Applied Engineering Letters **Vol.10, No.4, 193-210 (2025)** **e-ISSN: 2466-4847**

BTO: A BARREL THEORY-BASED OPTIMIZER FOR ENGINEERING DESIGN PROBLEMS

UDC:510.5:658.512.2
Original scientific paper https://doi.org/10.46793/aeletters.2025.10.4.2

Van Tai Tran[1,2], Quynh T.T. Nhu[1,2], Chitsutha Soomlek[1], Punyaphol Horata[1], Khamron Sunat[1*]

[1]College of Computing, Khon Kaen University, Khon Kaen, Thailand
[2]School of Computing and Information Technology, Eastern International University, Ho Chi Minh, Vietnam

Abstract:
This study introduces the Barrel Theory-based Optimizer (BTO), a novel metaheuristic algorithm inspired by the wooden barrel theory, where the weakest component constrains overall performance. In BTO, each solution is viewed as a barrel and each variable as a plank. Low-fitness solutions, which can be seen as the limiting planks, are updated frequently via a population-level adjustment strategy called Barrel Adjustment. As a result, the overall search capability improves. Besides, BTO uses an adaptive elite selection mechanism that gradually adjusts the number of elite solutions. It enables a smooth transition from exploration to exploitation. The elite set further guides directional updates with a gradually decreasing disturbance factor. The performance of BTO was tested on three groups of problems, including CEC2022 benchmark functions, classical benchmarks, and eight well-known engineering design problems from mechanical and structural engineering. Experimental results show that it achieves higher solution quality, faster convergence, and more stable performance than well-known algorithms, including Particle Swarm Optimization (PSO) and Grey Wolf Optimizer (GWO). These findings establish BTO as a reliable and effective algorithm for complex and real-world engineering optimization tasks.

ARTICLE HISTORY
Received: 11 September 2025
Revised: 1 November 2025
Accepted: 20 November 2025
Published: 15 December 2025

KEYWORDS
Metaheuristic optimization, Barrel theory, Adaptive strategy, Engineering design problems, Mechanical and structural design problems

1. INTRODUCTION

In the engineering design industry, there are many difficult problems to solve, as their objective functions are multimodal. They contain many local optima, which can trap optimization algorithms in local solutions and prevent them from achieving the global optimum. Such problems are usually considered black-box functions, and they need significant computational effort. This is because each evaluation might require complicated simulations or many models. This demand of heavy computational models restricts the number of solutions used for exploration. Traditional optimization methods, such as gradient-based algorithms, do not perform well in these cases. They need smooth and continuous functions to compute gradients and excessively depend on the starting point. As a result, they may converge too early and explore the search space poorly [1-5].

These challenges are common in engineering design problems such as pressure vessels, speed reducers, and truss structure optimization, where the search space is nonlinear and highly constrained. To handle such problems effectively, researchers often employ metaheuristic algorithms because of their flexibility and robustness under complex engineering conditions.

For these reasons, many researchers have turned to metaheuristic algorithms, which are more flexible and better suited for solving complex and expensive optimization problems. Metaheuristic algorithms perform well in difficult, nonlinear, or black-box problems because they usually don't require

*CONTACT: Khamron Sunat, e-mail: skhamron@kku.ac.th

gradient information. They employ smart strategies to progressively refine their candidate solutions as they search the space. Instead of using exact mathematical models, they follow basic rules inspired by nature, social behavior, or physical processes to obtain good solutions [6].

Metaheuristic algorithms are divided into several main types depending on their inspiration and search strategy.

Evolutionary algorithms are inspired by natural evolution. They improve solutions with time using ideas such as selection, crossover, and mutation. Well-known examples include Evolution Strategy (ES, 1964) [7], Genetic Algorithm (GA, 1975) [8], and Differential Evolution (DE, 1997) [9]. More recent algorithms in this group include Atom Search Optimization (ASO, 2019) [10] and Artemisinin Optimizer (AO, 2024) [11], Differentiated Creative Search (DCS, 2024) [12].

Swarm-based approaches imitate the behavior of animal groups like bird flocks or insect colonies when searching for food or moving together. Early representatives of this group are Particle Swarm Optimization (PSO, 1995) [13] and Ant Colony Optimization (ACO, 1996) [14], which laid the foundation for swarm intelligence. More recent methods are Grey Wolf Optimizer (GWO, 2014) [15], Sine Cosine Algorithm (SCA, 2016) [16], Whale Optimization Algorithm (WOA, 2016) [17], Slime Mould Algorithm (SMA, 2020) [18], Golden Jackal Optimization (GJO, 2022) [5], and Secretary Bird Optimization Algorithm (SBOA, 2024) [19].

Physics-based algorithms are models of physical processes, such as thermal annealing or gravitational attraction, to guide the search. Famous examples include Simulated Annealing (SA, 1983) [20]. More recent contributions, such as Physics-based Optimization (RIME, 2023) [21], show that the research direction of physics-inspired heuristics is still actively evolving.

Social or behavior-based algorithms are inspired by human or animal learning, including Social Network Search (SNS, 2021) [4], Human Memory Optimization Algorithm (HMOA, 2024) [22], and Escape Algorithm (ESC, 2024) [23].

To the best of the authors' knowledge, many established metaheuristic algorithms (e.g., AO, SCA, AOA, PSO, and in some cases GJO) have performance instability with relatively large standard deviations (STD). In some engineering problems, such as the Gear Train design, these methods fail entirely. They create solutions that deviate significantly from the feasible optimum. It shows the lack of robustness through different design tasks.

Additionally, algorithms like PSO, AO, and AOA are susceptible to premature convergence. They are often trapped in local optima without effective recovery, as shown in their poor mean results. Several algorithms also have limited generalization ability. For example, the GWO has a great performance in the Pressure Vessel and Gear Train designs, but it does not work well in the Piston Lever and Car Side Impact problems. Therefore, these algorithms are less applicable in real-world engineering, where problems can vary a lot.

Finally, existing methods often have an imbalance between exploration and exploitation. Because engineering design problems are generally nonlinear, constrained, and multimodal, excessive exploration leads to unstable behavior. Meanwhile, excessive exploitation causes premature convergence. The experimental results reinforce this imbalance, where high variability (STD) reflects over-exploration, and suboptimal mean performance indicates stagnation due to insufficient exploration.

To solve these limitations, BTO has a new search mechanism, which is inspired by barrel theory. This method makes a dynamic balance between exploration and exploitation. The design is expected to have consistently better performance, and BTO can achieve top rankings in most test problems with very low STD values and stable convergence characteristics. Besides, BTO is anticipated to not only provide improved best solutions but also ensure robust mean performance. Thus, it can address the performance gap in many existing algorithms.

The remainder of this paper is organized as follows. Section 2 describes the proposed BTO algorithm and its main components. Section 3 presents the experimental setup and results on benchmark and engineering problems. Finally, Section 4 is the conclusion.

2. PROPOSED METHOD

This part presents the Barrel Theory-based Optimizer (BTO) algorithm. Section 2.1 is the core concept of BTO, Section 2.2 outlines its components, Section 2.3 is the complete algorithm in pseudocode, and Section 2.4 is its time complexity.

2.1 Overview of BTO

The Barrel Theory-based Optimizer (BTO) is inspired by the barrel principle. In this analogy, the barrel is the population, and each plank is a candidate solution. Picture a wooden barrel with many vertical planks. The water level of the barrel is limited by its shortest plank. In a similar way, the weakest solutions in the population restrict the search progress. To solve this, BTO updates low-quality solutions more frequently to improve them. Meanwhile, stronger solutions are changed less to preserve their quality (Fig. 1).

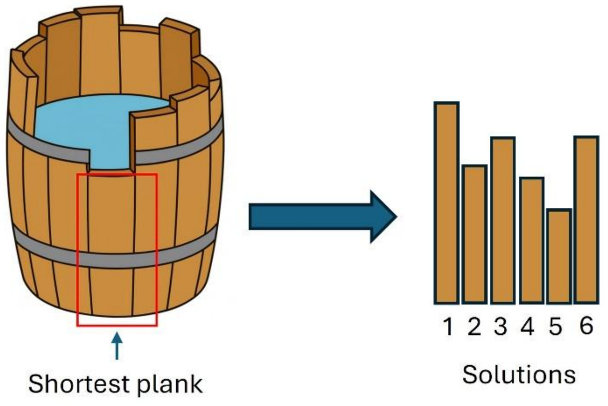

Fig. 1. Illustration of the barrel principle in BTO, where the shortest plank represents the weakest solution limiting the population.

This principle has been applied in many fields. In agriculture, Liebig's Law of the Minimum states that plant growth depends on the nutrient in the shortest supply [24]. According to the Theory of Constraints, a system's performance in business is determined by its weakest component [25]. In systems thinking, small changes at the right leverage points can make significant improvements [26]. Similar ideas are in Lean [27] and Six Sigma [28], each of which focuses on identifying and eliminating bottlenecks.

In BTO, each solution is treated as a barrel, and each variable is a plank. The algorithm identifies the variables that limit a solution and updates them more frequently. In other words, low-fitness solutions are updated more, while high-fitness solutions are changed less. Furthermore, BTO employs two approaches: one learns from the best solutions, and the other explores new directions. Together, they help the algorithm find better solutions efficiently and avoid premature stagnation (Fig. 2).

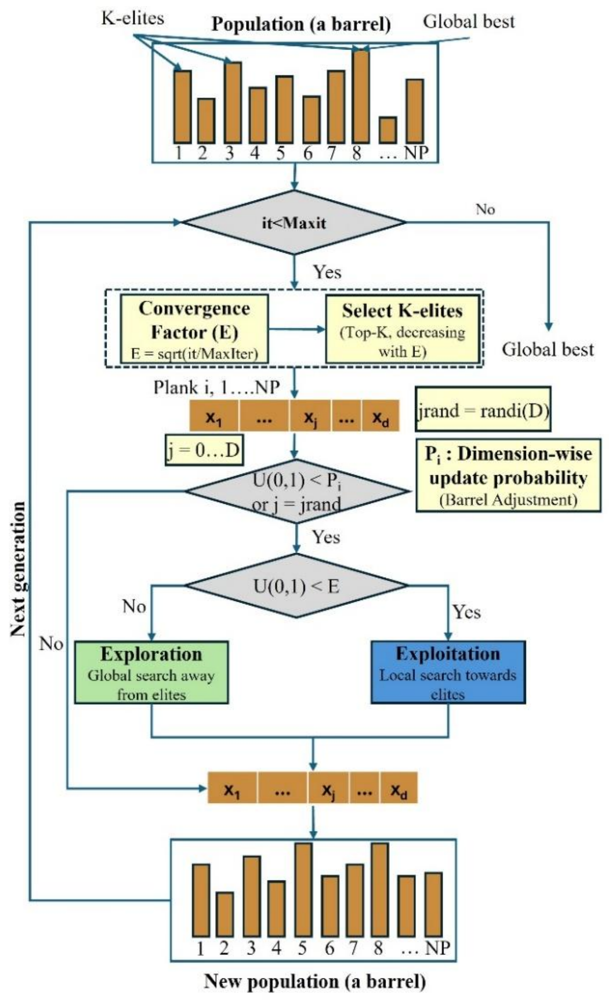

Fig. 2. Flowchart of the BTO algorithm

2.2 BTO Components

The proposed BTO algorithm follows the standard iterative structure of population-based metaheuristics. It includes steps such as population initialization, fitness evaluation, convergence control, elite selection, solution-wise update probability, boundary handling, and selection. The main steps are described as follows:

Step 1: Population Initialization

Each candidate solution $X_i \in R^D$ is initialized randomly within a lower bound LB_j and an upper bound UB_j.

$$X_{i,j} \sim \mathcal{U}(LB_j, UB_j), \qquad \forall i = 1,2,\dots,NP, \quad j = 1,2,\dots,D \tag{1}$$

where NP is the population size and D is the problem dimensionality. $\mathcal{U}(LB_j, UB_j)$ denotes the uniform distribution over the interval $[LB_j, UB_j]$, meaning that values within this range are equally likely to be selected.

Step 2: Fitness Evaluation

The fitness of each individual is evaluated using the objective function:

$$f(X_i), \quad \forall i = 1, \dots, NP \tag{2}$$

For a minimization problem, the individual with the lowest fitness is considered the best solution.

Step 3: Convergence Factor

The main principle is a balance between exploration and exploitation. In metaheuristic algorithms, there is an important trade-off between exploration and exploitation.

Exploration helps the algorithm check different and promising regions of the search space. This decreases the chance of convergence too early. At the same time, exploitation works on improving the current best solutions to have optimal or near-optimal results faster.

This balance changes with time:

At the start of iterations, more exploration is done to search widely and avoid getting stuck in local optima.

In later iterations, exploitation becomes dominant to fine-tune and polish the solutions discovered.

The selection is either exploration (global search) or exploitation (local refinement) through a convergence factor E, which is computed as:

$$E = \left(\sqrt{\frac{t}{T}}\right)^{p} \tag{3}$$

where t is the current iteration number and T is the total number of iterations. As the algorithm progresses (i.e., t increases), the convergence factor E also increases (Fig. 3). For p is 1, the marked point at $E = 0.5$, corresponding to a quarter of the total iterations, and reflects the algorithm's gradual transition from exploration to exploitation. The mean of $\mathrm{U}(0,1) < E$ is approximately 0.667 or 2/3. This implies that BTO conducts exploitation searches with a probability of two-thirds, while allocating the remaining one-third probability to exploration searches. Similar ratios are observed in natural and learning systems. In foraging theory, animals often spend about 60–70% of their effort exploiting known food patches before exploring new ones [29]. In reinforcement learning, ε-greedy strategies also dedicate most of the time (≈70%) to exploitation while reserving a smaller share (≈30%) for exploration [30].

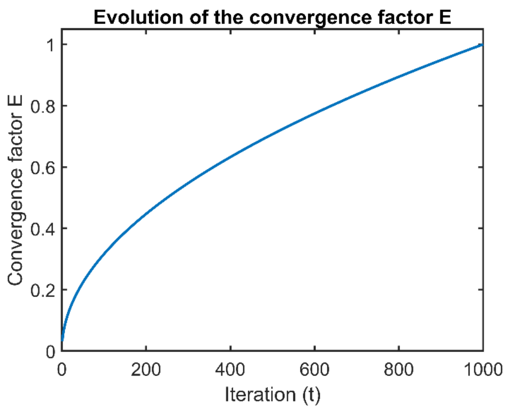

Fig. 3. Evolution of the convergence factor $\boldsymbol{E}$

Step 4: Elite Set Selection

In each iteration, a group of the best individuals, called the elite set ε, is selected from the population to help guide the update of other individuals. The number of elite members K depends on the convergence factor E:

$$K = \max(1, \lfloor \beta \cdot (1 - E) \rfloor), \tag{4}$$

where β is the maximum number of elite individuals ($\beta = 10$ in this study). As the optimization progresses (i.e., as E increases), the number of elite individuals K gradually decreases, allowing the algorithm to shift its focus from global exploration to local exploitation (Fig. 4). The elite set ε, consisting of the top K individuals with the best fitness values in the current population, is determined as follows:

$$\varepsilon = \text{Top-}K(\{X_i\}_{i=1}^{NP}), \tag{5}$$

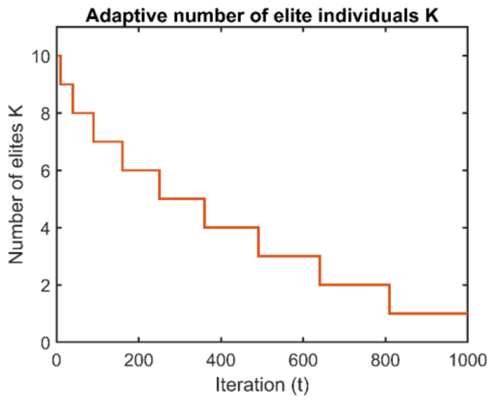

Fig. 4. Adaptive number of elite individuals $\boldsymbol{K}$

At the beginning stage of the search, many elite individuals are selected to expand the search space and increase global exploration. As the optimization progresses, the number of elites decreases slowly. This helps the algorithm concentrate on the most promising regions. In the final stage, only the best solution leads the population to precise convergence. This adaptive selection strategy helps BTO maintain a good balance between exploration and exploitation. At the same time, it can also reduce the risk of premature convergence.

Step 5: Dimension-wise Update Probability of Individual (Barrel Adjustment)

For determining the frequency of dimension updates in each solution, the BTO algorithm first assigns a probability value P_i to each individual. This is based on the barrel theory. Individuals having the lower fitness (leakier barrels) are assigned higher update probabilities, so their dimensions are more likely to be updated. In contrast, stronger individuals are updated less often to maintain stability. The update probability is computed as:

$$P_i = \lambda_{\min} + (\lambda_{\max} - \lambda_{\min}) \cdot f_i^{\text{norm}} \tag{6}$$

In this context, $\lambda_{\min}$ and $\lambda_{\max}$ represent the lower and upper bounds of the dynamic update probability range, which governs how frequently solution variables are modified during the search process.

In this formula, the term f_i^{norm} is the normalized fitness of the *i*-th individual. It shows how good that individual is relative to the current population on a scale of $[0, 1]$. This normalization formula is as:

$$f_i^{\text{norm}} = \frac{f_i - f_{min}}{f_{max} - f_{min}}, \tag{7}$$

where f_i is the raw fitness of the *i*-th individual. f_{min} and f_{max} are the minimum and maximum fitness values in the current population, respectively. This normalization makes sure that individuals with worse fitness (closer to f_{max}) have f_i^{norm} going to 1, and the best-performing individuals (closer to f_{min}) have f_i^{norm} going to 0, which fits the barrel adjustment logic.

The probability P_i is then scaled within a dynamic probability range controlled by two thresholds: λ_{min} and λ_{max}. These thresholds change dynamically based on the current iteration t, total iterations T, problem dimension D, and a scaling parameter η. Specifically, the minimum and maximum update probabilities are defined as:

$$\begin{aligned} \lambda_{min} &= \eta \cdot (1 - \sqrt{t/T}), \\ \lambda_{max} &= 1 - (1 - \eta \cdot (1 - D/\delta)) \cdot \sqrt{t/T}, \end{aligned} \tag{8}$$

where the scaling factor η is experimentally set to 0.3, and the dimensionality normalization constant δ is set to 100. As the algorithm advances, both of these thresholds decrease.

At the beginning of the search, the update probability is relatively high, allowing weaker individuals to modify more dimensions and explore new areas of the search space. In contrast, stronger individuals are updated less frequently to preserve good information and maintain population stability. As the optimization progresses, both the lower and upper probability limits gradually decrease, reducing the update frequency and promoting convergence. This adaptive strategy makes sure that at least one variable of each individual is always updated to prevent stagnation. The key idea is giving more updates to weaker solutions and fewer to stronger ones. With this principle, the algorithm improves poor individuals faster while preserving the quality of the best candidates. In short, the Barrel Adjustment mechanism helps BTO maintain diversity during the early stage and get smoother convergence in later iterations.

Step 6: Harmonic Disturbance

In this step, each individual has a harmonic disturbance vector F, which controls the update step size in all dimensions. This vector features controlled randomness based on a sine-wave pattern which changes over time. The disturbance vector F is formulated as:

$$F = 2(r_1 - 0.5) \circ \sin\left(2\pi r_2 \cdot \frac{t}{T/10}\right) \cdot \left(1 - \frac{t}{T}\right) \tag{9}$$

where r_1 and r_2 are two independently generated D –dimensional vectors of random numbers. Each component is picked from the interval $(0,1)$. The formula is then applied element-wise to have the disturbance vector F. t is the current iteration and T is the total number of iterations.

Fig. 5 shows how the disturbance vector F evolves over time with decreasing amplitude. In the implementation, this disturbance vector F is computed once per individual and is then used to update the solution vector.

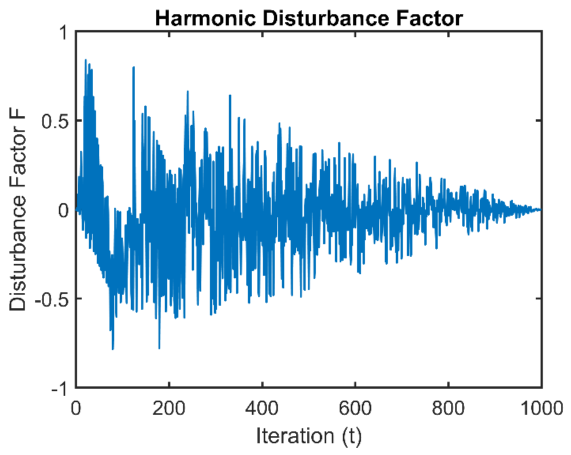

Fig. 5. Evolution of the harmonic disturbance factor $\boldsymbol{F}$ during optimization with $\boldsymbol{T = 1000}$

At the early phase of the optimization, the disturbance factor changes strongly within the range $[-1,1]$. Large update steps are produced, and it helps the algorithm to explore distant regions of the search space. When the process continues, the amplitude of the disturbance decreases. This leads to smaller and more stable movements. The controlled randomness helps BTO escape local optima in the early stage. It also helps improve fine-tuning accuracy in the later stage. Overall, the harmonic disturbance mechanism makes a smooth transition from exploration to exploitation and improves the stability of convergence.

Step 7: Plank Update Strategy

In BTO, a trial vector V_i is created for each individual X_i by updating its variables one by one. For each dimension, the algorithm decides whether to update it based on a probability P_i, or forces an update if it matches a randomly selected index j_{rand}. This mechanism ensures that at least one variable is always updated, even when all probability checks fail. Here, j_{rand} is simply a random dimension index drawn from $\{1,2,\ldots,D\}$. If the update condition is met, either Strategy A (elite-based learning) or Strategy B (Directional Adjustment) is applied, depending on the current value of the convergence factor E. An elite solution $X_{elite} \in \varepsilon$ is randomly chosen from the elite set to help guide the update.

Strategy A (elite-based learning): This strategy focuses on learning from elite individuals to guide updates more effectively. For each dimension j, the component $V_{i,j}^{(trial)}$ is updated using the following rule:

$$V_{i,j}^{(trial)} = X_{\text{sel},j}^{(t)} + F_j \cdot \left(X_{\text{elite},j}^{(t)} - X_{\text{sel},j}^{(t)}\right) + F_j \cdot \mathcal{N}(0,1).\eta_A.(1-\text{E}) \cdot \left(UB_j - LB_j\right) \tag{10}$$

In this update rule, $X_{\text{sel},j}^{(t)}$ is the current value at dimension j of a selected solution, while $X_{\text{elite},j}^{(t)}$ is the value at the same dimension from an elite solution chosen from the elite set ε. To ensure diversity, the selected individual is chosen randomly from the population, excluding the elite individual itself. This is implemented by repeatedly selecting a random index $sel = randi(NP)$ until it differs from the elite index.

The term F_j represents the harmonic disturbance factor, which helps control the update strength for each dimension. To introduce randomness and improve diversity, a value sampled from the Gaussian distribution $\mathcal{N}(0,1)$ is also added. $\eta_A \in (0,1)$ is a scaling coefficient that controls the maximum perturbation range (experimentally set, e.g., to 0.25). The terms UB_j and LB_j define the allowed range (upper and lower bounds) for dimension j, and $E \in (0,1)$ is the Convergence Factor.

Strategy B (Directional Adjustment): This strategy is designed to enhance exploration, especially at the beginning of the optimization process. For each individual X_i and each dimension j, the update is done based on a direction and step size. First, the update direction is computed.

$$\delta_{i,j} = -\left(X_{\text{elite},j}^{(t)} - X_{i,j}^{(t)} + \gamma\right) \tag{11}$$

where the correction term γ is defined as:

$$\gamma = (2 \cdot I[r < E] - 1) \cdot I\left[X_{\text{elite},j}^{(t)} = X_{i,j}^{(t)}\right] \tag{12}$$

Here, $r \sim U(0,1)$ is a uniformly distributed random number. This mechanism makes sure that even when the elite and current solution components are equal, a movement direction (either a positive one or a negative one) is still randomly assigned. This helps prevent stagnation. It also encourages continued exploration during the optimization process.

When the update direction $\delta_{i,j}$ is determined, the new value of dimension j for individual i is computed as:

$$V_{i,j}^{(\text{trial})} = X_{i,j}^{(t)} + \delta_{i,j} \cdot \left|F_j \cdot \left(UB_j - LB_j\right) \cdot \eta_B + \left(1 - |F_j|\right) \cdot E \cdot X_{i,j}^{(t)} \cdot \mathcal{N}(0,1) + \epsilon\right| \tag{13}$$

In this formulation, F_j denotes the j-th element of the disturbance vector F, which adjusts the intensity of the update, while η_B is a scaling coefficient that governs the basic step size (experimentally set, e.g., to 0.3). The parameter E is the Convergence Factor. $\mathcal{N}(0,1)$ introduces random variation through Gaussian noise. ϵ is a small constant preventing zero-magnitude updates. This update rule combines both fixed and random behaviors: the fixed component gives a baseline step based on the search range, while the random term dynamically changes the update strength according to the current solution's value and the convergence level. The direction $\delta_{i,j}$ decides whether the value along dimension j increases or decreases.

The plank update guarantees progress by forcing at least one dimension to change via j_{rand}, preventing idle iterations. The two modes are time-controlled by E: Strategy B encourages exploration in early iterations by deviating from the elite direction when needed, whereas Strategy A focuses on exploitation later by learning from elite solutions. The per-dimension disturbance F_j modulates step magnitude, yielding larger moves early and smaller, more stable moves later. Together, these choices mitigate premature convergence and stagnation while preserving convergence stability.

Step 8: Boundary Control

After the update step, some components of the trial vector $V_{i,j}$ may fall outside the valid search range defined by the lower and upper bounds LB_j and UB_j . To solve this, a boundary control mechanism is used to each dimension j of the individual i as follows:

$$V_{i,j}^{(trial)} = \begin{cases} \dfrac{\left(V_{i,j}^{(trial)} + LB_j\right)}{2}, V_{i,j}^{(trial)} < LB_j \\ \dfrac{\left(V_{i,j}^{(trial)} + UB_j\right)}{2}, V_{i,j}^{(trial)} > UB_j \\ V_{i,j}^{(trial)}, \quad otherwise \end{cases} \tag{14}$$

This correction method helps the updated solution stay within the feasible search space without abruptly clipping the values. By taking the average, the value is pulled back inside the limits, keeping useful information and continuity for smoother convergence behavior.

Step 9: Selection and Best Update

After generating the trial solution $V_i^{(trial)}$, the algorithm evaluates its quality by comparing $f(V_i^{(trial)})$with the current solution $f(X_i^{(t)})$. The selection rule is defined as:

$$X_i^{(t+1)} = \begin{cases} V_i^{(trial)}, if\ f(V_i^{(trial)}) \le f(X_i^{(t)}) \\ X_i^{(t)}, \quad otherwise \end{cases} \tag{15}$$

The best solution is updated accordingly:

$$X_{best}^{(t+1)} = \begin{cases} V_i^{(trial)}, if\ f(V_i^{(trial)}) \le f(X_{best}^{(t)}) \\ X_{best}^{(t)}, \quad otherwise \end{cases} \tag{16}$$

This approach ensures that new solutions are only accepted when they offer better or equal objective function values. It also guarantees that the global best solution is preserved and updated whenever an improvement is found.

2.3 Pseudocode

The pseudocode of the proposed BTO algorithm is presented in Algorithm 1 to summarize the main computational steps.

Algorithm 1. Pseudocode of the proposed BTO algorithm

Input: population size NP, maximum iterations T, search bounds $[LB, UB]$, dimension D

Output: Optimal solution X_{best}, best fitness f_{best}

```
1:  Initialize population using Eq. (1)
2:  Evaluate fitness using Eq. (2)
3:  Identify $X_{best}, f_{best}$
4:  for $t = 1$ to $T$ do
5:      Compute convergence factor $E$ using Eq. (3)
6:      Determine elite size $K$ and select elite set $\varepsilon$ (Eqs. (4)-(5))
7:      Compute update probabilities $P_i$ for each individual using Eq. (6)
8:      for $i = 1$ to $NP$ do
9:          Compute disturbance vector $F$ using Eq. (9)
10:         Select $elite\_idx \in \varepsilon$
11:         for $j = 1$ to $D$ do
12:             if rand < $P_i$ or j == jrand then
13:                 if rand < E then
14:                     Select $sel \neq elite\_idx$
15:                     Apply Strategy A to compute $V_{i,j}^{(trial)}$ (Eq. (10))
16:                 else
17:                     Apply Strategy B to compute $V_{i,j}^{(trial)}$ (Eqs. (11)- (13))
18:                 end if
19:             end if
20:         end for
21:     end for
22:     Apply boundary control (Eq. (14))
23:     Perform selection using Eqs. (15) - (16)
24: end for
25: return $X_{best}, f_{best}$
```

2.4 Time Complexity Analysis

The overall time complexity of the BTO algorithm is based on three main factors: NP denotes the number of individuals, D is the number of dimensions, and T is the maximum number of iterations. In each iteration, BTO evaluates the fitness of all NP individuals, selects elite solutions, and updates each variable in every solution. These steps take about $O(NP \times D)$ time per iteration. Since the process is repeated for T iterations, the total time complexity is approximately $O(NP \times D \times T)$. Other operations, such as computing adjustment probabilities and disturbance values, are simple and do not significantly affect the overall computational cost.

3. EXPERIMENTAL SETUP AND RESULTS

This section evaluates the performance of the proposed BTO algorithm through a series of experiments. First, the experimental setup is described, including benchmark problems, parameter settings, and evaluation criteria. After that, BTO is tested on the CEC2022 benchmark functions and compared with several selected well-known metaheuristic algorithms. Its performance is then assessed on classical benchmark functions to verify robustness, and on engineering design problems to evaluate its ability to handle practical, constrained optimization tasks. Finally, a parameter sensitivity analysis examines the effect of elite size on BTO's performance.

3.1 Experimental Setup

Three groups of benchmark problems were used to evaluate the performance of the proposed algorithm.

CEC2022 Benchmark Functions [31]: 12 functions from the CEC2022 set were tested in 20 dimensions in the first group. These are challenging problems often used in global optimization, with features like shifting, rotation, and complex combinations.

Classical Benchmark Functions [32]: 10 classical functions, such as Sphere, Rastrigin, and Ackley, were included in the second group. These functions are useful for evaluating convergence speed and solution accuracy.

Real-World Engineering Problems [4]: 8 real-world engineering problems with nonlinear objectives and constraints were tested in the third group. This is to examine the algorithm's ability to find good feasible solutions in practical conditions. or these constrained problems, only feasible solutions were considered, following the practice in Bayzidi et al. [4]. Infeasible solutions were discarded, which may lead to "NA" entries in the result tables when no feasible solution was found in a run.

Together, these three groups cover both standard mathematical tests and practical optimization tasks.

All algorithms were implemented in MATLAB R2024b and run on a computer with an Intel(R) Core(TM) i9-7980XE CPU @ 2.60 GHz, 64 GB RAM on Windows 11 Pro (64-bit). The population size was set to 30 for all problems. The maximum number of function evaluations depended on the problem type: 50,000 × D for CEC2022, 10,000 × D for classical functions, and 9000 for engineering problems, where D is the problem dimension (D = 20 for CEC2022 and D= 100 for classical functions). On average, 52 independent runs were performed for the CEC2022 and classical benchmark problems, and 104 runs for the engineering problems.

Ten well-known metaheuristic algorithms from 1995 to 2024 were used as benchmarks to compare with the proposed algorithm. They include Particle Swarm Optimization (PSO, 1995) [13], Differential Evolution (DE, 1997) [9], Grey Wolf Optimizer (GWO, 2014) [15], Moth-Flame Optimization (MFO, 2015) [33], Whale Optimization Algorithm (WOA, 2016) [17], Sine Cosine Algorithm (SCA, 2016) [16], Golden Jackal Optimization (GJO, 2022) [5], Arithmetic Optimization Algorithm (AOA, 2021) [34], Physics-based Optimization (RIME, 2023) [21], and the recently introduced Artemisinin Optimizer (AO, 2024) [11]. These algorithms range from classical mathematical models to modern bio-inspired strategies. All methods were implemented in MATLAB and tested under identical population sizes and stopping criteria to ensure fair comparison. The control parameters were taken from the original papers or kept at default values, as shown in

Table 1.

Table 1. Parameter Settings of Compared Algorithms

Abb	Parameter(s)	Value(s)
GWO	–	–
WOA	Φ, B	1,1
GJO	ϕ, λ	1.5 * (1 - t), 1.5
MFO	b	1
AO	ϕ, λ	exp(-4 * t), 0.6
SCA	a	2
AOA	μ, α	0.499, 5
PSO	c1, c2, w	2, 2, 1
RIME	w	5
DE	F, CR	0.5, 0.9

3.2 Results on CEC2022 Functions

The comparison results of BTO with the other algorithms are in

Table 2, with the best results for each function in bold. In most cases, BTO consistently outperformed the competing algorithms. Specifically, it had the lowest average rank (1.08) and ranked first in 11 of 12 functions. Furthermore, its performance was stable with relatively low standard deviation (STD) values on many functions. This stability indicates the algorithm's strong global search ability and good convergence for both simple and complex problems.

To statistically check these performance differences, the Friedman test was applied to all algorithms. The *p*-value was below 0.05, confirming that the performance differences were statistically significant. BTO had the best overall rank among the tested methods.

Table 2. Results of algorithms on CEC2022 benchmark functions (20-D)

Fun.	Type	BTO	GWO	WOA	GJO	MFO	AO	SCA	AOA	PSO	RIME	DE
F1	Mean	**3.00E+02**	6.82E+03	3.26E+02	1.13E+04	3.15E+04	3.28E+02	4.90E+03	1.56E+04	6.09E+04	3.00E+02	3.06E+02
	STD	**1.07E-05**	3.64E+03	2.70E+01	4.40E+03	2.28E+04	8.13E+01	1.27E+03	5.20E+03	2.33E+04	2.57E-04	2.09E+01
	Rank	**1**	7	4	8	10	5	6	9	11	2	3
F2	Mean	**4.20E+02**	4.89E+02	4.61E+02	5.63E+02	5.66E+02	4.53E+02	5.81E+02	7.21E+02	1.30E+03	4.46E+02	4.39E+02
	STD	**1.38E+01**	3.75E+01	2.13E+01	6.98E+01	1.69E+02	1.75E+01	2.88E+01	1.08E+02	3.77E+02	1.76E+01	2.14E+01
	Rank	**1**	6	5	7	8	4	9	10	11	3	2
F3	Mean	**6.00E+02**	6.03E+02	6.59E+02	6.20E+02	6.22E+02	6.00E+02	6.30E+02	6.55E+02	6.56E+02	6.00E+02	6.00E+02
	STD	**1.36E-06**	2.66E+00	1.44E+01	7.91E+00	9.06E+00	2.77E-01	3.87E+00	5.78E+00	7.45E+00	1.57E-02	1.40E-01
	Rank	**1**	5	11	6	7	4	8	9	10	2	3
F4	Mean	8.22E+02	8.47E+02	9.14E+02	8.78E+02	8.99E+02	**8.21E+02**	9.16E+02	8.82E+02	9.84E+02	8.55E+02	8.25E+02
	STD	**4.37E+00**	1.73E+01	2.89E+01	1.47E+01	3.14E+01	7.17E+00	1.14E+01	1.87E+01	1.16E+01	1.79E+01	1.11E+01
	Rank	2	4	9	6	8	**1**	10	7	11	5	3
F5	Mean	**9.02E+02**	1.08E+03	2.86E+03	1.55E+03	2.76E+03	9.08E+02	1.60E+03	2.42E+03	6.10E+03	9.26E+02	9.04E+02
	STD	**2.27E+00**	1.73E+02	8.84E+02	2.68E+02	8.81E+02	7.43E+00	2.12E+02	2.79E+02	1.52E+03	7.29E+01	8.62E+00
	Rank	**1**	5	10	6	9	3	7	8	11	4	2
F6	Mean	**3.44E+03**	6.20E+05	7.36E+03	8.99E+06	5.85E+06	3.98E+04	4.78E+07	6.67E+03	1.81E+08	7.47E+03	6.93E+03
	STD	**1.30E+03**	1.55E+06	5.97E+03	2.39E+07	1.20E+07	5.55E+04	2.50E+07	2.74E+03	4.09E+08	5.38E+03	5.54E+03
	Rank	**1**	7	4	9	8	6	10	2	11	5	3
F7	Mean	**2.02E+03**	2.06E+03	2.16E+03	2.09E+03	2.13E+03	2.07E+03	2.10E+03	2.19E+03	2.17E+03	2.05E+03	2.03E+03
	STD	**3.09E+00**	3.53E+01	4.49E+01	3.38E+01	6.18E+01	4.53E+01	1.38E+01	5.97E+01	5.25E+01	3.56E+01	1.56E+01
	Rank	**1**	4	9	6	8	5	7	11	10	3	2
F8	Mean	**2.22E+03**	2.24E+03	2.24E+03	2.24E+03	2.26E+03	2.23E+03	2.24E+03	2.40E+03	2.33E+03	2.23E+03	2.22E+03
	STD	**1.82E-01**	3.15E+01	1.08E+01	2.34E+01	4.86E+01	2.79E+01	4.19E+00	1.30E+02	8.06E+01	2.82E+01	4.14E+00
	Rank	**1**	5	7	6	9	4	8	11	10	3	2
F9	Mean	**2.48E+03**	2.50E+03	2.48E+03	2.55E+03	2.50E+03	2.49E+03	2.52E+03	2.64E+03	2.73E+03	2.48E+03	2.48E+03
	STD	**4.20E-06**	2.11E+01	4.36E-01	3.50E+01	2.45E+01	7.79E+00	1.34E+01	3.34E+01	1.38E+02	2.18E-04	2.46E+00
	Rank	**1**	7	3	9	6	5	8	10	11	2	4
F10	Mean	**2.44E+03**	3.07E+03	3.62E+03	3.01E+03	4.11E+03	2.59E+03	2.52E+03	4.21E+03	5.14E+03	2.52E+03	2.73E+03
	STD	**4.88E+01**	5.51E+02	9.82E+02	9.02E+02	1.01E+03	1.17E+02	4.54E+01	7.57E+02	1.03E+03	9.13E+01	2.87E+02
	Rank	**1**	7	8	6	9	4	3	10	11	2	5
F11	Mean	**2.83E+03**	3.45E+03	3.02E+03	4.18E+03	4.36E+03	2.92E+03	4.08E+03	6.19E+03	6.30E+03	2.92E+03	2.97E+03
	STD	**1.10E+02**	3.85E+02	7.12E+02	5.61E+02	8.97E+02	6.83E+01	3.52E+02	7.99E+02	1.41E+03	6.27E+01	9.70E+01
	Rank	**1**	6	5	8	9	3	7	10	11	2	4
F12	Mean	**2.94E+03**	2.96E+03	3.01E+03	3.01E+03	2.96E+03	2.97E+03	3.00E+03	3.60E+03	3.07E+03	2.96E+03	2.95E+03
	STD	**1.92E+00**	1.48E+01	7.03E+01	4.55E+01	1.50E+01	2.22E+01	1.44E+01	1.78E+02	9.00E+01	1.33E+01	1.34E+01
	Rank	**1**	5	9	8	4	6	7	11	10	3	2
Mean	**Rank**	**1.08**	5.67	7.00	7.08	7.92	4.17	7.50	9.00	10.67	3.00	2.92

Fig. 6 shows the convergence curves of BTO and other algorithms on different benchmark functions. These curves show how the best solution develops. In Fig. 6, BTO generally converged faster and got better solutions than the other algorithms in most tests. For other algorithms, they stopped improving early, particularly for functions F5 and F6. This showed possible entrapment in local optima. On the other hand, BTO continued to improve. It could escape local optima and keep searching for better solutions. This behavior showed BTO's ability to have a good balance between exploration and exploitation, making it effective for hard optimization problems.

To thoroughly understand the performance of BTO, Fig. 7 shows boxplots of the best results from different algorithms on six benchmark functions. These boxplots display both the solution quality and the stability of each algorithm via multiple runs.

In general, BTO had lower best scores and smaller box sizes on most functions, such as F1, F5, and F6. This indicates that it found better solutions with more consistent performance than the other algorithms. Although many algorithms exhibited large variations on functions F2 and F10, BTO still had stable results. These findings show that BTO is effective and reliable for complex optimization problems.

The Wilcoxon signed-rank test was conducted to compare BTO with other popular metaheuristic algorithms on the CEC 2022 test set with 20 dimensions. As shown in Table 3, BTO achieved wins on all 12 test functions against many algorithms, including GWO, WOA, GJO, MFO, SCA, AOA, and PSO. It also outperformed AO and RIME on 11 of 12 functions and showed competitive results against DE. All p-values were below 0.05. This confirms that the performance differences were statistically significant. Overall, the findings indicate that BTO performs reliably and has superior results on complex optimization problems compared with many existing algorithms.

Table 3. Wilcoxon test summary on CEC2022 (D = 20)

Algorithms	Win	Draw	Lose	*p*-value
GWO	12	0	0	**2.61E-08**
WOA	12	0	0	**1.31E-05**
GJO	12	0	0	**4.75E-10**
MFO	12	0	0	**1.46E-09**
AO	11	1	0	**1.15E-02**
SCA	12	0	0	**4.09E-10**
AOA	12	0	0	**8.70E-10**
PSO	12	0	0	**3.50E-10**
RIME	11	1	0	**1.03E-02**
DE	10	1	1	**3.10E-02**

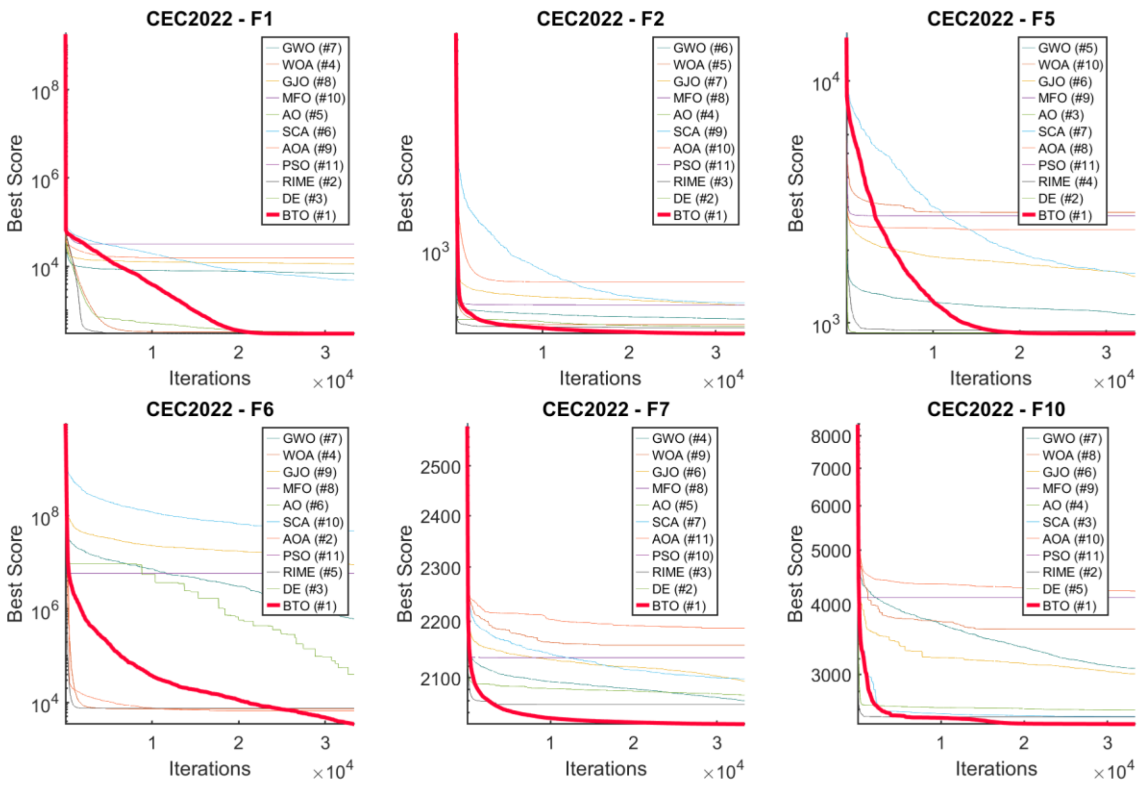

Fig. 6. CEC2022 convergence results on F1, F2, F5, F6, F7, F10 (D = 20)

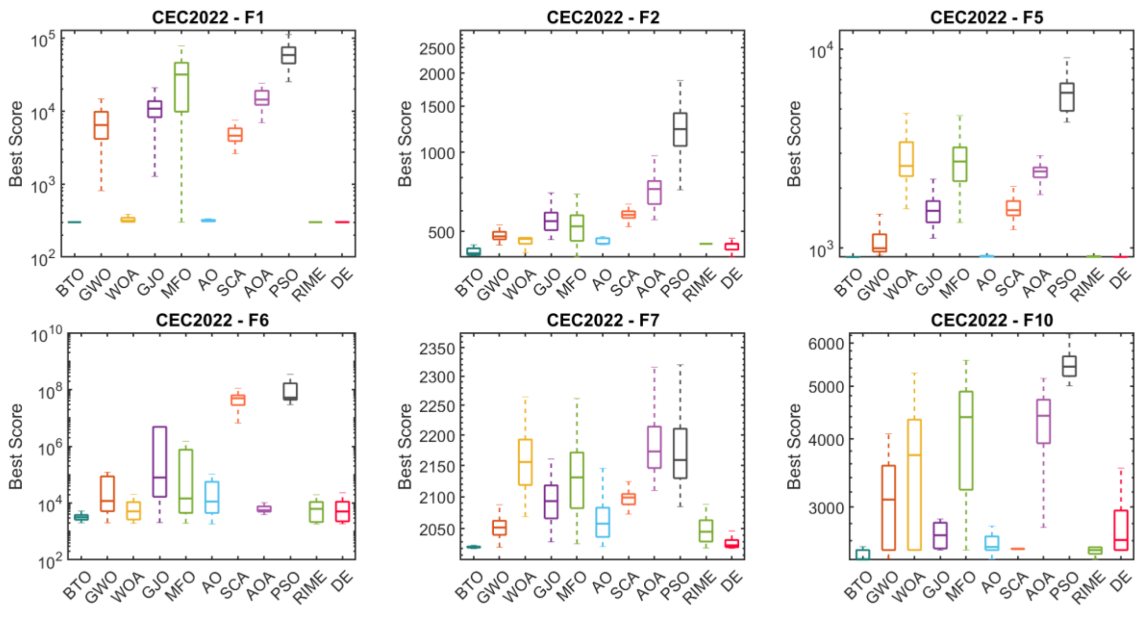

Fig. 7. Boxplot comparison on six benchmark functions from CEC2022 (D = 20)

3.3 Results on Classical Benchmark Functions

This section evaluates BTO on 10 classical benchmark functions selected from a well-known set of 23 test problems. These functions, including Sphere, Rastrigin, and Ackley, are widely used in the optimization field and represent both simple and complex search spaces. They are commonly used to estimate whether an algorithm can find good solutions and avoid local optima.

Table 4 shows the results of BTO compared with other metaheuristic algorithms on these 10 functions. BTO performed strongly, achieving the best rank in 7 out of 10 functions. Besides, it had a mean rank of 2.50, which was better than most other algorithms. The low standard deviation (STD) values on many functions showed the proposed algorithm's consistent performance and stable behavior across multiple runs. To verify that the differences in ranking were statistically significant, the Friedman test was applied. All *p*-values were below 0.05, indicating that the observed differences were significant. Collectively, BTO is a reliable and effective algorithm.

Table 4. Results of algorithms on 10 classical benchmark functions

Fun.	Type	BTO	GWO	WOA	GJO	MFO	AO	SCA	AOA	PSO	RIME	DE
F1	Mean	9.11E-07	**0.00E+00**	**0.00E+00**	**0.00E+00**	1.99E+04	**0.00E+00**	2.28E-03	3.27E-03	1.28E+05	1.52E-02	5.85E+02
	STD	1.52E-07	**0.00E+00**	**0.00E+00**	**0.00E+00**	1.43E+04	**0.00E+00**	8.80E-03	3.73E-03	3.17E+04	3.09E-03	6.07E+02
	Rank	5	**2.5**	**2.5**	**2.5**	10	**2.5**	6	7	11	8	9
F2	Mean	5.50E-04	**0.00E+00**	**0.00E+00**	**0.00E+00**	1.46E+02	**0.00E+00**	2.00E-40	**0.00E+00**	1.73E+47	7.59E-01	4.61E-01
	STD	3.76E-05	**0.00E+00**	**0.00E+00**	**0.00E+00**	5.85E+01	**0.00E+00**	1.43E-39	**0.00E+00**	1.02E+48	3.86E-01	1.30E+00
	Rank	7	**3**	**3**	**3**	10	**3**	6	**3**	11	9	8
F6	Mean	**8.79E-07**	8.72E+00	4.79E-05	1.66E+01	1.84E+04	1.12E-04	2.19E+01	1.45E+01	1.28E+05	1.48E-02	5.61E+02
	STD	**1.74E-07**	8.49E-01	1.01E-05	8.11E-01	1.41E+04	3.56E-05	5.38E+00	4.74E-01	3.19E+04	3.75E-03	6.68E+02
	Rank	**1**	5	2	7	10	3	8	6	11	4	9
F8	Mean	**-4.18E+04**	-1.55E+04	-4.15E+04	-9.26E+03	-2.49E+04	-3.66E+04	-8.59E+03	-1.74E+04	-1.34E+04	-3.65E+04	-2.93E+04
	STD	**1.27E+02**	1.99E+03	1.27E+03	3.53E+03	2.63E+03	7.27E+02	4.20E+02	1.03E+03	1.06E+03	7.91E+02	1.18E+03
	Rank	**1**	8	2	10	6	3	11	7	9	4	5
F10	Mean	1.25E-04	1.36E-14	2.43E-15	4.48E-15	1.98E+01	**4.44E-16**	1.79E+01	**4.44E-16**	1.99E+01	1.50E+00	9.64E+00
	STD	1.14E-05	1.93E-15	2.27E-15	1.22E-15	2.91E-01	**0.00E+00**	6.63E+00	**0.00E+00**	2.44E-01	4.23E-01	1.80E+00
	Rank	6	5	3	4	10	**1.5**	9	**1.5**	11	7	8
F12	Mean	**4.83E-10**	2.07E-01	8.72E-07	5.96E-01	5.61E+07	6.97E-05	5.77E+05	6.30E-01	1.21E+09	1.51E+00	2.07E+06
	STD	**9.13E-11**	4.42E-02	1.89E-07	7.83E-02	1.17E+08	3.45E-04	1.01E+06	2.21E-02	3.39E+08	9.19E-01	4.33E+06
	Rank	**1**	4	2	5	10	3	8	6	11	7	9
F13	Mean	**2.46E-08**	5.52E+00	3.03E-03	8.48E+00	1.77E+08	1.46E+00	1.23E+06	9.74E+00	2.24E+09	1.01E-02	4.19E+06
	STD	**4.25E-09**	4.63E-01	4.92E-03	2.79E-01	2.36E+08	1.38E+00	2.12E+06	1.14E-01	5.76E+08	1.75E-02	4.44E+06
	Rank	**1**	5	2	6	10	4	8	7	11	3	9
F14	Mean	**9.98E-01**	4.07E+00	2.58E+00	3.94E+00	3.08E+00	4.48E+00	1.64E+00	9.59E+00	1.03E+00	9.98E-01	1.13E+00
	STD	**2.26E-16**	4.09E+00	3.32E+00	3.67E+00	2.85E+00	4.53E+00	1.54E+00	4.02E+00	7.67E-02	1.10E-12	7.05E-01
	Rank	**1**	9	6	8	7	10	5	11	3	2	4
F20	Mean	**-3.32E+00**	-3.26E+00	-3.24E+00	-3.06E+00	-3.24E+00	-3.27E+00	-2.96E+00	-3.12E+00	-3.09E+00	-3.28E+00	-3.24E+00
	STD	**1.13E-11**	6.81E-02	1.03E-01	3.31E-01	6.20E-02	5.94E-02	3.51E-01	5.14E-02	2.62E-01	5.63E-02	5.33E-02
	Rank	**1**	4	5	10	6	3	11	8	9	2	7
F23	Mean	**-1.05E+01**	-1.04E+01	-8.80E+00	-9.76E+00	-7.75E+00	-6.36E+00	-4.89E+00	-4.14E+00	-4.45E+00	-9.89E+00	-9.97E+00
	STD	**7.76E-02**	7.43E-01	2.91E+00	2.02E+00	3.47E+00	3.55E+00	1.75E+00	1.40E+00	1.74E+00	1.81E+00	2.00E+00
	Rank	**1**	2	6	5	7	8	9	11	10	4	3
Mean	**Rank**	**2.50**	4.75	3.35	6.05	8.60	4.10	8.10	6.75	9.70	5.00	7.10

The convergence behavior of BTO and other algorithms is shown in Fig. 8, based on several classical benchmark functions. The plots represent each method improving its best-found solution with time. Mostly, BTO had better and faster results. It started to do better than the other algorithms when the number of iterations increased. For functions such as F13, F20, and F23, many algorithms slowed down early and stopped improving. This was possibly because of entrapment in local optima. Conversely, BTO continued to make progress, demonstrating its ability to escape local traps and keep searching for better results. These results support that BTO had a strong balance between exploration and exploitation. It could do well on both simple and complex optimization problems.

The Wilcoxon signed-rank test was used to compare BTO with other metaheuristic algorithms on the 10 classical benchmark functions. Table 5 shows the number of wins, draws, and losses, along with the corresponding *p*-values. BTO had strong results, outperforming most algorithms in many different test cases. It won all 10 comparisons against PSO and showed clear advantages over AOA, SCA, MFO, and AO. The *p*-values were under 0.05, confirming statistical significance. In contrast, comparisons with WOA, GJO, and DE yielded *p*-values slightly above 0.05. Thus, the differences in those cases were not significant. Taken together, these findings confirm that BTO has consistent performance on standard optimization problems, with meaningful improvements over many existing methods.

Table 5. Wilcoxon test on 10 classical functions

Algorithms	Win	Draw	Lose	*p*-value
GWO	6	0	4	**7.53E-05**
WOA	6	1	3	5.53E-02
GJO	6	1	3	5.06E-02
MFO	9	1	0	**7.91E-03**
AO	7	0	3	**2.83E-08**
SCA	8	0	2	**8.96E-04**
AOA	8	0	2	**2.39E-07**
PSO	10	0	0	**3.50E-10**
RIME	8	1	1	**3.38E-03**
DE	7	2	1	5.79E-02

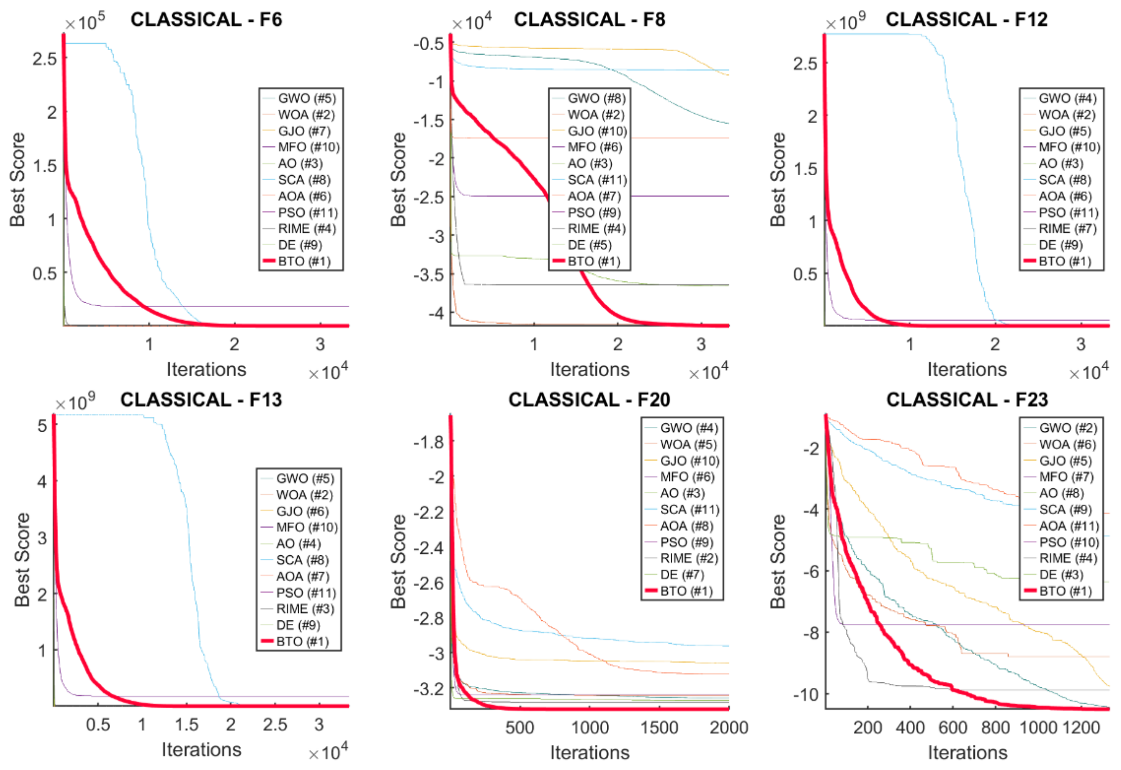

Fig. 8. Classical convergence results on F6, F8, F12, F13, F20, F23

3.4 Results on Engineering Design Problems

This section centers on solving real-life engineering design problems using metaheuristic algorithms. Table 6 contains eight well-known test problems from mechanical and structural design, including the gear train, pressure vessel, and truss structure [4]. These problems are different in the number of variables (D) and constraints (C). They are widely used in the optimization field to evaluate an algorithm's ability to handle realistic engineering conditions. By testing these problems, the capability of an algorithm to identify feasible, high-quality solutions in practical design tasks can be better understood.

Table 6. Summary of engineering design problems

Fun.	Problem	Optimum	D	C
F1	Speed reducer design	2.994424E+03	7	11
F2	Pressure vessel design	6.059714E+03	4	4
F3	Three-bar truss design problem	2.638958E+02	2	3
F4	Gear train design	2.700857E−12	4	0
F5	Piston lever	8.412698E+00	4	4
F6	Corrugated bulkhead design	6.842958E+00	4	6
F7	Car side impact design	2.284297E+01	11	1
F8	Reinforced concrete beam design	3.592080E+02	3	2

The performance of BTO and the other algorithms on the eight engineering design problems is presented in

Table 7. The test problems exhibit diversity in dimensionality and constraint complexity, covering a range of engineering design scenarios. BTO had strong and stable results. It ranked first in 5 of the 8

problems and got the lowest average rank of 1.63 among all algorithms. These results show that BTO often had better solutions. Also, the low standard deviation (STD) values suggest stable performance throughout different runs. Some algorithms, such as GWO and AO, worked well on certain problems (e.g., F2 and F4). However, their performance was less stable on more challenging problems such as F5, F6, and F7. In short, BTO is a robust metaheuristic optimizer. It is capable of giving high-quality solutions for both constrained and unconstrained engineering design problems.

Table 7. Results of algorithms on the engineering design problems

Fun.	Type	BTO	GWO	GJO	AO	SCA	AOA	PSO	RIME
F1	Best	**2.994430E+03**	3.006006E+03	3.006342E+03	3.024270E+03	3.066675E+03	3.081890E+03	3.025304E+03	NA
	Mean	**2.994431E+03**	3.007406E+03	3.012899E+03	3.055048E+03	3.115133E+03	3.150778E+03	3.095509E+03	NA
	STD	**9.308196E-04**	1.980447E+00	6.887656E+00	2.542959E+01	2.999130E+01	4.038927E+01	7.428571E+01	NA
	Rank	**1**	2	3	4	6	7	5	NA
F2	Best	6.074872E+03	**6.060194E+03**	6.064618E+03	6.082092E+03	6.542460E+03	6.852966E+03	6.223129E+03	6.078935E+03
	Mean	6.160040E+03	**6.069141E+03**	6.189030E+03	6.524594E+03	6.982891E+03	8.189016E+03	6.492794E+03	6.438081E+03
	STD	4.899269E+01	**1.192906E+01**	2.372035E+02	2.587468E+02	4.818504E+02	1.099830E+03	2.135882E+02	2.916822E+02
	Rank	2	**1**	3	6	7	8	5	4
F3	Best	**2.638960E+02**	2.638961E+02	2.638964E+02	2.643209E+02	2.639057E+02	2.639101E+02	2.638992E+02	2.638992E+02
	Mean	2.638973E+02	**2.638971E+02**	2.639002E+02	2.657049E+02	2.639558E+02	2.643656E+02	2.639301E+02	2.640587E+02
	STD	1.835250E-03	**7.718400E-04**	5.114219E-03	1.590988E+00	4.221930E-02	5.817781E-01	2.550712E-02	1.666341E-01
	Rank	2	**1**	3	8	5	7	4	6
F4	Best	**2.700857E-12**	**2.700857E-12**	**2.700857E-12**	2.307816E-11	**2.700857E-12**	6.602090E-10	1.545045E-10	**2.700857E-12**
	Mean	3.647076E-09	**5.842735E-10**	1.046784E-09	1.032775E-08	7.315753E-09	2.251768E-08	8.018256E-08	2.409654E-09
	STD	5.195994E-09	**7.601096E-10**	8.222924E-10	1.148079E-08	9.231444E-09	2.161387E-08	2.365163E-07	5.317016E-09
	Rank	4	**1**	2	6	5	7	8	3
F5	Best	8.417496E+00	8.419078E+00	8.418254E+00	**8.413624E+00**	8.509749E+00	2.027221E+02	8.460088E+00	8.474019E+00
	Mean	**8.509038E+00**	6.007065E+01	8.511100E+00	2.628011E+01	9.923896E+00	4.247127E+02	1.132615E+02	2.923582E+02
	STD	**3.335955E-01**	7.502881E+01	7.418659E-02	6.229367E+01	8.175923E-01	1.151135E+02	1.352121E+02	5.338954E+02
	Rank	**1**	5	2	4	3	8	6	7
F6	Best	**6.843024E+00**	6.846018E+00	6.848026E+00	6.846960E+00	7.050961E+00	7.243657E+00	7.079841E+00	6.843791E+00
	Mean	**6.843499E+00**	6.864006E+00	6.886030E+00	7.888029E+00	7.527380E+00	8.029575E+00	7.457171E+00	6.848433E+00
	STD	**6.887908E-04**	6.163411E-02	2.564943E-02	1.090158E+00	3.759724E-01	4.889006E-01	1.981395E-01	4.491012E-03
	Rank	**1**	3	4	7	6	8	5	2
F7	Best	2.285184E+01	2.285069E+01	2.289266E+01	2.288178E+01	2.387348E+01	2.312426E+01	2.303174E+01	**2.284558E+01**
	Mean	**2.292103E+01**	2.321185E+01	2.346535E+01	2.378752E+01	2.454231E+01	2.405493E+01	2.386128E+01	2.313710E+01
	STD	**5.863317E-02**	3.062765E-01	3.180235E-01	6.041977E-01	4.738910E-01	3.679272E-01	7.388443E-01	3.047513E-01
	Rank	**1**	3	4	5	8	7	6	2
F8	Best	**3.592080E+02**	3.592083E+02	3.592113E+02	**3.592080E+02**	3.592476E+02	3.594472E+02	3.592124E+02	3.592082E+02
	Mean	**3.598486E+02**	3.617896E+02	3.619046E+02	3.628465E+02	3.621962E+02	3.637218E+02	3.610682E+02	3.624523E+02
	STD	**1.041311E+00**	1.097217E+00	9.739983E-01	1.491893E+00	8.238104E-01	2.435364E+00	1.354264E+00	2.178483E+00
	Rank	**1**	3	4	7	5	8	2	6
Mean	**Rank**	**1.63**	2.38	3.13	5.88	5.63	7.50	5.13	4.29

Fig. 9 illustrates the convergence of BTO and other metaheuristic algorithms on eight engineering design problems. The graphs describe how the best solution found by each algorithm improved with the increase in the number of iterations. This trend shows their ability to approach optimal or near-optimal solutions over time. BTO steadily converged faster and obtained better solutions than most competing algorithms. It kept improving in all problems. Meanwhile, some algorithms slowed down or stagnated, particularly on hard problems like F5, F7, and F8. These results demonstrate that BTO balances searching widely and focusing deeply, helping it work well on complex engineering problems with constraints.

Fig. 10 is a box plot of solution scores for BTO and other metaheuristic algorithms on the same set of engineering design problems. These boxplots show the changes in the results over different runs, reflecting both solution quality and stability. In general, BTO had lower and more consistent scores. This indicates reliable and high-quality solutions. BTO's boxplots for problems F1, F2, and F7 were smaller and more consistent with less variance. The larger boxes and higher median scores of other algorithms suggested poorer performance. For example, AO and SCA showed larger variations on some problems. These findings demonstrate that BTO is more robust for solving challenging engineering design problems.

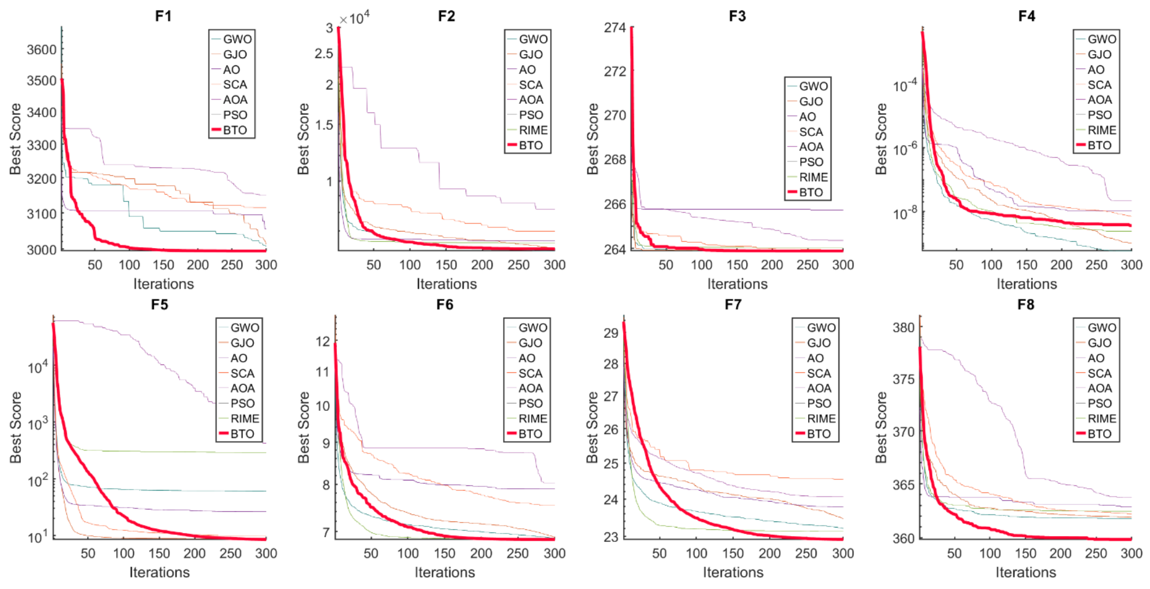

Fig. 9. Convergence curves of BTO and other algorithms on eight engineering design problems

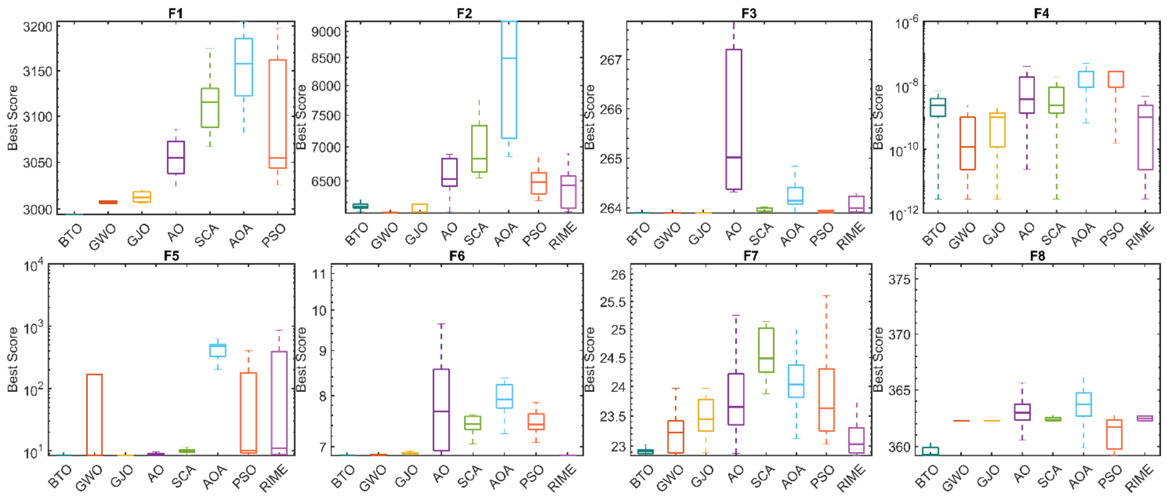

Fig. 10. Boxplot results on engineering design problems

3.5 Parameter Sensitivity Analysis on Elite Size

This section checks how the elite size (β), which defines the maximum number of top individuals guiding the search, affects the performance of BTO. The parameter β was tested with values $\{1, 3, 5, 7, 9, 10\}$ on the 20-dimensional CEC 2022 benchmark set. For each value, the average rank in all test functions was measured. As shown in Fig. 11, a small elite size (e.g., β = 1 or 3) gave weaker performance, likely because too few good individuals provided limited guidance for the search. When β went higher, the results got better ,with the best performance achieved at β = 10. Therefore, a larger elite size enables BTO to balance between exploring new solutions and learning from strong ones. Based on this analysis, β = 10 is used as the default setting in all experiments

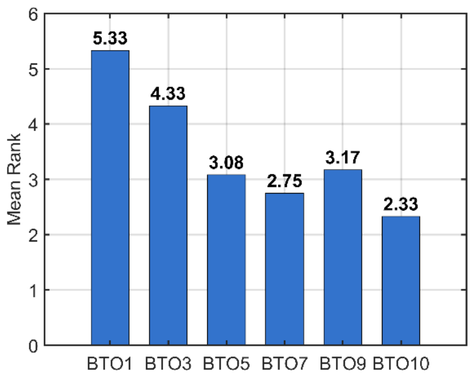

Fig. 11. Mean ranks of BTO with varying elite sizes (lower is better)

3.6 Ablation Study

This section investigates how each core mechanism contributes to the overall performance of the proposed BTO algorithm. Three simplified variants were prepared for comparison: BTO1 removes the Adaptive Elite

Selection mechanism, BTO2 removes the time-decreasing Harmonic Disturbance, where the disturbance factor F is is replaced by a random value uniformly sampled from [−1, 1], and BTO3 omits the Barrel Adjustment (BA) mechanism. All variants were tested under the same conditions on the CEC2022 benchmark set to ensure a fair comparison.

As

Fig. **12** shows, removing any single mechanism caused a clear drop in performance. This confirms that every component is important. The BTO3 variant (without Barrel Adjustment) produced the worst results, which highlights that the Barrel Adjustment mechanism is vital for stable convergence and preventing stagnation.

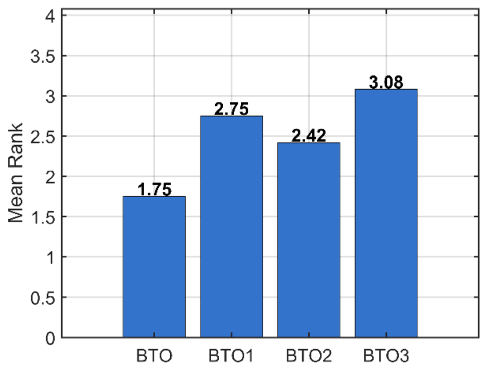

Fig. 12. Mean Friedman rank comparison of BTO and its ablated variants (lower is better)

The Wilcoxon signed-rank test ($\alpha = 0.05$) also confirms that the full BTO performed statistically better than all three ablated variants (with win/draw/loss counts of 5/6/1 against BTO1, 7/3/2 against BTO2, and 7/3/2 against BTO3).

Overall, the complete BTO algorithm earned the lowest mean rank (1.75). This demonstrates that the combined mechanisms work together to achieve a more effective balance between exploration and exploitation.

3.7 Behavioural Analysis

This section analyzes BTO's search behavior, focusing on its balance between exploration and exploitation. We track two metrics: population diversity and the exploration-exploitation ratio, using the method from Hussain et al. [35]. We selected three CEC2022 functions (F1, F6, and F10) to show how BTO works on different types of problems.

As shown in Fig. 13, on the simple unimodal function F1, BTO stays in exploration mode for most of the search. The smooth landscape allows the algorithm to search widely without getting stuck in local optima. For the hybrid function F6, exploration slowly decreases as exploitation increases. This shows BTO's adaptive nature: it starts with a broad search and then shifts to local refinement as it finds better solutions. On the complex composition function F10, exploration decreases quickly after BTO finds a good area and exploitation takes over. Near the end, exploration increases a bit. This small rebound is caused by the Harmonic Disturbance and Barrel Adjustment mechanisms. It enables BTO to avoid stagnation and fine-tune the final solutions in the optimal region.

Overall, BTO can dynamically offer a balance between exploration and exploitation in different landscape types. It can maintain diversity when necessary and exploit promising areas well to escape local optima.

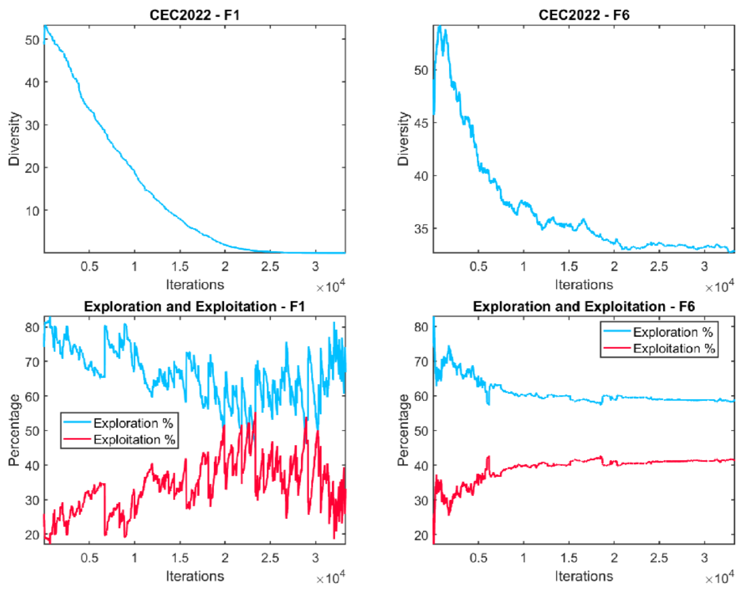

Fig. 13. Diversity and exploration–exploitation behaviour of BTO on representative CEC2022 functions (F1, F6, F10)

4. CONCLUSION

This paper proposed the Barrel Theory-based Optimizer (BTO), a new metaheuristic algorithm inspired by the idea of improving the weakest parts of a solution. BTO includes a dimension-wise update mechanism, called Barrel Adjustment, to focus on weak variables. It also features an adaptive elite selection strategy to balance exploration and exploitation. Besides, this algorithm has a hybrid update approach that combines elite learning with directional adjustments. BTO has been evaluated on the CEC2022 benchmark suite, classical benchmark functions, and real engineering design problems.

The results show that BTO consistently provides solutions that are competitive or superior to well-known algorithms such as PSO and GWO. BTO works stably and handles complex and constrained problems well. In addition, BTO has achieved robust and stable results on eight well-known engineering design problems. This confirms its effectiveness for constrained structural and mechanical optimization problems.

In summary, BTO is a reliable and helpful algorithm for solving many types of continuous optimization problems. Future studies will extend BTO to discrete, multi-objective optimization scenarios.

CONFLICTS OF INTEREST

The authors declare no conflict of interest.

DECLARATION OF AI USE

The authors declare the use of generative artificial intelligence (AI) and AI-assisted technologies in the preparation of this manuscript. Specifically, ChatGPT (version GPT-5, OpenAI) was used to improve the readability and language of the article. The AI tool was not used for data analysis, result interpretation, or drawing scientific conclusions. All content has been reviewed and verified by the authors to ensure accuracy and integrity.

REFERENCES

[1] W. Wang, H. Dong, X. Wang, P. Wang, J. Shen, Y. Jiang, Z. Wen, H. Zhu, Knowledge-guided global optimization for expensive and black-box constrained multi-objective engineering design problems. *Applied Soft Computing*, 177, 2025: 113216. https://doi.org/10.1016/j.asoc.2025.113216

[2] N.E. Chalabi, A. Attia, A.S. Almazayd, A.W. Mohamed, F. Werner, P. Jangir, M. Shokouhifar, MO-CBOA: Multi-objective chef-based optimization algorithm using hybrid dominance relations for solving engineering design problems. *Computer Modeling in Engineering and Sciences*, 143(1), 2025: 967–1008. https://doi.org/10.32604/cmes.2025.062332

[3] F. Wu, S. Li, J. Zhang, L. Yu, X. Xiong, L. Wu, Q-learning-based exponential distribution optimizer with multi-strategy guidance for solving engineering design problems and

robot path planning. *Results in Engineering*, 27, 2025: 105705. https://doi.org/10.1016/j.rineng.2025.105705

[4] H. Bayzidi, S. Talatahari, M. Saraee, C.P. Lamarche, Social network search for solving engineering optimization problems. *Computational Intelligence and Neuroscience*, 2021(1), 2021: 8548639. https://doi.org/10.1155/2021/8548639

[5] S. Mohapatra, P. Mohapatra, An improved Golden Jackal Optimization algorithm using opposition-based learning for global optimization and engineering problems. *International Journal of Computational Intelligence Systems*, 16(1), 2023: 147. https://doi.org/10.1007/S44196-023-00320-8

[6] M.O. Okwu, L.K. Tartibu, Metaheuristic Optimization: Nature-Inspired Algorithms, Swarm and Computational Intelligence – Theory and Applications. *Springer Cham*, 2021. https://doi.org/10.1007/978-3-030-61111-8

[7] N. Hansen, D.V. Arnold, A. Auger, Evolution strategies. in: Springer Handbook of Computational Intelligence. *Springer Handbooks,* 2015: pp.871–898. https://doi.org/10.1007/978-3-662-43505-2_44

[8] S. Forrest, Genetic algorithms. *ACM Computing Surveys (CSUR)*, 28(1), 1996: 77–80. https://doi.org/10.1145/234313.234350

[9] R. Storn, K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11, 1997: 341–359. https://doi.org/10.1023/A:1008202821328

[10] S. Kamel, H. Hamour, M.H. Ahmed, L. Nasrat, Atom search optimization algorithm for optimal radial distribution system reconfiguration. *Proceedings of the International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, 21-23 September 2019, Khartoum, Sudan

[11] C. Yuan, D. Zhao, A.A. Heidari, L. Liu, Y. Chen, Z. Wu, H. Chen, Artemisinin optimization based on malaria therapy: algorithm and applications to medical image segmentation. *Displays*, 84, 2024: 102740. https://doi.org/10.1016/j.displa.2024.102740

[12] P. Duankhan, K. Sunat, S. Chiewchanwattana, P. Nasa-ngium, The differentiated creative search (DCS): leveraging differentiated knowledge-acquisition and creative realism to address complex optimization problems. *Expert Systems with Applications*, 252(Part A), 2024: 123734. https://doi.org/10.1016/j.eswa.2024.123734

[13] J. Kennedy, R. Eberhart, Particle swarm optimization. *Proceedings of ICNN'95 – International Conference on Neural Networks,* 27 November - 1 December 1995, Perth, Australia, pp. 1942–1948.

[14] C. Blum, Ant colony optimization: introduction and recent trends. *Physics of Life Reviews*, 2(4), 2005: 353–373. https://doi.org/10.1016/j.plrev.2005.10.001

[15] S. Mirjalili, S.M. Mirjalili, A. Lewis, Grey wolf optimizer. *Advances in Engineering Software*, 69, 2014: 46–61. https://doi.org/10.1016/j.advengsoft.2013.12.007

[16] S. Mirjalili, SCA: a sine cosine algorithm for solving optimization problems. *Knowledge-Based Systems*, 96, 2016: 120–133. https://doi.org/10.1016/j.knosys.2015.12.022

[17] M.H. Nadimi-Shahraki, H. Zamani, Z. Asghari Varzaneh, S. Mirjalili, A systematic review of the Whale Optimization Algorithm: theoretical foundation, improvements, and hybridizations. *Archives of Computational Methods in Engineering*, 30, 2023: 4113–4159. https://doi.org/10.1007/S11831-023-09928-7

[18] S. Li, H. Chen, M. Wang, A.A. Heidari, S. Mirjalili, Slime mould algorithm: a new method for stochastic optimization. *Future Generation Computer Systems*, 111, 2020: 300–323. https://doi.org/10.1016/j.future.2020.03.055

[19] Y. Fu, D. Liu, J. Chen, L. He, Secretary bird optimization algorithm: a new metaheuristic for solving global optimization problems. *Artificial Intelligence Review*, 57, 2024: 123. https://doi.org/10.1007/s10462-024-10729-y

[20] P.J.M. Van Laarhoven, E.H.L. Aarts, Simulated annealing. in: Simulated Annealing: Theory and Applications. *Springer Dordrecht,* 1987: pp.7–15. https://doi.org/10.1007/978-94-015-7744-1

[21] H. Su, D. Zhao, A.A. Heidari, L. Liu, X. Zhang, M. Mafarja, H. Chen, RIME: a physics-based optimization. *Neurocomputing*, 532, 2023: 183–214. https://doi.org/10.1016/j.neucom.2023.02.010

[22] D. Zhu, S. Wang, C. Zhou, R. Tian, H. Zhang, J. Mao, Human memory optimization algorithm: a memory-inspired optimizer for global optimization problems. *Expert Systems with Applications*, 237(Part C), 2024: 121597. https://doi.org/10.1016/j.eswa.2023.121597

[23] K. Ouyang, S. Fu, Y. Chen, Q. Cai, A.A. Heidari, H. Chen, Escape: an optimization method based on crowd evacuation behaviors. *Artificial Intelligence Review*, 58, 2025: 19. https://doi.org/10.1007/s10462-024-11008-6

[24] H.J.W. de Baar, von Liebig's law of the minimum and plankton ecology (1899–1991). *Progress in Oceanography*, 33(4), 1994: 347–386. https://doi.org/10.1016/0079-6611(94)90022-1

[25] E.M. Goldratt, J. Cox, The Goal: A Process of Ongoing Improvement. *Routledge*, 2016.

[26] D. Meadows, Leverage Points – Places to Intervene in a System. *Sustainability Institute*, 2015.

[27] J.P. Womack, D.T. Jones, D. Roos, The Machine that Changed the World: The Story of Lean Production. Free Press, New York, 1990.

[28] R.D. Snee, R.W. Horerll, Leading Six Sigma: A Step-by-Step Guide Based on Experience with GE and Other Six Sigma Companies. *Prentice-Hall*, 2003.

[29] E.L. Charnov, Optimal foraging, the marginal value theorem. *Theoretical Population Biology*, 9(2), 1976: 129–136. https://doi.org/10.1016/0040-5809(76)90040-x

[30] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction. *MIT Press*, Cambridge, 2014.

[31] A. Kumar, K.V. Price, A.W. Mohamed, Problem definitions and evaluation criteria for the CEC 2022 special session and competition on single objective bound constrained numerical optimization, Technical Report. *Nanyang Technological University*, Singapore, 2022.

[32] C.-Y. Lee, X. Yao, Evolutionary programming using mutations based on the Lévy probability distribution. *IEEE Transactions on Evolutionary Computation*, 8(1), 2004: 1–13. https://doi.org/10.1109/TEVC.2003.816583

[33] S. Mirjalili, Moth-flame optimization algorithm: a novel nature-inspired heuristic paradigm. *Knowledge-Based Systems*, 89, 2015: 228–249. https://doi.org/10.1016/j.knosys.2015.07.006

[34] H. Liu, Z. Chen, X. Zhang, C.-C. Wu, J.N.D. Gupta, An improved arithmetic optimization algorithm based on reinforcement learning for global optimization and engineering design problems. *Swarm and Evolutionary Computation*, 96, 2025: 101985. https://doi.org/10.1016/j.swevo.2025.101985

[35] K. Hussain, M.N.M. Salleh, S. Cheng, Y. Shi, On the exploration and exploitation in popular swarm-based metaheuristic algorithms. *Neural Computing and Applications*, 31, 2019: 7665–7683. https://doi.org/10.1007/s00521-018-3592-0