results are compared with AO, MAO, WOA, SSA, and GOA. The comparison has been performed through the mean and STD (standard deviation) values by the considered algorithms across the course of the functions, as reported in Table 6. Moreover, the Friedman mean rank values and W/L/T are involved in the table's last lines (see Table 6). The results confirm the proposed DAO's superiority in dealing with these challenging testbed functions as it is classified as the best algorithm for half of these functions.

Meanwhile, AO succeeded for three functions, and MAO, WOA, SSA, and GOA for only one function out of this set. When it comes to the chain counterparts, DAO is positioned first in terms of sequence. The CPU runtime is mentioned in the last line of Table 6, which shows WOA taking much less time than the other algorithms. The convergence curves of Figure 4 show the efficiency of DAO in converging for high qualified solutions with significant convergence speed, as exhibited in F2, F6, F7, and F9.

Table 6. Mean and standard deviation (STD) obtained from objective function by standard AO, the proposed algorithm DAO, and other metaheuristic algorithms for 10-dimensional CEC 2019 benchmark functions.

Function	DAO	AO	MAO	WOA	SSA	GOA
F1 Mean	$\mathbf{9.900 \times 10^{1}}$	$\mathbf{9.900 \times 10^{1}}$	1.235×10^{9}	6.784×10^{6}	7.324×10^{9}	1.320×10^{10}
STD	0.000×10^{0}	2.053×10^{-8}	7.355×10^{8}	7.462×10^{6}	3.483×10^{9}	1.541×10^{10}
F2 Mean	$\mathbf{1.950 \times 10^{2}}$	$\mathbf{1.950 \times 10^{2}}$	2.825×10^{4}	7.663×10^{2}	2.001×10^{2}	1.739×10^{3}
STD	0.000×10^{0}	0.000×10^{0}	7.301×10^{3}	8.7317×10^{2}	2.079×10^{-2}	4.084×10^{2}
F3 Mean	2.948×10^{2}	2.937×10^{2}	2.862×10^{2}	2.951×10^{2}	2.970×10^{2}	$\mathbf{2.270 \times 10^{2}}$
STD	1.321×10^{0}	1.863×10^{0}	4.426×10^{-1}	1.923×10^{0}	1.776×10^{-15}	8.188×10^{-12}
F4 Mean	3.441×10^{2}	3.683×10^{2}	2.445×10^{2}	3.498×10^{2}	$\mathbf{3.423 \times 10^{1}}$	3.286×10^{2}
STD	1.376×10^{1}	9.697×10^{0}	2.501×10^{1}	2.496×10^{1}	1.077×10^{1}	1.971×10^{1}
F5 Mean	4.929×10^{2}	4.981×10^{2}	$\mathbf{3.059 \times 10^{2}}$	4.977×10^{2}	5.486×10^{2}	8.484×10^{2}
STD	4.413×10^{0}	1.826×10^{-1}	4.894×10^{1}	4.591×10^{-1}	8.533×10^{-1}	8.763×10^{-1}
F6 Mean	$\mathbf{5.918 \times 10^{2}}$	5.944×10^{2}	5.999×10^{2}	5.919×10^{2}	5.986×10^{2}	8.484×10^{2}
STD	1.787×10^{0}	1.440×10^{0}	9.224×10^{-1}	1.751×10^{0}	8.533×10^{-1}	8.763×10^{1}
F7 Mean	7.152×10^{2}	$\mathbf{3.011 \times 10^{2}}$	2.217×10^{3}	7.640×10^{2}	4.728×10^{2}	5.007×10^{2}
STD	2.553×10^{2}	2.936×10^{2}	2.924×10^{2}	3.001×10^{2}	9.776×10^{-1}	2.191×10^{2}
F8 Mean	7.953×10^{2}	8.957×10^{2}	7.644×10^{2}	$\mathbf{5.953 \times 10^{2}}$	9.088×10^{2}	8.587×10^{2}
STD	1.998×10^{-1}	3.015×10^{-1}	2.387×10^{-1}	3.216×10^{-1}	6.135×10^{-1}	4.300×10^{-1}
F9 Mean	$\mathbf{8.985 \times 10^{2}}$	9.365×10^{2}	8.993×10^{3}	8.985×10^{3}	2.416×10^{3}	9.664×10^{2}
STD	1.639×10^{-1}	1.427×10^{-1}	8.679×10^{-1}	2.006×10^{-1}	5.956×10^{-1}	1.827×10^{-1}
F10 Mean	$\mathbf{9.785 \times 10^{2}}$	9.996×10^{2}	9.852×10^{2}	9.953×10^{2}	2.101×10^{3}	9.923×10^{2}
STD	7.688×10^{-1}	4.637×10^{0}	1.350×10^{-1}	1.330×10^{-1}	3.562×10^{1}	3.718×10^{-4}
(W/L/T)	5/5/2	3/7/2	1/9/0	1/9/0	1/9/0	1/9/0
Rank	**2.65**	3.25	3.15	3.65	4.30	4.00
CPU Runtime	3.11×10^{4}	3.02×10^{4}	2.16×10^{4}	5.02×10^{4}	$\mathbf{4.22 \times 10^{3}}$	1.26×10^{4}

Note: bold is used to indicate the best result.

Figure 4 shows the convergence capacity of six algorithms on test functions. The average fitness value is displayed as the "Mean". Because of its exceptional exploration capabilities, DAO converges quickly with iterative computation, as illustrated in the figures. Regarding the trend that is gradually convergent, this is because the DOL technique is capable of being exploited.

Table 7 represents the Wilcoxon rank sum test results. The totals of ranks for positive and negative differences are represented by $\sum R^{+}$ and $\sum R^{-}$, respectively. When compared to other algorithms, DAO has a greater positive rank sum in most of the cases. Additionally, in the table, the corresponding z and p values are provided. The significant threshold of difference is $\alpha = 0.05$. This table shows that the performance of DAO is equivalently acceptable when compared to other metaheuristic algorithms.

Bonferroni–Dunn's test is used for the DAO algorithm to identify significant differences, and the results are shown in the last line of Table 7. Among all the other algorithms, DAO was found to have the lowest mean rank. The Bonferroni–Dunn graphic in Figure 5 shows the variation in ranks for each method at D = 10. In this figure, the smallest bar will show the best-performing algorithm, or the one with the lowest ranking bar. Algorithms with a higher rank bar are deemed to perform worse than the control algorithm. As a result, it is evident from the use of the Bonferroni–Dunn technique that DAO is also performing well when compared with other metaheuristic algorithms, and the worst performance is from the SSA algorithm.

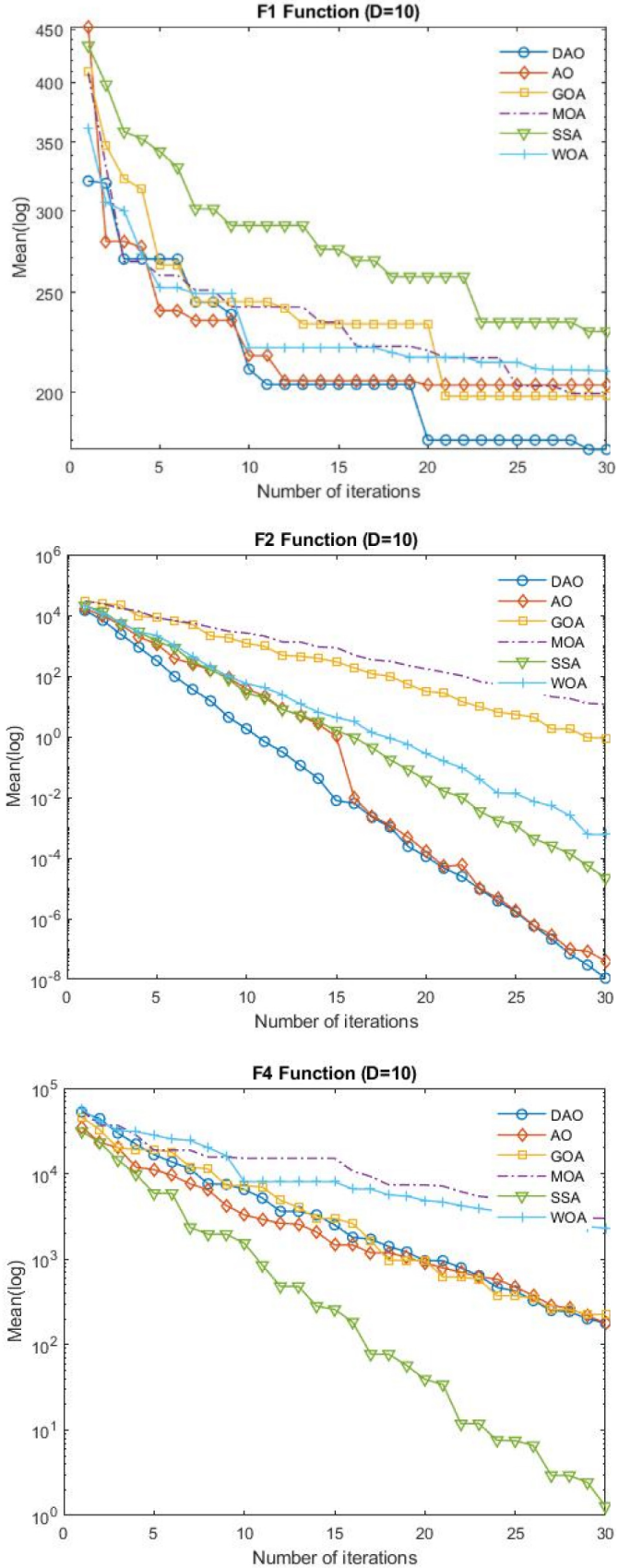

Figure 4. *Cont*.

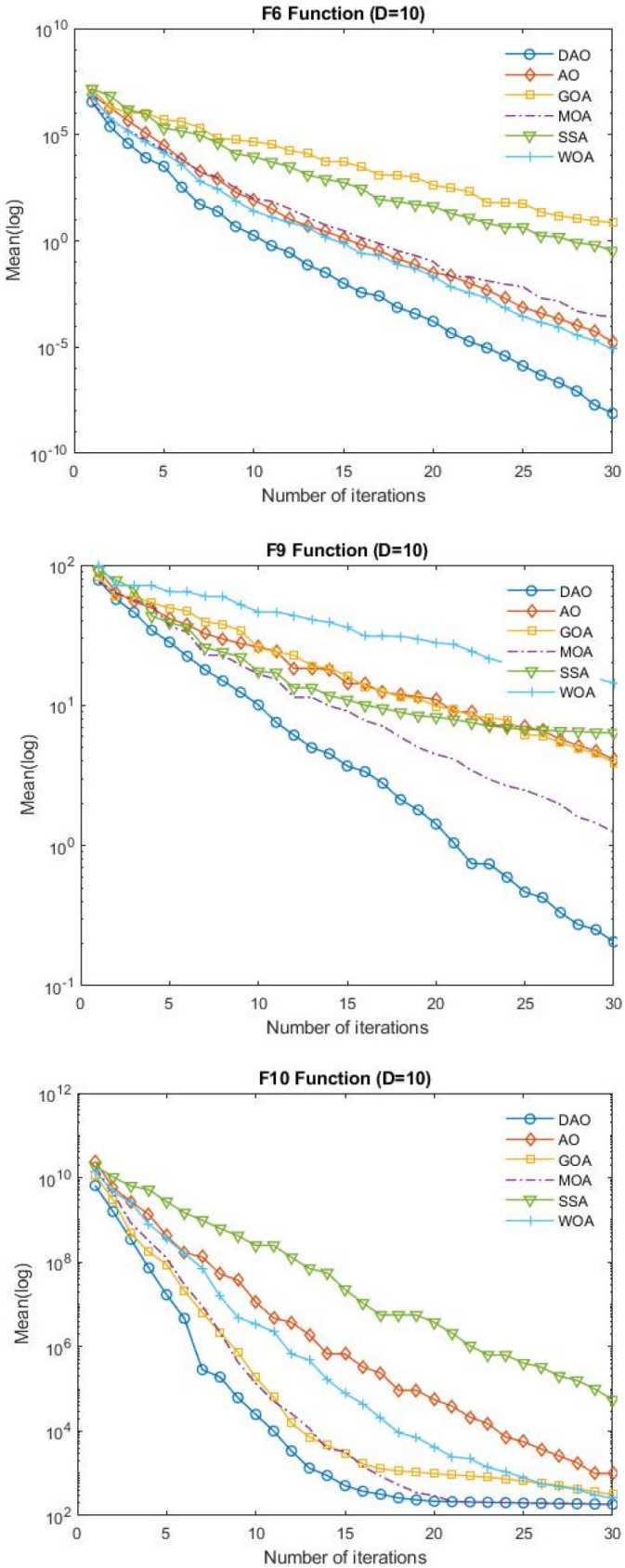

Figure 4. Convergence graphs of F1, F2, F4, F6, F9, and F10 CEC 2019 benchmark functions.

Table 7. Summary of non-parametric statistical results obtained from Wilcoxon test and Bonferroni–Dunn test.

Algorithms		ΣR^+	ΣR^-	z-Value	*p*-Value	Sign
DAO vs.	AO	6	2	1.260	0.208	=
	MAO	5	5	0.866	0.386	=
	WOA	6	3	1.599	0.110	=
	MPA	8	2	1.478	0.139	=
	GOA	7	3	1.478	0.139	=

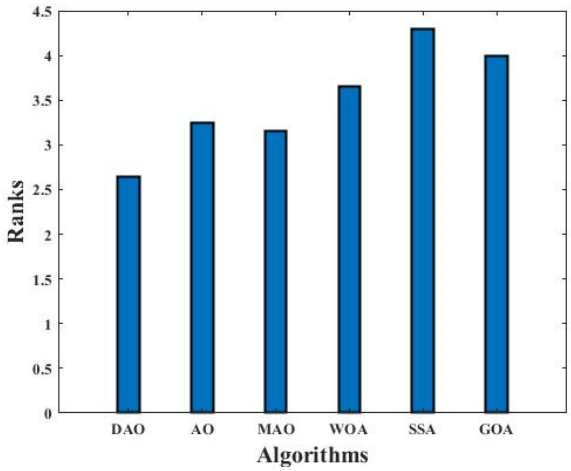

Figure 5. Bonferroni–Dunn bar chart for D = 10. The bar represents the rank of the correspondence algorithm.

6. DAO for Engineering Design Problems

Three relevant engineering benchmarks are used in this section to confirm that DAO improves when tackling real-world problems: problem of cantilever beam design (CBD), welded beam design problem (WBD), and pressure vessel design (PVD) problem and work. Thirty independent runs of each problem were carried out in order to examine the statistical features of the outcomes, and all the parameters are taken at their best.

6.1. CBD Problem

The goal of the CBD problem is to minimize a cantilever beam's weight while accounting for the vertical displacement constraint. There are five hollow square blocks, and each of the five side length values z_1, z_2, z_3, z_4, z_5 needs to be optimized [47]. The following is an explanation of the mathematical model:

Consider

$$z = [z_1\ z_2\ z_3\ z_4\ z_5]$$

Minimize

$$f(z) = 0.6224(z_1 + z_2 + z_3 + z_4 + z_5),$$

Subject to

$$p(z) = \frac{60}{z_1^3} + \frac{27}{z_2^3} + \frac{19}{z_3^3} + \frac{7}{z_4^3} + \frac{1}{z_5^3} - 1 \leq 0$$

Variable range is

$$0.01 \leq z_1, z_2, z_3, z_4, z_5 \leq 100$$

Table 8 displays the results of the CBD problem compared with six different MAs, such as COA, AO, GWO, ROA, WOA, and SCA. The results indicate that the proposed algorithm DAO is able to provide better results than other state-of-the-art algorithms. Thus, DAO is the optimal method for addressing the CBD problem. CPU runtime of the given

set of algorithms is calculated, which shows that WOA takes very little time to compute the CBD problem.

Table 8. Comparison of DAO and other algorithms for CBD problem.

	Optimum Attributes						
Algorithms	z_1	z_2	z_3	z_4	z_5	**Optimum Weight**	**CPU Runtime (s)**
DAO	6.0112	5.1211	4.8221	3.2114	2.1510	**1.3302**	1.986
COA [48]	6.0172	5.3071	4.4912	3.5081	2.1499	1.3999	2.001
AO [9]	5.8492	5.5413	4.3778	3.5978	2.1026	1.3596	1.926
ROA [49]	6.0156	5.1001	4.303	3.7365	2.3183	1.3456	1.256
GWO [50]	5.9956	5.4121	4.5986	3.5689	2.3548	1.3586	1.112
WOA [6]	5.8393	5.1582	4.9917	3.693	2.2275	1.3467	**0.606**
SCA [51]	5.9264	5.9285	4.5223	3.3267	1.9923	1.3581	1.111

Note: bold is used to indicate better result.

6.2. WBD Problem

The goal of the WBD challenge is to reduce the cost of manufacturing a welded beam [9]. The optimization parameters include thickness (H), height (T_T), length of the clamping bar (L), and thickness (B_B). It is important to take into account seven limitations. The optimization model can be stated as follows:

Consider

$$z = [z_1\ z_2\ z_3\ z_4] = [H\ L\ T_T\ B_B]$$

Minimize

$$f(z) = 1.10471 z_1^2 z_z + 0.04811 z_3 z_4 (14.0 + z_2)$$

Subject to the constraint,

$$\begin{aligned}
&p_1(z) = \tau(z) - \tau_{\max} \le 0,\\
&p_2(z) = \sigma(z) - \sigma_{\max} \le 0,\\
&p_3(z) = \delta(z) - \delta_{\max} \le 0,\\
&p_4(z) = z_1 - z_4 \le 0,\\
&p_5(z) = P - P_c(z) \le 0,\\
&p_6(z) = 0.125 - z_1 \le 0,\\
&p_7(z) = 1.10471 z_1^2 + 0.04811 z_3 z_4 (14 + z_2) - 5 \le 0
\end{aligned}$$

Variable range

$$\begin{aligned}
&0.1 \le z_1 \le 2,\\
&0.1 \le z_2 \le 10,\\
&0.1 \le z_3 \le 10,\\
&0.1 \le z_4 \le 2
\end{aligned}$$

where

$$\begin{aligned}
&\tau(z) = \sqrt{(\tau')^2 + 2\tau'\tau'' \tfrac{z_2}{2R} + (\tau'')^2},\\
&\tau' = \tfrac{p}{\sqrt{2 z_1 z_2}},\ \tau'' = \tfrac{MR}{J}\\
&M = P\left(L + \tfrac{z_2}{2}\right),\\
&R = \sqrt{\tfrac{z_2^2}{4} + \left(\tfrac{z_1 + z_3}{2}\right)^2},\\
&J = 2\left\{\sqrt{z_1 z_2}\left[\tfrac{z_2^2}{4} + \left(\tfrac{z_1 + z_3}{2}\right)^2\right]\right\},\\
&\sigma(z) = \tfrac{6PL}{z_4 z_3^2}, \delta(z) = \tfrac{6PL^3}{E z_3^2 z_4}\\
&P_c(z) = \tfrac{4.013E\sqrt{\frac{z_3^2 z_4^6}{36}}}{L^2}\left(1 - \tfrac{z_3}{2L}\sqrt{\tfrac{E}{4G}}\right),
\end{aligned}$$

$$P = 6000 \text{ lb}, L = 14 \text{ in.}, \delta_{\max} = 0.25 \text{ in.},$$
$$E = 30 \times 1^6 \text{ psi}, G = 12 \times 10^6 \text{ psi},$$
$$\tau_{\max} = 13600 \text{ psi}, \sigma_{\max} = 30000 \text{ psi}$$

Table 9 reports the outcomes of the WBD problem. It is clear that DAO is not able to provide a better solution than other algorithms. However, with the exception of AO, DAO also has a very close value to provide an optimal result. This suggests that DAO is a stable and effective solution to the WBD problem. CPU runtime of the given set of algorithms is calculated, which shows that WOA takes very little time to compute the WBD problem.

Table 9. Comparison of DAO and other algorithms for WBD problem.

Optimum Attributes						
Algorithms	**H**	**L**	**T_T**	**B_B**	**Optimum Cost**	**CPU Runtime (s)**
DAO	0.2138	3.2154	9.0275	0.2052	1.6960	2.410
COA [48]	0.2456	3.2563	9.0403	0.2057	1.6963	2.031
AO [9]	0.1631	3.3652	9.0202	0.2067	**1.6566**	2.399
SSA [44]	0.2057	3.4714	9.0366	0.2057	1.7249	2.121
WOA [6]	0.2054	3.4843	9.0374	0.2062	1.7305	**1.037**

Note: bold is used to indicate better result.

6.3. PVD Problem

The PVD problem, a classical and representative optimization issue in engineering, is typically employed to verify the efficacy of optimization techniques. Its goal is to reduce a tension/compression spring's cost [41]. The design parameters are thickness of the shell T_S, thickness of the head T_H, inner radius r, and the length of the cylindrical shell L_{CS}. The following is the expression for the mathematical formulation [47]:

Consider

$$z = [z_1\ z_2\ z_3\ z_4] = [T_S\ T_H\ r\ L_{CS}]$$

Minimize

$$f(z) = 0.6224 z_1 z_3 z_4 + 1.7781 z_2 z_3^2 + 3.1661 z_1^2 z_4 + 19.84 z_1^2 z_3,$$

Subject to

$$p_1(z) = -z_1 + 0.0193 z_3 \le 0,$$
$$p_2(z) = -z_3 + 0.00954 z_3 \le 0,$$
$$p_3(z) = -\pi z_3^2 z_4 - \frac{4}{3}\pi z_3^3 + 1296000 \le 0,$$
$$p_4(z) = z_4 - 240 \le 0$$

Variable range is

$$\begin{cases} 0 \le z_1 \le 99, \\ 0 \le z_2 \le 99, \\ 10 \le z_3 \le 200, \\ 10 \le z_4 \le 200 \end{cases}$$

Table 10's results for the TSD problem demonstrate that ROA is the optimal method for solving it, followed by COA and DAO, but we can say that DAO is a competitive and stable solution. CPU runtime of the given set of algorithms is calculated, which shows that WOA takes very little time to compute the PVD problem.

Table 10. Comparison of DAO and other algorithms for PVD problem.

Optimum Attributes						
Algorithms	T_S	T_H	r	L_{CS}	**Optimum Cost**	**CPU Runtime (s)**
DAO	0.7885	0.3254	42.3275	189.892	5877.1000	2.432
COA [48]	0.7437	0.3705	40.3238	199.9414	5735.2488	2.356
AO [9]	1.0540	0.1828	59.6219	38.8050	5949.2258	2.222
GWO [50]	0.8125	0.4345	42.0891	176.7587	6051.5639	1.345
ROA [49]	0.7295	0.2226	40.4323	198.5537	**5311.9175**	2.252
RSA [5]	0.8071	0.4426	43.6335	142.5359	6213.8317	1.125
WOA [6]	0.8125	0.4375	42.0982	76.6389	6059.7410	**0.872**

Note: bold is used to indicate better result.

The outcomes of three classic engineering challenges are shown in this section, demonstrating how well and consistently DAO performs when handling real-world issues. In particular, DAO performs noticeably better than the AO algorithm.

7. Conclusions

In order to replace the expanded exploration regarding AO, this study has proposed a low-complexity DRW method that strikes a fair balance between exploitation and exploration. The aim of this technique is to increase computational efficiency and to avoid stagnation. Moreover, to achieve a balance between exploration and exploitation, the DOL technique is introduced. The CPU runtime clearly shows Aquila Optimizer's computing efficiency. Then, the results obtained from the benchmark functions of CEC 2017 and CEC 2019 demonstrate its superiority. Furthermore, the convergence graphs, the Wilcoxon rank sum tests, the Friedman test, and the Bonferroni test show its accessibility. Then, it is also applied to real-world structural engineering design problems, which provides better results than AO. All these results show that the DRW and DOL approaches provide great additions to AO. DAO performs far better than AO as well as compared to most of the other MAs.

8. Future Scope

DAO could be applied in additional real-world applications given its great performance. Additionally, other optimization jobs including image processing, cloud and fog computing, and others could use the DAO optimization method.

Author Contributions: Conceptualization, M.V. and P.K.; methodology, M.V. and P.K.; software, M.V. and P.K.; validation, M.A. and Y.G.; formal analysis, M.V. and P.K.; investigation, M.A. and Y.G.; resources, P.K., M.A. and Y.G.; data curation, M.V. and P.K.; writing—original draft preparation, M.V. and P.K.; writing—review and editing, M.A. and Y.G.; visualization, M.V. and P.K.; supervision, M.A., P.K. and Y.G.; project administration, M.A. and P.K.; funding acquisition, M.A. and Y.G. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Deanship of Scientific Research, the Vice Presidency for Graduate Studies and Scientific Research, King Faisal University, Saudi Arabia (GrantA016).

Institutional Review Board Statement: Not applicable.

Data Availability Statement: Since no datasets were created or examined in the current investigation, data sharing is not relevant to this topic.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Abdel-Basset, M.; Abdel-Fatah, L.; Sangaiah, A. Metaheuristic Algorithms: A Comprehensive Review. In *Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications*; Academic Press: Cambridge, MA, USA, 2018; pp. 185–231.
2. Goldberg, D.E. *Genetic Algorithms*; Pearson Education: Bangalore, India, 2006.
3. Storn, R.; Price, K. Differential Evolution- A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *J. Glob. Optim.* **1997**, *11*, 341–359. [CrossRef]

4. Kennedy, J.; Eberhart, R. Particle Swarm Optimization. *Proc. IEEE Int. Conf. Neural Netw.* **1995**, *4*, 1942–1948.
5. Abualigah, L.; Elaziz, M.A.; Sumari, P.; Geem, Z.W.; Gandomi, A.H. Reptile Search Algorithm (RSA): A Nature-Inspired Meta-Heuristic Optimizer. *Expert Syst. Appl.* **2022**, *191*, 116158. [CrossRef]
6. Mirjalili, S.; Lewis, A. The Whale Optimization Algorithm. *Adv. Eng. Softw.* **2016**, *95*, 51–67. [CrossRef]
7. Shi, Y. Brain Storm Optimization Algorithm. In *International Conference in Swarm Intelligence*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 303–309.
8. Rao, R.; Savsani, V.; Vakharia, D. Teaching-learning based optimization: A novel method for constrained mechanical design optimization problems. *Comput. Aided Des.* **2011**, *43*, 303–315. [CrossRef]
9. Abualigah, L.; Yousri, D.; Abd Elaziz, M.; Ewees, A.A.; Al-qaness, M.A.A.; Gandomi, A.H. Aquila Optimizer: A Novel MetaHeuristic Optimization Algorithm. *Comput. Ind. Eng.* **2021**, *157*, 107250. [CrossRef]
10. Li, L.; Pan, J.; Zhuang, Z.; Chu, S. A Novel Feature Selection Algorithm Based on Aquila Optimizer for COVID-19 Classification. In *International Conference on Intelligent Information Processing*; Springer International Publishing: Cham, Switzerland, 2022; pp. 30–41.
11. Chaudhari, S.V.; Dhipa, M.; Ayoub, S.; Gayathri, B.; Siva, M.; Banupriya, V. Modified Aquila Optimization based Route Planning Model for Unmanned Aerial Vehicles Networks. In Proceedings of the 2022 International Conference on Automation, Computing and Renewable Systems (ICACRS), Pudukkottai, India, 13–15 December 2022; pp. 370–375.
12. Abualigah, L.; Elaziz, M.A.; Khodadadi, N.; Forestiero, A.; Jia, H.; Gandomi, A.H. Aquila Optimizer Based PSO Swarm Intelligence for IoT Task Scheduling Application in Cloud Computing. In *Part of the Studies in Computational Intelligence Book Series*; Springer International Publishing: Cham, Switzerland, 2022; Volume 1038, pp. 481–497.
13. Wolpert, D.H.; Macready, W.G. No Free Lunch Theorems for Optimization. *IEEE Trans. Evol. Comput.* **1997**, *1*, 67–82. [CrossRef]
14. Sasmal, B.; Hussien, A.G.; Das, A.; Dhal, K.G. A Comprehensive Survey on Aquila Optimizer. *Arch. Comput. Methods Eng.* **2023**, *30*, 4449–4476. [CrossRef]
15. Xu, Y.; Yang, Z.; Li, X.; Kang, H.; Yang, X. Dynamic opposite learning enhanced teaching–learning-based optimization. *Knowl. Based Syst.* **2020**, *104966*, 188. [CrossRef]
16. Dong, H.; Xu, Y.; Li, X.; Yang, Z.; Zou, C. An improved antlion optimizer with dynamic random walk and dynamic opposite learning. *Knowl. Based Syst.* **2021**, *106752*, 216. [CrossRef]
17. Rahnamayan, S.; Tizhoosh, H.R.; Salama, M.M.A. Quasi-oppositional differential evolution. In Proceedings of the 2007 IEEE Congress on Evolutionary Computation, Singapore, 25–28 September 2007; pp. 2229–2236. [CrossRef]
18. Ergezer, M.; Simon, D.; Du, D. Oppositional biogeography-based optimization. In Proceedings of the 2009 IEEE International Conference on Systems, Man and Cybernetics, San Antonio, TX, USA, 11–14 October 2009; IEEE: Piscataway, NJ, USA, 2009; pp. 1009–1014. [CrossRef]
19. Zhou, J.; Zhang, Y.; Guo, Y.; Feng, W.; Menhas, M.; Zhang, Y. Parameters Identification of Battery Model Using a Novel Differential Evolution Algorithm Variant. *Front. Energy Res.* **2022**, *10*, 794732. [CrossRef]
20. Liu, Z.H.; Wei, H.L.; Li, X.H.; Liu, K.; Zhong, Q.C. Global identification of electrical and mechanical parameters in PMSM drive based on dynamic self-learning PSO. *IEEE Trans. Power Electron.* **2018**, *33*, 10858–10871. [CrossRef]
21. Tizhoosh, H.R. 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), Vienna, Austria, 22 May 2006*; IEEE: Piscataway, NJ, USA, 2005; Volume 1, pp. 695–701.
22. Mohamed, A.; Abualigah, L.; Alburaikan, A.; Khalifa, H.A.E.-W. AOEHO: A New Hybrid Data Replication Method in Fog Computing for IoT Application. *Sensors* **2023**, *23*, 2189. [CrossRef]
23. Nirmalapriya, G.; Agalya, V.; Regunathan, R.; Belsam Jeba Ananth, M. Fractional Aquila Spider Monkey Optimization Based Deep Learning Network for Classification of Brain Tumor. *Biomed. Signal Process. Control.* **2023**, *79*, 104017. [CrossRef]
24. Perumalla, S.; Chatterjee, S.; Kumar, A.P.S. Modelling of Oppositional Aquila Optimizer with Machine Learning Enabled Secure Access Control in Internet of Drones Environment. *Theor. Comput. Sci.* **2023**, *941*, 39–54. [CrossRef]
25. Duan, J.; Zuo, H.; Bai, Y.; Chang, M.; Chen, X.; Wang, W.; Ma, L.; Chen, B. A Multistep Short-Term Solar Radiation Forecasting Model Using Fully Convolutional Neural Networks and Chaotic Aquila Optimization Combining WRF-Solar Model Results. *Energy* **2023**, *271*, 126980. [CrossRef]
26. Ramamoorthy, R.; Ranganathan, R.; Ramu, S. An Improved Aquila Optimization with Fuzzy Model Based Energy Efficient Cluster Routing Protocol for Wireless Sensor Networks. *Yanbu J. Eng. Sci.* **2022**, *19*, 51–61. [CrossRef]
27. Huang, C.; Huang, J.; Jia, Y.; Xu, J. A Hybrid Aquila Optimizer and Its K-Means Clustering Optimization. *Trans. Inst. Meas. Control* **2023**, *45*, 557–572. [CrossRef]
28. Zhang, Y.; Xu, X.; Zhang, N.; Zhang, K.; Dong, W.; Li, X. Adaptive Aquila Optimizer combining niche thought with dispersed chaotic swarm. *Sensors* **2023**, *23*, 755. [CrossRef] [PubMed]
29. Ekinci, S.; Izci, D.; Abualigah, L. A Novel Balanced Aquila Optimizer Using Random Learning and Nelder–Mead Simplex Search Mechanisms for Air–Fuel Ratio System Control. *J. Braz. Soc. Mech. Sci. Eng.* **2023**, *45*, 68. [CrossRef]
30. Alangari, S.; Obayya, M.; Gaddah, A.; Yafoz, A.; Alsini, R.; Alghushairy, O.; Ashour, A.; Motwakel, A. Wavelet Mutation with Aquila Optimization-Based Routing Protocol for Energy-Aware Wireless Communication. *Sensors* **2022**, *22*, 8508. [CrossRef] [PubMed]
31. Das, T.; Roy, R.; Mandal, K.K. A Novel Weighted Adaptive Aquila Optimizer Technique for Solving the Optimal Reactive Power Dispatch Problem. *Researchsquare*, 2022; *preprint*.

32. Bas, E. Binary Aquila Optimizer for 0–1 Knapsack Problems. *Eng. Appl. Artif. Intell.* **2023**, *118*, 105592. [CrossRef]
33. Long, H.; Liu, S.; Chen, T.; Tan, H.; Wei, J.; Zhang, C.; Chen, W. Optimal reactive power dispatch based on multi-strategy improved Aquila optimization algorithm. *IAENG Int. J. Comput. Sci.* **2022**, *49*, 4.
34. Wang, Y.; Jin, C.; Li, Q.; Hu, T.; Xu, Y.; Chen, C.; Zhang, Y.; Yang, Z. A Dynamic Opposite Learning-Assisted Grey Wolf Optimizer. *Symmetry* **2022**, *14*, 1871. [CrossRef]
35. Cao, D.; Xu, Y.; Yang, Z.; Dong, H.; Li, X. An enhanced whale optimization algorithm with improved dynamic opposite learning and adaptive inertia weight strategy. *Complex Intell. Syst.* **2023**, *9*, 767–795. [CrossRef]
36. Sharma, S.; Kaur, M.; Sing, B. A Self-adaptive Bald Eagle Search optimization algorithm with dynamic opposition-based learning for global optimization problems. *Expert Syst.* **2023**, *40*, e13170. [CrossRef]
37. Wang, Y.; Xiao, Y.; Guo, Y.; Li, J. Dynamic Chaotic Opposition-Based Learning-Driven Hybrid Aquila Optimizer and Artificial Rabbits Optimization Algorithm: Framework and Applications. *Processes* **2022**, *10*, 2703. [CrossRef]
38. Ali, M.H.; Salawudeen, A.T.; Kamel, S.; Salau, H.B.; Habil, M.; Shouran, M. Single- and Multi-Objective Modified Aquila Optimizer for Optimal Multiple Renewable Energy Resources in Distribution Network. *Mathematics* **2022**, *10*, 2129. [CrossRef]
39. Saremi, S.; Mirjalili, S.; Lewis, A. Grasshopper Optimisation Algorithm: Theory and Application. *Adv. Eng. Softw.* **2017**, *105*, 30–47. [CrossRef]
40. García, S.; Molina, D.; Lozano, M.; Herrera, F. A Study on the Use of Non-Parametric Tests for Analyzing the Evolutionary Algorithms' Behaviour: A Case Study on the CEC'2005 Special Session on Real Parameter Optimization. *J. Heuristics* **2009**, *15*, 617–644. [CrossRef]
41. García, S.; Fernández, A.; Luengo, J.; Herrera, F. Advanced Nonparametric Tests for Multiple Comparisons in the Design of Experiments in Computational Intelligence and Data Mining: Experimental Analysis of Power. *Inf. Sci.* **2010**, *180*, 2044–2064. [CrossRef]
42. Luengo, J.; García, S.; Herrera, F. A Study on the Use of Statistical Tests for Experimentation with Neural Networks: Analysis of Parametric Test Conditions and Non-Parametric Tests. *Expert Syst. Appl.* **2009**, *36*, 7798–7808. [CrossRef]
43. Wu, G.; Mallipeddi, R.; Suganthan, P.N. *Problem Definitions and Evaluation Criteria for the CEC 2017 Competition and Special Session on Constrained Single Objective Real-Parameter Optimization*; National University of Defense Technology: Changsha, China; Kyungpook National University: Daegu, Republic of Korea; Nanyang Technological University: Singapore, 2016.
44. Mirjalili, S.; Gandomi, A.H.; Mirjalili, S.Z.; Saremi, S.; Faris, H.; Mirjalili, S.M. Salp Swarm Algorithm: A Bio-Inspired Optimizer for Engineering Design Problems. *Adv. Eng. Softw.* **2017**, *114*, 163–191. [CrossRef]
45. Ahmadianfar, I.; Asghar Heidari, A.; Noshadian, S.; Chen, H.; Gandomi, A.H. INFO: An efficient optimization algorithm based on weighted mean of vectors. *Expert Syst. Appl.* **2022**, *116516*, 195. [CrossRef]
46. Jing-Chang, L.; Qu, B.; Suganthan, P. Problem Definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization, Computer science. *Mathematics* **2014**, *635*, 2014.
47. Varshney, M.; Kumar, P.; Ali, M.; Gulzar, Y. Using the Grey Wolf Aquila Synergistic Algorithm for Design Problems in structural Engineering. *Biomimetics* **2024**, *9*, 54. [CrossRef]
48. Jia, H.; Rao, H.; Wen, C.; Mirjalili, S. Crayfish Optimization Algorithm. *Artif. Intell.* **2023**, *56*, 1919–1979. [CrossRef]
49. Jia, H.; Peng, X.; Lang, C. Remora Optimization Algorithm. *Expert Syst. Appl.* **2021**, *185*, 115665. [CrossRef]
50. Mirjalili, S.; Mirjalili, S.M.; Lewis, A. Grey Wolf Optimizer. *Adv. Eng. Softw.* **2014**, *69*, 46–61. [CrossRef]
51. Mirjalili, S. SCA: A Sine Cosine Algorithm for Solving Optimization Problems. *Knowl. Based Syst.* **2016**, *96*, 120–133. [CrossRef]