

Freshwater Snail Optimizer: A Bio-Inspired Optimizer for Engineering Design Problems

Gao-Yuan Liu¹, Jeng-Shyang Pan^{2,3,*}, Shu-Chuan Chu², Ru-Yu Wang⁴,
Bo-Xiong Yang^{5,*}, Ying Yu¹ and Seyedali Mirjalili^{6,7,8}

¹ School of Information and Intelligent Engineering, University of Sanya, Sanya, 572022, China

² School of Artificial Intelligence/School of Future Technology, Nanjing University of Information Science and Technology, Nanjing, 210044, China

³ Department of Information Management, Chaoyang University of Technology, Taichung, 40601, Taiwan

⁴ College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao, 266590, China

⁵ Hainan Environmental Big Data and Digital Governance Key Laboratory of Philosophy and Social Sciences, University of Sanya, Sanya, 572022, China

⁶ Centre for Artificial Intelligence Research and Optimisation, Torrens University Australia, Brisbane, QLD 825001, Australia

⁷ University Research and Innovation Center, Obuda University, Budapest, 1034, Hungary

⁸ Faculty of Electrical Engineering and Computer Science, VSB-Technical University of Ostrava, Ostrava, 70800, Czech Republic

INFORMATION

Keywords:

Freshwater snail optimizer
metaheuristic algorithms
swarm intelligence
engineering optimization
benchmarks
artificial intelligence

DOI: 10.23967/j.rimni.2025.10.72114

Revista Internacional
Métodos numéricos
para cálculo y diseño en ingeniería

RIMNI



UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

In cooperation with
CIMNE[®]

Freshwater Snail Optimizer: A Bio-Inspired Optimizer for Engineering Design Problems

Gao-Yuan Liu¹, Jeng-Shyang Pan^{2,3,*}, Shu-Chuan Chu², Ru-Yu Wang⁴, Bo-Xiong Yang^{5,*}, Ying Yu¹ and Seyedali Mirjalili^{6,7,8}

¹School of Information and Intelligent Engineering, University of Sanya, Sanya, 572022, China

²School of Artificial Intelligence/School of Future Technology, Nanjing University of Information Science and Technology, Nanjing, 210044, China

³Department of Information Management, Chaoyang University of Technology, Taichung, 40601, Taiwan

⁴College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao, 266590, China

⁵Hainan Environmental Big Data and Digital Governance Key Laboratory of Philosophy and Social Sciences, University of Sanya, Sanya, 572022, China

⁶Centre for Artificial Intelligence Research and Optimisation, Torrens University Australia, Brisbane, QLD 825001, Australia

⁷University Research and Innovation Center, Obuda University, Budapest, 1034, Hungary

⁸Faculty of Electrical Engineering and Computer Science, VSB-Technical University of Ostrava, Ostrava, 70800, Czech Republic

ABSTRACT

As the scale and nonlinearity of optimization problems continue to increase, traditional deterministic solution strategies are becoming increasingly flawed in the face of the exponential growth of search space dimensions and multimodal objective functions. Metaheuristic algorithms, with their probability-driven global search capabilities and local development capabilities, have gradually become an essential tool for solving complex optimization tasks. We propose a Freshwater Snail Optimizer (FSO), inspired by the social behavior of water snails in terms of movement and collision, as a metaheuristic algorithm. FSO combines the floating of water snails' air chambers, movement in the water, and population collisions and divides them into groups during initialization to balance exploration and development, achieving gratifying optimization results, especially in high-dimensional problems. We utilized CEC 2017 and CEC 2022 to qualitatively analyze FSO in various problems, and employed the Friedman test and Wilcoxon rank sum test for statistical testing. Experimental results show that our proposed FSO achieved 32 first-place results on 41 problems compared with 9 classic algorithms, and 27 first-place results when compared with 9 emerging algorithms that appeared in the past two years. FSO has also achieved the first comparison results in six engineering optimizations on multiple occasions, proving that FSO possesses well optimization capabilities and practicality for real-world problems. The source code accompanying this article has been released at: <https://github.com/leogalaxy0603/Freshwater-Snail-Optimizer> (accessed on 12 October 2025).

OPEN ACCESS

Received: 19/08/2025

Accepted: 13/10/2025

Published: 15/12/2025

DOI

10.23967/j.rimni.2025.10.72114

Keywords:

Freshwater snail optimizer
metaheuristic algorithms
swarm intelligence
engineering optimization
benchmarks
artificial intelligence

1 Introduction

Optimization problems are common in various fields, including engineering design, data analysis, and decision science, and their solution methods have a direct impact on system performance [1]. This is usually described formally through mathematical modeling, which involves giving an objective function $f(x)$ and solving for the decision variable x that minimizes or maximizes $f(x)$ under constraints. The general form can be expressed as finding $x^* = \arg \min_{x \in \Omega} f(x)$ or $\arg \max_{x \in \Omega} f(x)$ in the feasible domain Ω . The properties of the objective function $f(x)$ are crucial in optimization theory, and different properties will significantly affect the selection of optimization strategies and the effectiveness of the algorithm. Differentiability determines whether gradient information can be used to guide the search. Suppose $f(x)$ is differentiable in the domain, and the gradient is easily calculable. In that case, algorithms based on first-order or second-order derivative information, such as gradient descent [2] and Newton's method [3], can be employed. These traditional algorithms have the advantage of fast convergence when the objective function is smooth and unimodal without local optimum, and the optimal solution can be found by setting the derivative to zero. However, when $f(x)$ is a piecewise function, a discontinuous function, or other non-differentiable function or the gradient calculation cost is high, the gradient method is challenging to apply. In this case, heuristic or meta-heuristic algorithms that do not rely on derivatives are usually required [4]. In addition, convexity and multimodality have a profound impact on the difficulty of optimization and algorithm selection. Convex functions have a single global optimal solution and no local minima. When the objective function is convex and smooth, the gradient method, conjugate gradient method, or Gauss–Newton method can efficiently find the global optimum and usually converge quickly [5]. However, for non-convex functions such as multi-peak and multi-modal functions, there may be multiple local optimal solutions to the optimization problem. Traditional gradient algorithms are prone to fall into local extreme points due to the nature of greedy descent [6], and cannot guarantee to find the global optimum. Currently, a global search strategy is necessary to address this issue. Metaheuristic algorithms have a greater probability of jumping out of the local extreme point trap through random perturbations and diversified searches [7]. This also reflects the outstanding advantages of metaheuristic algorithms. They have few requirements on the objective function and do not require strict conditions such as convexity, differentiability, or continuity. They are also highly adaptable to discrete variables, nonlinear constraints, etc.

In general, we refer to the collective intelligent behavior that emerges from the aggregation of simple biological or non-biological individuals as Swarm Intelligence (SI) which was first proposed by Gerardo and Wang in 1989 when describing the collective intelligence phenomenon of cellular robots [8]. Since the 1990s, a large number of intelligent algorithms inspired by animal behavior in nature have emerged, becoming an important branch of meta-heuristic algorithms. This type of algorithm primarily simulates the complex and coordinated behaviors exhibited by the mutual influence between individuals in large-scale groups, continuously improving the adaptability of the population to its environment. As a result, it can approach an approximate optimal solution to the problem being solved, as shown in Fig. 1. According to current research, the theoretical basis of metaheuristic algorithms primarily encompasses mathematical, biological, and sociological foundations. In terms of mathematical basis, researchers have tried to clarify the intelligence emergence, evolution and self-organization laws and characteristics of metaheuristic algorithms [9] from the perspectives of random processes [10], Markov chains [11], convergence analysis [12], etc.; in terms of biological basis, drawing on various group evolution phenomena and mechanisms contained in nature, researchers have explored new theories and models of evolutionary computing from the perspectives of survival of the

fittest, natural selection, biological evolution, genetic laws, human brain simulation, biological loss, immune evolution, etc. [13]. For example, the genetic algorithm (GA) [14] and differential evolution algorithm (DE) [15], both based on Darwin's 'theory of evolution', search for the optimal solution through crossover, mutation, and greedy selection. The proposed method is based on the idea of immune evolution. In terms of sociological foundations, scholars have drawn on various collective intelligence phenomena that have emerged in sociological biology and even human society, such as collective intelligence, group competition, group games, group collaboration, etc., to provide new theories and new approaches to understand, analyze and regulate the emergence and evolution of collective intelligence.

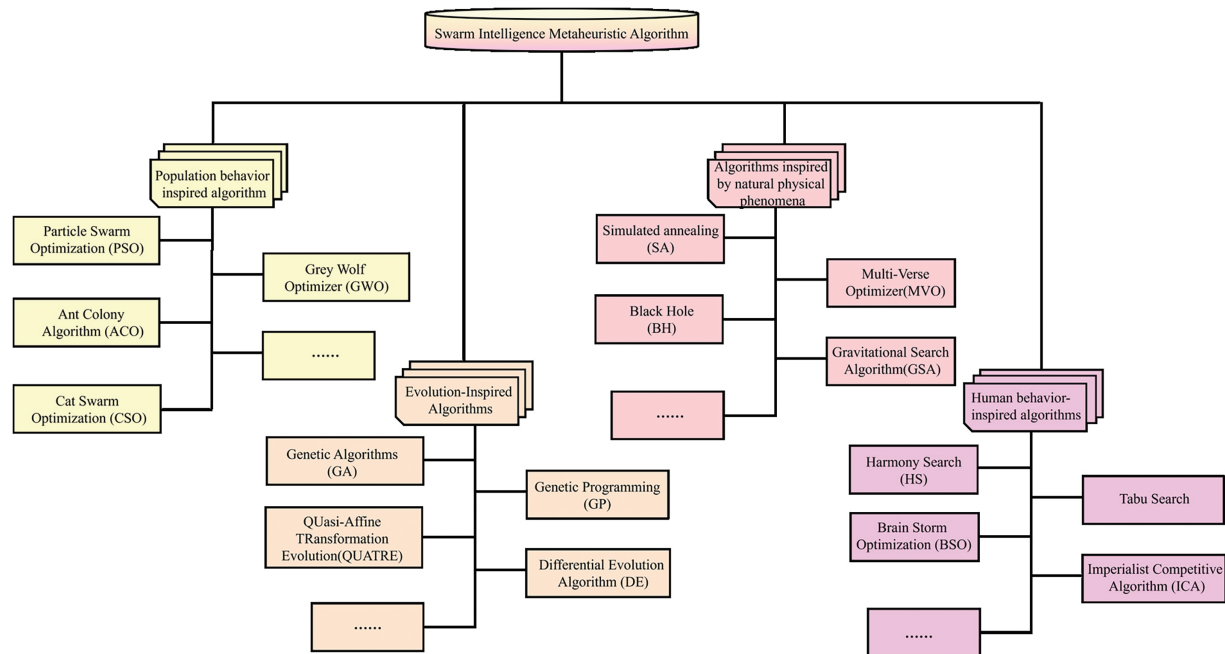


Figure 1: Swarm intelligence algorithm framework

For example, particle swarm optimization (PSO), which is inspired by the flying foraging behavior of bird flocks and biological group games and collaboration [16], was proposed by Eberhart and Kennedy in 1995 and is one of the most widely used metaheuristic algorithms to date; the ant colony algorithm (ACO) proposed by Blum [17] is inspired by the pheromone trails left by ant colonies when foraging, and is good at solving discrete path optimization problems [18]; the Firefly Algorithm (FA) proposed by Yang and Slowik [19] is based on the mechanism that fireflies attract each other through the brightness of their flashes, and can solve continuous optimization problems; the Cat Swarm Optimization (CSO) proposed by Chu et al. [20] uses the behavior of cat groups in the process of searching for food to perform global search and local optimization; the Imperialist Competitive Algorithm (ICA) [21], based on collective intelligence and group competition, was proposed by Atashpaz et al. in 2007 based on the competition between countries and optimizes through national competition and mergers; Brain Storm Optimization (BSO) proposed by Shi [22] simulates the process of human brainstorming meetings, in which members conduct free divergent thinking and collective discussions around a topic, promoting population diversity and global exploration; Teaching-Learning-Based Optimization (TLBO) proposed by Rao et al. [23] imitates the role division and knowledge dissemination in the human teaching-learning process, so that the algorithm can use

the 'teachers' to guide the global search in the early stage, and use mutual learning among students to enhance local development in the later stage. In addition, after 2016, a series of new animal behavior algorithms have emerged, including Grey Wolf Optimizer (GWO) [24], Whale Optimization Algorithm (WOA) [25], Snake Optimizer (SO) [26], Kirchhoff's law algorithm (KLA) [27], as well as Newton Metaheuristic Algorithm (NMA) [28] and Quantum Snowflake Algorithm (QSA) [29] based on natural physical phenomena and mathematical laws. Each algorithm draws inspiration from nature to improve its optimization effect in different ways.

In metaheuristic optimization, "exploration," and "exploitation" are important concepts for measuring search strategies, which correspond to global search capabilities and local search capabilities, respectively [30]. Exploration refers to the ability of the algorithm to search the solution space extensively, that is, to use a variety of solutions to try to discover areas that have not yet been explored, aiming to avoid the local optimal trap; exploitation refers to the ability of the algorithm to deeply explore the area near the current excellent solution, and to quickly improve the quality of the solution by performing a local fine search on the existing good solutions. Ideally, the algorithm should strike a balance between exploration and exploitation, with sufficient exploration to avoid missing the area where the global optimal solution is located, and sufficient exploitation to use the currently discovered excellent solutions to converge towards a better direction. If too much emphasis is placed on exploitation, the algorithm may stay in a local optimal state and be unable to jump out, falling into the local optimal state, resulting in premature convergence; on the contrary, too much emphasis on exploration may cause the algorithm to wander in the solution space, converge slowly, and find it difficult to find a high-quality solution in time. The 'no free lunch' theory [31] holds that no heuristic method can always have a good solution for all possible problems. Among the numerous algorithms, swarm intelligence [32] and biological foraging [33] remain the primary design prototypes. However, the mechanisms underlying different biological behaviors vary, and there are still many undiscovered natural inspirations that can be utilized, providing a rich source of inspiration for further innovation. The main research contents and contributions of this paper are as follows:

- A new bio-inspired Freshwater Snail Optimizer (FSO) is proposed to simulate the mobility and social behavior of freshwater Snail;
- The FSO algorithm simulates floating, crawling, and collision. Floating behavior helps global exploration, crawling behavior helps local development, and innovatively proposed dynamic explorer-exploiter Initialization Strategy and free step movement model based on collision behavior;
- We compared FSO with 18 algorithms on two CEC benchmark test functions (CEC 2017, CEC 2022) and performed with Friedman test and Wilcoxon rank sum test. The results show that FSO has good optimization ability and robustness;
- The effectiveness and practicality of FSO are verified through six engineering-constrained optimization problems.

The rest of this paper is organized as follows. [Section 2](#) introduces the proposed FSO algorithm. [Section 3](#) performs benchmark tests using CEC 2017, and CEC 2022, as well as a performance analysis on FSO. [Section 4](#) discusses the optimization of engineering problems. Finally, conclusion and future prospects.

2 Freshwater Snail Optimizer (FSO)

2.1 Inspiration

Freshwater snails are widely distributed in aquatic ecosystems such as rivers and lakes and play an important role in material circulation and balance maintenance. They exhibit unique adaptive behaviors in the natural environment, providing a rich source of inspiration for optimization algorithms.

2.1.1 Movement by Buoyancy

Freshwater snails can control their buoyancy by adjusting the gas content in their air chambers (similar to lung sacs) to change their specific gravity [34]. In water, they inhale or release gas to make their body density lower or higher than that of water, allowing their bodies to float to the surface or sink to the bottom. This physiological regulation mechanism will enable snails to float or remain attached to the benthic environment. When the surrounding environment is scarce or the conditions are unfavorable, they loosen their attachment to the substrate, float up, and drift with the water flow to move to new areas in search of food or oxygen. This drifting behavior significantly expands the range of activities of water snails [35] which shown in Fig. 2.



Figure 2: Freshwater snail movement by buoyancy (*Planorbis planorbis*). **Source:** Photograph by Uccio D'Agostino, [Wikimedia Commons](#), CC BY 4.0

2.1.2 Crawl

Water snails can also cling to and crawl on surfaces [36]. When they find a suitable habitat surface, such as water plants, rocks, or the wall of a water tank, they will use their ventral feet to cling to it and crawl slowly with the help of the suction force generated by mucus [37], carefully searching for food and oxygen in a local area. Water snails can crawl on a variety of surfaces [38], including bottom sediments, plant surfaces which shown in Fig. 3, and even tension membranes under the water surface, thereby making full use of environmental resources at a microscopic scale.



Figure 3: Freshwater snail (*Lymnaea stagnalis*) crawling on a reed stem underwater. **Source:** Photograph by Kk, [Wikimedia Commons](#), CC BY-SA 3.0

2.1.3 Collision Aggregation Behavior

Most organisms living under natural conditions tend to be distributed in clusters or clumps, and freshwater snails follow this rule both in the wild and in laboratory conditions. Studies have shown [39] that freshwater snails are highly social and will adjust their aggregation areas based on external environmental factors such as resource distribution and water flow conditions and behavioral conditions driven by internal conditions, as shown in Fig. 4, and prevent the disorderly increase of aggregation areas through random collision behavior.



Figure 4: Aggregation behavior of freshwater snails (*Planorbarius corneus*). **Source:** Photograph by Claus Ableiter, [Wikimedia Commons](#), CC BY SA 3.0

2.2 Mathematical Models

This section provides a detailed introduction to the mathematical model of FSO proposed in this paper. FSO is divided into three types of movement: “floating (exploration stage)-crawling

(exploitation stage)-collision”. Although FSO uses sine and cosine formulas to replace water flow fluctuations in the update, its simulation method is only related to the position of freshwater snails and the upper and lower limits of the problem. This is fundamentally different from the PSO improved by borrowing the sine and cosine ideas [40] and the hybrid SCA-PSO method [41]. In addition, FSO does not involve the historical record of individual optimality. Table 1 lists the symbols and terms used. In this algorithm, each freshwater snail individual represents a feasible solution, and each freshwater snail has its position composed of dim variables.

Table 1: Parameters and their interpretations in the water-snail optimisation algorithm

Parameter	Interpretation
A	Buoyancy
D_{avg}	Average distance between water-snail individuals
ER	Exploration ratio
gf	Global guidance coefficient
$gbestPos$	Global optimum position
$Iter, maxIter$	Current and maximum iteration numbers
lf	Local random-step coefficient
pop	Water-snail population (all individuals)
pop_{ub}	Problem upper bound
pop_{lb}	Problem lower bound
$pop_{explorer}, pop_{exploiter}$	Explorer and exploiter sub-populations
$\bar{V}_{best}$	Mean value at the global best position
$\bar{V}$	Mean value at a colliding position
$Wave$	Fluctuation induced by water flow
λ	Free-step movement factor

2.2.1 Initialization

As a meta-heuristic algorithm, FSO first needs to initialize a set of random solutions, as shown in Eq. (1):

$$pop = U(0, 1)_{popsize \times D} \times (pop_{ub} - pop_{lb}) + pop_{lb} \quad (1)$$

where $U(0, 1)$ represents a uniform distribution from 0 to 1, popsize is the population size, D is the problem dimension, and pop_{ub} and pop_{lb} represent the upper and lower limits of the problem to be optimized, respectively. The contradictory unity of exploration and development presents a difficulty in both theory and practice: how to characterize and dynamically regulate the balance between the two quantitatively. Therefore, in this paper, we divide the initialized pop into two parts: “exploration” and “development”. In this paper, the number of explorers and developers is not fixed, but is dynamically adjusted with the iteration process to achieve a smooth transition from “large-scale exploration” to “fine development”, as shown in Eq. (2):

$$ER_{Iter} = ER_{init} - (ER_{init} - ER_{final}) \frac{Iter}{maxIter} \quad (2)$$

where $ER_{init} = 0.5$ is the initial explorer ratio, $ER_{final} = 0.1$ is the final exploration, $Iter$ is the current iteration number, and $maxIter$ is the maximum iteration number. In the $Iter$ generation, the calculated ER_{iter} determines how many individuals in the current generation are marked as “explorers,” as shown in Eq. (3):

$$EC = \text{round}(ER_{Iter} \times \text{popsize}) \quad (3)$$

2.2.2 Exploration Stage

In natural water bodies, freshwater snails rapidly adjust their buoyancy by modulating the gas content of their pulmonary sacs. When the internal gas volume increases, buoyancy rises instantly, lifting the individual from the benthic surface and allowing it to drift upward with the surrounding flow. As the gas gradually diffuses, buoyancy decreases and the snail sinks again or is advected downstream. Superimposed on the intrinsic periodic fluctuations of the flow field, this rise and sink cycle produces a quasi-periodic floating-drifting trajectory that enables the snail to leave its original position quickly, reach new waters, and thus markedly expand its activity range, as shown in Eq. (4):

$$\text{pop}_i(\text{iter} + 1) = A \cdot (\text{gbestPos} - \text{pop}_{\text{explorer}}) + \text{Wave} \cdot (\text{gbestPos} - \text{snailPbestPos}_i) \quad (4)$$

where gbestPos is the global optimal position of the freshwater snail, snailPbestPos_i is the individual optimal position of the water snail, A and Wave are the buoyancy and the fluctuation affected by the water flow, as shown in Eqs. (5) and (6):

$$A = A_{\text{rand}} \cdot e^{\beta \frac{\text{Iter}}{\text{maxIter}}} \quad (5)$$

where $A_{\text{rand}} \in (0, 1)$ is a random number, and $\beta = 7$ is an exponential decay rate, which controls the decay speed of buoyancy with iteration.

$$\text{Wave} = \begin{cases} \sin\left(2\pi \frac{\|\text{gbestPos} - \text{snailPbestPos}_i\|_2}{\text{pop}_{ub} - \text{pop}_{lb}}\right), & \text{if rand} \leq 0.5, \\ \cos\left(2\pi \frac{\|\text{gbestPos} - \text{snailPbestPos}_i\|_2}{\text{pop}_{ub} - \text{pop}_{lb}}\right), & \text{if rand} > 0.5. \end{cases} \quad (6)$$

where $\|\text{gbestPos} - \text{snailPbestPos}_i\|_2$ is the Euclidean distance between the global optimal solution and the personal optimal solution, pop_{ub} and pop_{lb} are the upper and lower bounds of the search space; $\sin$ and $\cos$ are functions of periodic fluctuations. For ease of understanding, the schematic diagram of this stage is shown in Fig. 5.

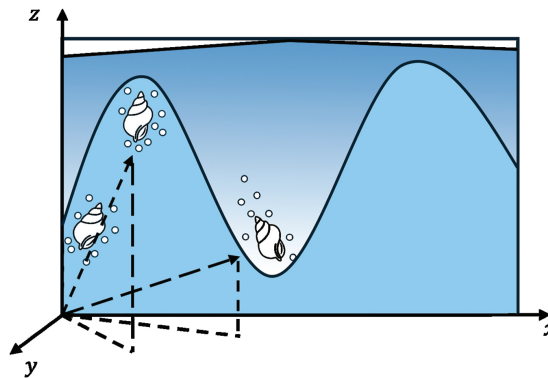


Figure 5: The process of exploring the floating of freshwater snails

2.2.3 Exploitation Stage

Compared with floating and drifting, the crawling behavior of freshwater snails corresponds to the development process in the optimization algorithm. When a snail finds an area rich in resources or suitable for survival, it will use its ventral foot to firmly attach to the nearby substrate and crawl slowly to explore the area in detail to maximize the intake of resources. Each snail remains attached and applies a small perturbation or incremental adjustment to the current solution to search for a better neighborhood solution in the solution space, as shown in Eq. (7):

$$pop_i(iter + 1) = snailPbestPos_i + lf \cdot r + gf \cdot (gbestPos - pop_{exploiter}) \quad (7)$$

where lf is the local random step coefficient, gf is the global guidance coefficient, $r \in (0, 1)$ is a random number, lf and gf are as shown in Eqs. (8) and (9):

$$lf = 0.1 \cdot \left(1 - \frac{Iter}{maxIter}\right) \quad (8)$$

$$gf = 0.2 \cdot \frac{Iter}{maxIter} \quad (9)$$

For ease of understanding, a schematic diagram of the development phase is shown in Fig. 6.

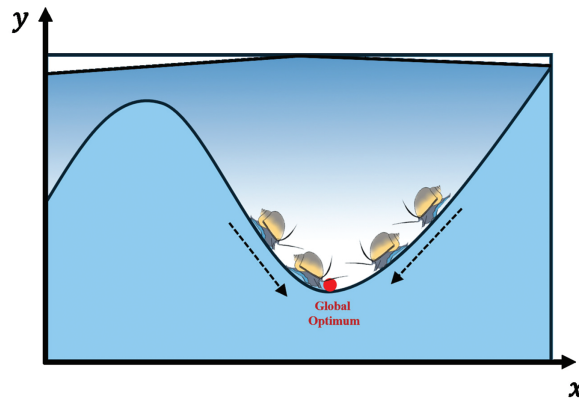


Figure 6: The process of exploiting the crawling of freshwater snails

2.2.4 Free-Step Movement by Collision Aggregation Behavior

Freshwater snails are highly social, and the degree of group aggregation is determined by random collisions during population movement [42]. The number of collisions between freshwater snails is determined by the density or number of freshwater snails per unit area N and the relative speed V_r . During this period, any freshwater snail will sweep out a strip path that becomes a free-movement path. Inspired by this, we propose a free-step movement factor λ .

Due to every solution trying to approach the global optimum in iteration, factors may overlap or approach the limit during the movement. Then, within the time T , the width between individuals is D , as shown in Eq. (10):

$$d = \sqrt{\sum_{k=1}^{dim} (pop_{ik} - pop_{jk})^2} \quad (10)$$

If $d < \text{threshold}$, it means that a collision has occurred between the two particles. In this paper, $\text{threshold} = 0.5$, designed by sensitivity test which contains 6 benchmark functions of CEC 2017 (which will be introduced in detail in [Section 3.3.2](#)), namely benchmark functions 1, 4, 10, 11, 20, and 21. We performed 1500 iterations on each function, with a dimension of 30 and a population size of 50. To prevent the colliding particles from falling into the local optimum, it is necessary to determine the distance relationship between the colliding particles and the global optimum, as shown in [Eq. \(11\)](#). To avoid the distance difference being too large or too small, we use the average distance calculation.

$$D_{\text{avg}} = \sum_{m \in \text{Colpop}} \|pop_m - gBest\|_2 \quad (11)$$

where $m \in \text{Colpop}$ represents the number of collisions and $\|\cdot\|_2$ represents the Euclidean norm. Therefore, the free movement step factor λ can be expressed by [Eq. \(12\)](#):

$$\lambda = \frac{d}{n} = \frac{\bar{X}_{\text{best}}}{2D_{\text{avg}} N^{\bar{V}}} \quad (12)$$

where $\bar{V}_{\text{best}}$ represents the average value of the global optimal particle, and $\bar{V}$ represents the average value of the collision particles. As the number of collisions increases, it also reflects that the interval and speed between particles are gradually decreasing. At this time, they need to be corrected as shown in [Eq. \(13\)](#), and the equations are rewritten as

$$\lambda = \frac{1}{2\beta D N} \quad (13)$$

$$\beta \approx \frac{4}{\pi} \left(1 + 0.4 \left(\frac{\sigma}{\bar{V}}\right)^2\right) \quad (14)$$

$$\lambda = \begin{cases} \frac{\bar{V}_{\text{best}}}{2D_{\text{avg}} N \bar{V}}, & m < pop_{\text{size}}, \\ \frac{1}{2\beta D N}, & m > pop_{\text{size}}. \end{cases} \quad (15)$$

where σ represents the variance of the velocity. Therefore, the particle update using the free movement step factor λ will be expressed as [Eq. \(16\)](#), and the free-step movement factor λ varies as shown in [Fig. 7](#). To facilitate understanding of FSO, the pseudo code is shown in Algorithm 1.

$$pop_i = pop_i + \lambda \cdot pop_i \quad (16)$$

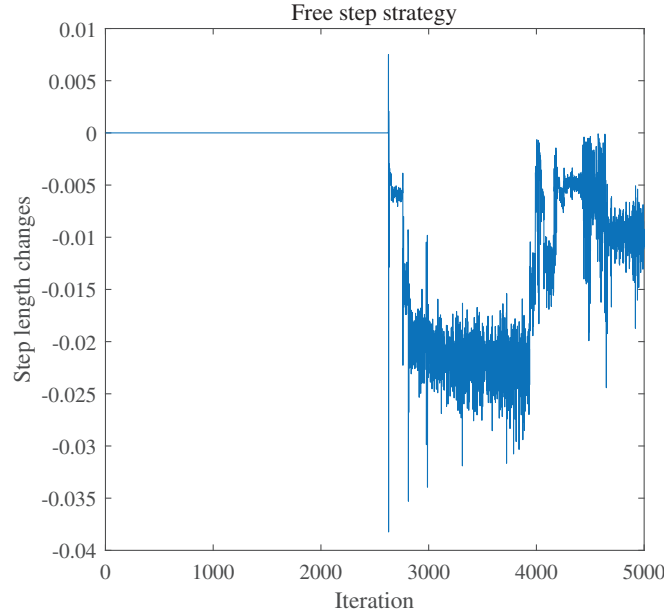


Figure 7: FSO free-step factor λ variation curve

Algorithm 1 : Pseudocode of freshwater snail optimizer

- 1: **Initialization:** Initialize the population size Pop_{size} , dimension D , initialize the population pop by Eq. (1), initialize the velocity by Eq. (2).
 - 2: Initialize individual optimal, global optimal, individual optimal value and global optimal value.
 - 3: **while** $Iter < Iter_{max}$ **do**
 - 4: **for each** pop **do**
 - 5: Calculate the buoyancy coefficient A by Eq. (5).
 - 6: **if** $V_{air} > r_1$ **then**
 - 7: Update the new freshwater snail position of pop by Eq. (8).
 - 8: **else**
 - 9: Update the new freshwater snail position of pop by Eq. (9).
 - 10: **end if**
 - 11: Evaluate the fitness value of new pop and compare it to the global optimum.
 - 12: Calculate the number of collisions and the free step factor λ .
 - 13: Update the new freshwater snail position of pop by Eq. (16).
 - 14: **end for**
 - 15: **end while**
 - 16: **Output:** the global optimum $gBest$ and $gBest_{vel}$.
-

2.3 Time Complexity Analysis

The time complexity of the metaheuristic algorithm comes from its primary calculation process. The time complexity analysis of the FSO proposed in this paper can be divided into the following parts:

- (1) **Initialization:** It takes $O(D \times Pop_{size})$ time to generate Pop_{size} individuals with D dimensions during the initialization process.
- (2) **Exploration and exploitation phases:** The time complexity of calculating A is $O(1)$, and updating the position of the water snail requires $O(Pop_{size})$. Therefore, the total time complexity for the exploration and exploitation stages over $maxGen$ iterations is $O(maxGen \times D \times Pop_{size})$.
- (3) **Calculation of free step length:** The time complexity of calculating the number of collisions and the average distance is $O(1)$, and updating the position of the water snail is $O(D \times Pop_{size})$.
- (4) **Fitness evaluation:** Let E be the time required for a single function evaluation. During $maxGen$ iterations, each iteration evaluates the fitness of Pop_{size} individuals, which takes $O(maxGen \times Pop_{size} \times E)$ time in total.

Therefore, the total time complexity of FSO is

$$\begin{aligned}
 \text{Timecomplexity}(FSO) &= O(\text{Init}) + O(\text{Explore\&Exploit}) + O(\text{Free step strategy}) + O(\text{Calculate fitness}) \\
 &= O(D \times Pop_{size}) + O(maxGen \times D \times Pop_{size}) \\
 &\quad + O(D \times Pop_{size}) + O(maxGen \times Pop_{size} \times E) \\
 &= O(maxGen \times D \times Pop_{size} + maxGen \times Pop_{size} \times E).
 \end{aligned}$$

Although we increase the computation of A , the time complexity does not increase compared to other metaheuristic algorithms [43]. As shown in Table 2, FSO exhibits comparable computational complexity to classical algorithms like PSO and SCA, with $O(maxGen \times D \times Pop_{size} + maxGen \times Pop_{size} \times E)$. The main difference lies in the specific operations within each iteration:

- **FSO:** Beyond the standard position update, one additional free-step-length computation and a A update are performed. Both are vectorized across individuals and dimensions, so the per-iteration cost remains $O(N \times D)$ with only a small constant factor increase, while yielding stronger global exploration.
- **PSO:** Each particle performs a velocity update followed by a position update. The per-iteration complexity is likewise $O(N \times D)$, but PSO lacks FSO's adaptive step-size/exploration mechanism.
- **GWO:** Encircling with three leaders (α, β, δ) induces roughly three groups of vector operations per individual. The asymptotic complexity remains $O(N \times D)$, with a slightly larger constant in practice.
- **SCA:** Position updates rely on trigonometric functions. Although still $O(N \times D)$ in theory, the use of $\sin / \cos$ typically raises the constant factor.

Table 2: Computational complexity comparison of different algorithms

Algorithm	Initialization	Main loop	Total complexity
FSO	$O(N \times D)$	$O(T \times N \times D)$	$O(T \times N \times D + T \times N \times E)$
PSO	$O(N \times D)$	$O(T \times N \times D)$	$O(T \times N \times D + T \times N \times E)$
GWO	$O(N \times D)$	$O(T \times N \times 3D)$	$O(T \times N \times D + T \times N \times E)$
SCA	$O(N \times D)$	$O(T \times N \times D)$	$O(T \times N \times D + T \times N \times E)$

The key advantage of FSO is that while maintaining the same order of complexity, the A mechanism provides better solution quality without significant computational overhead.

2.4 FSO Convergent Proof

To analyze the convergence properties of the FSO algorithm, we convert to the optimization process as a discrete-time Markov chain. Let $\mathcal{X}_t = \{X_1^{(t)}, \dots, X_{popsize}^{(t)}\}$ denote the population state at iteration t , where $X_i^{(t)} \in \mathcal{S} \subseteq \mathbb{R}^D$ represents the position of the i -th individual in the D -dimensional search space. The state space of the entire population is defined as $\Omega = \mathcal{S}^{popsize}$. The Eq. (17) shows the state transition of FSO satisfies the Markov property.

$$P(\mathcal{X}_{t+1} | \mathcal{X}_0, \dots, \mathcal{X}_t) = P(\mathcal{X}_{t+1} | \mathcal{X}_t) \quad (17)$$

This property indicates that the future state depends only on the current state, independent of the historical trajectory.

2.4.1 Transition Probability

The overall transition probability kernel of the FSO algorithm can be decomposed as a composition of three phase-specific kernels in Eq. (19):

$$\mathcal{P}(\mathcal{X}_t, \mathcal{X}_{t+1}) = \mathcal{P}_{exp} \circ \mathcal{P}_{expl} \circ \mathcal{P}_{col} \quad (18)$$

where $\mathcal{P}_{exp}$, $\mathcal{P}_{expl}$, and $\mathcal{P}_{col}$ correspond to the exploration, exploitation, and collision-aggregation phases, respectively.

2.4.2 Convergence Analysis

We establish the global convergence property of the FSO algorithm through the following theorem.

Theorem 1 (Global Convergence). *The Markov chain $\{\mathcal{X}_t\}_{t=0}^{\infty}$ generated by the FSO algorithm converges almost surely to the global optimal solution set $\mathcal{S}^*$.*

Proof: To establish global convergence, we verify three essential properties of the Markov chain:

(i) **Irreducibility.** The stochastic nature of the exploration phase ensures that any state in the search space can be reached from any other state with positive probability. Specifically, for any $\mathcal{X}, \mathcal{Y} \in \Omega$, there exists a positive integer n such that:

$$\mathcal{P}^n(\mathcal{X}, \mathcal{Y}) > 0 \quad (19)$$

This property guarantees that the algorithm maintains global search capability throughout the optimization process.

(ii) **Aperiodicity.** The collision mechanism introduces self-loop transitions with probability $P(d \geq threshold) > 0$, ensuring that the chain is aperiodic. This prevents the algorithm from being trapped in cyclic behaviors.

(iii) **Monotonicity.** The elite preservation mechanism through $gbestPos$ ensures monotonic improvement. We define the Lyapunov function as Eq. (20):

$$V(\mathcal{X}_t) = \min_{i=1, \dots, popsize} f(X_i^{(t)}) - f^* \quad (20)$$

where f^* denotes the global optimal value. This function satisfies the contraction property by Eq. (21):

$$\mathbb{E}[V(\mathcal{X}_{t+1})|\mathcal{X}_t] \leq \rho \cdot V(\mathcal{X}_t) \quad (21)$$

with contraction factor Eq. (22):

$$\rho = 1 - \frac{ER_{Iter}}{popsize} \cdot P_{improvement} < 1 \quad (22)$$

where ER_{Iter} is the explorer ratio at iteration $Iter$, and $P_{improvement}$ represents the probability of finding an improved solution. $\square$

2.4.3 Convergence Rate Analysis

The dynamic adjustment of the explorer ratio plays a crucial role in balancing exploration and exploitation:

$$ER_{Iter} = ER_{init} - (ER_{init} - ER_{final}) \frac{Iter}{maxIter} \quad (23)$$

This linear decrease from $ER_{init} = 0.5$ to $ER_{final} = 0.1$ ensures a smooth transition from global exploration to local exploitation. Under this scheme, the convergence rate can be characterized as Eq. (24):

$$\|\mathcal{X}_t - \mathcal{X}^*\| \leq \mathcal{O}(\rho^t) \quad (24)$$

indicating geometric convergence to the optimal solution. Therefore, The convergence analysis reveals that the FSO algorithm achieves global convergence through the synergistic interaction of its three phases:

- The exploration phase maintains global search capability by ensuring irreducibility of the Markov chain;
- The exploitation phase accelerates local convergence through adaptive step size control;
- The collision-aggregation mechanism balances population diversity and convergence speed while ensuring aperiodicity.

Therefore, with the recommended parameter settings, our proposed FSO satisfies Eq. (25):

$$\lim_{t \rightarrow \infty} P(\mathcal{X}_t \in \mathcal{N}_\epsilon(\mathcal{X}^*)) = 1 \quad (25)$$

where $\mathcal{N}_\epsilon(\mathcal{X}^*)$ denotes the ϵ -neighborhood of the global optimal solution. This result establishes the almost sure convergence of the FSO algorithm to the global optimum.

3 Experimental Analysis

All experimental results in this article are from a ROG laptop equipped with an Intel (R) Core (TM) i9-14900HX processor, 32 GB memory, and 64-bit Windows 11 operating system. The software used in the experiment is MATLAB R2023b. The experimental results are analyzed based on the optimal value of each run, and the optimal average (Avg.) and standard deviation (Std.) are analyzed. The bold parts in all tables in this section represent the optimal results of that row.

3.1 Benchmark Test Sets

We will compare the proposed FSO algorithm with various other algorithms on the benchmarks of CEC 2017, and CEC 2022, which testing in 50 dimensions in CEC 2017, 10 and 20 dimensions in CEC 2022.

3.2 Parameter Settings

In the experiment, the maximum population size $popSize = 50$ of each algorithm, and the maximum number of evaluations is $maxGen = 1500$ and $maxGen = 3000$ in 50-dimension. Each algorithm is run 20 times independently, and the comparison algorithm parameter settings are shown in Tables 3 and 4. All independent experimental results were statistically tested using the Wilcoxon rank sum test. “+” indicates that FSO is significantly better than the comparison algorithm, and “ $\approx$ ” indicates that there is no significant difference between FSO and the comparison algorithm.

Table 3: Parameter settings for the classic metaheuristic algorithms

Algorithm	Parameter(s)
AOA	$\alpha = 5, \mu = 0.499, MOP_{\max} = 1, MOP_{\min} = 0.2$
COA	$I \in \{1, 2\}$
RSO	$x = 1, y = 5$
GWO	$ae \in [2, 0]$
PSO	$c_1 = c_2 = 0.5$
SCA	$a = 2$
BH	None
WOA	$ae \in [2, 0]$
MFO	$b = 1$

Table 4: Parameter settings for the emerging metaheuristic algorithms

Algorithm	Parameter(s)
GGO	None
GOOSE	$coe = 0.17$
HO	$\beta = 1.5$
SGA	$T = 1$
PKO	$BF = 8, PE_{\max} = 0.5, PE_{\min} = 0$
PO	$\beta = 1.5$
SCHO	$ct = 3.6, u = 0.388, m = 0.45, \varepsilon = 0.003, n = 0.5, \alpha = 4.6, \beta = 1.55, p = 10, q = 9$
NRBO	$DF = 0.6, \text{Flag} = 1$
NOA	$p_{rp} = 0.2, p_{a_2} = 0.4, s = 0.05$

3.3 Experimental Results and Analysis

We designed two sets of comparative experiments. The first set compared the proposed FSO with nine classic algorithms, including AOA [44], COA [45], RSO [46], GWO, PSO, SCA [47], BH [48],

WOA, and MFO [49]. The second set compared the proposed FSO with nine new algorithms published in the past year, including GGO [50], GOOSE [51], HO [52], SGA [53], PKO [54], PO [55], SCHO [56], NRBO [57], and NOA [58].

We first marked CEC 2017 (50 dimensions), and CEC 2022 (10 and 20 dimensions) with the classic algorithms. Then, we used CEC 2017 (50 dimensions), and CEC 2022 (20 dimensions) to compare with the new algorithms and perform related data analysis in parallel. This hierarchical approach allows us to assess both the fundamental optimization capabilities and the scalability of FSO in challenging search spaces.

3.3.1 CEC 2017 Benchmark Test Experiment Results and Analysis

The CEC 2017 benchmark test set [59] contains 29 real-valued continuous optimization problems, including 2 unimodal functions (F1~F3, without F2) with a single global optimal solution, which aims to examine the global search ability of the algorithm; 7 multimodal functions (F4~F10) with multiple local optimal solutions, which challenge the algorithm's ability to find the global optimal solution and escape from the local optimal solution; 10 hybrid functions (F11~F20) that combine unimodal and multimodal characteristics to test the algorithm's adaptability in complex search spaces; and 10 composition functions (F21~F30) that are used to evaluate the algorithm's global search and local development capabilities, which are composed of multiple unimodal or multimodal functions.

As shown in Table 5, FSO achieved the best results in 24 of the 29 test functions, and only slightly lagged behind the current optimal algorithm in F3, F10, F19, F20, and F30, but the gap with the optimal value remained at a minimal level. Meanwhile, the standard deviation of FSO on most functions is also at its lowest or second-lowest level, indicating that the algorithm not only possesses a strong global search capability but also exhibits excellent stability and reproducibility in a high-dimensional search space. Overall, the experimental results demonstrate that FSO has excellent adaptability and robustness in high-dimensional, complex optimization problems, providing strong support for its promotion and application in larger-scale, more complex engineering optimization tasks.

In the comparison of emerging algorithms, as shown in Table 6, FSO demonstrated significant advantages in the 50-dimensional CEC 2017 test, particularly in the context of multi-dimensional, complex optimization problems, showing its excellent global search ability and optimization efficiency. In the 50-dimensional benchmark test, FSO only achieved suboptimal or inferior place in the optimal mean in the test functions of F1, F3, F9, F10, F11, F12, F13, F15, F18, F19, and F30. In the vast majority of other benchmark tests, FSO's optimization performance surpassed other metaheuristic algorithms proposed in the past year. This result fully demonstrates the excellent performance and stability of FSO in high-dimensional optimization tasks. In addition, FSO's advantage in the STD evaluation indicator is even more prominent. In 14 benchmark tests, FSO successfully won first place. As a new metaheuristic optimization algorithm, FSO exhibits good robustness and state-of-the-art (SOTA) performance compared with other emerging algorithms.

Table 5: Comparison results with classic algorithm groups on the 50-dimensional CEC 2017 test function

F	Val	FSO		AOA		COA		RSO		GWO		PSO		SCA		BH		WOA		MFO	
		Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T
F1	avg	223900	1.08E+11	+ 13667522	+ 7.36E+10	+ 5.79E+09	+ 6.81E+08	+ 6.81E+08	+ 6.81E+08	+ 5.79E+09	+ 6.81E+08	+ 6.81E+08	+ 6.81E+08	+ 4.75E+10	+ 3.23E+10	+ 1.61E+08	+ 3.42E+10	+ 1.61E+08	+ 3.42E+10	+ 1.61E+08	+ 3.42E+10
	std	49848.07	7.69E+09	15016255	9.89E+09	3.64E+09	1.44E+09	1.44E+09	1.44E+09	3.64E+09	1.44E+09	1.44E+09	1.44E+09	6.8E+09	1.18E+09	46725552	1.5E+10	46725552	1.5E+10	46725552	1.5E+10
F2	avg	0	0	+ 0	+ 0	+ 0	+ 0	+ 0	+ 0	+ 0	+ 0	+ 0	+ 0	+ 0	+ 0	+ 0	+ 0	+ 0	+ 0	+ 0	+ 0
	std	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F3	avg	88799.92	172202.4	+ 209937.6	+ 160848.1	+ 89286.74	+ 10360.17	+ 89286.74	+ 89286.74	+ 89286.74	+ 89286.74	+ 89286.74	+ 89286.74	+ 119099.6	+ 139490.4	+ 147477.1	+ 189127	+ 147477.1	+ 189127	+ 147477.1	+ 189127
	std	11419.83	22775.04	41819.72	1.55E+04	22142.3	4194.331	22142.3	22142.3	22142.3	22142.3	22142.3	22142.3	15862.2	17963.34	56379.66	98018.23	56379.66	98018.23	56379.66	98018.23
F4	avg	537.9823	29227.21	+ 599.3411	+ 17496.22	+ 1051.913	+ 664.1674	+ 1051.913	+ 1051.913	+ 1051.913	+ 1051.913	+ 664.1674	+ 664.1674	+ 7707.077	+ 6584.865	+ 854.701	+ 4848.733	+ 6584.865	+ 854.701	+ 4848.733	+ 6584.865
	std	36.63633	4486.704	71.88941	2954.751	310.4525	142.5707	310.4525	310.4525	310.4525	310.4525	142.5707	142.5707	1349.084	460.5101	113.0078	2609.039	1349.084	460.5101	113.0078	2609.039
F5	avg	636.7167	1125.62	+ 871.5851	+ 1076.723	+ 713.2062	+ 777.7405	+ 713.2062	+ 713.2062	+ 713.2062	+ 713.2062	+ 777.7405	+ 777.7405	+ 1072.805	+ 910.9909	+ 958.9401	+ 974.0107	+ 1072.805	+ 910.9909	+ 958.9401	+ 974.0107
	std	29.9843	39.40149	20.73915	34.69133	28.4437	38.93469	28.4437	28.4437	28.4437	28.4437	38.93469	38.93469	37.11922	68.80682	90.90039	76.97034	37.11922	68.80682	90.90039	76.97034
F6	avg	603.1575	690.9327	+ 653.3493	+ 694.3322	+ 614.8454	+ 659.0615	+ 614.8454	+ 614.8454	+ 614.8454	+ 614.8454	+ 659.0615	+ 659.0615	+ 674.0782	+ 676.7913	+ 683.846	+ 652.3646	+ 674.0782	+ 676.7913	+ 683.846	+ 652.3646
	std	5.489538	6.529082	10.41037	5.008233	4.817236	3.668136	4.817236	4.817236	4.817236	4.817236	4.817236	4.817236	6.470367	6.428433	10.73526	8.175246	6.470367	6.428433	10.73526	8.175246
F7	avg	961.2793	1913.8	+ 1667.709	+ 1776.634	+ 1061.283	+ 1277.961	+ 1061.283	+ 1061.283	+ 1061.283	+ 1061.283	+ 1277.961	+ 1277.961	+ 1682.907	+ 1678.488	+ 1715.161	+ 1812.902	+ 1682.907	+ 1678.488	+ 1715.161	+ 1812.902
	std	41.55341	67.63632	116.6059	63.91906	71.15406	109.2017	71.15406	71.15406	71.15406	71.15406	109.2017	109.2017	85.10938	150.0885	137.2567	370.1102	85.10938	150.0885	137.2567	370.1102
F8	avg	922.5409	1454.546	+ 1204.057	+ 1390.002	+ 1028.312	+ 1119.973	+ 1028.312	+ 1028.312	+ 1028.312	+ 1028.312	+ 1119.973	+ 1119.973	+ 1382.746	+ 1248.4	+ 1249.757	+ 1262.289	+ 1382.746	+ 1248.4	+ 1249.757	+ 1262.289
	std	21.18716	43.08762	32.14039	45.34692	36.53211	39.96796	36.53211	36.53211	36.53211	36.53211	39.96796	39.96796	31.4623	52.21616	65.27025	73.46465	31.4623	52.21616	65.27025	73.46465
F9	avg	2237.079	25295.73	+ 19780.46	+ 31582.39	+ 6871.095	+ 15792.46	+ 6871.095	+ 6871.095	+ 6871.095	+ 6871.095	+ 15792.46	+ 15792.46	+ 24061.48	+ 19170.88	+ 26729.39	+ 16203.28	+ 24061.48	+ 19170.88	+ 26729.39	+ 16203.28
	std	1870.793	2299.296	3016.254	3246.157	4064.644	2354.269	4064.644	4064.644	4064.644	4064.644	2354.269	2354.269	4073.125	5477.667	9289.802	3195.099	4073.125	5477.667	9289.802	3195.099
F10	avg	8184.998	13175.01	+ 10275.09	+ 13343.84	+ 7065.538	+ 8397.473	+ 7065.538	+ 7065.538	+ 7065.538	+ 7065.538	+ 8397.473	+ 8397.473	+ 14690.68	+ 12810.7	+ 10663.68	+ 8971.717	+ 14690.68	+ 12810.7	+ 10663.68	+ 8971.717
	std	843.2895	655.8861	1346.101	1387.572	866.4877	960.7367	866.4877	866.4877	866.4877	866.4877	960.7367	960.7367	486.6018	1378.936	1313.856	1128.879	1378.936	1313.856	1313.856	1128.879
F11	avg	1357.635	20635.79	+ 1504.084	+ 12381.17	+ 4135.387	+ 1502.906	+ 4135.387	+ 4135.387	+ 4135.387	+ 4135.387	+ 1502.906	+ 1502.906	+ 6635.588	+ 8062.921	+ 1982.337	+ 10675.98	+ 6635.588	+ 8062.921	+ 1982.337	+ 10675.98
	std	37.65011	2689.438	136.9878	1797.872	1973.398	87.61963	1973.398	1973.398	1973.398	1973.398	87.61963	87.61963	1131.321	762.4347	233.9019	10416.92	1131.321	762.4347	233.9019	10416.92
F12	avg	17446916	6.53E+10	+ 20158159	+ 5.11E+10	+ 8.85E+08	+ 3.7E+08	+ 8.85E+08	+ 8.85E+08	+ 8.85E+08	+ 8.85E+08	+ 3.7E+08	+ 3.7E+08	+ 1.36E+10	+ 1.52E+10	+ 3.99E+08	+ 4.45E+09	+ 1.36E+10	+ 1.52E+10	+ 3.99E+08	+ 4.45E+09
	std	12118112	1.48E+10	10539965	7.47E+09	9.85E+08	7.92E+08	9.85E+08	9.85E+08	9.85E+08	9.85E+08	7.92E+08	7.92E+08	2.7E+09	1.5E+09	2.06E+08	3.92E+09	2.7E+09	1.5E+09	2.06E+08	3.92E+09
F13	avg	90858.62	2.19E+10	+ 127691.6	+ 2.46E+10	+ 99043603	+ 26376163	+ 99043603	+ 99043603	+ 99043603	+ 99043603	+ 26376163	+ 26376163	+ 3.8E+09	+ 3.66E+09	+ 1560570	+ 9.46E+08	+ 3.8E+09	+ 3.66E+09	+ 1560570	+ 9.46E+08
	std	66931.71	7.03E+09	61808.68	6.51E+09	94969514	75510300	94969514	94969514	94969514	94969514	75510300	75510300	1.48E+09	7.78E+08	1332855	2E+09	1.48E+09	7.78E+08	1332855	2E+09
F14	avg	32539.32	21587165	+ 204107.8	+ 10418534	+ 535932.9	+ 755400.6	+ 535932.9	+ 535932.9	+ 535932.9	+ 535932.9	+ 755400.6	+ 755400.6	+ 3014060	+ 2741354	+ 1680616	+ 1185993	+ 3014060	+ 2741354	+ 1680616	+ 1185993
	std	21632.6	20210111	127624.5	4272343	597167.2	2132211	597167.2	597167.2	597167.2	597167.2	2132211	2132211	1286602	862162.9	1049373	2479370	1286602	862162.9	1049373	2479370
F15	avg	28461.85	4.54E+08	+ 32808.6	+ 2.4E+09	+ 10758262	+ 35926.37	+ 10758262	+ 10758262	+ 10758262	+ 10758262	+ 35926.37	+ 35926.37	+ 3.78E+08	+ 1491906	+ 181799.7	+ 2.29E+08	+ 3.78E+08	+ 1491906	+ 181799.7	+ 2.29E+08
	std	13295.95	5.56E+08	10425.09	6.53E+08	18885727	18868.3	18885727	18885727	18885727	18885727	18868.3	18868.3	1.82E+08	854956.4	118875.7	5.12E+08	1.82E+08	854956.4	118875.7	5.12E+08
F16	avg	2864.406	7357.394	+ 3517.901	+ 5866.421	+ 3020.271	+ 3626.437	+ 3020.271	+ 3020.271	+ 3020.271	+ 3020.271	+ 3626.437	+ 3626.437	+ 5612.298	+ 6140.45	+ 5155.592	+ 4171.965	+ 5612.298	+ 6140.45	+ 5155.592	+ 4171.965
	std	425.481	1763.87	381.5185	799.9905	341.0352	487.0185	799.9905	799.9905	799.9905	799.9905	487.0185	487.0185	435.7695	650.0323	748.6927	575.7849	435.7695	650.0323	748.6927	575.7849
F17	avg	2750.466	5850.738	+ 3083.126	+ 5558.219	+ 2832.42	+ 3296.359	+ 2832.42	+ 2832.42	+ 2832.42	+ 2832.42	+ 3296.359	+ 3296.359	+ 4457.271	+ 3745.861	+ 4042.417	+ 3996.339	+ 4457.271	+ 3745.861	+ 4042.417	+ 3996.339
	std	171.5351	984.0885	378.6129	664.1061	302.7432	298.6119	302.7432	302.7432	302.7432	302.7432	298.6119	298.6119	237.3425	365.4874	502.5168	433.5614	237.3425	365.4874	502.5168	433.5614

(Continued)

Table 5 (continued)

F	Val	FSO	AOA		COA		RSO		GWO		PSO		SCA		BH		WOA		MFO	
			Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T
F18	avg	539865.3	56464257	+	1877408	+	35937059	+	3902995	+	885173.7	+	22496485	+	9041000	+	8109012	+	12323799	+
	std	352048.7	34256001		1555925		34696045		3992705		552497.3		7262663		1563274		5001070		24475645	
F19	avg	228421.9	3.63E+08	+	25944.24	+	3.4E+09	+	3162140	+	86192.29	≈	3.66E+08	+	497884.3	+	3586244	+	52258008	+
	std	227964.6	4.14E+08		14599.45		9.16E+08		5912121		61937.77		1.96E+08		183689		3437902		1.87E+08	
F20	avg	2813.457	3622.883	+	3525.191	+	3692.171	+	2776.089	+	3307.296	+	3992.925	+	3494.885	+	3702.355	+	3598.391	+
	std	196.2873	248.4432		338.7036		276.4171		248.6081		368.0281		189.6254		328.8753		344.7407		440.4968	
F21	avg	2401.815	3043.535	+	2681.497	+	3056.417	+	2500.864	+	2677.83	+	2900.299	+	2906.71	+	2930.611	+	2735.672	+
	std	23.4734	74.65066		106.8433		59.29899		59.19675		45.70399		39.77202		87.78896		80.20558		55.09027	
F22	avg	4588.316	15284.16	+	12270.91	+	16079.79	+	8775.485	+	10684.07	+	16354.06	+	14891.01	+	12606.03	+	10571.16	+
	std	3619.936	747.3238		1030.263		626.2147		1345.98		1123.345		395.7523		1146.026		832.6882		773.7732	
F23	avg	2823.769	4368.484	+	3205.925	+	3845.788	+	2965.649	+	3616.917	+	3547.718	+	4130.712	+	3649.615	+	3175.24	+
	std	20.24655	244.2143		130.2636		96.51086		73.74598		218.4391		65.86595		159.7131		167.7354		59.06018	
F24	avg	2988.661	4919.214	+	3296.463	+	4207.935	+	3131.104	+	3664.003	+	3723.683	+	4476.711	+	3715.736	+	3234.874	+
	std	27.58914	257.4798		104.1845		150.8787		72.5593		161.4156		54.48765		146.398		129.0776		68.0968	
F25	avg	3027.9	14804.52	+	3119.413	+	12346.6	+	3564.883	+	3114.054	+	6598.893	+	5500.439	+	3261.597	+	6114.779	+
	std	13.5144	1873.826		38.54563		877.278		200.0171		31.90462		575.2098		146.4642		89.87756		2299.446	
F26	avg	2901.75	16181	+	10596.94	+	13790.07	+	6295.707	+	10217.95	+	12442.72	+	12611.03	+	13001.47	+	8253.896	+
	std	0.229152	1082.462		2237.9		718.3481		508.2311		2650.101		635.7655		824.2698		1544.705		914.3894	
F27	avg	3280.532	6713.103	+	3639.896	+	5358.243	+	3542.282	+	3866.447	+	4539.874	+	5991.326	+	4401.704	+	3586.947	+
	std	24.40481	416.7966		136.2516		563.9839		70.50362		254.5208		244.0413		268.673		540.796		121.4149	
F28	avg	3285.073	11713.09	+	3383.278	+	8171.183	+	4168.957	+	3399.334	+	7162.5	+	6255.682	+	3785.722	+	7846.64	+
	std	22.38702	1212.228		46.6459		568.4161		398.2249		127.4588		557.8903		139.4436		218.6257		1074.274	
F29	avg	4152.411	22495.79	+	5051.381	+	15648.93	+	4369.214	+	5560.192	+	7440.47	+	9419.107	+	7511.131	+	5294.481	+
	std	210.7282	11251.07		305.734		3302.696		320.0599		412.151		539.8834		715.0711		757.858		540.6186	
F30	avg	56377310	3.57E+09	+	3945801	+	3.69E+09	+	94330194	+	18454584	+	8.15E+08	+	2.51E+08	+	1.57E+08	+	83931187	+
	std	8686235	3.17E+09		1393737		1.02E+09		31432088		9973929		2.52E+08		15041391		82556585		1.35E+08	
avg/std win times		24/17	0/0		2/7		0/0		2/0		1/2		0/3		0/0		0/0		0/0	

Note: Bold values indicate the best performance for each test function. “+” and “≈” denote the results of the Wilcoxon rank-sum test compared with FSO ($\alpha = 0.05$).

Table 6: Comparison results with emerging algorithm groups on the 50-dimensional CEC 2017 test function

F	Val	FSO	GGO		HO		SGA		PKO		PO		SCHO		NRBO50		NOA			
			Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T		
F1	avg	223900	7.5E+10	+	7433.097	+	2926790	+	1.05E+09	+	1504.287	+	2.51E+08	+	2.54E+10	+	4.31E+10	+	1.98E+11	+
	std	49848.07	1.02E+10		6211.858		6218792		1.31E+09		1758.946		3.01E+08		8.16E+09		6.78E+09		1.11E+10	
F2	avg	0	0	+	0	+	0	+	0	+	0	+	0	+	0	+	0	+	0	+
	std	0	0		0		0		0		0		0		0		0		0	
F3	avg	88799.92	249760	+	3408.637	+	78904.79	+	68997.52	+	104693.3	+	94106.51	≈	111341.4	+	104670.8	+	442711.9	+
	std	11419.83	54356.21		3514.437		1.01E+04		16606.25		18539.78		11754.98		20293.82		14418.97		85985.87	
F4	avg	537.9823	17809.03	+	541.2877	+	765.4609	+	751.1076	+	550.1033	+	746.5508	+	4513.278	+	6091.141	+	66341.93	+
	std	36.63633	5162.884		39.02944		66.69696		90.80396		30.64991		64.0242		1739.785		1467.323		7276.05	
F5	avg	636.7167	1043.567	+	1078.835	+	850.5297	+	864.1213	+	653.1266	+	876.9592	+	906.6508	+	1071.838	+	1523.708	+
	std	29.9843	72.3226		165.2091		31.35854		55.17382		42.74758		44.64055		41.78722		55.9519		45.7405	
F6	avg	603.1575	675.8899	+	676.3705	+	663.4983	+	671.5049	+	605.2165	+	668.8698	+	660.7174	+	685.6052	+	731.2986	+
	std	5.489538	8.250811		8.799026		5.898928		6.837951		1.896024		6.333848		7.851195		8.072546		5.566536	
F7	avg	961.2793	1845.916	+	2892.625	+	1555.853	+	1655.481	+	968.6253	+	1586.272	+	1441.227	+	1688.949	+	5155.854	+
	std	41.55341	144.1577		1512.614		51.29432		114.513		28.1761		136.0051		102.1715		122.8185		243.6299	
F8	avg	922.5409	1356.662	+	1420.808	+	1137.62	+	1208.986	+	949.1316	+	1209.654	+	1199.586	+	1391.112	+	1812.646	+
	std	21.18716	58.87151		130.8428		30.48513		58.46599		25.58442		40.93835		53.14015		67.05377		59.30928	
F9	avg	2237.079	24029.91	+	18642.52	+	12739.63	+	14731.9	+	1547.726	+	17181.34	+	30318.97	+	25765.42	+	75009.51	+
	std	1870.793	4041.36		7267.006		1734.807		2458.138		780.284		3290.357		8272.732		4182.683		6577.412	
F10	avg	8184.998	12345.26	+	9254.458	+	7969.87	+	9821.381	≈	7675.852	+	9707.835	+	9854.648	+	13212.46	+	16386.82	+
	std	843.2895	848.7371		1125.194		597.2683		1195.405		885.0964		1421.46		1188.987		925.692		438.987	
F11	avg	1357.635	20891.1	+	1373.673	+	1597.212	+	1734.471	+	1244.886	+	1916.277	+	7208.461	+	6276.585	+	50424.22	+
	std	37.65011	7011.124		90.41805		122.9049		186.7907		46.39135		203.7611		3427.868		1755.488		8776.74	
F12	avg	17446916	4.33E+10	+	9699121	+	2.17E+08	+	1.89E+08	+	2979238	+	3.31E+08	+	6.68E+09	+	1.03E+10	+	8.48E+10	+
	std	12118112	9.75E+09		5036969		3.29E+08		1.38E+08		1313740		1.48E+08		6.2E+09		6.33E+09		1.42E+10	
F13	avg	90858.62	1.96E+10	+	162871.6	+	180849.1	+	269983	+	6973.099	+	3915976	+	3.25E+09	+	1.8E+09	+	3.93E+10	+
	std	66931.71	5.51E+09		84477.69		99955.47		176586.8		5760.875		3322014		5.13E+09		7.67E+08		9.75E+09	
F14	avg	32539.32	39983817	+	46251.29	+	760521.5	+	335568.8	+	282251.8	+	1016404	+	1891163	+	1194275	+	82079535	+
	std	21632.6	25931968		21972.13		611714.7		215113.8		207346.1		816964.2		1722420		909824		25219467	
F15	avg	28461.85	2.34E+09	+	57417.77	+	68347.54	+	52238.13	+	16233.05	+	243678.8	+	3.44E+08	+	82798206	+	1.38E+10	+
	std	13295.95	1.84E+09		59703.87		68001.55		31387.43		11255.38		437505.6		8.26E+08		66299918		4.34E+09	
F16	avg	2864.406	6463.38	+	4515.755	+	4561.021	+	4418.934	+	3199.422	+	4547.628	+	4273.357	+	5253.478	+	9883.632	+
	std	425.481	781.1174		468.1634		543.6516		665.0438		466.0673		606.354		726.6868		357.5436		961.1891	
F17	avg	2750.466	4349.426	+	3713.957	+	3626.796	+	3842.225	+	2757.823	+	3831.121	+	3360.529	+	4298.499	+	69959.09	+
	std	171.5351	359.0221		238.3537		228.9922		312.065		295.9317		468.4363		280.6945		362.8846		54544.18	

(Continued)

Table 6 (continued)

F	Val	FSO	GGO		GOOSE		HO		SGA		PKO		PO		SCHO		NRBO50		NOA	
			Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T
F18	avg	539865.3	86881930	+	317275.7	+	3914998	+	1672738	+	1020237	+	4872407	+	14604105	+	6978194	+	3.06E+08	+
	std	352048.7	35826230		165306.4		2167434		1577101		798714.1		4508083		20424160		3813028		1.25E+08	
F19	avg	228421.9	1.05E+09	+	927838	+	2748373	+	1371801	+	9803.78	+	4420728	+	17265566	+	4694727	+	5.79E+09	+
	std	227964.6	7.21E+08		403770.8		1484718		2262734		9196.746		3227230		63081388		36720876		1.31E+09	
F20	avg	2813.457	3694.994	+	3948.561	+	3074.495	+	3525.942	+	2912.891	+	3505.722	≈	3460.046	+	3550.935	+	4784.339	+
	std	196.2873	435.9603		304.0011		204.1631		301.3147		346.5784		275.242		294.0012		217.3744		150.5878	
F21	avg	2401.815	2925.458	+	3060.138	+	2742.761	+	2842.069	+	2438.701	+	2748.977	+	2738.414	+	2934.166	+	3325.469	+
	std	23.4734	56.99038		91.18413		64.76305		118.1249		35.43141		70.47937		37.69108		60.44939		69.41192	
F22	avg	4588.316	14729.78	+	10765.71	+	10653.38	+	12232.82	+	9488.721	+	11614.78	+	12087.28	+	15371.12	+	18250.45	+
	std	3619.936	894.4558		1088.911		812.6967		1351.034		1099.081		1319.008		904.6949		770.435		509.1725	
F23	avg	2823.769	4308.491	+	4179.826	+	3438.174	+	3663.926	+	2880.879	+	3407.314	+	3461.682	+	3514.015	+	4508.512	+
	std	20.24655	196.904		216.1125		146.2886		212.8891		35.9853		124.5682		145.7043		146.4518		167.0182	
F24	avg	2988.661	4811.223	+	4361.902	+	3786.396	+	3862.485	+	3039.446	+	3586.376	+	3695.62	+	3707.11	+	4921.663	+
	std	27.58914	262.339		157.32		163.9599		173.6964		39.21578		167.4878		110.528		79.85589		210.3417	
F25	avg	3027.9	10298.92	+	3056.46	+	3272.549	+	3229.232	+	3071.116	+	3231.288	+	5085.495	+	5973.577	+	39524.55	+
	std	13.5144	1092.835		26.5995		98.39039		61.03039		22.17379		74.00725		974.0779		775.2211		4172.829	
F26	avg	2901.75	15021.27	+	13129.83	+	11790.2	+	12060.44	+	5225.492	+	9843.359	+	10721.92	+	11955.11	+	23572.2	+
	std	0.229152	892.2395		7196.472		1748.629		1688.924		340.1367		3262.101		1157.554		1790.102		1503.981	
F27	avg	3280.532	6438.607	+	5408.582	+	4247.862	+	4157.21	+	3364.823	+	3970.183	+	4249.121	+	4194.945	+	6862.053	+
	std	24.40481	571.4697		577.2842		396.5224		378.9843		61.20894		248.573		355.2336		295.1008		406.1497	
F28	avg	3285.073	9361.304	+	3301.404	+	3883.476	+	3493.653	+	3332.596	+	3625.272	+	5574.135	+	6453.341	+	18448.73	+
	std	22.38702	1002.889		17.12218		163.1572		91.35169		19.29122		115.8431		568.1744		604.5204		1116.008	
F29	avg	4152.411	14608.04	+	6211.279	+	7414.644	+	6873.883	+	4497.212	+	6748.916	+	5774.17	+	7036.107	+	81895.23	+
	std	210.7282	4658.487		579.6792		931.8687		867.1311		238.5708		909.5146		567.998		709.4287		51382.88	
F30	avg	56377310	1.67E+09	+	21031053	+	1.44E+08	+	83282760	+	1031839	+	1.71E+08	+	1.48E+08	+	3.91E+08	+	8.9E+09	+
	std	8686235	9.86E+08		4469467		52054068		35399816		240948.9		57651723		62087533		1.28E+08		1.92E+09	
avg/std win times		18/13	0/0	2/3	0/0	0/0	9/10	0/0	0/0	0/0	0/1	0/2								

Note: Bold values indicate the best performance for each test function. “+” and “≈” denote the results of the Wilcoxon rank-sum test compared with FSO ($\alpha = 0.05$).

The Friedman test [60] is a nonparametric statistical method that can effectively eliminate the uncertainty associated with sample distribution assumptions. FSO consistently ranks first in the 30, and 50-dimensional scenarios of the CEC 2017 problem set, with mean ranks of 1.65, and 1.24, respectively. As shown in Fig. 8, compared with the classical algorithm group, COA (3.21), which ranked second in the 30-dimensional experiments, was pulled away by FSO by about 1.57 ranks, and the relative advantage increased by about 49%; in the 50-dimensional case, the mean rank of FSO further dropped to 1.24, and the advantage over the second-ranked GWO (3.13) expanded to about 60%. At the same time, AOA and RSO are both stable at the end, about 9.38 and 8.97 respectively, indicating that these algorithms have a significant degradation trend in high-dimensional optimization problems, while FSO's performance is not negatively affected by the increase in dimension, but shows more outstanding scalability and robustness. Further comparison of FSO with the nine emerging metaheuristic methods proposed in recent years shows, the mean rank of FSO is 1.62, which is again far ahead of the following ranked HO (3.45) by about 53%, and the gap with GGO and NOA is 4 to 6 times, highlighting the absolute competitiveness of FSO in emerging algorithms.

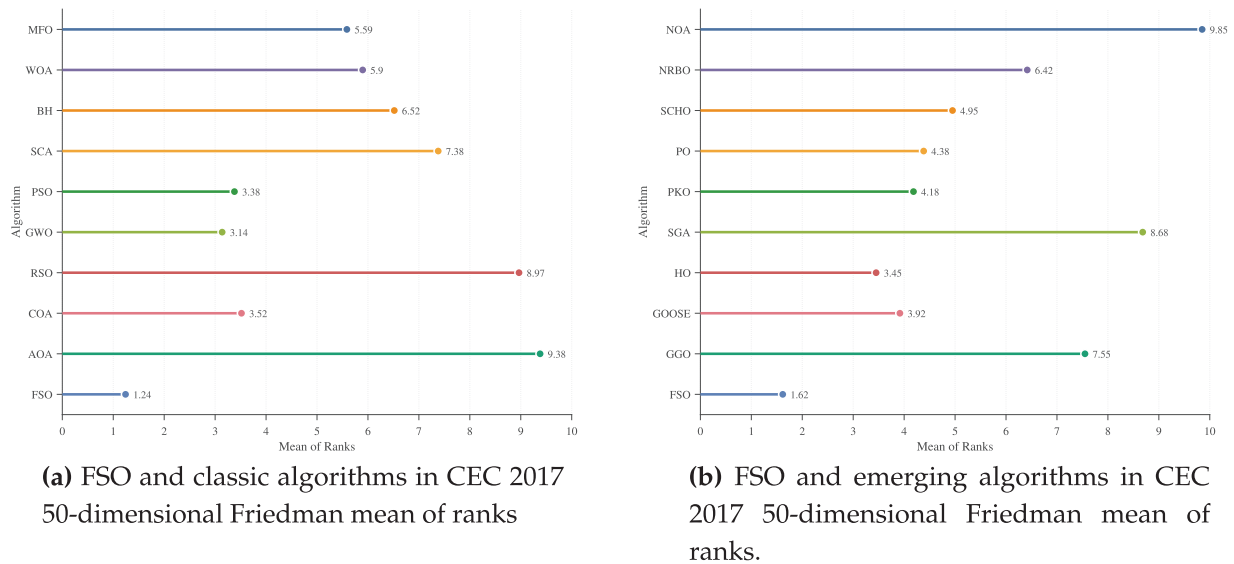


Figure 8: Friedman mean of ranks in CEC 2017. FSO and classic algorithms (a), FSO and emerging algorithms (b)

3.3.2 Parameter Sensitivity Analysis

The setting of key parameters has a crucial impact on the algorithm's optimization capabilities. To systematically evaluate the effect of the Threshold parameter, which used to detect swarm collisions in FSO, on search behavior, we conducted a parameter sensitivity analysis on F1, F3, F10, F11, F20, and F21 Benchmark Functions from the CEC 2017. We used the CEC 2017 benchmark function to conduct 20 independent experiments on F1, F3, F10, F11, F20, and F21 in 50 dimensions, with 1500 iterations per experiment for each Threshold $\in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. The experimental results in Table 7 show that F1 is most sensitive to Threshold, showing a clear U-shaped response. When Threshold = 0.5, FSO achieves the optimal average fitness of 1.3710×10^5 , which achieves 10.6% improvement compared with Threshold = 0.1 in F1. The mean differences between the functions F3 and F11 are less than 5%, indicating that the algorithm is robust to threshold settings for this problem.

The F10 monotonically degrades with increasing Threshold, suggesting that a smaller threshold is more conducive to maintaining the necessary exploration for this problem. The mean changes of F20 and F21 are insignificant, and their standard deviations are below 100. Fig. 9 shows the parameter sensitivity curves. When Threshold = 0.5, the optimal compromise is achieved and the combined average fitness of the six functions drops to 2.9762×10^4 , with a corresponding standard deviation of 6.3686×10^3 , the lowest among the five settings. The coefficient of variation is approximately 0.214, lower than the 0.269 for Threshold = 0.1 and the 0.245 for Threshold = 0.9, indicating that Threshold = 0.5 not only can improve expected performance but also maintains robustness. This indicates consistent with the mechanism by which the threshold regulates the intensity of exploration and exploitation in FSO, which a too small Threshold makes collision judgments overly sensitive, leading to premature clustering and maturation. A too large Threshold delays clustering signals, searching excessively randomly, and hindering convergence. A moderate Threshold = 0.5 promotes efficient exploitation while maintaining necessary exploration, and is adopted as the default setting.

Table 7: Sensitivity analysis of population collision threshold parameters

Function Value		Threshold = 0.1	Threshold = 0.3	Threshold = 0.5	Threshold = 0.7	Threshold = 0.9
F1	avg	153,454.9525	149,892.1132	137,104.9981	145,984.0041	158,110.6224
	std	46,897.87457	36,330.96214	33,244.97299	43,263.67516	43,462.14878
F3	avg	30,123.99887	30,031.54164	30,506.52146	30,599.96759	31,709.74248
	std	4770.285316	5362.594123	4191.344367	6275.674787	4953.816526
F10	avg	4823.357258	4988.838200	5018.760349	4862.691797	4877.457415
	std	519.4766093	679.8720887	670.7003784	556.8173946	670.8722559
F11	avg	1205.91333	1202.249371	1207.48284	1218.398523	1211.519482
	std	27.67435429	30.25022353	21.35307236	29.76592176	25.78174926
F20	avg	2409.237731	2408.688707	2369.021416	2360.910214	2400.810912
	std	101.1864141	69.24262517	63.86584283	95.63303205	61.08272572
F21	avg	2365.202234	2363.637659	2368.772509	2372.459103	2372.472829
	std	17.07739186	12.46630887	19.39137064	15.94973207	17.33533622

3.3.3 CEC 2022 Benchmark Test Experiment Results and Analysis

The CEC 2022 benchmark test set [61] further expands the diversity and complexity of optimization problems. The problems in the test set are designed with consideration of the complexity of the real world, aiming to improve the algorithm's performance when dealing with complex scenarios in practical applications. They include 1 unimodal function (F1), 4 Basic Functions (F2–F5), 3 Hybrid Functions (F6–F8), and 4 Composition Functions (F9–F12).

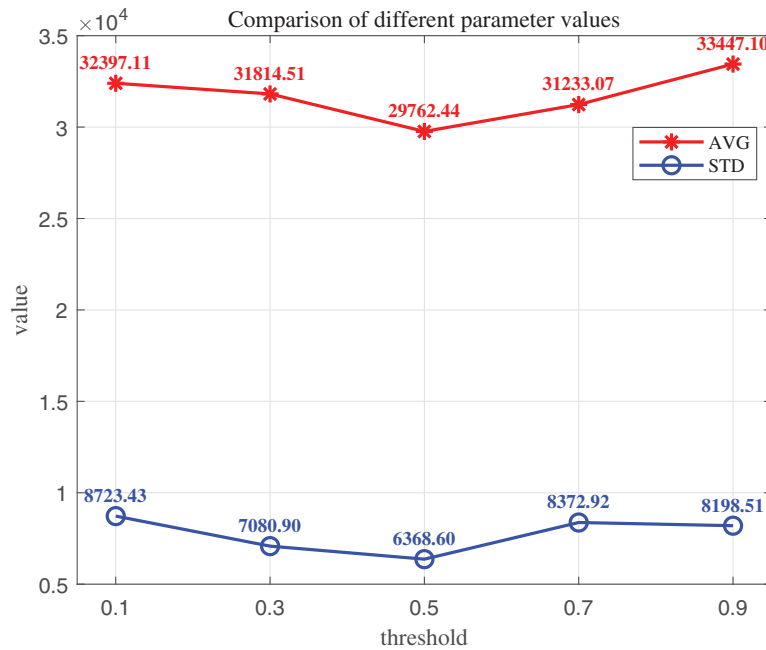


Figure 9: Comparison of different Threshold values

Table 8 shows the results of the FSO algorithm in the CEC 2022 10-dimensional test. FSO shows strong overall competitiveness. It achieves the best mean performance on all test functions except F7 and F10 and exhibits low standard deviation in most cases, indicating that the algorithm possesses good stability and robustness in the search process. Although the standard deviation performance on some test functions such as F4, F7, F8, F10, and F11 is not optimal, in general, the FSO algorithm can balance global exploration and local development capabilities in a relatively balanced manner, effectively avoiding premature convergence problems, and achieve efficient solutions to complex, multi-peak and non-convex problems. In the 20-dimensional test, as shown in Table 9, although the FSO algorithm failed to achieve the best results on a few functions, such as F1, F7, F9, F10, and F11, its performance still maintains an overall advantage and can show relatively stable search performance in most high-dimensional problems.

In Table 10, compared with the emerging published optimization algorithms in the CEC 2022 20-dimensional test results, the FSO algorithm achieves relatively excellent mean and volatility indicators in most test functions, indicating that it possesses high robustness and adaptability in multidimensional, complex optimization problems.

Fig. 10 shows the exploration and exploitation utilization rate, population diversity, and convergence curve of FSO at CEC 2022. Population diversity [62] is a core element for maintaining the search space coverage capability. Its main function is to prevent the algorithm from falling into the local optimal solution too early. Higher population diversity can ensure that the distribution of candidate solutions covers a wider area during the search process, thereby giving the algorithm stronger global search capabilities in the early stages. The population diversity calculation uses the moment of inertia I_c for calculation, as shown in Eqs. (17) and (18):

Table 8: Comparison results of FSO and classic algorithm groups on the 10-dimensional CEC 2022 test function

F	Val	FSO	AOA		COA		RSO		GWO		PSO		SCA		BH		WOA		MFO	
			Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T
F1	avg	300.216	1336.77	+	570.9015	+	2907.913	+	3718.991	+	1242.04	+	2211.424	+	4140.394	+	30767.89	+	13831.34	+
	std	1.19E-01	6617.295		438.8554		1509.878		2537.344		2503.398		968.3292		1424.847		15789.13		14287.42	
F2	avg	402.078	783.9987	+	412.3706	+	835.1609	+	417.7912	+	409.1488	+	455.3747	+	428.1278	+	431.0821	+	417.5527	+
	std	3.38239	322.4148		21.30091		271.1295		17.05345		15.94853		17.04464		23.16351		33.90877		19.65893	
F3	avg	601.366	642.8054	+	614.5584	+	645.038	+	605.2451	+	627.0475	+	622.3057	+	634.9946	+	642.0487	+	622.3154	+
	std	2.20909	7.365199		1.41E+01		5.40E+00		5.930711		15.61357		4.722684		9.37E+00		16.08643		15.44729	
F4	avg	815.479	840.4884	+	830.6192	+	838.9175	+	820.1349	+	827.1254	+	846.094	+	829.8322	+	847.3747	+	845.6515	+
	std	4.668489	8.370722		3.55714		5.958691		7.676464		11.24843		8.278558		14.80569		17.1852		15.0567	
F5	avg	900.024	1411.259	+	1248.433	+	1288.026	+	976.6712	+	1034.651	+	1113.618	+	1188.313	+	1546.941	+	1523.125	+
	std	0.02246	165.9724		250.0968		130.059		131.1973		315.9418		192.9621		206.8121		368.2215		569.31	
F6	avg	2537.57	1.98E+08	+	4600.305	+	526182	+	5956.24	+	6592.488	+	5398839	+	34383.88	+	5789.061	+	4805.797	+
	std	691.947	4.91E+08		2275.914		1896177		2489.361		6946.977		3200515		137183.9		3443.328		1964.153	
F7	avg	2055.625	2110.394	+	2043.217	+	2078.387	+	2042.46	+	2085.791	+	2064.515	+	2067.124	+	2091.148	+	2065.995	+
	std	21.44297	34.02318		42.7023		11.60784		23.44219		44.78223		10.20287		17.54228		36.82847		55.30401	
F8	avg	2226.8	2378.201	+	2248.736	+	2251.221	≈	2243.227	+	2271.617	+	2237.278	+	2232.146	+	2237.089	+	2236.088	≈
	std	9.985032	153.078		45.2449		28.45276		54.09594		56.33149		3.03623		6.697501		11.1786		27.79706	
F9	avg	2531.98	2776.141	+	2550.703	+	2646.88	+	2597.966	+	2531.805	+	2580.93	+	2653.517	+	2616.721	+	2567.951	+
	std	2.72027	59.61073		36.18603		74.47255		40.3854		6.085834		21.94774		9.857565		49.76958		47.86789	
F10	avg	2683.268	2857.61	+	2614.6	+	2510.18	≈	2658.262	+	2704.081	+	2511.038	+	2516.928	+	2578.045	≈	2831.884	≈
	std	303.8637	224.5734		156.0717		4.04548		164.726		256.92		35.53908		40.0303		72.35844		401.8387	
F11	avg	2646.41	3870.248	+	2873.081	+	3040.116	+	2946.163	+	2852.923	+	2784.694	+	2801.166	+	2883.731	+	2884.557	+
	std	110.4082	559.1236		178.6645		214.9003		188.8405		195.6433		15.1016		182.0507		193.8111		203.5819	
F12	avg	2864.14	3098.466	+	2905.548	+	2889.133	+	2878.274	+	2910.469	+	2871.271	+	2952.486	+	2924.143	+	2968.567	+
	std	2864.14	96.06768		38.8334		27.63033		16.73474		58.43526		2.195011		20.69643		54.69413		6.056646	
avg/std win times		8/5	0/0	0/1	0/1	1/1	1/0	0/3	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0

Note: Bold values indicate the best performance for each test function. “+” and “≈” denote the results of the Wilcoxon rank-sum test compared with FSO ($\alpha = 0.05$).

Table 9: Comparison results of FSO and classic algorithm groups on the 20-dimensional CEC 2022 test function

F	Val	FSO	AOA		COA		RSO		GWO		PSO		SCA		BH		WOA		MFO	
			Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T
F1	avg	5348.408	30985.8	+	12258.42	+	23706.42	+	8218.898	+	354.6064	+	10869.27	+	18599.41	+	11047.61	+	36687.86	+
	std	3.45E+03	11700.77		9451.691		4766.228		4220.248		81.61077		1887.199		3790.38		4548.838		19301.04	
F2	avg	446.7348	1858.689	+	453.7492	+	1268.021	+	487.2766	+	455.2012	+	661.5027	+	682.4532	+	535.3001	+	532.6231	+
	std	9.649156	523.746		13.60965		200.7899		38.14918		20.11272		46.10952		21.6862		47.07341		94.61474	
F3	avg	600.6478	656.5763	+	619.2911	+	679.1976	+	603.1186	+	641.553	+	640.7457	+	658.3539	+	663.0254	+	615.158	+
	std	0.581467	9.892534		1.24E+01		6.32E+00		2.275759		9.561884		7.05907		9.26E+00		11.83857		9.132866	
F4	avg	843.4809	930.4825	+	886.8132	+	943.2612	+	843.4805	≈	859.6341	+	941.0953	+	889.7254	+	927.1438	+	897.9137	+
	std	7.917932	16.52464		10.66223		16.60444		19.25549		12.82412		9.798464		18.0878		30.59309		29.21421	
F5	avg	900.0889	2715.959	+	2123.909	+	2875.482	+	1054.067	+	1435.789	+	2188.262	+	2391.947	+	3636.892	+	2511.043	+
	std	0.043793	476.7966		616.2925		278.5292		160.0356		373.0285		420.7837		685.3292		1024.444		832.0519	
F6	avg	5761.799	1.39E+08	+	4534.676	+	2.01E+08	+	682169.6	+	10738.39	+	99389371	+	6924.113	≈	47146.95	+	4486823	+
	std	2080.832	2.25E+08		4033.671		2.07E+08		2704193		5570.854		80305191		3371.41		64763.99		11291683	
F7	avg	2093.179	2206.053	+	2095.01	+	2173.698	+	2060.35	+	2117.438	+	2137.61	+	2128.015	+	2196.523	+	2090.853	≈
	std	15.49796	90.68395		44.01085		20.80411		19.93125		26.60274		21.93212		26.39941		57.07137		36.86744	
F8	avg	2231.199	2408.698	+	2250.154	+	2394.8	+	2251.924	≈	2267.396	+	2260.522	+	2292.296	+	2251.79	+	2246.13	≈
	std	2.823377	140.5924		43.06574		94.919		48.50184		55.47362		12.57195		72.2193		22.20317		32.59748	
F9	avg	2481.355	2959.018	+	2480.785	+	2687.561	+	2501.823	+	2489.413	+	2576.181	+	2679.7	+	2526.445	+	2483.001	+
	std	0.282928	182.9662		0.002946		38.78647		21.81418		13.13186		29.29042		21.81023		32.06567		4.329059	
F10	avg	2987.277	5199.594	+	3400.858	+	3873.012	+	3185.688	≈	3744.853	+	2989.448	≈	2546.82	≈	4530.616	+	3289.675	+
	std	696.9677	954.3014		730.7027		1447.873		638.2598		1121.862		1123.6		79.7182		1039.179		1051.074	
F11	avg	2943.021	7701.394	+	2931.243	+	6870.104	+	3413.069	+	3065.698	+	4615.938	+	4404.813	+	3093.184	+	4166.533	+
	std	92.23197	928.9198		46.3383		804.0252		226.4348		248.3637		398.4567		399.0845		127.1097		622.8111	
F12	avg	2945.652	3702.688	+	2982.096	+	3302.595	+	2966.143	+	3072.689	+	3047.492	+	3424.228	+	3041.829	+	2950.16	+
	std	7.128024	203.4501		58.09038		164.8665		21.18086		74.92711		27.33212		97.63662		55.61203		6.161881	
avg/std win times		6/7	0/0		3/2		0/0		1/0		1/2		0/0		1/1		0/0		0/1	

Note: Bold values indicate the best performance for each test function. “+” and “≈” denote the results of the Wilcoxon rank-sum test compared with FSO ($\alpha = 0.05$).

Table 10: Comparison results of FSO and emerging algorithm groups on the 20-dimensional CEC 2022 test function

F	Val	FSO	GGO		GOOSE		HO		SGA		PKO		PO		SCHO		NRBO		NOA	
			Val		T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	Val	T	
F1	avg	5348.408	38785.660	+	300.0014	≈	7912.420	+	38658.671	+	2842.2819	+	9047.109	+	13658.003	+	9680.1796	+	133183.1572	+
	std	3453.8843	9993.77	0.000542		3252.2449		6468.39		1977.0126		3365.374		5090.066249		2996.3864		133505.3795		
F2	avg	446.7348	1095.736	+	447.4051	+	489.8527	+	2012.0711	+	477.88543	+	495.26889	+	568.934	+	646.93635	+	4573.413835	+
	std	9.649156	234.0063502		16.77261		40.072829		379.913		37.346676		42.644718		77.6998029		95.285246		987.7274478	
F3	avg	600.6478	655.6108	+	673.28882	+	648.27787	+	673.035	+	650.62426	+	648.06276	+	635.8511471	+	652.31	+	698.385	+
	std	0.581467	7.50673161		1.08E+01		9.70E+00		6.731345		12.877295		12.36061		1.18E+01		11.265179		12.9109591	
F4	avg	843.4809	918.9417	+	928.67958	+	871.54148	+	999.352	+	888.47961	≈	881.11908	+	902.5512024	+	918.8429465	+	1060.347949	+
	std	7.917932	20.54571166		36.559751		9.1961484		14.7371714		22.041679		21.474022		27.99886485		14.245145		19.66636191	
F5	avg	900.0889	2546.570702	+	3846.0462	+	2068.0267	+	5630.616	+	2354.1208	+	2185.3158	+	3138.326492	+	2065.3535	+	9344.214593	+
	std	0.043793	1127.888865		1094.3714		228.7349		594.644		431.29138		330.87233		1296.603934		350.81186		1405.211732	
F6	avg	5761.799	3.423E+08	+	6794.9761	≈	10877.144	+	8.457E+08	+	7447.9462	+	46175.648	+	4828548.967	+	4158548.2	+	3.535E+09	+
	std	2080.832	3.718E+08		6337.1432		8111.9426		3.268E+08		5451.9584		123192.44		13992798.35		4387961.9		9.821E+08	
F7	avg	2093.179	2148.638662	+	2276.3038	+	2128.7255	+	2178.31152	+	2171.4696	+	2125.5244	+	2158.446536	+	2179.0655	+	2314.563712	+
	std	15.49796	52.23471026		107.22999		15.636241		19.59423026		65.979952		23.402513		69.8952551		41.60358		68.32668133	
F8	avg	2231.199	2382.097	+	2576.993	+	2238.237	+	2357.592	+	2310.071	+	2263.576	+	2302.612	+	2286.060	+	2860.272	+
	std	2.823377	138.282		198.579		7.788		63.915		74.522		48.133		73.447		59.877		240.091	
F9	avg	2481.355	2939.612	+	2480.85	≈	2524.615	+	2805.930	+	2506.444	+	2537.935	+	2566.545	+	2597.100	+	3240.717	+
	std	0.283	174.284	0.046648		39.437		63.456		42.355		41.735		69.354		57.734		161.223		
F10	avg	2987.277	4895.519	+	5093.466	+	3517.532	+	2705.846	+	3810.749	≈	2535.155	+	3899.054	≈	4212.339	+	6459.163	+
	std	696.968	1170.020		673.011		906.953		89.722		1355.612		83.592		895.707		1551.367		1369.504	
F11	avg	2943.021	6333.726	+	2963.151	+	2997.489	≈	6894.231	+	3100.260	+	3084.641	+	4419.408	+	4355.314	+	10423.289	+
	std	92.232	837.239		105.710		187.495		890.623		161.826		173.389		878.696		535.134		1058.500	
F12	avg	2945.652	3634.100	+	3654.682	+	3076.270	+	3368.801	+	3016.252	+	3011.821	+	3122.645	+	3019.897	+	3784.815	+
	std	7.128	122.770		254.742		68.710		56.499		50.962		29.190		82.251		62.832		162.303	
avg/std win times		9/9	0/0	2/2		0/0	0/0	0/0		0/0	0/0	1/1		0/0		0/0		0/0	0/0	

Note: Bold values indicate the best performance for each test function. “+” and “≈” denote the results of the Wilcoxon rank-sum test compared with FSO ($\alpha = 0.05$).

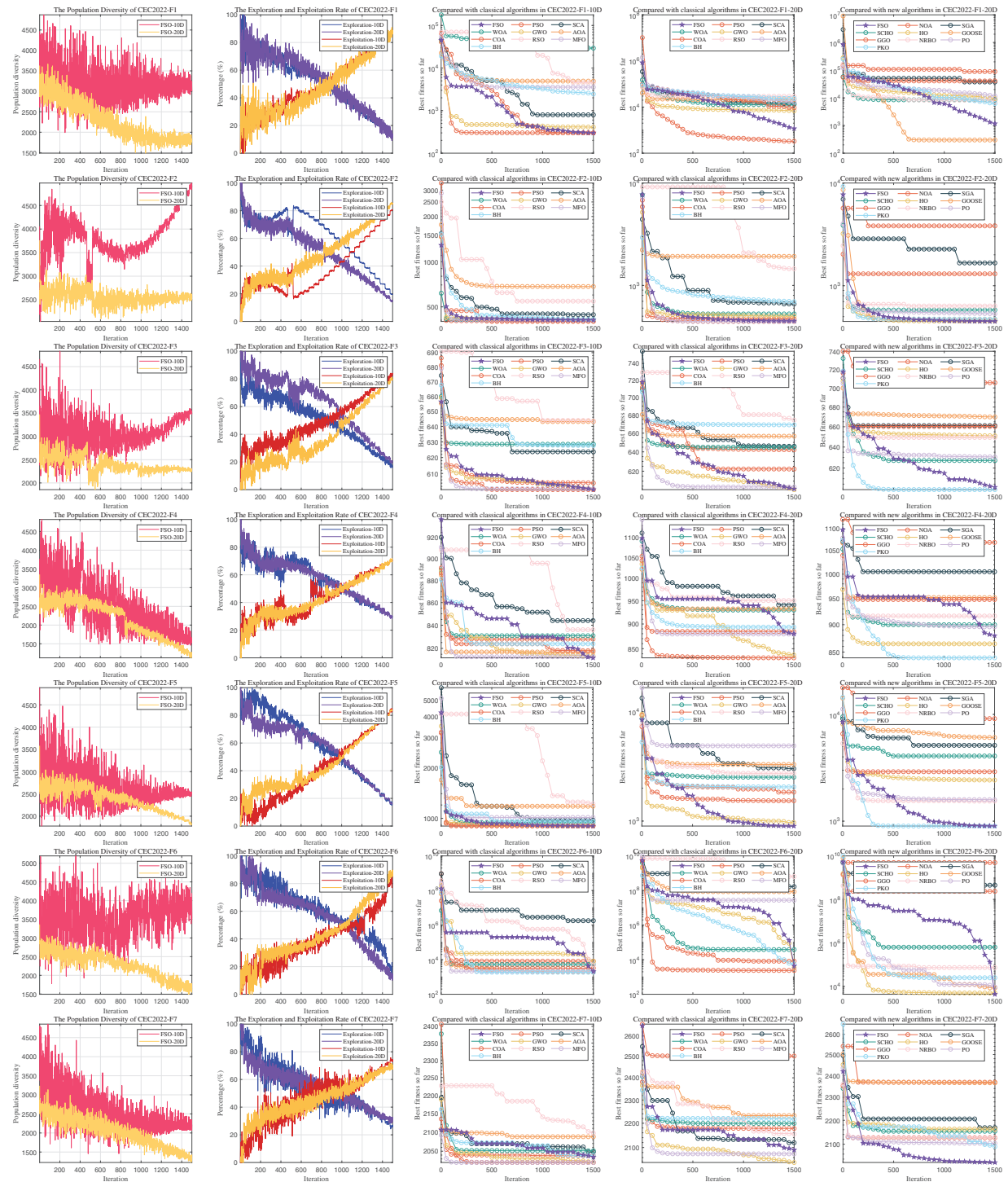


Figure 10: (Continued)

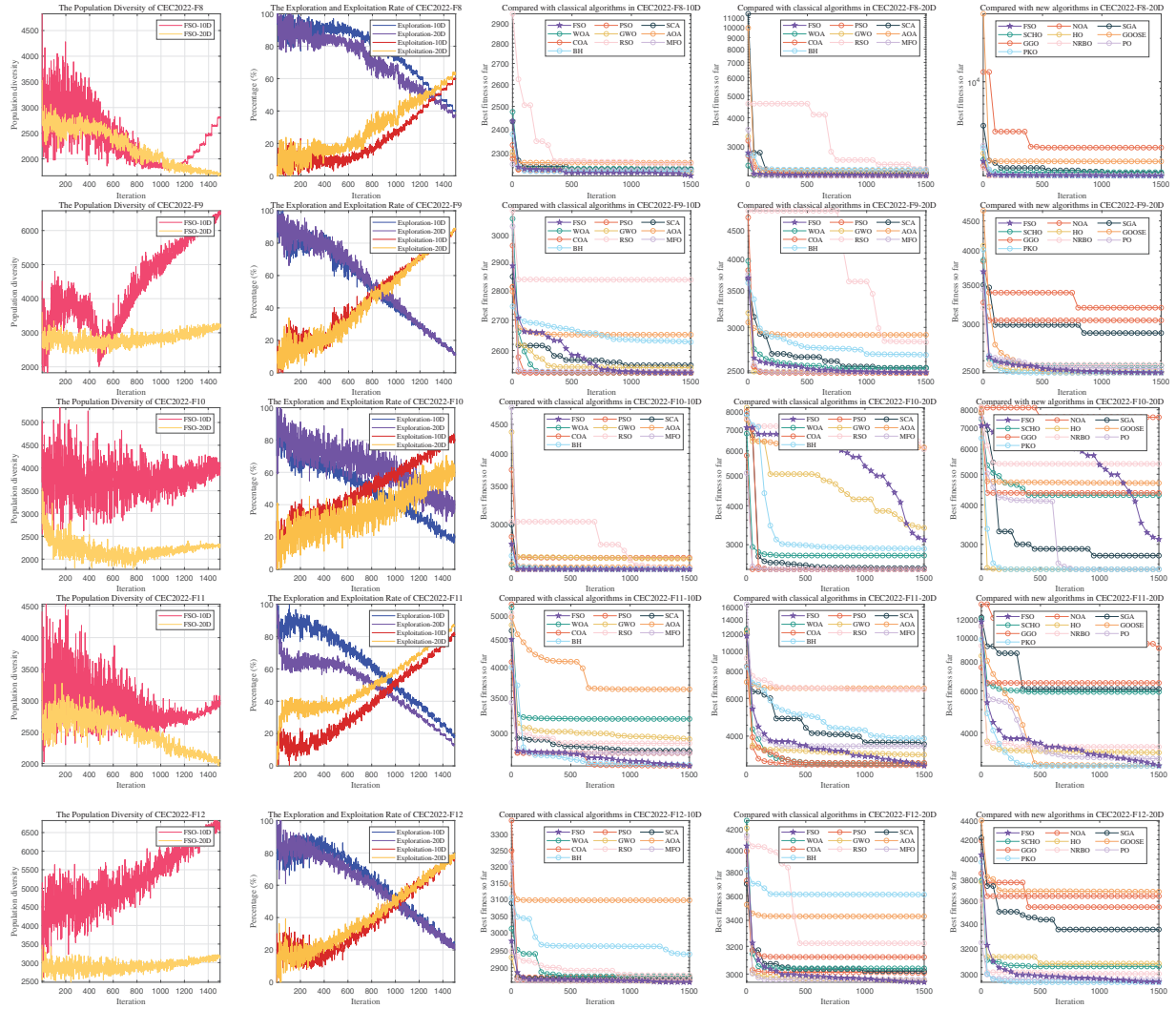


Figure 10: FSO population diversity curve, exploration-exploitation utilization rate curve and convergence curve at CEC 2022

$$I_c(t) = \sqrt{\sum_{i=1}^{popsize} \sum_{d=1}^D (x_{id}(t) - c_d(t))^2} \quad (26)$$

$$c_d(t) = \frac{1}{D} \sum_{i=1}^{popsize} x_{id}(t) \quad (27)$$

where $x_{id}(t)$ is the value of the i -th decision variable in the t -th generation, and $c_d(t)$ is the center coordinate of the population in the d -th dimension of the t -th generation. The exploration-development rate [63] is a key indicator that describes the algorithm's resource allocation between global search (exploration) and local search (development). A high exploration rate means that the algorithm tends to use larger-scale or higher-intensity perturbation operators in each generation, making the search

path more random and globally comprehensive; a high development rate means that the algorithm focuses more on utilizing the local information around the current high-quality candidate solution. The exploration and development rate calculations are shown in Eqs. (19) to (21).

$$Exploration(\%) = \frac{Div(t)}{Div_{max}} \times 100\% \quad (28)$$

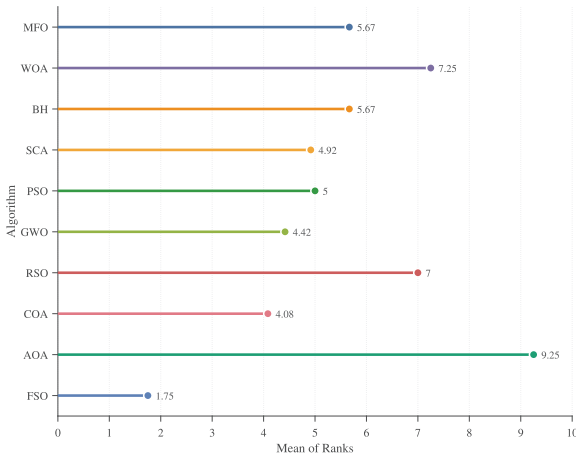
$$Exploitation(\%) = \frac{|Div(t) - Div_{max}|}{Div_{max}} \times 100\% \quad (29)$$

$$Div = \frac{1}{D} \sum_{d=1}^D \frac{1}{N} \sum_{i=1}^{Pop^{size}} |\text{median}(pop_d(t) - pop_{id}(t))| \quad (30)$$

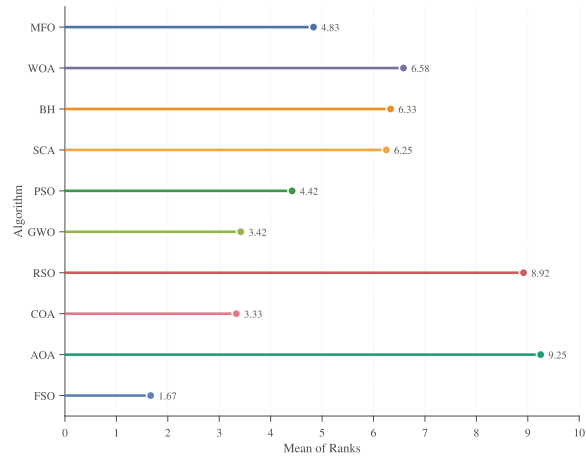
As shown in Fig. 10, the FSO has multiple advantages for the unimodal problem. The population diversity curve shows that the algorithm can maintain a fairly high solution space coverage in the early stage in both 10-dimensional and 20-dimensional scenarios. Then the diversity slowly decays monotonically rather than converges abruptly, indicating that FSO avoids premature convergence in the global exploration stage and gradually concentrates on the optimal solution space in the middle and late stages. Compared with 10-dimensional, the decay rate of 20-dimensional diversity is slower, which means that the algorithm can adapt to the longer exploration cycle required for high-dimensional search. The exploration-exploitation rate further quantifies this adjustment process. The initial exploration rate is close to 90% and the exploitation rate is close to 10%, which is consistent with the exploration-exploitation clustering strategy designed in this paper. As the iteration proceeds, the two lines cross linearly. At 900 iterations, the exploitation rate exceeds 60% in both 10 and 20 dimensions, indicating that FSO employs a progressive strategy of first conducting an extensive search and then refining the profile. At the same time, in the high-dimensional case, the exploration-dominated stage is extended to ensure sufficient information collection. In the comparison of convergence trajectories in the 10-dimensional case, FSO compressed the target value by two orders of magnitude in less than 50 iterations and approached the numerical limit after 300 iterations, while algorithms such as PSO and GWO remained in the $10^3 \sim 10^4$ range until 1500 iterations, and WOA and BH showed a long-term platform phenomenon, highlighting that classical algorithms are prone to fall into local optimality or limited convergence rate; after the dimension increased to 20, the performance of most classical methods further deteriorated, and most curves showed premature stagnation near 10^4 , while FSO, although its decline rate slowed down slightly in the early high-dimensional period, still maintained monotonic decay and entered the second position after 1400 iterations, and continued to retain room for decline; in the same 20-dimensional setting, the convergence comparison of nine new algorithms proposed in the past year showed that except for PO, which briefly approached FSO within 600 iterations, the other methods fell into local optimality in the range of $10^4 \sim 10^5$, while FSO, after 1000 iterations, continued to slowly decrease until it fell below the 10^3 level, eventually maintaining a significant advantage over the algorithm.

Fig. 11, in the comparison between our proposed FSO and nine classic algorithms, including AOA, COA, RSO, GWO, PSO, SCA, BH, WOA, and MFO, FSO won the first place with a significant advantage in both the 10-dimensional and 20-dimensional experiments, and its Friedman mean ranking was 1.75 and 1.67, respectively. Compared with the second place, FSO achieved a relative advantage of about 57.1% and 49.9%, respectively. Fig. 12 shows the Friedman mean ranking results of FSO and the emerging nine new algorithms. FSO also ranked far ahead of other algorithms in the CEC 2022 20D experiment. Compared with HO and PO, which followed closely, FSO has an improvement

of 59.5%, further demonstrating that FSO not only has an advantage in the classic comparison but also maintains a leading position in technology ahead of the emerging algorithms.



(a) FSO and classic algorithms in CEC 2022 10D
Friedman mean of ranks



(b) FSO and classic algorithms in CEC 2022 20D
Friedman mean of ranks

Figure 11: Friedman mean of ranks in CEC 2022 compared with classic algorithms. 10D Friedman mean of ranks (a), 20D Friedman mean of ranks (b)

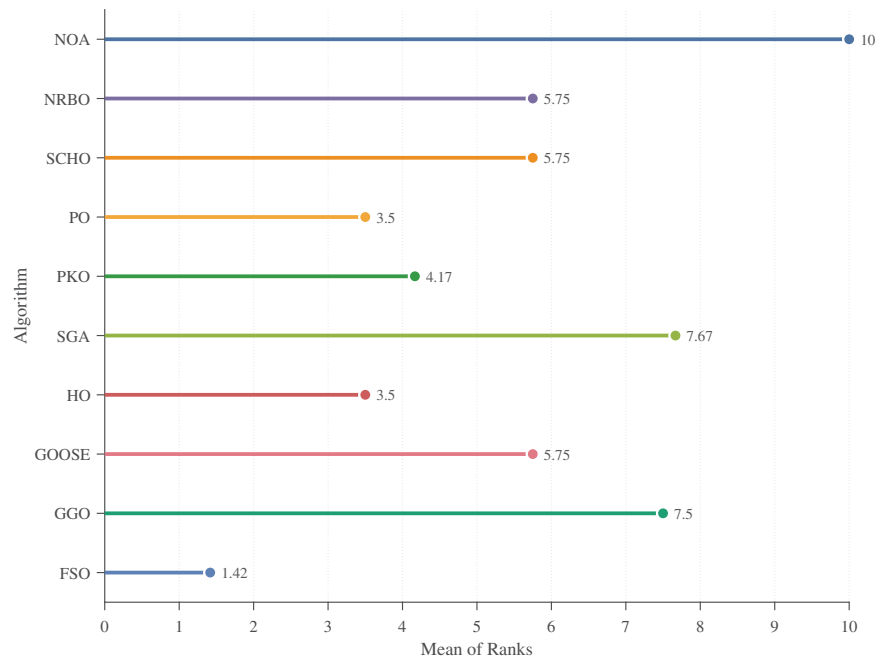


Figure 12: Friedman mean of ranks in CEC 2022 compared with emerging algorithms

4 Application of FSO in Engineering Optimization Problems

In this section, we compare FSO with seven algorithms, namely SCA, PSO, AOA, BH, SO, DOA [64], and RSO, among which SO and DOA are SOTA algorithms proposed in recent years. We conduct comparative tests on six actual engineering applications to demonstrate the optimization ability of FSO in actual engineering problems. The six engineering constraint optimization problems we use are as follows:

- Tension/compression spring design [65]
- Welded beam design [66]
- Pressure vessel design [67]
- Weight minimization of a speed reducer [68]
- Gear train design problem [69]
- Three-bar truss design problem [70]

The platform used in this experiment is equipped with an Intel(R) Core(TM) i9-14900HX processor with a central frequency of 2.2 GHz, 32 GB of memory, a 64-bit Windows 11 system, and the MATLAB 2023b experimental platform. In the experiment, the maximum population size is 100, the maximum number of iterations is 1500.

4.1 Tension/Compression Spring Design

The design problem of tension/compression springs is usually regarded as a classic nonlinear constrained engineering optimization problem. This problem has significant multimodal and non-convex characteristics due to the strong nonlinear coupling between the objective and the constraints. Its core task is to minimize the deadweight of the spring structure while satisfying several mechanical and geometric constraints [71], as shown in Fig. 13. Let the vector $\mathbf{x}_i = [d, D, N]^T$, $i = 1, 2, 3$, where d represents the wire diameter, D represents the average coil diameter, and N represents the number of effective coils. In addition, to ensure the safety and functional performance of the spring during service, the upper limit of shear stress $c_1(x)$, the lower limit of burst (impact) frequency $c_2(x)$, and the minimum deflection $c_3(x)$ constraint functions are added.

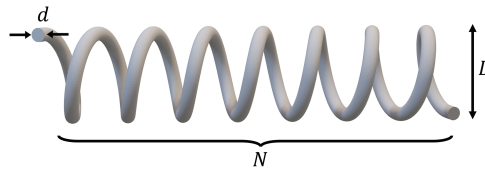


Figure 13: Tension/compression spring design problem

Minimize:

$$\min_x (x_3 + 2) x_2 x_1^2$$

Subject to:

$$c_1(x) = 1 - \frac{x_3^2 x_1}{71785 x_1^4} \leq 0, \quad c_2(x) = \frac{4x_2^2 - x_1 x_2}{12566 (x_2 x_1^3 - x_1^4)} + \frac{1}{5108 x_1^2} \leq 0,$$

$$c_3(x) = 1 - \frac{140.451 x_1^2}{x_2 x_3} \leq 0, \quad c_4(x) = \frac{x_1 + x_2}{1.5} - 1 \leq 0.$$

Variable range:

$$0.05 \leq x_1 \leq 2.00, \quad 0.25 < x_2 < 1.30, \quad 2.00 \leq x_3 \leq 15.00.$$

In numerical experiments on a tension/compression spring design problem, the FSO algorithm demonstrated excellent convergence and stability. As shown in Table 11, the optimal objective value obtained by FSO across multiple independent runs was 1.2694×10^{-2} . FSO reduced the optimal values of the SCA, RSO, AOA, BH, DOA, SO, and PSO by approximately 0.46%, 2.65%, 3.81%, 4.24%, 9.05%, 28.58%, and 58.45%, respectively. The FSO solution was $d = 0.054425$, $D = 0.426119$, and $N = 8.17945$. This means that while satisfying the constraints, a small wire diameter and a medium coil diameter are selected, and the number of effective turns is kept within a moderate range. This is consistent with the structural relationship that spring weight increases with increasing N .

Table 11: Optimal design variables and best objective value for the tension/compression spring problem

Algorithms	Tension/compression spring design			Best value
	$x_1(d)$	$x_2(D)$	$x_3(N)$	
FSO	0.054425	0.426119	8.17945	1.26941E-02
SCA	0.051400	0.348398	11.96288	1.27531E-02
PSO	0.081640	0.907702	7.68668	3.05490E-02
AOA	0.050000	0.312115	15.00000	1.31973E-02
BH	0.060184	0.595906	4.477879	1.32561E-02
SO	0.069064	0.936499	1.988497	1.77732E-02
DOA	0.153698	1.300000	15.00000	1.39573E-02
RSO	0.050000	0.392126	6.583604	1.30400E-02

Note: Bold values indicate the best performance for each test function.

4.2 Welded Beam Design Problem

The welded beam design problem is a classic problem in the field of structural optimization. Its goal is to minimize the overall manufacturing cost of the welded beam by optimizing the design variables under the premise of satisfying a series of constraints [72], as shown in Fig. 14. The design variables of this problem include the weld thickness h , the steel bar length l , the beam height t , and the beam thickness b . The main constraints include the bending stress θ , the shear stress τ , the beam end deflection δ , the steel bar buckling load P_c , etc., which together determine the mechanical properties and manufacturing cost of the beam.

Minimize:

$$\min_x 1.10471x_1^2 x_2 + 0.04811x_3 x_4 (14.0 + x_2)$$

Subject to:

$$c_1(x) = \tau(x) - \tau_{\max} \leq 0, \quad c_2(x) = \sigma(x) - \sigma_{\max} \leq 0, \quad c_3(x) = \delta(x) - \delta_{\max} \leq 0,$$

$$c_4(x) = x_1 - x_4 \leq 0, \quad c_5(x) = P - P_c(x) \leq 0, \quad c_6(x) = 0.125 - x_1 \leq 0,$$

$$c_7(x) = 1.10471x_1^2 + 0.04811x_3 x_4 (14.0 + x_2) - 5.0 \leq 0.$$

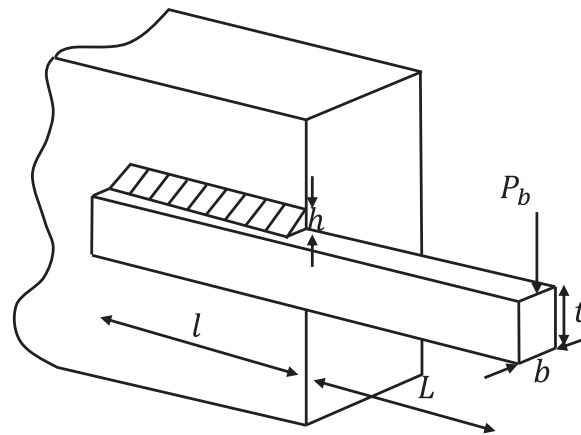


Figure 14: Welded beam design problem

Variable range:

$$0.1 \leq x_1 \leq 2.0, \quad 0.1 \leq x_2 \leq 10.0, \quad 0.1 \leq x_3 \leq 10.0, \quad 0.1 \leq x_4 \leq 2.0.$$

In Table 12, FSO achieved an optimal objective value of 1.72784 across multiple independent runs, significantly outperforming competing algorithms such as SCA (1.79602), PSO (3.07717), and AOA (1.86225). SO's result of 1.089×10^{24} was attributed to a significant penalty term triggered by severe constraint violations. The solution of FSO, with $h = 0.205166$, $l = 3.493986$, $t = 9.033529$, and $b = 0.205890$, aligns with the known structural characteristics of the problem, requiring symmetry between thin welds and thin flanges while satisfying shear/bending strength constraints. This approach reduces overall cost while meeting deflection and strength constraints.

Table 12: Optimal design variables and best objective value for the welded beam design problem

Algorithms	Welded beam design variables				Best value
	$x_1(h)$	$x_2(l)$	$x_3(t)$	$x_4(b)$	
FSO	0.205166	3.493986	9.033529	0.205890	1.72784E+00
SCA	0.201029	3.823178	8.908540	0.212772	1.79602E+00
PSO	0.276355	3.295999	7.960959	0.422542	3.07717E+00
AOA	0.202010	3.464458	9.995854	0.203136	1.86225E+00
BH	0.224855	4.274271	8.518026	0.231543	1.97272E+00
SO	1.157178	1.871254	2.000000	2.000000	1.08900E+24
DOA	0.209560	3.365336	9.107217	0.219862	1.83610E+00
RSO	0.211171	3.745182	8.512591	0.244318	1.96005E+00

Note: Bold values indicate the best performance for each test function.

4.3 Pressure Vessel Design Problem

The pressure vessel design problem is a vital engineering optimization task. Its main goal is to minimize the total cost of manufacturing cylindrical containers through reasonable design. The

total cost includes material cost, molding cost and welding cost [73], as shown in Fig. 15. Let $\mathbf{x}_i = [T_s, T_h, L, R]^T$, $i = 1, 2, 3, 4$, where T_s is the shell thickness, T_h is the head thickness, R is the inner radius, and L is the length of the cylindrical section ignoring the head. To fully consider the structural performance and manufacturing cost of the container, it is also necessary to optimize and adjust the shell thickness constraint $c_1(x)$, the head thickness constraint $c_2(x)$, the minimum volume constraint $c_3(x)$ and the maximum length constraint $c_4(x)$ during the design process.

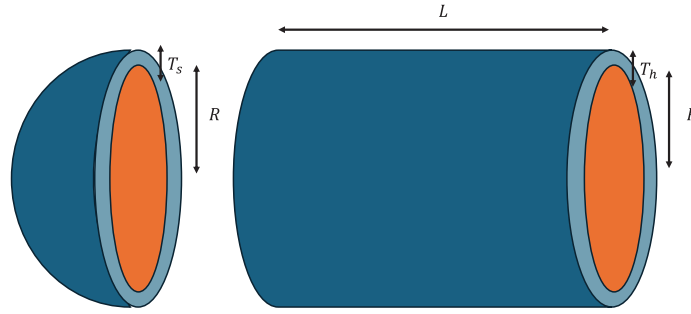


Figure 15: Pressure vessel design problem

Minimize:

$$\min_x 0.6224x_1 x_3 x_4 + 1.7781x_2 x_3^2 + 3.166x_1^2 x_4 + 19.84x_1^2 x_3$$

Subject to:

$$c_1(x) = -x_1 + 0.0193x_3 \leq 0, \quad c_2(x) = -x_2 + 0.00954x_3 \leq 0,$$

$$c_3(x) = -\pi x_3^2 x_4 - \frac{4}{3} x_3^3 + 1\,296\,000 \leq 0, \quad c_4(x) = x_4 - 240 \leq 0.$$

Variable range:

$$0 \leq x_1 \leq 99, \quad 0 \leq x_2 \leq 99, \quad 10 \leq x_3 \leq 200, \quad 10 \leq x_4 \leq 200.$$

From Table 13, the FSO for pressure vessel design achieves high-level thickness variables $t = 0.844252$ and $b = 157.8897$, satisfying both volume constraints and structural strength requirements. Furthermore, the cylinder height $h = 0.844252$ and the hemispherical head radius $l = 0.424508$ are both near the lower bound, significantly reducing material consumption and manufacturing costs. The FSO achieves the lowest objective value of 6.04731×10^3 for this problem, still achieving a cost reduction of approximately 0.66% compared to the suboptimal SCA, demonstrating a more refined search near the boundary of the feasible region.

4.4 Weight Minimization of a Speed Reducer Problem

The reducer weight minimization problem is a classic and representative constrained optimization benchmark in the field of mechanical transmission optimization. Its research goal is to coordinate the structure and gear parameters [74], as shown in Fig. 16. Assumption $\mathbf{x}_i = [b, m, z, l_1, l_2, d_1, d_2]^T$, $i = 1, 2, 3, 4, 5, 6, 7$, where b is the gear face width, m is the gear module, z is the number of pinion teeth, l_1 is the length of the shaft section between the first bearing, l_2 is the length of the shaft section between the second bearing, d_1 is the diameter of the first shaft, and d_2 is the diameter of the second shaft. To ensure the reliability of the reducer under actual working conditions, the model also introduces 11

inequality constraints, namely, limiting the bending stress of gear teeth $c_1(x)$, tooth surface contact stress $c_2(x)$, the deflection constraints of the first and second transmission shafts $c_3(x)$ and $c_4(x)$, the shear-bending composite stress constraints of the first and second shafts $c_5(x)$ and $c_6(x)$, the gear center distance geometric constraint $c_7(x)$, the gear face width and module relationship constraint $c_8(x)$, the face width lower limit constraint $c_9(x)$, the axis $\neq 1$ aspect ratio constraint $c_{11}(x)$ and the axis $\neq 2$ aspect ratio constraint $c_{12}(x)$.

Table 13: Optimal design variables and best objective value for the pressure vessel design problem

Algorithms	Pressure vessel design variables				Best value
	$x_1(h)$	$x_2(l)$	$x_3(t)$	$x_4(b)$	
FSO	0.844252	0.424508	43.73671	157.8897	6.04731E+03
SCA	0.786103	0.414491	40.66118	200.0000	6.08720E+03
PSO	2.596780	16.22233	51.37895	140.4864	9.76839E+04
AOA	0.896261	0.411381	42.36283	200.0000	7.22279E+03
BH	1.678686	2.013314	48.50170	118.4676	1.81934E+04
SO	0.963410	0.476214	49.91763	99.0000	6.28333E+03
DOA	1.673597	2.023968	45.93828	135.4026	1.78275E+04
RSO	1.439973	0.627803	65.46044	10.0000	8.12869E+03

Note: Bold values indicate the best performance for each test function.

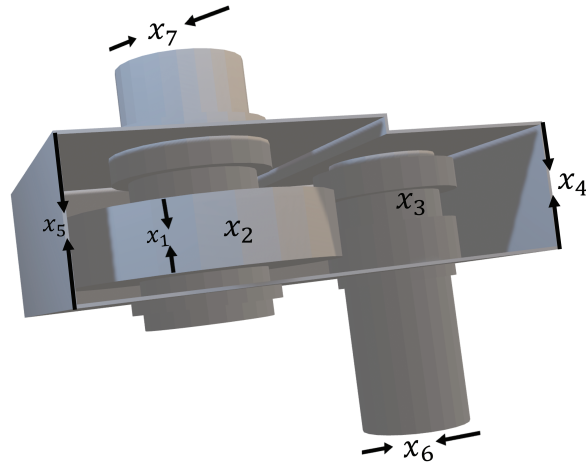


Figure 16: Weight minimization of a speed reducer

Minimize:

$$\min_x 0.7854x_1^2x_2^2(3.3333x_3^2 + 14.9334x_3 - 43.0934) - 1.5084x_1(x_2^6 + x_2^7) \\ + 7.4777(x_3^6 + x_3^7) + 0.7854(x_4x_2^6 + x_5x_2^7).$$

Subject to:

$$\begin{aligned}
c_1(x) &= \frac{27}{x_1^2 x_2 x_3} - 1 \leq 0, & c_2(x) &= \frac{397.5}{x_1^2 x_2^2 x_3} - 1 \leq 0, \\
c_3(x) &= \frac{1.93 x_2^3 x_4}{x_2^6 x_3} - 1 \leq 0, & c_4(x) &= \frac{1.93 x_3^5}{x_2^4 x_7} - 1 \leq 0, \\
c_5(x) &= \frac{\left(\frac{745 x_4}{x_2 x_3}\right)^2 + 16.9 \times 10^6}{100 x_3^6} - 1 \leq 0, & c_6(x) &= \frac{\left(\frac{745 x_5}{x_2 x_3}\right)^2 + 157.5 \times 10^6}{852 x_3^7} - 1 \leq 0, \\
c_7(x) &= \frac{x_2 x_3}{40} - 1 \leq 0, & c_8(x) &= \frac{5 x_2^2}{x_1} - 1 \leq 0, \\
c_9(x) &= \frac{x_1}{12 x_2^2} - 1 \leq 0, & c_{10}(x) &= 1.5 x_6 + 1.9 - \frac{1}{x_4} \leq 0, \\
c_{11}(x) &= 1.1 x_7 + 1.9 - \frac{1}{x_5} \leq 0.
\end{aligned}$$

Variable range:

$$\begin{aligned}
2.6 \leq x_1 \leq 3.6, \quad 0.7 \leq x_2 \leq 0.8, \quad 17 \leq x_3 \leq 28, \quad 7.3 \leq x_4 \leq 8.3, \\
7.3 \leq x_5 \leq 8.3, \quad 2.9 \leq x_6 \leq 3.9, \quad 5.0 \leq x_7 \leq 5.5.
\end{aligned}$$

From the distribution of decision variables in Table 14, FSO tends to fix the modulus m and the number of teeth z at the lower bound combination allowed by the problem to reduce the mass of the meshing pair and, at the same time, by slightly increasing the face width b and the bearing diameter d_2 , reduce material consumption under the premise of meeting the bending strength and contact fatigue constraints. Statistical indicators further verify the robustness of FSO. FSO ranks first among the eight compared algorithms. Compared with the Best result of 2.99547×10^3 of the second-ranked DOA, FSO can still achieve an absolute improvement of 9.0×10^{-2} . Compared with SCA, AOA, and BH, its weight is reduced by 0.83%, 3.19%, and 4.50%, respectively.

Table 14: Optimal design variables and best objective value for the weight minimization of a speed reducer

Algorithms	Weight minimization of a speed reducer							Best value
	$x_1(b)$	$x_2(m)$	$x_3(z)$	$x_4(l_1)$	$x_5(l_2)$	$x_6(d_1)$	$x_7(d_2)$	
FSO	3.50003	0.7	17	7.356022	7.726576	3.350344	5.286855	2.99538E+03
SCA	3.526305	0.7	17	7.402812	7.913655	3.365822	5.296929	3.02062E+03
PSO	3.571946	0.706582	25.23448	7.658492	7.985637	3.346364	5.409497	1.08520E+08
AOA	3.6	0.7	17	7.986437	8.3	3.498114	5.288941	3.09386E+03
BH	3.558615	0.711569	17.21610	7.801354	7.961622	3.381829	5.298845	3.13633E+03
SO	3.6	2.6	3.6	3.6	3.6	3.086665	3.6	1.27061E+16
DOA	3.500432	0.700020	17.00247	7.300029	7.716003	3.350389	5.287161	2.99547E+03
RSO	3.536949	0.708461	25.59718	8.115698	7.637894	3.666141	5.275455	1.10403E+11

Note: Bold values indicate the best performance for each test function.

4.5 Gear Train Design Problem

The gear train design problem is a simplified engineering optimization model with the core goal of minimizing the transmission ratio. Its research goal is to accurately adjust the number of teeth of each gear [75], as shown in Fig. 17. Let's assumption $\mathbf{x}_i = [T_1, T_2, T_3, T_4]^T$, $i = 1, 2, 3, 4$, correspond to the number of teeth of the four gears A , B , D , and F in the gear train.

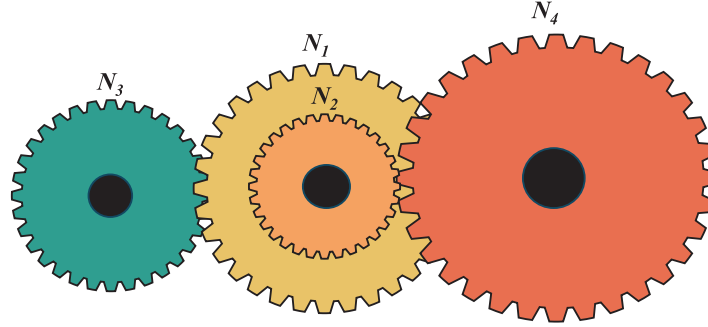


Figure 17: Gear train design

Minimize:

$$\min_x \frac{1}{6.931} + \frac{x_2 x_3}{x_1 x_4}$$

Subject to:

$$12 \leq x_i \leq 60, \quad i = 1, 2, 3, 4.$$

As shown in Table 15, the FSO algorithm achieves an optimal objective function value of 2.60422×10^{-16} for a classic four-stage gear train error minimization benchmark problem. While the SO algorithm achieves a numerically lower objective function value of 1.38233×10^{-26} , it is important to note that when the optimization accuracy reaches the order of $\mathcal{O}(10^{-16})$, it far exceeds the accuracy required for practical engineering applications. The FSO algorithm yields a tooth number vector of approximately $(n_1, n_2, n_3, n_4) = (31.79, 17.56, 12.96, 49.62) \approx (32, 18, 13, 50)$, demonstrating the effectiveness of the FSO optimization.

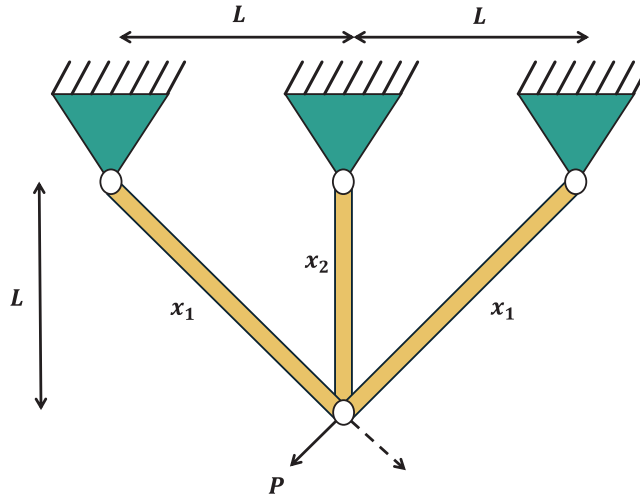
4.6 Three-Bar Truss Design Problem

The core task of the three-bar truss design is to minimize the weight of the truss while satisfying the constraints of structural safety and service performance [76], as shown in Fig. 18. Let the vector $\mathbf{x} = [x_1, x_2]^T$, where x_1 and x_2 represent the dimensionless cross-sectional area ratios of different members, respectively, and the constraints are the axial stress limits $c_1(x)$, $c_2(x)$, and vertical deflection $c_3(x)$ of the two groups of compression/tension members.

Table 15: Optimal design variables and best objective value for the gear train design problem

Algorithms	Gear train design				Best value
	$x_1(n_1)$	$x_2(n_2)$	$x_3(n_3)$	$x_4(n_4)$	
FSO	31.78686	17.55625	12.96305	49.62348	2.60422E-16
SCA	35.72559	16.64936	12.00000	38.76103	4.12202E-15
PSO	50.37303	22.60940	15.31228	48.04618	1.52410E-06
AOA	33.42441	12.88471	22.45663	60.00000	1.45447E-17
BH	33.38853	12.32445	23.00589	58.85797	3.86042E-23
SO	53.56361	12.00000	34.44703	53.48834	1.38233E-26
DOA	53.41485	13.08966	34.82841	59.15561	1.61112E-20
RSO	30.66056	12.00000	12.00000	32.42878	3.00811E-07

Note: Bold values indicate the best performance for each test function.


Figure 18: Three-bar truss design problem

Minimize:

$$\min_x (2\sqrt{2}x_1 + x_2) L$$

Subject to:

$$c_1(x) = \frac{P\sqrt{2x_1 + x_2}}{\sqrt{2x_1^2 + 2x_1x_2}} - \sigma \leq 0, \quad c_2(x) = \frac{Px_2}{\sqrt{2x_1^2 + 2x_1x_2}} - \sigma \leq 0,$$

$$c_3(x) = \frac{P}{x_1 + \sqrt{2}x_2} - \sigma \leq 0.$$

where $L = 100$ cm is the beam length, $P = 2$ kN/cm² is the external load, and $\sigma = 2$ kN/cm² is the allowable stress of the material.

As shown in Table 16, the FSO algorithm yielded the same optimal value as the BH and SO algorithms, which is 2.63896×10^2 . While guaranteeing that member stresses satisfy the restrictions, the FSO solution, $x_1 = 0.788300$ and $x_2 = 0.409311$, strikes an ideal balance between material utilization and structural weight. The PSO, RSO, and AOA algorithms, on the other hand, produce much larger x_2 values, which may result in redundant designs and increase material usage.

Table 16: Optimal design variables and best objective value for the three-bar truss design problem.

Algorithm	Three-bar truss design problem		Best value
	x_1	x_2	
FSO	0.788300	0.409311	2.63896E+02
SCA	0.791629	0.400003	2.63907E+02
PSO	0.760721	0.517676	2.66932E+02
AOA	0.778931	0.436972	2.64012E+02
BH	0.789210	0.406739	2.63896E+02
SO	0.788669	0.408266	2.63896E+02
DOA	0.791372	0.400938	2.63928E+02
RSO	0.731384	0.600909	2.64373E+02

Note: Bold values indicate the best performance for each test function.

Fig. 19 shows the runtime distribution of FSO and seven other algorithms for optimizing engineering problems. Among the eight metaheuristic algorithms evaluated, FSO ranked third with a total runtime of 2.38 s, slightly ahead of SCA's 0.94 s and PSO's 1.62 s, though SCA and PSO have a slight advantage in runtime, their optimization results lag behind FSO, with FSO demonstrating the best balance between solution quality and computational cost. Compared to SOTA algorithms such as SO and DOA, despite FSO's inferiority to SO on the Gear Train Design Problem, the optimization time has halved for FSO. The result demonstrates the FSO algorithm's superiority in search mechanism design and convergence strategy.

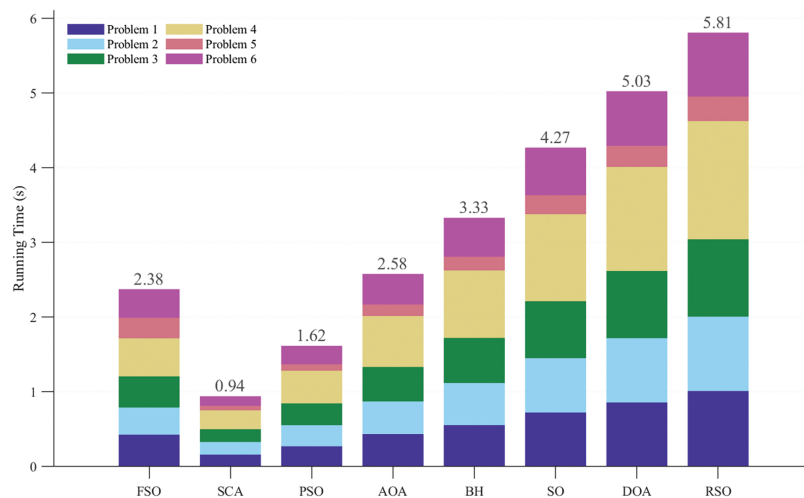


Figure 19: Running time comparison of different optimization algorithms

5 Conclusion

In response to the increasingly complex and highly nonlinear modern optimization scenarios [77], we proposed a new metaheuristic algorithm named Freshwater Snail Optimizer (FSO). Through the diversified solution space constructed by swarm initialization, the global search is expanded by floating in the air chamber during the exploration process, and feasible solutions are scanned over an extensive range in the solution space. During the development process, local development is carried out in a small range by imitating the way water snails move in the water. To prevent the algorithm from falling into the local optimum, we designed a free-step strategy by imitating the collision social behavior of water snails to assist FSO in escaping the local optimum. To systematically verify the optimization ability of FSO, we conducted a performance comparison and mechanism analysis, with the CEC series benchmark evaluation serving as the core. The experimental design encompasses CEC 2017 50 dimensions and CEC 2022 (10, 20 dimensions) tests, providing a comprehensive examination of dynamic indicators such as exploration-exploitation ratio, and convergence curve. The results are ranked using the Friedman test at the statistical level. Comprehensively comparing nine classic metaheuristic algorithms and nine new algorithms proposed in the past year, FSO showed significant advantages in both the mean and variance of the optimal value, proving that FSO has good robustness, consistency and the ability to balance exploration and development. It has repeatedly ranked first in six engineering constraint optimization examples, fully demonstrating its excellent adaptability and generalization potential for practical problems.

Kulkarni et al. [78] proposed the Snail Homing and Mating Search (SHMS) algorithm in their paper, inspired by snails, the FSO algorithm proposed in this paper is significantly different from SHMS in terms of biological abstraction and operator mechanism. FSO simulates the fluid dynamics of freshwater snorkels that rely on lung sacs to adjust for floating and sinking. It performs global exploration by superimposing exponential decay buoyancy and sine-cosine drift, and then completes local exploitation and maintains population discreteness through adhesion, sliding, and collision with the shell. SHMS is derived from the mucus nest-seeking and mate-selection behavior of terrestrial snails, dividing individuals into 'snails' and 'nests', utilizing fixed-probability switching trajectories and Lévy flight, and requiring adjustments to five constants, such as mucus attenuation and nest density. In addition, the buoyancy adaptive mechanism of FSO enables fast early convergence and avoids premature maturity in the later stage, whereas the convergence curve of SHMS is asymptotic Sigmoid, and its ability to jump out of the local area is limited. Therefore, there are significant differences in the metaheuristic algorithm design.

In the future, we will focus on three main lines: first, introduce surrogate-assisted technology and deep transfer learning strategy to improve further the real-time response capability on ultra-high-dimensional and dynamic multi-peak objective functions; second, expand FSO to multi-objective and multi-task optimization framework to further enhance its optimization capability for multiple problems and multiple objectives; third, combine binary mechanism to meet the algorithm's needs in feature selection on sparse data, so as to solve real-world problems better.

Acknowledgement: Not applicable.

Funding Statement: This research was funded by the Educational and Teaching Reform Research for Higher Education in Hainan Province (Grant No. Hnjg2024ZC-11) and supported by the Environmental Big Data and Digital Governance Key Laboratory of Philosophy and Social Sciences in Hainan Province.

Author Contributions: The authors confirm contribution to the paper as follows: Wrote the original draft, Gao-Yuan Liu; conceived the study, Jeng-Shyang Pan and Shu-Chuan Chu; conducted the experiments, Ru-Yu Wang and Ying Yu; funding acquisition, Bo-Xiong Yang and Ying Yu; analyzed the results and software, Bo-Xiong Yang and Seyedali Mirjalili. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: Not applicable.

Ethics Approval: Not applicable.

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

References

1. Adeyeye O, Akanbi I. Optimization in systems engineering: a review of how data analytics and optimization algorithms are applied. *Comput Sci IT Res J*. 2024;5(4):809–23. doi:10.51594/csitrj.v5i4.1027.
2. Haji SH, Abdulazeez AM. Comparison of optimization techniques based on gradient descent algorithm: a review. *PalArch's J Archaeol Egypt Egyptol*. 2021;18(4):2715–43.
3. Battiti R. First-and second-order methods for learning: between steepest descent and Newton's method. *Neural Comput*. 1992;4(2):141–66. doi:10.1162/neco.1992.4.2.141.
4. Turgut OE, Turgut MS, Kırtepe E. A systematic review of the emerging metaheuristic algorithms on solving complex optimization problems. *Neural Comput Appl*. 2023;35(19):14275–378. doi:10.1007/s00521-023-08481-5.
5. Corriou JP. Numerical methods of optimization. Cham, Switzerland: Springer; 2021.
6. Mei Y, Lan T, Imani M, Subramaniam S. A Bayesian optimization framework for finding local optima in expensive multimodal functions. In: *ECAI 2023*. Amsterdam, Netherlands: IOS Press; 2023. p. 1704–11.
7. Chawla M, Duhan M. Levy flights in metaheuristics optimization algorithms—a review. *Appl Artif Intell*. 2018;32(9–10):802–21. doi:10.1080/08839514.2018.1508807.
8. Beni G, Wang J. Swarm intelligence in cellular robotic systems. In: *Robots and biological systems: towards a new bionics?* Berlin, Germany: Springer; 1993. p. 703–12. doi:10.1007/978-3-642-58069-7_38.
9. Bandaru S, Deb K. Metaheuristic techniques. In: *Decision sciences*. Boca Raton, FL, USA: CRC Press; 2016. p. 709–66.
10. Kareem SW, Ali KWH, Askar S, Xoshaba FS, Hawezi R. Metaheuristic algorithms in optimization and its application: a review. *JAREEJAREE (J Adv Res Electr Eng)*. 2022;6(1):7–12. doi:10.12962/jaree.v6i1.216.
11. Yang XS, Ting TO, Karamanoglu M. Random walks, Lévy flights, Markov chains and metaheuristic optimization. In: *Future information communication technology and applications: ICFICE 2013*. Cham, Switzerland: Springer; 2013. p. 1055–64. doi:10.1007/978-94-007-6516-0_116.
12. Gutjahr WJ. Convergence analysis of metaheuristics. In: *Matheuristics: hybridizing metaheuristics and mathematical programming*. Berlin, Germany: Springer; 2009. p. 159–87. doi:10.1007/978-1-4419-1306-7_6.
13. Yang X, Wang R, Li K, Ishibuchi H. Meta-Black-Box optimization for evolutionary algorithms: review and perspective. *Swarm Evol Comput*. 2025;93(4):101838. doi:10.1016/j.swevo.2024.101838.
14. Holland JH. Genetic algorithms. *Sci Am*. 1992;267(1):66–73.
15. Storn R, Price K. Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. *J Glob Optim*. 1997;11(4):341–59. doi:10.1023/A:1008202821328.
16. Kennedy J, Eberhart R. Particle swarm optimization. In: *Proceedings of ICNN'95-International Conference on Neural Networks*; 1995; Perth, WA, Australia. Piscataway, NJ, USA: IEEE; 1995, Vol. 4, p. 1942–8. doi:10.1109/ICNN.1995.488968.

17. Blum C. Ant colony optimization: introduction and recent trends. *Phys Life Rev.* 2005;2(4):353–73. doi:10.1016/j.plrev.2005.10.001.
18. Reshamwala A, Vinchurkar DP. Robot path planning using an ant colony optimization approach: a survey. *Int J Adv Res Artif Intell.* 2013;2(3):65–71. doi:10.14569/ijarai.2013.020310.
19. Yang XS, Slowik A. Firefly algorithm. In: *Swarm intelligence algorithms*. Boca Raton, FL, USA: CRC Press; 2020. p. 163–74.
20. Chu SC, Tsai PW, Pan JS. Cat swarm optimization. In: *Pacific Rim International Conference on Artificial Intelligence*; 2006; Berlin, Germany: Springer; 2006. p. 854–8. doi:10.1007/978-3-540-36668-3_94.
21. Atashpaz-Gargari E, Lucas C. Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In: *2007 IEEE Congress on Evolutionary Computation*; 2007 Sep 25–28; Singapore. Piscataway, NJ, USA: IEEE; 2007. p. 4661–7. doi:10.1109/CEC.2007.4425083.
22. Shi Y. Brain storm optimization algorithm. In: *Advances in Swarm Intelligence: Second International Conference, ICSI 2011*; 2011 Jun 12–15; Chongqing, China. Berlin, Germany: Springer; 2011. p. 303–9. doi:10.1007/978-3-642-21515-5_36.
23. Rao RV, Savsani VJ, Vakharia DP. Teaching-learning-based optimization: a novel method for constrained mechanical design optimization problems. *Comput Aided Des.* 2011;43(3):303–15. doi:10.1016/j.cad.2010.12.015.
24. Mirjalili S, Mirjalili SM, Lewis A. Grey wolf optimizer. *Adv Eng Softw.* 2014;69:46–61. doi:10.1016/j.advengsoft.2013.12.007.
25. Mirjalili S, Lewis A. The whale optimization algorithm. *Adv Eng Softw.* 2016;95(12):51–67. doi:10.1016/j.advengsoft.2016.01.008.
26. Hashim FA, Hussien AG. Snake optimizer: a novel meta-heuristic optimization algorithm. *Knowl Based Syst.* 2022;242(10):108320. doi:10.1016/j.knosys.2022.108320.
27. Ghasemi M, Khodadadi N, Trojovský P, Li L, Mansor Z, Abualigah L, et al. Kirchhoff's law algorithm (KLA): a novel physics-inspired non-parametric metaheuristic algorithm for optimization problems. *Artif Intell Rev.* 2025;58(10):325. doi:10.1007/s10462-025-11289-5.
28. Zaerreza A, Gholizadeh S, Mohammadi M. An improved newton metaheuristic algorithm for design optimization of steel moment-resisting frames. *Adv Eng Softw.* 2025;207(8):103960. doi:10.1016/j.advengsoft.2025.103960.
29. Oralhan Z, Oralhan B. Quantum snowflake algorithm (QSA): a snowflake-inspired, quantum-driven metaheuristic for large-scale continuous and discrete optimization with application to the traveling salesman problem. *Appl Sci.* 2025;15(9):5117. doi:10.3390/app15095117.
30. Morales-Castañeda B, Zaldivar D, Cuevas E, Fausto F, Rodríguez A. A better balance in metaheuristic algorithms: does it exist? *Swarm Evol Comput.* 2020;54(1):100671. doi:10.1016/j.swevo.2020.100671.
31. Adam SP, Alexandropoulos SAN, Pardalos PM, Vrahatis MN. No free lunch theorem: a review. In: *Approximation and optimization: algorithms, complexity and applications*. Cham, Switzerland: Springer; 2019. p. 57–82. doi:10.1007/978-3-030-12767-1_5.
32. Monga P, Sharma M, Sharma SK. A comprehensive meta-analysis of emerging swarm intelligent computing techniques and their research trend. *J King Saud Univ Comput Inf Sci.* 2022;34(10):9622–43. doi:10.1016/j.jksuci.2021.11.016.
33. Velasco L, Guerrero H, Hospitaler A. A literature review and critical analysis of metaheuristics recently developed. *Arch Comput Methods Eng.* 2024;31(1):125–46. doi:10.1007/s11831-023-09990-1.
34. Cheatum EP. Limnological investigations on respiration, annual migratory cycle, and other related phenomena in fresh-water pulmonate snails. *Trans Am Microsc Soc.* 1934;53(4):348–407.
35. Kappes H, Haase P. Slow, but steady: dispersal of freshwater molluscs. *Aquat Sci.* 2012;74(1):1–14. doi:10.1007/s00027-011-0187-6.

36. Deliagina TG, Orlovsky GN. Control of locomotion in the freshwater snail *Planorbis corneus* I. Locomotory repertoire of the snail. *J Exp Biol.* 1990;152(1):389–404. doi:10.1242/jeb.152.1.389.
37. Newar J, Ghatak A. Studies on the adhesive property of snail adhesive mucus. *Langmuir.* 2015;31(44):12155–60. doi:10.1021/acs.langmuir.5b03498.
38. Bush JWM, Hu DL. Walking on water: biolocomotion at the interface. *Annu Rev Fluid Mech.* 2006;38(1):339–69. doi:10.1146/annurev.fluid.38.050304.092157.
39. Dalesman S, Lukowiak K. Social snails: the effect of social isolation on cognition is dependent on environmental context. *J Exp Biol.* 2011;214(24):4179–85. doi:10.1242/jeb.064857.
40. Chen K, Zhou F, Yin L, Wang S, Wang Y, Wan F. A hybrid particle swarm optimizer with sine cosine acceleration coefficients. *Inf Sci.* 2018;422(1):218–41. doi:10.1016/j.ins.2017.09.015.
41. Chegini SN, Bagheri A, Najafi F. PSOSCALF: a new hybrid PSO based on sine cosine algorithm and levy flight for solving optimization problems. *Appl Soft Comput.* 2018;73(3):697–726. doi:10.1016/j.asoc.2018.09.019.
42. Simpson AW, Thomas JD, Townsend CR. Social behavior in the freshwater pulmonate snail *Biomphalaria glabrata* (Say). *Behav Biol.* 1973;9(6):731–40. doi:10.1016/S0091-6773(73)80133-6.
43. Oliveto PS, He J, Yao X. Time complexity of evolutionary algorithms for combinatorial optimization: a decade of results. *Int J Autom Comput.* 2007;4(3):281–93. doi:10.1007/s11633-007-0281-3.
44. Dhal KG, Sasmal B, Das A, Ray S, Rai R. A comprehensive survey on arithmetic optimization algorithm. *Arch Comput Methods Eng.* 2023;30(5):3379–404. doi:10.1007/s11831-023-09902-3.
45. Jia H, Rao H, Wen C, Mirjalili S. Crayfish optimization algorithm. *Artif Intell Rev.* 2023;56(Suppl 2):1919–79. doi:10.1007/s10462-023-10567-4.
46. Zebiri I, Zeghida D, Redjimi M. Rat swarm optimizer for data clustering. *Jordan J Comput Inf Technol.* 2022;8(3):297–307. doi:10.5455/jjcit.71-1652735477.
47. Mirjalili S. SCA: a sine cosine algorithm for solving optimization problems. *Knowl Based Syst.* 2016;96(63):120–33. doi:10.1016/j.knsys.2015.12.022.
48. Hatamlou A. Black hole: a new heuristic optimization approach for data clustering. *Inf Sci.* 2013;222:175–84. doi:10.1016/j.ins.2012.08.023.
49. Shehab M, Abualigah L, Al Hamad H, Alabool H, Alshinwan M, Khasawneh AM. Moth-flame optimization algorithm: variants and applications. *Neural Comput Appl.* 2020;32(14):9859–84. doi:10.1007/s00521-019-04570-6.
50. El-Kenawy ESM, Khodadadi N, Mirjalili S, Abdelhamid AA, Eid MM, Ibrahim A. Greylag goose optimization: nature-inspired optimization algorithm. *Expert Syst Appl.* 2024;238(22):122147. doi:10.1016/j.eswa.2023.122147.
51. Hamad RK, Rashid TA. GOOSE algorithm: a powerful optimization tool for real-world engineering challenges and beyond. *Evol Syst.* 2024;15(4):1249–74. doi:10.1007/s12052-023-00345-7.
52. Amiri MH, Mehrabi Hashjin N, Montazeri M, Mirjalili S, Khodadadi N. Hippopotamus optimization algorithm: a novel nature-inspired optimization algorithm. *Sci Rep.* 2024;14(1):5032. doi:10.1038/s41598-024-54910-3.
53. Tian AQ, Liu FF, Lv HX. Snow Geese Algorithm: a novel migration-inspired meta-heuristic algorithm for constrained engineering optimization problems. *Appl Math Model.* 2024;126(8):327–47. doi:10.1016/j.apm.2023.10.045.
54. Bouaouda A, Hashim FA, Sayouti Y, Hussien AG. Pied kingfisher optimizer: a new bio-inspired algorithm for solving numerical optimization and industrial engineering problems. *Neural Comput Appl.* 2024;36(25):15455–513. doi:10.1007/s00521-024-09781-0.
55. Lian J, Hui G, Ma L, Zhu T, Wu X, Heidari AA, et al. Parrot optimizer: algorithm and applications to medical problems. *Comput Biol Med.* 2024;172(5):108064. doi:10.1016/j.combiomed.2024.108064.

56. Bai J, Li Y, Zheng M, Khatir S, Benaissa B, Abualigah L, et al. A sinh cosh optimizer. *Knowl Based Syst.* 2023;282(1):111081. doi:10.1016/j.knosys.2023.111081.
57. Sowmya R, Premkumar M, Jangir P. Newton-Raphson-based optimizer: a new population-based meta-heuristic algorithm for continuous optimization problems. *Eng Appl Artif Intell.* 2024;128:107532. doi:10.1016/j.engappai.2023.107532.
58. Abdel-Basset M, Mohamed R, Jameel M, Abouhawwash M. Nutcracker optimizer: a novel nature-inspired metaheuristic algorithm for global optimization and engineering design problems. *Knowl Based Syst.* 2023;262(3):110248. doi:10.1016/j.knosys.2022.110248.
59. Stanovov V, Akhmedova S, Semenkin E. LSHADE algorithm with rank-based selective pressure strategy for solving CEC, 2017 benchmark problems. In: 2018 IEEE Congress on Evolutionary Computation (CEC); 2018 Jul 8–13; Rio de Janeiro, Brazil. Piscataway, NJ, USA: IEEE; 2018. p. 1–8. doi:10.1109/CEC.2018.8477977.
60. Demšar J. Statistical comparisons of classifiers over multiple data sets. *J Mach Learn Res.* 2006;7:1–30. doi:10.5555/1248547.1248548.
61. Biedrzycki R. Revisiting CEC 2022. ranking: a new ranking method and influence of parameter tuning. *Swarm Evol Comput.* 2024;89(6):101623. doi:10.1016/j.swevo.2024.101623.
62. Osuna-Enciso V, Cuevas E, Castañeda BM. A diversity metric for population-based metaheuristic algorithms. *Inf Sci.* 2022;586(1):192–208. doi:10.1016/j.ins.2021.11.073.
63. Cuevas E, Diaz P, Camarena O. Experimental analysis between exploration and exploitation. In: *Metaheuristic computation: a performance perspective*. Cham, Switzerland: Springer; 2021. p. 249–69. doi:10.1007/978-3-030-58100-8_10.
64. Lang Y, Gao Y. Dream optimization algorithm (DOA): a novel metaheuristic optimization algorithm inspired by human dreams and its applications to real-world engineering problems. *Comput Methods Appl Mech Eng.* 2025;436(2):117718. doi:10.1016/j.cma.2024.117718.
65. Tzanetos A, Blondin M. A qualitative systematic review of metaheuristics applied to tension/compression spring design problem: current situation, recommendations, and research direction. *Eng Appl Artif Intell.* 2023;118(1):105521. doi:10.1016/j.engappai.2022.105521.
66. Kamil AT, Saleh HM, Abd-Alla IH. A multi-swarm structure for particle swarm optimization: Solving the welded beam design problem. *J Phys Conf Ser.* 2021;1804:012012. doi:10.1088/1742-6596/1804/1/012012.
67. Chopra N, Ansari MM. Golden jackal optimization: a novel nature-inspired optimizer for engineering applications. *Expert Syst Appl.* 2022;198(5):116924. doi:10.1016/j.eswa.2022.116924.
68. Sahu SS, Satapathy SC, Naik A. Hybridization of social group optimization and differential evolution algorithm for solving speed reducer design problem. *Indian J Sci Technol.* 2024;17(23):2455–62. doi:10.17485/IJST/v17i23.965.
69. Onwubolu GC, Babu BV, Zelinka I, Lampinen J. Mechanical engineering problem optimization by SOMA. In: *New optimization techniques in engineering*. Berlin, Germany: Springer; 2004. p. 633–53. doi:10.1007/978-3-540-39930-8_26.
70. Eker E. Assessment of GTO: performance evaluation via constrained benchmark function, and optimized of three bar truss design problem. *Dicle Univ Muhendislik Fak Muhendislik Derg.* 2023;14(1):27–33. doi:10.24012/dumf.1211918.
71. Cadet G, Paredes M. Optimized dimensioning of helical compression springs. *Eur J Mech A Solids.* 2024;107(4):105385. doi:10.1016/j.euromechsol.2024.105385.
72. Alkurdi AAH. Optimization of welded beam design problem using water evaporation optimization algorithm. *Acad J Nawroz Univ.* 2023;12(3):499–509. doi:10.25007/ajnu.v12n3a1753.
73. Hassan S, Kumar K, Raj CD, Sridhar K. Design and optimisation of pressure vessel using metaheuristic approach. *Appl Mech Mater.* 2014;465:401–6. doi:10.4028/www.scientific.net/AMM.465-466.401.

74. Abd Elaziz M, Elsheikh AH, Oliva D, Abualigah L, Lu S, Ewees AA. Advanced metaheuristic techniques for mechanical design problems. *Arch Comput Methods Eng.* 2022;29(9):695–716. doi:10.1007/s11831-021-09589-4.
75. Korta JA, Mundo D. A population-based meta-heuristic approach for robust micro-geometry optimization of tooth profile in spur gears considering manufacturing uncertainties. *Meccanica.* 2018;53(1):447–64. doi:10.1007/s11012-017-0703-4.
76. Shaikh MS, Lin H, Xie S, Dong X, Lin Y, Shiva CK, et al. An intelligent hybrid grey wolf-particle swarm optimizer for optimization in complex engineering design problem. *Sci Rep.* 2025;15(1):18313. doi:10.1038/s41598-025-02154-0.
77. Banerjee M, Garg V, Deep K, Jasser MB, Kamel S, Mohamed AW. Enhancing sine-cosine mutation strategy with Lorentz distribution for solving engineering design problems. *Int J Syst Assur Eng Manag.* 2024;15(4):1536–67. doi:10.1007/s13198-023-02106-9.
78. Kulkarni AJ, Kale IR, Shastri A, Khandekar A. Snail homing and mating search algorithm: a novel bio-inspired metaheuristic algorithm. *Soft Comput.* 2024;28(17):10629–68. doi:10.1007/s00500-024-09754-4.