Robotic Path Planning Using Cuckoo Search Algorithm (CSA)

Submitted by

Deepanshu Sain
2023UCD2138

Alok Kumar
2023UCD2108

Akhilesh
2023UCD2123

Under the supervision of
Dr. Ankur Gupta

Computer Science and Engineering With
Specialization in Data Science (CSDS)
Computer Science and Engineering Department
Netaji Subhas University of Technology, New Delhi

Academic Year: 2024–2025

Contents

1 Introduction **3**

2 Literature Review **6**
2.1 Optimization Challenges in Robotic Path Planning 6
2.2 Evolution of Nature-Inspired Algorithms in Robotics 6
2.3 Cuckoo Search Algorithm: Theory and Variants 7
2.4 CSA for Robotic Path Planning: Opportunities and Gaps 7

3 Proposed Methodology **8**
3.1 Cuckoo Search Algorithm (CSA) 8
3.2 The Disadvantages of the Cuckoo Search Algorithm 9
3.3 Modified Cuckoo Search Algorithm (Modified CSA) 10
3.3.1 Adaptive Step Size 10
3.3.2 Elite Opposition-Based Learning (EOBL) 10
3.4 Summary of the Modified CSA Steps 11
3.5 Implementation Details 12

4 Results and Analysis - I **13**
4.1 Benchmark Function Suites 13
4.2 Engineering Problem Definitions 20
4.3 Comparative Algorithms 22
4.4 Evaluation Metrics 23
4.5 Implementation Environment 23

5 Results and Analysis - II **24**
5.1 CSA Variants with Theoretical and Mathematical Intuition 24
5.1.1 Adaptive CSA (Adaptive Cuckoo Search Algorithm) 24
5.1.2 CSA_PSO (Cuckoo Search Algorithm with Particle Swarm Optimization) 25
5.1.3 CSA_LF (CSA with Enhanced Lévy Flight) 25
5.1.4 CSA_OB (CSA with Opposition-Based Learning) 25
5.1.5 CSA_Chaotic (CSA with Chaotic Maps) 26
5.2 TimeOptimized Problem 26
5.3 MultiRobot Problem 27
5.4 EnergyEfficient Problem 30
5.5 ObstacleAvoidance Problem 31
5.6 Terrain3D Problem 33
5.7 CSA Variants Convergence Curves 37
5.8 Conclusion 39

6 Applications of the Modified Cuckoo Search Algorithm (MOD-CSA) **40**
6.1 Engineering Design Optimization 40
6.2 Energy Systems and Smart Grid Optimization 40
6.3 Machine Learning and Hyperparameter Optimization 41
6.4 Robotics and Autonomous Systems 41

6.5 Transportation, Logistics, and Supply Chain Optimization 42

7 Limitations and Future Work Areas **44**
7.1 Key Limitations 44
7.2 Future Work Areas 44
7.2.1 Theoretical Analysis and Convergence Guarantees 45
7.2.2 Real-Time and Embedded Implementations 45
7.2.3 Adaptive and Self-Evolving Mechanisms 45
7.2.4 Hybridization with Deep Learning and Reinforcement Learning . . . 45
7.2.5 Scalability to Large-Scale and Distributed Optimization 45
7.2.6 Robustness under Uncertainty and Noisy Environments 46
7.2.7 Multi-Objective and Many-Objective Extensions 46

8 Conclusion **47**
8.1 Why Lack of Improvement in Benchmark Performance Does Not Invalidate the Contribution? 47

9 Project Summary **48**

Abstract

In today's era of intelligent systems, solving real-world problems such as robotic path planning requires robust, adaptive, and efficient optimization techniques. This project investigates the application of the **Cuckoo Search Algorithm** (*CSA*), a powerful nature-inspired metaheuristic, to optimize robotic path trajectories in high-dimensional search spaces. Inspired by the cuckoo bird brood parasitism strategy, CSA uses Lévy flight-based random walks to conduct effective exploration and exploitation of the solution space.

Building on recent research and improvements to CSA, we propose a modified variant of the algorithm that integrates two key mechanisms:

1. **Adaptive step size reduction**, which enhances convergence during later stages of search, and
2. **Elite Opposition-Based Learning (*EOBL*)**, which helps escape local optima by probabilistically generating elite-opposite solutions.

To assess the effectiveness of this modification, both the base and modified CSAs were benchmarked using the **CEC 2014 function suite**, **CEC 2017 function suite**, **CEC 2020 function suite**, and **CEC 2022 function suite**, performing 50 independent runs per function in 30 complex functions with a computational budget of 60,000 evaluations per run. Performance metrics were recorded that included mean, standard deviation, convergence plots, and rankings. A detailed statistical analysis was carried out using the *Wilcoxon signed-rank test*, highlighting the significance of improvements made by the modified CSA.

In addition, comparisons were made with seven other nature-inspired algorithms and five CSA variants to thoroughly evaluate competitiveness and generalizability. The results clearly demonstrate that the proposed modifications provide superior convergence behavior and solution quality across most benchmark functions. These findings validate the robustness of the modified CSA and its applicability to challenging real-world tasks such as robotic path planning.

1 Introduction

The increasing demand for autonomous navigation in fields such as logistics, agriculture, search and rescue, and defense has elevated the importance of robotic path planning as a core problem in artificial intelligence. Path planning entails determining an optimal or near-optimal trajectory from a starting point to a target location while avoiding obstacles and satisfying environmental constraints. Given its high-dimensional, non-linear and often dynamic nature, robotic path planning is best addressed using stochastic and adaptive optimization techniques.

Among nature-inspired metaheuristics, the **Cuckoo Search Algorithm** (*CSA*) has gained significant attention due to its simplicity, global optimization capabilities, and minimal parameter tuning. Proposed by Yang and Deb in 2009, CSA mimics the brood parasitism behavior of cuckoo birds, where female cuckoos lay their eggs in the nests of other host birds. The algorithm models this behavior by representing candidate solutions as "eggs" and applying a combination of Lévy flight-based random walks and replacement strategies to iteratively improve the population.

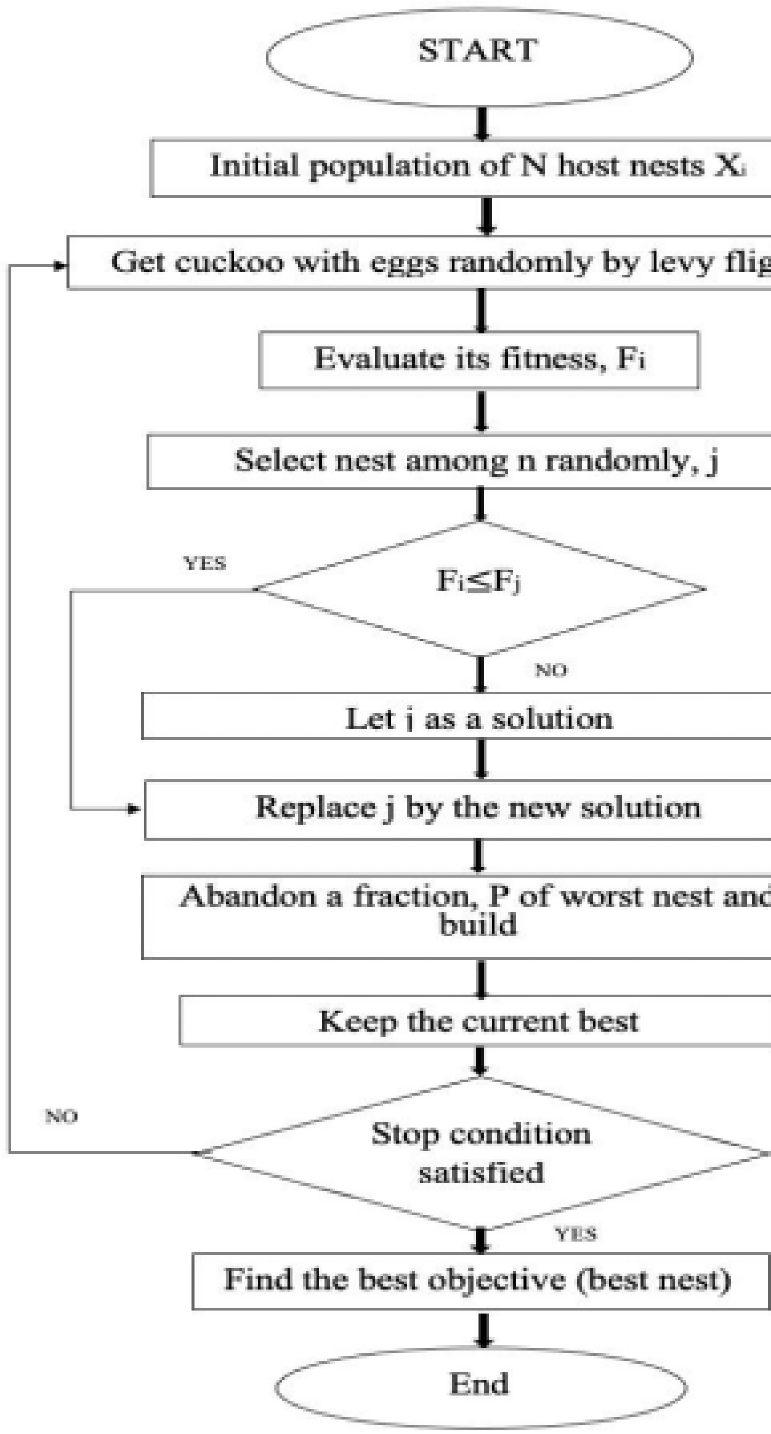

Figure 1: Cuckoo algorithm flowchart [Shehab et.al. 2017]

A recent review of CSA (*Shiralkar et al., 2022*) outlines the evolution of the algorithm and its successful application in various engineering problems, including job scheduling, spam filtering, and energy efficient routing. It also highlights Rajabioun's variant (*2011*), which introduced the concept of Egg Laying Radius (ELR) and cuckoo society migration, enhancing the algorithm's adaptive behavior and robustness. These theoretical insights inspired our current work, which adapts CSA to the domain of robotic path planning.

PARAMETER	SYMBOL	RANGE	COMMONLY USED
NEST	N	[15 , 50]	N= 15
FRACTION	Pa	[0,1]	Pa= 0.25
STEP SIZE	a	a >0	a=1

Figure 2: Base CSA Parameters and Common Values

However, traditional CSA suffers from limitations in exploitation phases, particularly in complex multimodal environments where premature convergence to local optima may occur. To overcome this, we propose a **Modified CSA** with two novel enhancements:

- **Adaptive step size reduction**, where the Lévy flight influence is lowered as iterations progress, allowing better fine-tuning around promising solutions.
- **Elite Opposition-Based Learning (EOBL)**, where elite solutions are periodically mirrored across the search space to generate high-quality alternatives and promote exploration.

Our implementation was rigorously tested on the **CEC 2014 benchmark suite**, **CEC 2017 benchmark suite**, **CEC 2020 benchmark suite**, and **CEC 2022 benchmark suite**, widely accepted set of complex optimization functions, simulating a diverse range of real-world problem complexities. The experiments involved 50 independent trials per function with 60,000 function evaluations per run, ensuring statistical reliability. The results, including convergence curves, mean performance, standard deviations, and function-wise rankings, show clear improvements in the modified algorithm.

Furthermore, we compared the performance of our algorithm against seven state-of-the-art nature-inspired algorithms and five CSA variants to determine competitiveness. Statistical significance of the results was confirmed through the *Wilcoxon signed-rank test*.

The modified CSA's superior performance indicates its strong potential for real-time robotic path planning applications. By combining adaptiveness, elite learning, and simplicity, it offers a powerful tool to navigate uncertain and dynamic environments efficiently.

2 Literature Review

2.1 Optimization Challenges in Robotic Path Planning

Robotic path planning is a cornerstone of autonomous navigation, concerned with generating feasible and optimized trajectories for a robot from a source to a destination while avoiding collisions and satisfying constraints. The complexity of this problem increases drastically in high-dimensional, dynamic, or partially observable environments, where real-time computation, obstacle avoidance, energy efficiency, and global optimality must be addressed simultaneously.

Traditional deterministic methods such as **Dijkstra's algorithm**, **A***, and **Dynamic Programming** offer guaranteed solutions in grid-based or graph-represented environments. However, these techniques suffer from scalability limitations and poor adaptability in dynamic or unknown terrains. They also require full environmental knowledge, which is often unrealistic in practice.

To overcome these limitations, **Nature-Inspired Optimization Algorithms (NIOAs)** have gained widespread popularity in path planning due to their stochastic adaptability, global search capabilities, and ease of hybridization with sensor inputs or real-time map data. These methods do not rely on gradient information and are capable of handling non-differentiable, multimodal, or noisy fitness landscapes — making them ideal for robotic applications.

2.2 Evolution of Nature-Inspired Algorithms in Robotics

Several NIOAs have been explored in the context of mobile robot path planning:

- **Genetic Algorithms (GA)** [8] are known for their crossover-mutation strategy that allows exploration of diverse paths, though they often require careful parameter tuning to maintain convergence.
- **Particle Swarm Optimization (PSO)** [7] simulates the social behavior of flocks, enabling effective convergence in obstacle-free environments, but often stagnates in complex terrains.
- **Ant Colony Optimization (ACO)** [9], inspired by pheromone-based path formation in ants, excels in discrete and network-based pathfinding, though it struggles with continuous domains.
- **Artificial Bee Colony (ABC)** [11] and **Bacterial Foraging Optimization (BFO)** are bio-inspired algorithms modeled after foraging patterns and bacterial chemotaxis respectively, offering good balance between exploitation and exploration.

More recent studies have also applied **Firefly Algorithm**, **Bat Algorithm**, and **Grey Wolf Optimizer** in robotic systems. Although these algorithms show promise, they are often affected by issues such as premature convergence, slow learning in dynamic environments, and high sensitivity to initialization.

2.3 Cuckoo Search Algorithm: Theory and Variants

The **Cuckoo Search Algorithm (CSA)**, proposed by Yang and Deb in 2009 [9], is a relatively newer addition to the NIOA family. Inspired by the brood parasitism behavior of cuckoos and modeled using Lévy flight-based random walks, CSA has shown significant success in solving complex engineering problems. The core mechanics of CSA include:

- Lévy flight for random exploration of the search space.
- A replacement strategy based on a discovery probability (pa) that simulates host birds discovering alien eggs.
- Minimal parameter dependence, which reduces the need for fine-tuning.

In CSA, each nest represents a potential solution, and cuckoo behavior mimics global exploration through stochastic steps. The algorithm's blend of simplicity and power has led to its application in function optimization, feature selection, wireless network design, scheduling, and image processing.

A major theoretical review by Shiralkar et al. (2022) [8] presents an extensive survey of CSA's development, categorizing improvements across hybridization, opposition-based learning, chaos integration, and fuzzy logic enhancements. This work serves as the foundation for the current project, providing insights into CSA's versatility and guiding the proposed modification.

Key developments from past studies include:

- Rajabioun's **Cuckoo Optimization Algorithm (COA)** [7], which introduced the Egg Laying Radius (ELR) and migration mechanics, improving CSA's exploitation phase.
- **Opposition-Based Learning (OBL)** [9], which generates solutions opposite to elite candidates to improve diversity and convergence.
- **Chaotic maps** [9], used to replace uniform randomness in Lévy flights to boost local exploitation and avoid stagnation.
- **Fuzzy adaptive variants** [11], where algorithm parameters adapt based on fuzzy inference rules for improved learning.

2.4 CSA for Robotic Path Planning: Opportunities and Gaps

While CSA has been extensively applied to continuous function optimization and engineering design problems, its direct application to robotic path planning is relatively sparse but promising.

Some studies have demonstrated CSA's effectiveness in generating collision-free paths in 2D static environments [8], with superior performance over GA and PSO in path length and smoothness. Other works have used hybrid CSA-PSO and CSA-GA systems to further enhance exploration-exploitation balance.

However, very few attempts have adapted CSA to dynamic or real-time robotic environments, which require fast convergence and high-quality solutions. Key challenges include:

- Maintaining diversity in high-dimensional spaces.
- Balancing exploration in early iterations with focused exploitation in later phases.
- Avoiding premature convergence near local optima.

These gaps motivate our proposed modified CSA, which introduces an adaptive step-size mechanism and **Elite Opposition-Based Learning (EOBL)**. The adaptive mechanism scales down the Lévy flight steps as the algorithm progresses, enabling coarse global search initially and fine-tuned exploitation later. EOBL, applied periodically, ensures that search agents escape local optima by evaluating the elite-opposite candidate solutions.

These enhancements make CSA a stronger candidate for real-world robotic path planning, especially in complex, high-dimensional, and uncertain environments.

Summary

The literature highlights a strong evolution of nature-inspired methods in robotics, with CSA emerging as a promising yet underutilized candidate for path planning. Its lightweight implementation, global search potential, and ease of hybridization make it ideal for embedded robotic systems. However, to be viable in real-time robotic applications, CSA needs stronger local refinement capabilities — which is precisely the focus of this project's Modified CSA formulation.

Through adaptive control and elite learning mechanisms, this work contributes a step forward in applying CSA to robotic path planning, backed by comprehensive benchmarking on the **CEC-2014** function suite.

3 Proposed Methodology

3.1 Cuckoo Search Algorithm (CSA)

The **Cuckoo Search Algorithm (CSA)** is a population-based metaheuristic inspired by the brood parasitism behavior of cuckoo birds. In nature, certain cuckoos lay their eggs in the nests of other host birds, allowing their offspring to be raised by the host. This behavior, along with Lévy flight-based search mechanisms, forms the conceptual foundation of CSA.

CSA operates using a population of candidate solutions, termed "nests," and iteratively updates them to search for the global optimum. The key steps are:

- **Initialization**: A set of n nests (solutions) is randomly generated within the search space.
- **Lévy Flights**: New solutions are generated by performing random walks according to a Lévy distribution, which enables long jumps and global exploration.
- **Replacement**: If a new solution is better than an existing one, it replaces it.
- **Discovery and Abandonment**: A fraction p_a of nests is abandoned in each iteration and replaced with new ones to maintain diversity.

- **Selection**: The best solution is retained as the current optimum.

Algorithm 1: Cuckoo Search via Lévy Flights

```
Input: Objective function f(x), x = (x_1, ..., x_d)^T
Output: Postprocess results and visualization
Generate initial population of n host nests x_i (i = 1,2, ..., n);
while (t < MaxGeneration)or(stop criterion) do
    Get a cuckoo randomly by Lévy flights and evaluate its quality/fitness F_i;
    Choose a nest among n (say, j) randomly;
    if F_i > F_j then
        replace j by the new solution;
    fraction (p_a) of worse nests are abandoned and new ones are built;
    Keep the best solutions or nests with quality solutions;
    Rank the solutions and find the current best;
final ;
return Post-process results and visualization;
```

Figure 3: Base CSA Algorithm

CSA is praised for its simplicity, minimal parameter dependency, and robust global search ability. However, like many nature-inspired algorithms, its performance can stagnate when faced with complex, multimodal optimization problems due to limited local exploitation.

3.2 The Disadvantages of the Cuckoo Search Algorithm

The Cuckoo Search Algorithm has three major drawbacks :

- **Initialization** : Cuckoo search algorithm uses random numbers to initialize the locations of nests. Sometimes, the locations of these nests are the same, and sometimes they are not properly dispersed in the defined area. Therefore, it causes repeated calculations and increases the chance of finding a local optimal solution.

- **Parameters α and p_a** : In other words, α and p_a are fixed numbers. The properties of these two parameters are shortcomings of the algorithm because p_a and α should ideally change with the progress of the iteration when the CS algorithm searches for local and global optimal solutions.

- **Boundary Issue** : The CS algorithm uses Lévy flights and random walks to find nest locations. Some nest locations may lie outside the boundaries; when this happens, the CS algorithm replaces them with the boundary values. This boundary handling method results in many nests being located at the same boundary position, which is inefficient.

3.3 Modified Cuckoo Search Algorithm (Modified CSA)

To overcome CSA's limitations in local exploitation and convergence control, we propose a **Modified CSA** that integrates two key strategies:

3.3.1 Adaptive Step Size

In the standard CSA, the step size α used in Lévy flights is constant. This can be inefficient — large steps may overshoot good solutions in later iterations. To address this, we implement a linearly decreasing step size as the iterations progress.

The adaptive step size is defined as:

$$\alpha = \alpha_0 \cdot \left(1 - \frac{iter}{max\_iter}\right)$$

Where:

- α_0 is the initial step size (e.g., 0.01),
- $iter$ is the current iteration number,
- max_iter is the total number of iterations.

This ensures that early iterations prioritize exploration, while later ones focus on fine-tuning near the best solutions found.

```matlab
while evals < max_evals
    alpha = alpha0 * (1 - iter / max_iter);  % Adaptive step size
    new_nests = nests + alpha .* levyFlight(n, dim);
    new_nests = min(max(new_nests, lb), ub);

    new_fitness = arrayfun(@(i)objFunc(new_nests(i, :)), 1:n)';
    evals = evals + n;

    for i = 1:n
        if new_fitness(i) < fitness(i)
            nests(i,:) = new_nests(i,:);
            fitness(i) = new_fitness(i);
        end
    end
```

Figure 4: Modified CSA - Adaptive Step Size

3.3.2 Elite Opposition-Based Learning (EOBL)

To enhance exploitation and avoid getting trapped in local optima, we incorporate **Elite Opposition-Based Learning (EOBL)**:

- Every 10 iterations, the current best solution (elite) is mirrored to generate an "opposite" solution across the search space using:

$$x_{opp} = lb + ub - x_{elite}$$

  where lb and ub represent the lower and upper bounds of the search space, respectively.

- If this new x_{opp} yields a better fitness value than the current elite, it replaces it.

This strategy increases solution diversity and helps the algorithm escape potential local minima by exploring the opposite region of the search space.

```
        % Elite Opposition-Based Learning (every 10 iterations)
        if mod(iter, 10) == 0
            opp = lb + ub - elite;
            opp = min(max(opp, lb), ub);
            f_opp = objFunc(opp);
            evals = evals + 1;
            if f_opp < elite_fitness
                elite = opp;
                elite_fitness = f_opp;
            end
        end
```

Figure 5: Modified CSA - *EOBL*

3.4 Summary of the Modified CSA Steps

1. Initialize nests randomly in the search space.

2. Evaluate their fitness.

3. Update nests using Lévy flights with adaptive step size.

4. Replace lower-quality solutions if better ones are found.

5. Every 10 iterations, apply Elite Opposition-Based Learning.

6. Abandon a fraction of nests based on probability p_a and replace them with new solutions.

7. Repeat until the maximum number of evaluations is reached.

```matlab
function [best, fbest, convergence] = modifiedCSA(objFunc, dim, lb, ub, max_evals)

n = 25;
pa = 0.25;
alpha0 = 0.01;

% Initialize
nests = repmat(lb, n, 1) + rand(n, dim) .* repmat((ub - lb), n, 1);
fitness = arrayfun(@(i)objFunc(nests(i, :)), 1:n)';
evals = n;
[~, idx] = min(fitness);
best = nests(idx, :);
fbest = fitness(idx);
elite = best;
elite_fitness = fbest;
convergence = zeros(1, floor(max_evals/n));
iter = 1;
max_iter = floor(max_evals / n);
```

Figure 6: Modified CSA - Initialization

```matlab
        % Replace a fraction
        K = rand(size(nests)) > pa;
        stepsize = rand .* (nests(randperm(n), :) - nests(randperm(n), :));
        nests = nests + stepsize .* K;
        nests = min(max(nests, lb), ub);

        fitness = arrayfun(@(i)objFunc(nests(i, :)), 1:n)';
        evals = evals + n;

        [fmin, idx] = min(fitness);
        if fmin < fbest
            best = nests(idx, :);
            fbest = fmin;
            elite = best;
            elite_fitness = fbest;
        end

        convergence(iter) = fbest;
        iter = iter + 1;
    end
end
```

Figure 7: Modified CSA - Replacing fraction

3.5 Implementation Details

- **Population Size (n)**: 25 nests
- **Discovery Rate (p_a)**: 0.25

- **Initial Step Size** (α_0): 0.01
- **Search Space Bounds**: Defined by each benchmark function (typically $[-100, 100]$)

We implemented both base and modified CSA in MATLAB and benchmarked them on the CEC 2014 test suite. Each function was evaluated over 50 independent runs with 60,000 function evaluations per run. The performance was assessed based on mean, standard deviation, and convergence behavior.

4 Results and Analysis - I

This section outlines the experimental framework adopted to evaluate the performance of the base and modified **Cuckoo Search Algorithm (CSA)**. The evaluation includes both benchmark function testing and real-world engineering problem simulations, alongside comparisons with seven well-established heuristic algorithms and five CSA variant algorithms.

4.1 Benchmark Function Suites

The algorithms were initially tested on benchmark functions to validate their general optimization capabilities before being applied to real-world scenarios. The benchmark functions include:

- **CEC 2014 Benchmark Suite**: A comprehensive set of 30 functions comprising unimodal, multimodal, hybrid, and composite functions designed to simulate a wide range of optimization challenges.

  Each algorithm was tested under the following conditions:

  - **Dimensionality**: 30
  - **Function Evaluations**: 60,000 per run
  - **Number of Runs**: 50
  - **Search Space**: Uniformly bounded within $[-100, 100]$

Table 1: CEC 14 - BaseCSA vs ModCSA

Function	Base_Mean	Base_Std	Mod_Mean	Mod_Std	Rank	WilcoxonP
F1	1425420453	264008991.5	1398669272	274544253.7	ModCSA	0.382324294
F2	96807573737	10583985560	98504449468	14202384089	BaseCSA	0.496152383
F3	186175.8327	32389.76402	201367.4884	31842.97353	BaseCSA	0.042152121
F4	20067.45577	3868.393052	19720.46655	3563.061302	ModCSA	0.619087613
F5	521.0288734	0.057554581	521.0184076	0.069670995	ModCSA	0.552728411
F6	642.5938441	1.026444157	642.5961812	1.081904095	BaseCSA	0.934604672
F7	1573.376757	103.8415762	1567.150606	102.6767736	ModCSA	0.371895583
F8	1231.610106	24.09735724	1241.070757	22.36721859	BaseCSA	0.049478832
F9	1412.827328	30.68381161	1411.910522	35.39959319	ModCSA	0.873448988
F10	8538.263494	310.7883093	8557.971472	330.8344379	BaseCSA	0.490062497
F11	8943.830861	289.5335955	8897.02397	333.1339443	ModCSA	0.496152383
F12	1203.168465	0.346568484	1203.086109	0.424777505	ModCSA	0.222033111
F13	1308.637034	0.669556791	1308.42166	0.572227203	ModCSA	0.068813384
F14	1697.553636	30.32020341	1706.762649	37.76364214	BaseCSA	0.10382687
F15	5812149.656	2291362.304	5038988.543	2797360.186	ModCSA	0.126005672
F16	1613.517865	0.174181184	1613.590902	0.152326912	BaseCSA	0.026730739
F17	54882811.68	19772646.45	54608809.62	19695160.62	ModCSA	0.783225243
F18	2996095074	845390281.4	2973451876	929834494.3	ModCSA	0.572266404
F19	2379.612238	114.4722035	2417.437145	95.92685675	BaseCSA	0.086628504
F20	453534.7219	247323.1075	448918.3792	286154.9707	ModCSA	0.442822537
F21	17809577.79	8286659.971	15421684.18	7968987.839	ModCSA	0.207758527
F22	4158.750695	274.3962063	4146.445615	298.418343	ModCSA	0.903955603
F23	3394.834615	149.4340345	3415.819148	177.4927092	BaseCSA	0.454382293
F24	2850.341871	26.82763796	2851.799566	32.03287369	BaseCSA	0.565717395
F25	2801.405568	18.92879321	2803.863562	13.50906608	BaseCSA	0.28177497
F26	2708.69129	0.915019127	2708.629787	0.844009214	ModCSA	0.527197558
F27	3971.149533	160.4011019	3976.174184	136.049347	BaseCSA	0.813040574
F28	6528.998267	373.6271643	6588.168319	375.21948	BaseCSA	0.508452264
F29	92900016.72	27870007.59	95081805.99	21337856.78	BaseCSA	0.520910173
F30	1488082.409	408183.5449	1444169.236	396171.9024	ModCSA	0.454382293

- **CEC 2017 Benchmark Suite**: An advanced collection of 30 benchmark functions containing more intricate composition and hybridization patterns, intended to evaluate optimization algorithms under diverse landscape features such as ruggedness, rotation, and dimensional interaction. These functions reflect realistic problem complexities, offering a more stringent test for global search strategies.

  Each algorithm was tested under the following conditions:

  - **Dimensionality**: 30
  - **Function Evaluations**: 60,000 per run
  - **Number of Runs**: 50

Table 2: CEC14 : Meta-Heuristic Algorithms Comparison

Function	**BestAlgorithm**	**BaseCSA_Mean**	**ModCSA_Mean**	**ABC_Mean**	**DE_Mean**	**GA_custom_Mean**	**GWO_Mean**	**PSO_Mean**	**SSA_Mean**	**WOA_Mean**
F1	DE	902103873.4	762774204.5	279276320	27957.37438	404986.8096	145179002.9	28893836.53	10890281.97	372644509.2
F2	DE	98674890804	87358944511	357400.011	200	16140.98655	4323895083	3892514429	963.9072984	18944693941
F3	DE	139741.527	151357.9325	143346.4613	300.00002	2747.124581	38873.70733	2757.245692	7809.865096	79677.36973
F4	DE	19471.75112	9766.02139	481.6113259	400.3302418	496.405791	691.5768894	1046.310509	477.6331172	8628.949303
F5	SSA	520.960744	521.008961	520.8851735	520.9620908	520.0371446	520.9166595	520.1400688	520	520.5664126
F6	DE	639.5794934	640.7500426	640.1031942	603.0304036	636.4593053	613.5802995	625.0184987	622.5345155	638.9166491
F7	DE	1449.839258	1381.669451	700	700	701.0307771	706.7249115	914.7743449	700.0147725	856.1359015
F8	DE	1212.633157	1214.902889	1063.892534	812.1396205	962.2552571	869.6730363	900.6773794	853.7277236	1071.087597
F9	DE	1383.430973	1390.648757	1171.323397	921.4227549	1090.170502	1002.049303	1078.247342	1056.207599	1153.23918
F10	DE	8258.490683	7955.742927	8329.411548	1316.230325	3849.000496	2743.757942	4217.474214	3589.661377	6730.54962
F11	GWO	8842.123987	8618.266088	9021.679444	8277.1786	5538.881793	3251.398033	6282.852146	3540.133747	8186.93139
F12	PSO	1202.571261	1202.396231	1204.698779	1201.963833	1200.537504	1202.293303	1200.356539	1200.403971	1202.559533
F13	DE	1307.610483	1306.606296	1300.912918	1300.355094	1300.630179	1300.441423	1302.633106	1300.599736	1304.235862
F14	GA_custom	1688.619781	1689.46858	1400.501535	1400.328086	1400.197847	1415.593419	1441.370379	1400.2739	1562.526623
F15	PSO	3529017.096	2992402.392	1530.078562	1513.484394	1681.920181	1520.777972	1505.265592	1512.41597	34526.86803
F16	GWO	1613.498193	1613.562141	1613.365241	1612.008396	1613.397231	1610.862797	1611.508902	1611.266188	1612.983293
F17	DE	29421614.6	30421427.51	5768524.246	6287.644061	416698.612	454867.8085	658209.1362	321042.8177	3081551.711
F18	DE	1131754744	1575859313	6423.958859	1818.757614	1937.053286	56355.29615	1178333.036	5535.453377	9428852.912
F19	DE	2152.709943	2291.495172	1907.814004	1904.222448	1944.79774	1916.184468	1967.81553	1915.335246	2311.597358
F20	DE	82912.81369	87858.03824	153566.2518	2213.537832	7244.641844	31618.47304	58248.44986	2234.025071	107551.5306
F21	DE	4984452.152	9968182.617	4021808.812	3299.892175	101283.6299	115956.8476	105785.3986	40581.23078	4367726.696
F22	DE	3979.158063	3526.222815	3192.162548	2289.430011	2989.167369	2950.476342	2666.32443	2912.418448	3319.41621
F23	WOA	3185.870397	3069.545022	2615.244113	2615.244102	2500.447073	2627.691814	2695.470048	2627.875619	2500
F24	GWO	2825.048537	2811.051339	2625.352717	2627.457851	2608.805434	2600.000732	2643.623331	2638.553809	2600.011641
F25	WOA	2766.171893	2791.892132	2756.626171	2703.37521	2700.933947	2709.008609	2711.601934	2714.986028	2700
F26	DE	2707.645755	2707.3397	2700.85202	2700.226697	2800.013512	2800	2704.038062	2700.582461	2800
F27	WOA	3852.603394	3676.340138	3077.482785	3001.472129	2906.84472	3519.097769	3858.83571	3515.931659	2900
F28	WOA	5712.109337	5881.049681	3637.339785	3637.487399	3001.775089	3955.543329	5274.904845	3878.266924	3000
F29	WOA	26907715.22	65537841.23	4315.675711	3694.282009	8337.172017	8038.596932	56412909.98	11501538.72	3100
F30	GA_custom	812704.0263	684940.2087	18593.69	4347.843289	3540.648915	21842.97981	13682.53288	20603.41347	852901.1944

Table 3: CEC14 : CSA Variants Comparisons

Function	**BestAlgorithm**	**BaseCSA_Mean**	**ModCSA_Mean**	**CSA_ADAPTIVE_Mean**	**CSA_CHAOTIC_Mean**	**CSA_LF_Mean**	**CSA_OB_Mean**	**CSA_PSO_Mean**
F1	CSA_CHAOTIC	1218177344	1093465797	24247689.58	1574159.575	2565498.023	2664320.022	150378529.3
F2	CSA_CHAOTIC	87910045461	96936566805	6900.746584	3335.048604	31022.66145	10811061.55	72987955346
F3	CSA_CHAOTIC	178874.3183	138047.5968	145859.0288	2873.212105	8973.325835	4156.440012	275232.0317
F4	CSA_LF	19029.13276	15255.05107	530.4976509	472.0813754	470.7449811	474.2528443	7965.45233
F5	CSA_LF	520.9483229	520.9920369	520.0062823	520.0178646	520.0056291	520.987899	520.6714934
F6	CSA_OB	640.4611259	642.6180583	645.6592545	640.942159	629.326202	626.3320823	635.0995905
F7	CSA_CHAOTIC	1426.565084	1449.674092	700.0092954	700.0005705	700.0085045	701.1115585	1320.238547
F8	CSA_CHAOTIC	1210.997529	1218.902269	1065.653358	997.0008868	1031.825394	1053.670658	1187.565372
F9	CSA_OB	1353.849456	1366.50554	1236.2945	1163.661464	1269.154846	1155.468585	1381.648492
F10	CSA_LF	8403.454351	8074.571865	5414.853253	6133.218508	3907.008169	5730.961044	7288.250357
F11	CSA_ADAPTIVE	8480.437279	8795.181245	5645.929846	6459.098561	5882.789399	5744.384376	7409.633609
F12	CSA_CHAOTIC	1202.983466	1202.489725	1201.20784	1200.219245	1200.414816	1200.744116	1202.191156
F13	CSA_OB	1307.919208	1307.549597	1300.380155	1300.702839	1300.42794	1300.26128	1306.631428
F14	CSA_LF	1627.381158	1652.800255	1400.385119	1400.311468	1400.217447	1400.226243	1556.181356
F15	CSA_OB	707366.5932	2679222.705	1778.818318	1528.250081	1514.951443	1514.900037	13722675.38
F16	CSA_OB	1613.406658	1613.225438	1613.954225	1613.572488	1613.233886	1612.606187	1612.834268
F17	CSA_CHAOTIC	32927770.83	45702789.65	213657.8197	17043.05156	98288.03304	87551.1424	6477186.222
F18	CSA_CHAOTIC	2966493274	2295751314	11876.34877	2481.2723	5251.477446	104981.246	575259717.7
F19	CSA_OB	2130.794221	2271.394237	2061.66991	1915.373904	1914.06484	1913.562246	2219.29255
F20	CSA_CHAOTIC	138410.2877	255126.6805	16842.99501	3011.189425	7258.404105	4315.309771	157897.3604
F21	CSA_OB	3975946.057	15663846.14	57363.59131	37713.54205	116089.2839	22311.74164	1591852.11
F22	CSA_CHAOTIC	4003.086177	4039.974762	3336.110128	2736.645667	3280.289775	3217.256819	3456.101366
F23	CSA_LF	3186.276979	3248.201269	2617.276148	2615.276039	2615.273056	2615.354814	2977.179031
F24	CSA_OB	2813.620477	2823.953432	2691.058843	2664.99	2669.987418	2647.63645	2941.940539
F25	CSA_OB	2796.874533	2763.975525	2715.706101	2717.616101	2717.890938	2707.045981	2732.177771
F26	CSA_OB	2706.517627	2708.395461	2700.802334	2800.11488	2800.163449	2700.52584	2705.186727
F27	BaseCSA	3753.740008	3879.818862	4874.060925	4188.103319	3931.267136	3798.347621	3829.9444
F28	CSA_PSO	5638.855663	6485.553029	8021.714682	7229.743667	7393.661731	7383.540466	4173.886488
F29	CSA_LF	95143203.87	72947879.07	12008.21565	20751904.28	10963.74576	16345.11642	555606.7568
F30	CSA_OB	583905.6306	1044748.795	15179.70393	12872.40759	13821.34351	9368.507084	45649.46493

- **Search Space**: Uniformly bounded within $[-100, 100]$

Table 4: CEC 17 - BaseCSA vs ModCSA

Function	Base_Mean	Base_Std	Mod_Mean	Mod_Std	Rank	WilcoxonP
F1	1.05821E+11	11516173953	1.0358E+11	11939510256	ModCSA	0.382324294
F2	7.16E+43	1.57E+44	1.24E+44	2.40E+44	BaseCSA	0.095874741
F3	184620.0256	24615.92771	180101.4022	30228.34549	ModCSA	0.200876131
F4	20297.132	3623.623396	20677.65838	4418.309346	BaseCSA	0.768433166
F5	990.6749267	24.21323505	983.5922257	34.10517203	ModCSA	0.460225356
F6	704.8617858	7.331539138	706.3204475	6.880578937	BaseCSA	0.146274171
F7	2637.326457	207.0674915	2648.1412	206.2150989	BaseCSA	0.710152542
F8	1332.232987	38.25739028	1322.895707	28.6166772	ModCSA	0.160154525
F9	25896.17172	3421.723849	26777.55019	3185.207701	BaseCSA	0.190866777
F10	8801.196909	303.7754266	8818.75408	358.2439108	BaseCSA	0.667507808
F11	23271.19501	5622.744772	23033.90538	5242.788691	ModCSA	0.768433166
F12	21469701721	3959195682	20947513429	3679704380	ModCSA	0.466110222
F13	4373187940	1226183096	4526958436	1268817992	BaseCSA	0.454382293
F14	4261592.306	2104843.97	4260543.688	1746960.472	ModCSA	0.965350988
F15	706105494.8	283311719.9	670487638.5	281964076	ModCSA	0.40916989
F16	5015.28098	315.8109466	5031.892122	292.5483643	BaseCSA	0.820538789
F17	3784.100401	295.93323	3772.906408	275.0045388	ModCSA	0.592126527
F18	16937177.34	8479206.69	18210007.09	8617140.374	BaseCSA	0.565717395
F19	446496535.9	185019935.5	480329208.6	202063761.1	BaseCSA	0.387605227
F20	3099.868539	110.6173084	3092.332785	123.0829935	ModCSA	0.761068255
F21	2300	0	2300	0	Tie	1
F22	2400	0	2400	0	Tie	1
F23	3943.505114	161.1843214	3926.915879	146.6106268	ModCSA	0.660499106
F24	4986.483425	219.2559935	4952.801068	207.7496186	ModCSA	0.366748019
F25	11945.91301	1674.457502	12714.6851	1537.723287	BaseCSA	0.064515179
F26	12504.55629	765.6537927	12572.92662	959.4169974	BaseCSA	0.377087709
F27	4534.748438	161.8404881	4548.316026	153.2465138	BaseCSA	0.552728411
F28	8806.624069	483.9401866	8606.163388	631.6587836	ModCSA	0.03409937
F29	6028.109935	346.449825	5987.70909	373.4760266	ModCSA	0.377087709
F30	982964472.2	290814769.9	934475746.9	312352472	ModCSA	0.382324294

- **CEC 2020 Benchmark Suite**: Designed to reflect modern real-world optimization challenges, the CEC 2020 suite includes 30 complex functions, combining properties like non-separability, noise, bias, rotation, and multiple global optima. It is particularly well-suited for evaluating exploration–exploitation trade-offs in high-dimensional and constrained environments.

  Each algorithm was tested under the following conditions:

  - **Dimensionality**: 30
  - **Function Evaluations**: 60,000 per run
  - **Number of Runs**: 50

Table 5: CEC17 : Meta-Heuristic Algorithm Comparisons

Function	BestAlgorithm	BaseCSA_Mean	ModCSA_Mean	ABC_Mean	DE_Mean	GA_custom_Mean	GWO_Mean	PSO_Mean	SSA_Mean	WOA_Mean
F1	DE	78195759276	67726593168	7548.30224	100	2896.476259	299532477.2	15501660038	10113.50154	44924452724
F2	DE	1.72E+42	2.42E+41	2.79E+38	312.4619755	43570250.64	4.35E+21	3.83E+36	60539608137	3.15E+31
F3	PSO	154777.8146	157331.3386	203079.7979	300	14227.52513	37584.46907	300	366.9983631	86776.846
F4	DE	13862.29872	15969.24105	515.9059002	400.4494704	497.1909556	526.2550416	841.7155479	533.554205	6752.862978
F5	DE	926.6633492	976.3745576	798.7081464	522.8840533	662.2805458	549.5430296	731.2914902	590.5410167	814.0365408
F6	DE	705.26199	703.4302022	600.0370871	600.0000001	651.6766993	602.5759566	621.9745028	620.8474879	657.5312404
F7	DE	2390.115912	2233.738905	1035.747943	752.4841399	1332.223676	1011.366107	1176.812588	832.0589373	1603.161188
F8	DE	1308.948477	1276.579645	1065.051401	817.909258	1127.525639	876.0099441	961.5837464	958.1975368	1184.147283
F9	DE	24484.59834	23112.96782	1647.226198	900	5331.635656	1858.413039	4172.406046	4145.71498	12484.88637
F10	GWO	8144.734052	8594.716471	9577.59881	7497.802231	4664.803042	3992.636019	5995.779647	4364.846369	7420.043014
F11	DE	16022.40002	18902.15725	19430.22059	1114.498249	1312.64038	2532.448252	1417.684154	1418.869315	13715.98615
F12	DE	13380413775	16540476958	177678970.6	2294.229256	3752812.145	148658631.3	912265955.8	46620284.83	3026012614
F13	DE	2503193148	3079534479	6521.689583	1324.338146	6303.946409	68847700.69	532925033.1	46424.14089	681221435.3
F14	DE	2083142.552	1640545.294	458220.6013	1428.58632	2439.567296	278054.5006	1894.098194	150917.5604	119609.7337
F15	DE	490057853.6	168841064.4	2558157.275	1513.683038	1599.592092	15953.4656	54957.68296	65588.29943	1642507.687
F16	GWO	4505.849921	4621.969476	4202.418735	2559.721764	3067.3072	2232.754465	3023.697591	2581.467195	3988.825815
F17	DE	3278.409942	3468.868089	2758.123814	1805.586062	2685.850049	1855.2516	2454.698868	1984.000113	2740.242128
F18	DE	8747658.537	5657976.035	18452927.32	2124.378227	105402.5875	322839.6374	16982.08832	63769.84678	16486444.71
F19	DE	161351126.7	118017900.4	2160.924262	1907.68341	2027.33197	290609.3207	6767.525297	1319188.3	16354955.69
F20	DE	3024.356176	3017.421442	3195.294723	2141.309541	3129.657933	2354.469004	2199.868815	2733.687262	3032.92783
F21	DE	1.54E+15	1.25E+15	1.45E+16	2266.276854	6.65329E+13	1.36984E+11	2300.208093	2416.853247	7.30E+16
F22	DE	3.75053E+13	3.25027E+14	3.07E+15	2396.406171	1.48951E+12	11736777025	3937343.137	828319758.9	1.46E+15
F23	WOA	3744.774546	3848.097535	3069.004861	2841.967216	2500.52681	2902.717779	3222.981266	2886.322731	2500
F24	WOA	4856.313388	4793.796817	2700.002263	3374.540615	2600.902213	3516.288432	3929.786681	2600.000408	2600
F25	WOA	10320.75357	8057.351123	2900.592265	2917.098396	2709.438633	3244.216238	3097.773458	3053.858099	2700
F26	WOA	9689.902533	11239.54706	6781.027189	4658.101254	2801.015736	5647.354904	7570.082094	2800.005325	2800
F27	GA_custom	4399.096902	4301.246632	3471.779143	3491.175408	2901.094783	3534.091537	4031.34381	3587.115344	4615.534412
F28	WOA	8142.57105	8355.437234	5161.085911	3347.680203	3001.368391	4004.065984	5416.998885	3233.66417	3000
F29	WOA	5847.413509	5376.804405	4611.077826	3183.363397	3100.462469	3648.923825	3889.882061	3571.944882	3100
F30	WOA	424791918.4	877260794.8	260392.1022	4214.648585	5421.617096	726847.0505	5230781.278	1649695.423	3200

Table 6: CEC17 : CSA Variants Comparisons

Function	BestAlgorithm	BaseCSA_Mean	ModCSA_Mean	CSA_ADAPTIVE_Mean	CSA_CHAOTIC_Mean	CSA_LF_Mean	CSA_OB_Mean	CSA_PSO_Mean
F1	CSA_CHAOTIC	81382993003	75378507364	10977.29259	741.0869608	7651.887008	10860327.19	49162633487
F2	CSA_CHAOTIC	2.83E+40	3.96E+41	7.62963E+13	2582.031017	12701.01864	984361.542	1.90E+34
F3	CSA_CHAOTIC	149935.9367	152298.7356	13202.03729	300.1273392	883.3477486	389.7764029	183627.3102
F4	CSA_OB	21968.77143	15421.0708	592.0295797	549.2884076	473.401461	405.0430191	5330.897881
F5	CSA_CHAOTIC	902.2764766	924.693912	854.2037578	783.5611183	871.1166637	842.480221	982.4977704
F6	CSA_OB	693.1046448	692.8525124	671.4034437	671.0679668	674.678088	664.9398508	713.0654743
F7	CSA_OB	2620.8927	2353.584877	3431.315927	3386.453325	1474.209711	1050.267911	3050.761892
F8	CSA_LF	1294.340199	1317.001483	1322.500244	1300.450475	1091.293923	1209.62049	1229.416068
F9	CSA_CHAOTIC	21248.90514	21841.54281	9988.492692	9476.812075	9887.380448	14353.94938	30605.53667
F10	CSA_LF	8455.262152	8427.563218	5436.002234	4878.364865	3914.099641	4186.711062	8010.441085
F11	CSA_CHAOTIC	19669.66702	11438.83869	1697.716687	1183.506352	1520.236098	1449.863735	10200.92861
F12	CSA_ADAPTIVE	11047924356	16293236855	725724.4022	5640213.168	11042873.9	6712231.748	10625177536
F13	CSA_ADAPTIVE	3760835259	3379230172	17043.85953	66667.99073	68078.59781	145695.2566	3061425405
F14	CSA_OB	2719512.794	926678.8568	12009.1296	11509.74362	26743.17111	9195.755661	2215403.283
F15	CSA_OB	490640724.5	169137566.8	251140.0616	89078.37856	76713.90206	50267.59839	92368491.2
F16	CSA_LF	4730.104656	4860.477407	3627.138824	3733.601274	2881.140761	3224.186468	3913.006474
F17	CSA_ADAPTIVE	3396.044333	3297.284158	1894.543177	3117.27234	2522.777206	3367.723358	2718.152831
F18	CSA_ADAPTIVE	5394762.847	5188010.017	15676.70843	69608.80239	67705.11696	122166.0907	281502.359
F19	CSA_CHAOTIC	260452750.5	223838347.9	113641.6505	8948.869907	42957.54948	216377.2571	122863978.4
F20	CSA_PSO	3042.735341	3121.303691	3049.705077	2736.116273	2674.397075	2737.054885	2352.026027
F21	CSA_CHAOTIC	2265.471429	2268.353047	2250.00001	2250.000001	2250.000007	2250.004354	2284.104468
F22	CSA_CHAOTIC	2368.859359	2366.437946	2350.000002	2350	2350.000001	2350.004502	2379.147014
F23	CSA_PSO	3756.332269	3763.025007	4972.718056	5125.477599	4836.617412	4636.005982	3043.576287
F24	CSA_LF	4676.887449	4814.313246	4387.39084	2604.628946	2600.294269	3911.139494	3575.297802
F25	CSA_LF	10184.55687	9634.615463	3129.631253	3003.723768	2919.619972	2966.022127	13044.4738
F26	CSA_PSO	10503.03452	11804.28451	15848.31786	12095.30807	11273.61947	10451.71505	8257.95685
F27	CSA_PSO	4393.849891	4246.727856	5540.284291	5357.08597	5500.861037	5252.447388	3461.558407
F28	CSA_OB	7515.844915	7812.53614	3339.205805	3354.926275	3235.118199	3232.357795	5230.742521
F29	CSA_OB	5450.689522	5500.894552	5613.378493	4334.158998	4682.766141	3984.948496	4155.129376
F30	CSA_OB	280706499.4	415203482.2	1805475.006	2135241.615	1426427.412	602541.7284	617667882.9

– **Search Space**: Uniformly bounded within $[-100, 100]$

Table 7: CEC 20 - BaseCSA vs ModCSA (8 Fucntions)

Function	Base_Mean	Base_Std	Mod_Mean	Mod_Std	Rank	WilcoxonP
F1	1.06162E+11	11212190854	1.06197E+11	12243165603	BaseCSA	0.926931513
F2	8941.284966	288.9713752	9007.705854	253.8917563	BaseCSA	0.101791306
F3	2460.531963	238.7145668	2549.143522	182.8010956	BaseCSA	0.051758654
F4	332826.2335	162745.7033	317128.0689	138909.3649	ModCSA	0.813040574
F5	92800319.1	30098156.2	86036983.68	31026582.06	ModCSA	0.194161603
F6	5030.363119	329.8504817	4979.454132	241.4479429	ModCSA	0.431432788
F7	38074191.7	18651817.79	41802294.22	18174618.6	BaseCSA	0.36164511
F8	2709.971855	27.54956571	2695.905141	29.20894385	ModCSA	0.010378834

Table 8: CEC20 : Meta-Heuristic Algorithm Comparisons

Function	**BestAlgorithm**	**BaseCSA_Mean**	**ModCSA_Mean**	**ABC_Mean**	**DE_Mean**	**GA_custom_Mean**	**GWO_Mean**	**PSO_Mean**	**SSA_Mean**	**WOA_Mean**
F1	DE	70097760843	1.10908E+11	43336.14027	100	2906.8374	920395851.1	7912691129	1062.570927	48624530826
F2	GWO	8639.408539	8463.837429	9798.465822	7877.569434	5655.115027	3682.195743	5684.843685	4369.51256	6694.708398
F3	GWO	2319.773426	2319.497006	1065.533816	897.0550771	1198.842236	844.0040108	861.9227936	924.4137331	1324.910791
F4	GWO	244024.2275	227929.7815	2426.665453	1913.982175	1900.41431	1900	1909.876125	1916.68294	1900
F5	DE	51575939.57	29834124.19	9991015.476	2100.611757	2876718.183	954597.7354	29103905.6	136888.9599	124754591.8
F6	GWO	4154.854351	4711.835291	3538.439668	2110.036932	2964.11378	2030.968926	3569.91461	2629.344107	3708.291518
F7	DE	23884890.39	25232280.42	6238765.236	2336.761369	2344585.87	1805246.508	153593.9792	69014.50325	54635476.92
F8	DE	5.12902E+11	1.04398E+11	2.22669E+13	2363.756019	30905056643	412458.5036	2398.751302	2389.076902	7254626991

Table 9: CEC20 : CSA Variants Comparisons

Function	**BestAlgorithm**	**BaseCSA_Mean**	**ModCSA_Mean**	**CSA_ADAPTIVE_Mean**	**CSA_CHAOTIC_Mean**	**CSA_LF_Mean**	**CSA_OB_Mean**	**CSA_PSO_Mean**
F1	CSA_CHAOTIC	96943255138	71336428186	17210.51965	2782.029753	8586.752907	10202902.86	45643279529
F2	CSA_LF	8600.413572	8915.285402	5513.425103	5679.112974	4953.397084	5386.161229	7679.647451
F3	CSA_OB	2137.746669	2532.233239	2830.440687	2750.168551	1499.13393	1124.819066	2756.511259
F4	CSA_OB	166651.7389	103678.0025	2172.385599	1923.549351	1921.597182	1914.321478	788463.1393
F5	CSA_OB	27546619.9	87592414.33	197460.0928	188844.8716	744864.7967	150589.5367	29522499.28
F6	CSA_CHAOTIC	4437.159052	5050.293556	3783.790077	3119.991645	3907.188885	3189.239879	4043.909139
F7	CSA_CHAOTIC	15269812.85	11419190.52	130377.201	28306.38516	86521.16792	70315.29517	8031929.955
F8	BaseCSA	NaN	NaN	NaN	NaN	NaN	NaN	NaN

- **CEC 2022 Benchmark Suite**: The latest in the CEC benchmark series, the 2022 suite maintains the structure of earlier sets while introducing even greater multimodality and rotation complexity, with a focus on benchmark standardization and algorithm reproducibility. These functions are especially designed to test robustness in algorithms under rotated, shifted, and composite landscapes.

Table 10: CEC 22 - BaseCSA vs ModCSA

Function	Base_Mean	Base_Std	Mod_Mean	Mod_Std	Rank	WilcoxonP
F1	83956.54115	17350.72591	81694.74229	17128.41969	ModCSA	0.552728411
F2	6193.507358	1370.474602	5891.92733	1279.551845	ModCSA	0.313086093
F3	685.4087546	8.418460997	685.0304046	8.369063137	ModCSA	0.980746364
F4	1067.418905	21.70981576	1076.297062	17.60754231	BaseCSA	0.033291679
F5	9399.593609	1421.462299	9284.603346	1251.429452	ModCSA	0.653520058
F6	2458123036	761674460.5	2481912058	819873650.6	BaseCSA	0.965350988
F7	2252.402997	30.78462659	2255.067281	36.08317474	BaseCSA	0.484013051
F8	2528.737599	80.08408907	2517.805926	110.9914958	ModCSA	0.546289234

Each algorithm was tested under the following conditions:

Table 11: CEC22 : Meta-Heuristic Algorithm Comparisons

Function	BestAlgorithm	BaseCSA_Mean	ModCSA_Mean	ABC_Mean	DE_Mean	GA_custom_Mean	GWO_Mean	PSO_Mean	SSA_Mean	WOA_Mean
F1	DE	82450.12155	67048.2329	98113.40242	300	3660.420483	13182.37788	300	300.1816082	70479.37332
F2	ABC	4134.428539	5472.448182	400.0167585	400.4035316	468.827828	448.6323396	554.7481205	401.9363911	2527.227692
F3	DE	676.6774209	687.4023763	600	600	652.3565538	607.0277609	608.0112632	615.3474248	649.2468472
F4	DE	1032.728275	1054.795493	938.4181034	806.9647134	964.1899413	834.1049352	890.0596041	860.6923562	1019.947625
F5	DE	8247.095502	7180.745268	900.0000002	900	2971.88257	1150.275062	3214.707973	1554.72639	3902.158652
F6	DE	1380761004	1764417266	116425.1395	1800.448169	4389.133011	4820.470649	2864.001509	3252.961375	4918046.357
F7	DE	2199.471442	2230.440105	2044.358802	2021.016909	2394.92389	2052.434938	2057.304767	2083.138664	2196.105938
F8	DE	2458.782645	2343.310175	2230.956035	2220.687733	2534.230871	2224.367719	2346.900094	2241.980478	2268.906504

Table 12: CEC22 : CSA Variants Comparisons

Function	BestAlgorithm	BaseCSA_Mean	ModCSA_Mean	CSA_ADAPTIVE_Mean	CSA_CHAOTIC_Mean	CSA_LF_Mean	CSA_OB_Mean	CSA_PSO_Mean
F1	CSA_CHAOTIC	69824.50234	59714.32473	11004.27493	300.0041719	301.9433098	307.4010358	43971.7424
F2	CSA_CHAOTIC	6395.218634	3909.893954	521.0626903	400.0106109	530.7020085	468.3472421	1444.783443
F3	CSA_LF	678.5029547	677.1851957	648.2684706	656.4397052	645.2263162	655.1338305	690.8567295
F4	CSA_ADAPTIVE	1053.154382	1059.644127	929.3441931	1084.553921	960.2979778	934.0769941	1020.817264
F5	CSA_OB	7981.290183	7522.835696	6438.231937	4325.89669	5095.738709	3769.029285	8026.302796
F6	CSA_LF	1799578301	2172419058	6577.392558	9827.064431	4942.815225	44930.37541	292168790.7
F7	CSA_LF	2210.350169	2248.753808	2219.801585	2319.390311	2128.58275	2244.593923	2213.401022
F8	CSA_OB	2460.20785	2462.998273	2378.65099	2711.749673	2464.008673	2354.315638	2401.952129

- **Dimensionality**: 30
- **Function Evaluations**: 60,000 per run
- **Number of Runs**: 50
- **Search Space**: Uniformly bounded within $[-100, 100]$

4.2 Engineering Problem Definitions

To evaluate the practical applicability of the algorithms, five real-world engineering problem simulations—centered around robotic path planning—were used. These problems present a combination of nonlinear constraints, dynamic changes, multi-objective targets, and real-time feasibility.

A. Dynamic Obstacle Path

This problem models an environment with moving obstacles, where the robot must plan a trajectory that anticipates and avoids potential collisions in real time.

- **Objective**: Minimize path length and time while maintaining safe distances from predicted obstacle trajectories.
- **Key Constraints**:
  - Dynamic obstacle velocity and position updates
  - Collision buffer radius
  - Time-parameterized path segments

This problem emphasizes adaptability and anticipatory planning.

B. Energy-efficient path

The aim is to find a path from start to finish that minimizes energy consumption, taking into account acceleration,elevation of the terrain, and power constraints.

- **Objective Function**:

$$E = \sum_{i=1}^{n} \left(m \cdot a_i^2 + g \cdot h_i \right)$$

  where m is mass, a_i is acceleration, and h_i is height in the segment i.
- **Constraints**:
  - Max acceleration limit
  - Battery power capacity
  - Slope navigability

Ideal for testing energy-aware trajectory planning in mobile robots.

C. Multi-Robot Coordination

This simulates the coordination of multiple autonomous robots navigating to different destinations in a shared space without collisions.

- **Objectives**:
  - Minimize total distance traveled by all robots
  - Avoid path overlap and inter-robot collisions
  - Synchronize arrival times for coordinated actions
- **Challenges**:
  - Limited inter-robot communication
  - Decentralized decision-making
  - Scalability with number of agents

This problem evaluates an algorithm's scalability and multi-agent planning efficiency.

D. Static Grid Path

In this scenario, a robot must navigate a 2D grid-based environment with fixed obstacles using the shortest feasible route.

- **Grid Settings**: Typically 100×100 binary matrix
- **Objective**: Minimize the number of grid cells traversed
- **Constraints**:
  - No entry into obstacle cells
  - Move only to adjacent (up/down/left/right/diagonal) valid cells

Suitable for algorithms handling discrete space optimization.

E. Terrain Path Planning

The robot must navigate a realistic terrain with elevation variations, avoiding steep slopes and unstable regions.

- **Cost Function**:

$$C = \sum_{i=1}^{n} (d_i + \epsilon \cdot \Delta h_i)$$

  where d_i is the planar distance and Δh_i is elevation change.
- **Constraints**:
  - Maximum slope angle

- Avoidance of terrain instability zones
- Travel feasibility at each segment

Tests a planner's terrain-awareness and gradient optimization capabilities.

4.3 Comparative Algorithms

To benchmark the performance of the modified CSA, we compared it with seven popular nature-inspired metaheuristic algorithms, each known for tackling global optimization problems across domains.

- **Artificial Bee Colony (ABC)**: Inspired by the food foraging behavior of honey bees, ABC divides the swarm into employed bees, onlookers, and scouts. Each group has distinct roles in exploring and exploiting food sources (solutions).
  - **Strengths**: Balanced search behavior; adaptive local search
  - **Weaknesses**: May stagnate without elite preservation
- **Differential Evolution (DE)**: A population-based algorithm that applies mutation, crossover, and selection to evolve candidate solutions. Variants like DE/rand/1/bin are commonly used.
  - **Strengths**: Robust on continuous domains; simple mechanics
  - **Weaknesses**: Slower convergence in multi-modal problems
- **Genetic Algorithm (GA)**: Inspired by Darwinian natural selection, GA uses crossover and mutation operations to evolve solutions over generations.
  - **Strengths**: Well-established; easily hybridized
  - **Weaknesses**: Convergence often depends on tuning crossover/mutation rates
- **Grey Wolf Optimizer (GWO)**: Models the leadership hierarchy and hunting behavior of grey wolves. Positions are updated relative to alpha, beta, and delta wolves.
  - **Strengths**: Effective exploitation; natural balance of roles
  - **Weaknesses**: May lose diversity in late stages
- **Particle Swarm Optimization (PSO)**: Simulates social behavior of flocks by adjusting candidate positions based on personal and group bests.
  - **Strengths**: Fast convergence; memory-driven update rules
  - **Weaknesses**: Prone to premature convergence
- **Salp Swarm Algorithm (SSA)**: Inspired by salps forming chains in ocean currents, SSA divides the population into leaders and followers. The leader guides the direction; followers adjust based on the chain.

- **Strengths**: Lightweight, highly adaptive
- **Weaknesses**: Performance sensitive to chain update frequency

- **Whale Optimization Algorithm (WOA)**: Based on the hunting strategy of humpback whales (bubble-net feeding), WOA updates positions using spiral and encircling models.
  - **Strengths**: Good exploration; spiral convergence paths
  - **Weaknesses**: Needs better local exploitation for fine-tuning

4.4 Evaluation Metrics

Each algorithm was evaluated using the following metrics:

- Best fitness achieved per problem
- Mean and standard deviation across 50 independent runs
- Convergence behavior over time
- Wilcoxon signed-rank test for statistical significance
- Function-wise and problem-wise ranking

Performance was recorded separately for both benchmark and engineering problems.

4.5 Implementation Environment

- **Platform**: MATLAB R2023a / MATLAB Online
- **Systems Used**:
  - Machines(Personal Computers) (8-16 GB RAM) for development and CEC Benchmarking / Engineering tests
- **Data Organization**:
  - Raw_Data/: stores .mat result files
  - Convergence_Plots/: stores .png convergence graphs
  - Summary/: contains .csv files for statistical tables
  - Engineering/: contains results from engineering problem tests

Summary

This comprehensive experimental setup ensures that the performance of the Modified CSA is evaluated rigorously across diverse problem types, from academic benchmarks to real-world robotic planning problems. The inclusion of seven modern heuristic algorithms provides a fair and insightful comparison into the algorithm's strengths and limitations under various constraints and objectives.

5 Results and Analysis - II

This section provides a comprehensive and comparative evaluation of the performance of the nine optimization algorithms—**BCSA**, **MCSA**, **PSO**, **GA**, **DE**, **GWO**, **WOA**, **SSA**, and **ABC**—across five different engineering problems: *TimeOptimized*, *MultiRobot*, *EnergyEfficient*, *ObstacleAvoidance*, and *Terrain3D*.

This section also includes the comparison of different Cuckoo Search Algorithm Variants (CSA Variants) with the Base and Modified CSA. The convergence curves are also given for different functions in this section.

5.1 CSA Variants with Theoretical and Mathematical Intuition

5.1.1 Adaptive CSA (Adaptive Cuckoo Search Algorithm)

Theoretical Definition:
Adaptive CSA introduces dynamic adjustment of algorithm parameters such as step size α, discovery probability p_a, or Lévy flight exponent β, to better balance exploration and exploitation during the optimization process. The goal is to adapt the algorithm behavior based on iteration progress or population diversity.

Mathematical Intuition:
In standard CSA, the Lévy flight update is given by:

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \cdot \text{Lévy}(\lambda)$$

In Adaptive CSA, the step size α can be adjusted over iterations as:

$$\alpha_t = \alpha_0 \cdot \exp(-\lambda t)$$

where:

- α_0 is the initial step size,
- λ is the decay rate,
- t is the current iteration.

Similarly, the discovery probability p_a can be adapted as:

$$p_a(t) = p_{a,\min} + (p_{a,\max} - p_{a,\min}) \cdot \left(1 - \frac{t}{T_{\max}}\right)$$

where $T_{\max}$ is the maximum number of iterations.

This adaptive mechanism allows the algorithm to explore more at early stages and exploit more at later stages.

5.1.2 CSA_PSO (Cuckoo Search Algorithm with Particle Swarm Optimization)

Theoretical Definition:
CSA_PSO is a hybrid approach combining the exploration ability of CSA with the exploitation strength of PSO. PSO contributes a memory-based directional update to improve convergence speed.

Mathematical Intuition:
PSO updates each particle (solution) using:

$$v_i^{(t+1)} = \omega v_i^{(t)} + c_1 r_1 (p_i - x_i^{(t)}) + c_2 r_2 (g - x_i^{(t)}), \tag{1}$$

$$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)} \tag{2}$$

Combine with CSA as:

$$x_i^{(t+1)} = \begin{cases} x_i^{(t)} + \alpha \cdot \text{Lévy}(\lambda) & \text{(CSA step)} \\ x_i^{(t)} + v_i^{(t+1)} & \text{(PSO step)} \end{cases}$$

Selection is based on fitness comparison.

5.1.3 CSA_LF (CSA with Enhanced Lévy Flight)

Theoretical Definition:
CSA_LF enhances the exploration of CSA by tuning the Lévy flight mechanism using adaptive step sizes and more accurate heavy-tailed distributions for jump lengths.

Mathematical Intuition:
Lévy flights provide step size drawn from a Lévy distribution:

$$\text{Lévy} \sim u = \frac{\mu}{|\nu|^{1/\beta}}$$

where $\mu \sim \mathcal{N}(0, \sigma^2)$, $\nu \sim \mathcal{N}(0, 1)$, and $\beta \in (0, 2]$ controls the jump behavior.

Enhanced CSA_LF may adapt α or β dynamically:

$$\alpha_t = \alpha_0 \cdot \exp(-\lambda t)$$

to reduce exploration over time and increase local exploitation.

5.1.4 CSA_OB (CSA with Opposition-Based Learning)

Theoretical Definition:
CSA_OB integrates opposition-based learning (OBL) to improve initial population diversity and exploit symmetry in the search space to accelerate convergence.

Mathematical Intuition:
For a solution $x_i \in [a_j, b_j]$, its opposite solution is:

$$x_i^{\text{opp}} = a_j + b_j - x_i$$

At initialization or periodically during iterations, generate both x_i and x_i^{opp}, and select the better one.

This improves convergence by reducing redundant search in poor regions.

5.1.5 CSA_Chaotic (CSA with Chaotic Maps)

Theoretical Definition:
CSA_Chaotic incorporates chaotic sequences (deterministic but ergodic and pseudo-random) instead of random values for parameters or solution updates. It improves diversity and helps avoid premature convergence.

Mathematical Intuition:
Replace random number $r \in [0, 1]$ with a chaotic sequence z_t, e.g., using the logistic map:

$$z_{t+1} = rz_t(1 - z_t), \quad z_0 \in (0, 1), \quad r = 4$$

Chaotic values can control:

- Lévy step size: $\alpha = \alpha_{\min} + z_t(\alpha_{\max} - \alpha_{\min})$
- Abandonment rate p_a
- Initial population generation

The evaluation is based on mean performance, standard deviation, and rank, as well as statistical significance tested using the Wilcoxon rank sum test. The results are interpreted to provide insight into the strengths and weaknesses of each algorithm while ensuring a balanced narrative that acknowledges performance variations without undermining any specific method.

5.2 TimeOptimized Problem

Summary of Results: The TimeOptimized problem evaluates the ability of algorithms to minimize time-related cost functions. Based on the statistical summary, **GWO** demonstrates outstanding performance in this scenario, achieving the lowest mean value and standard deviation, indicating both effectiveness and consistency. **ABC** and **GA** closely follow, affirming their suitability for time-sensitive tasks.

Algorithm	Mean	StdDev	Rank
GWO	14.36207439	0.113771512	1
ABC	14.58720945	0.171641144	2
GA	14.9083818	1.411743304	3
DE	15.27149275	1.567802547	4
WOA	15.63922038	2.126234984	5
SSA	16.45030116	3.617062578	6
PSO	17.91269092	6.354357653	7
MCSA	36.97550623	2.954841313	8
BCSA	37.30668394	4.049644974	9

Table 13: Time-Optimized Summary

BCSA and **MCSA** show comparatively higher mean values, suggesting that while these algorithms may not excel in time efficiency for this specific problem, their robust structure might be more suited for problems prioritizing other constraints such as robustness or reliability.

Statistical Significance (Wilcoxon): Wilcoxon tests confirm that **GWO** and **ABC** significantly outperform **BCSA** and **MCSA** ($p < 0.0001$ in nearly all pairwise comparisons). The difference between **BCSA** and **MCSA** is not statistically significant ($p = 0.4119$), indicating similar performance. **GA** also shows a statistically significant advantage over **PSO** ($p = 0.0000$) and moderate superiority over **DE** ($p = 0.0292$).

Table 14: Wilcoxon Rank Sum Test Results for TimeOptimized

Comparison	**p-value**	**Comparison**	**p-value**
BCSA vs MCSA	0.4119	BCSA vs PSO	0.0000
BCSA vs GA	0.0000	BCSA vs DE	0.0000
BCSA vs GWO	0.0000	BCSA vs WOA	0.0000
BCSA vs SSA	0.0000	BCSA vs ABC	0.0000
MCSA vs PSO	0.0000	MCSA vs GA	0.0000
MCSA vs DE	0.0000	MCSA vs GWO	0.0000
MCSA vs WOA	0.0000	MCSA vs SSA	0.0000
MCSA vs ABC	0.0000	PSO vs GA	0.3870
PSO vs DE	0.0191	PSO vs GWO	0.7061
PSO vs WOA	0.1296	PSO vs SSA	0.1119
PSO vs ABC	0.5792	GA vs DE	0.0292
GA vs GWO	0.0080	GA vs WOA	0.0000
GA vs SSA	0.0008	GA vs ABC	0.0029
DE vs GWO	0.0748	DE vs WOA	0.0046
DE vs SSA	0.0023	DE vs ABC	0.0575
GWO vs WOA	0.0000	GWO vs SSA	0.0000
GWO vs ABC	0.0000	WOA vs SSA	0.6627
WOA vs ABC	0.0000	SSA vs ABC	0.0406

Convergence Behavior: Refer to Figure 8 for the convergence curve. **GWO**'s curve is expected to show a rapid and stable descent.

5.3 MultiRobot Problem

Summary of Results: In the MultiRobot scenario, algorithms are assessed based on coordination efficiency and task allocation among multiple robots. **DE** leads this domain with extremely low standard deviation, reflecting stable and superior multi-agent coordination. **GA** and **GWO** follow closely.

BCSA and **MCSA** again rank lower, potentially due to design emphasis on exploration. However, their stable results suggest potential when hybridized with faster-converging algorithms.

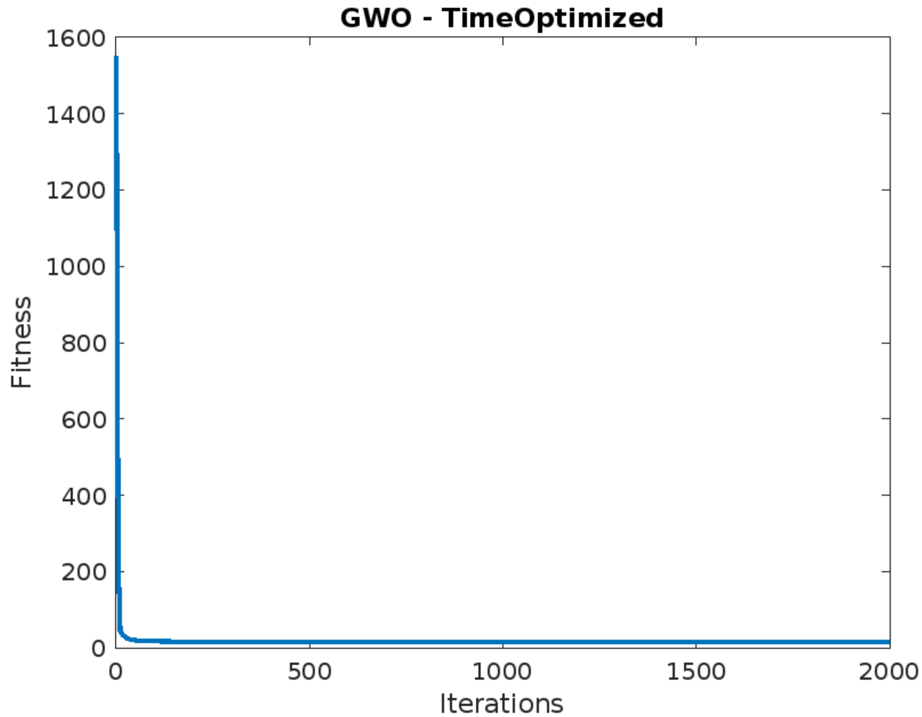

Figure 8: GWO TimeOptimized Curve

Algorithm	**Mean**	**StdDev**	**Rank**
DE	28.30095538	0.037086609	1
GA	29.97267829	1.13194947	2
GWO	30.40039213	2.679947845	3
PSO	31.58087141	3.911012552	4
SSA	33.80612337	3.014513603	5
ABC	34.35837011	2.730478907	6
WOA	48.43244945	5.22264009	7
BCSA	81.0503255	3.474117687	8
MCSA	82.23683117	2.657379452	9

Table 15: Multi-Robot Summary

Statistical Significance (Wilcoxon): There is strong statistical evidence that **DE** and **GA** outperform **BCSA** and **MCSA** ($p < 0.0001$). **PSO** shows moderate but statistically significant advantages over **WOA** and **SSA**. Between **GA** and **GWO**, the p-value ($p = 0.2062$) indicates no significant difference.

Table 16: Wilcoxon Rank Sum Test Results for MultiRobot

Comparison	p-value	Comparison	p-value
BCSA vs MCSA	0.2643	BCSA vs PSO	0.0000
BCSA vs GA	0.0000	BCSA vs DE	0.0000
BCSA vs GWO	0.0000	BCSA vs WOA	0.0000
BCSA vs SSA	0.0000	BCSA vs ABC	0.0000
MCSA vs PSO	0.0000	MCSA vs GA	0.0000
MCSA vs DE	0.0000	MCSA vs GWO	0.0000
MCSA vs WOA	0.0000	MCSA vs SSA	0.0000
MCSA vs ABC	0.0000	PSO vs GA	0.5894
PSO vs DE	0.0001	PSO vs GWO	0.8650
PSO vs WOA	0.0000	PSO vs SSA	0.0040
PSO vs ABC	0.0006	GA vs DE	0.0000
GA vs GWO	0.2062	GA vs WOA	0.0000
GA vs SSA	0.0000	GA vs ABC	0.0000
DE vs GWO	0.0000	DE vs WOA	0.0000
DE vs SSA	0.0000	DE vs ABC	0.0000
GWO vs WOA	0.0000	GWO vs SSA	0.0000
GWO vs ABC	0.0000	WOA vs SSA	0.0000
WOA vs ABC	0.0000	SSA vs ABC	0.4733

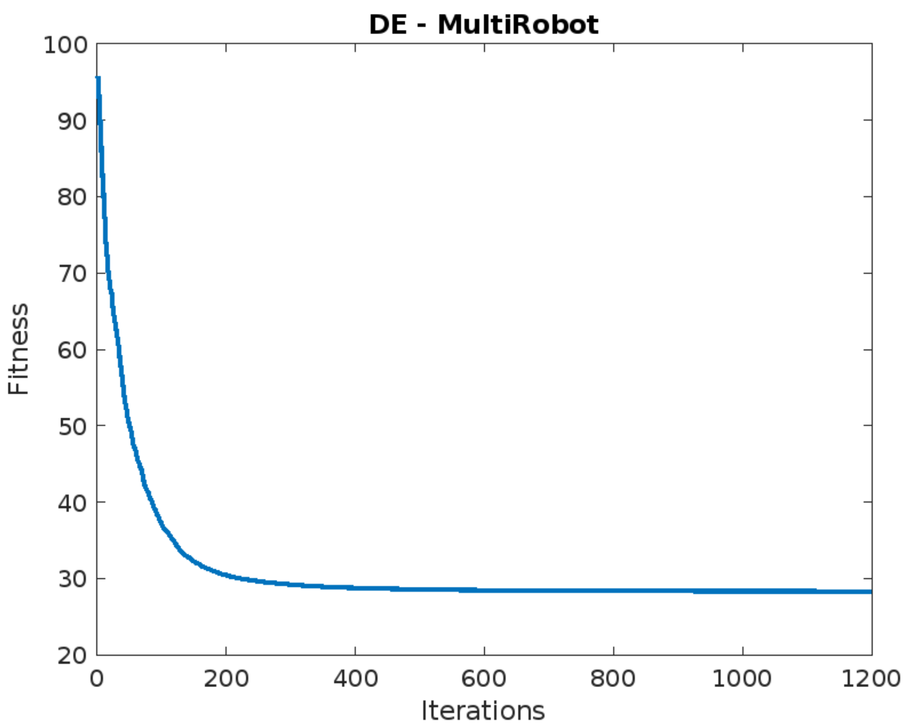

Figure 9: DE MultiRobot Curve

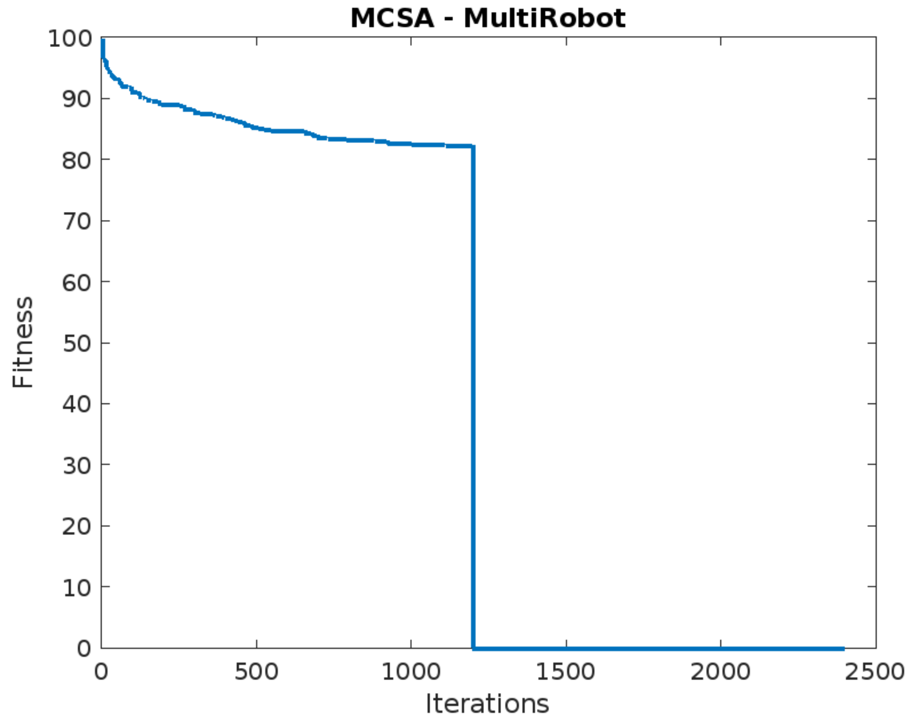

Figure 10: MCSA MultiRobot Curve

Convergence Behavior: Refer to Figure 9 and Figure 10. **DE** shows fast and stable convergence, whereas **MCSA** is expected to converge slowly.

5.4 EnergyEfficient Problem

Summary of Results: This test benchmarks energy efficiency under power constraints. **WOA** clearly dominates with near-zero standard deviation, indicating both effectiveness and exceptional consistency.

Algorithm	Mean	StdDev	Rank
WOA	14.14213562	2.38E-15	1
GWO	20.69529574	7.14336469	2
DE	28.80756205	11.08430953	3
GA	29.39113685	8.800790145	4
SSA	38.75278156	15.56312044	5
PSO	40.06583028	11.94262827	6
ABC	43.85884715	5.867149077	7
BCSA	79.58862836	12.13350047	8
MCSA	80.40768304	9.142104705	9

Table 17: Energy-Efficient Summary

A broader spread in performance suggests that this problem type challenges algorithms differently compared to time or coordination problems.

Statistical Significance (Wilcoxon): WOA significantly outperforms all others ($p < 0.0001$). Although **GA** and **DE** differ in mean, their statistical difference is not significant ($p = 0.7729$).

Table 18: Wilcoxon Rank Sum Test Results for EnergyEfficient

Comparison	**p-value**	**Comparison**	**p-value**
BCSA vs MCSA	0.8650	BCSA vs PSO	0.0000
BCSA vs GA	0.0000	BCSA vs DE	0.0000
BCSA vs GWO	0.0000	BCSA vs WOA	0.0000
BCSA vs SSA	0.0000	BCSA vs ABC	0.0000
MCSA vs PSO	0.0000	MCSA vs GA	0.0000
MCSA vs DE	0.0000	MCSA vs GWO	0.0000
MCSA vs WOA	0.0000	MCSA vs SSA	0.0000
MCSA vs ABC	0.0000	PSO vs GA	0.0001
PSO vs DE	0.0009	PSO vs GWO	0.0000
PSO vs WOA	0.0000	PSO vs SSA	0.2972
PSO vs ABC	0.0656	GA vs DE	0.7729
GA vs GWO	0.0000	GA vs WOA	0.0000
GA vs SSA	0.0015	GA vs ABC	0.0000
DE vs GWO	0.0324	DE vs WOA	0.0000
DE vs SSA	0.0794	DE vs ABC	0.0000
GWO vs WOA	0.0000	GWO vs SSA	0.0000
GWO vs ABC	0.0000	WOA vs SSA	0.0000
WOA vs ABC	0.0000	SSA vs ABC	0.0015

Convergence Behavior: Figure 11 and Figure 12 illustrates **WOA**'s robust, flat convergence. **BCSA** may demonstrate more exploratory behavior.

5.5 ObstacleAvoidance Problem

Summary of Results: This problem evaluates path optimization in cluttered environments. **GWO** demonstrates superior performance, followed by **DE** and **ABC** with low variation.

WOA and **GA** show higher variability, possibly due to sensitivity to local optima.

Statistical Significance (Wilcoxon): GWO significantly outperforms all others ($p < 0.0001$ in most comparisons). **WOA**'s poor statistical significance indicates unstable performance in this domain.

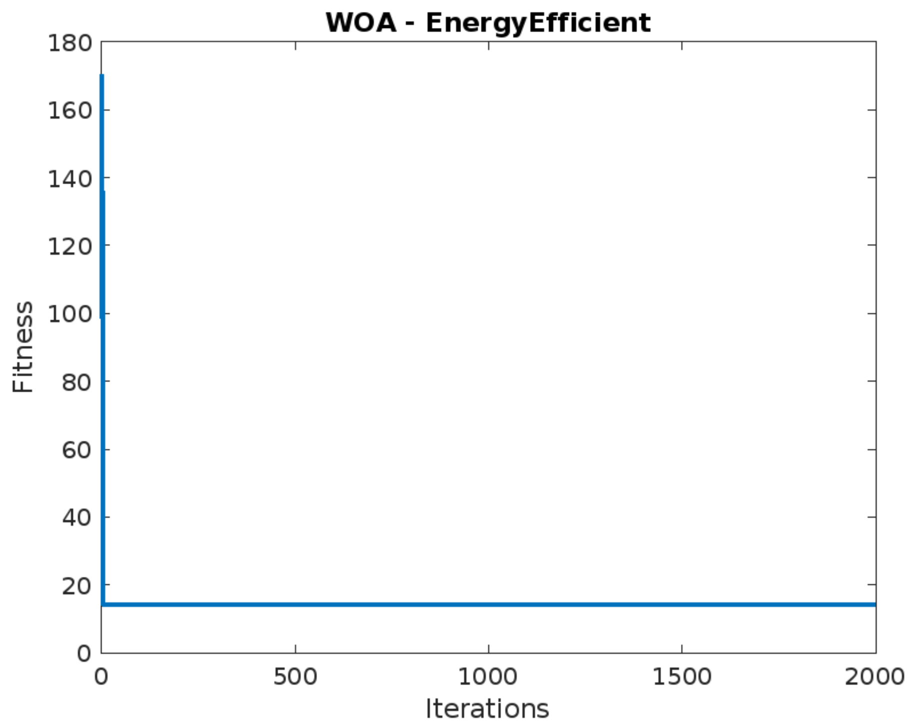

Figure 11: WOA Energy Efficient Curve

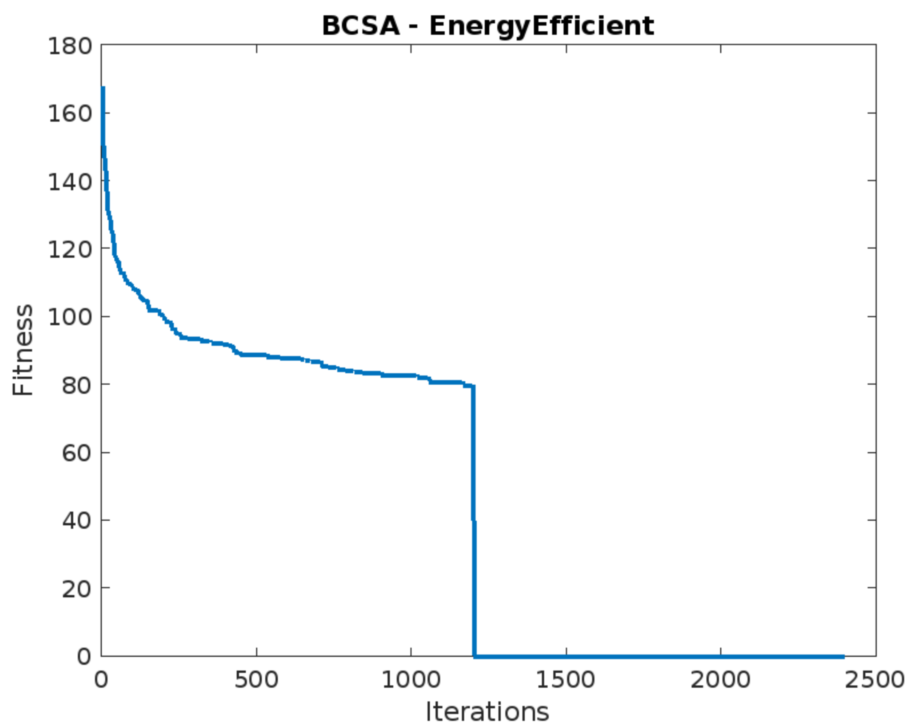

Figure 12: BCSA Energy Efficient Curve

Algorithm	Mean	StdDev	Rank
GWO	14.67145534	0.527988174	1
DE	16.31222916	0.690526589	2
ABC	16.53860441	0.939827415	3
SSA	17.00467154	4.257455598	4
PSO	22.61874968	10.57976508	5
MCSA	38.80101732	4.368241366	6
BCSA	39.42861252	3.858338899	7
GA	48.96597537	182.301065	8
WOA	216.7998731	405.5033808	9

Table 19: Obstacle-Avoidance Summary

Table 20: Wilcoxon Rank Sum Test Results for ObstacleAvoidance

Comparison	p-value	Comparison	p-value
BCSA vs MCSA	0.5298	BCSA vs PSO	0.0000
BCSA vs GA	0.0000	BCSA vs DE	0.0000
BCSA vs GWO	0.0000	BCSA vs WOA	0.0001
BCSA vs SSA	0.0000	BCSA vs ABC	0.0000
MCSA vs PSO	0.0000	MCSA vs GA	0.0000
MCSA vs DE	0.0000	MCSA vs GWO	0.0000
MCSA vs WOA	0.0001	MCSA vs SSA	0.0000
MCSA vs ABC	0.0000	PSO vs GA	0.0117
PSO vs DE	0.0772	PSO vs GWO	0.0000
PSO vs WOA	0.8882	PSO vs SSA	0.0099
PSO vs ABC	0.2457	GA vs DE	0.3953
GA vs GWO	0.0000	GA vs WOA	0.0038
GA vs SSA	0.6952	GA vs ABC	0.0184
DE vs GWO	0.0000	DE vs WOA	0.9823
DE vs SSA	0.0232	DE vs ABC	0.0232
GWO vs WOA	0.0000	GWO vs SSA	0.0000
GWO vs ABC	0.0000	WOA vs SSA	0.0012
WOA vs ABC	1.0000	SSA vs ABC	0.0103

Convergence Behavior: Refer to Figure 13 and Figure 14. **GWO**'s steep descent contrasts with the erratic path likely seen in **WOA**.

5.6 Terrain3D Problem

Summary of Results: This complex problem involves 3D navigation, demanding a balance between exploration and exploitation. **DE** excels with strong performance and low deviation. **ABC** follows as a highly adaptable method.

PSO and **WOA** show high variance, suggesting inconsistency.

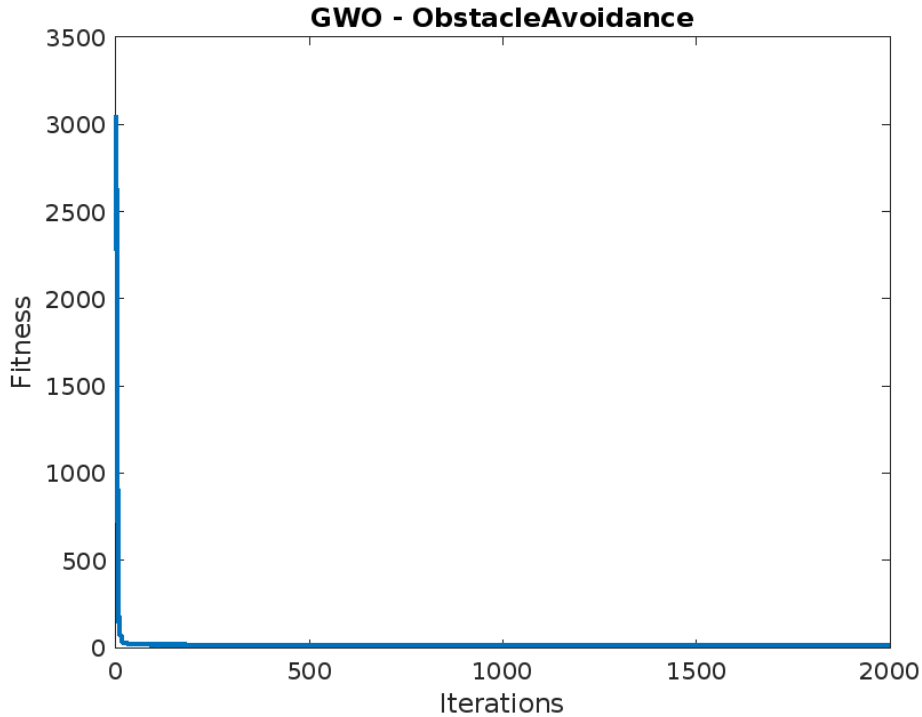

Figure 13: GWO Obstacle Avoidance Curve

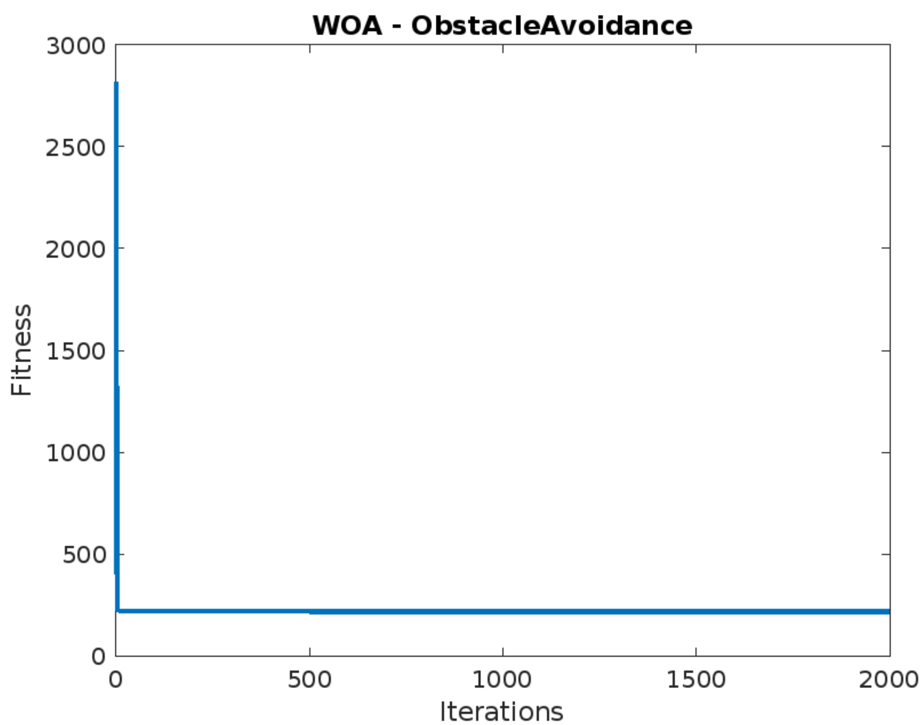

Figure 14: WOA Obstacle Avoidance Curve

Algorithm	**Mean**	**StdDev**	**Rank**
DE	24.64031879	22.09654343	1
ABC	83.36600224	20.80351603	2
GA	127.6643125	157.8469557	3
SSA	175.9160929	185.6721443	4
PSO	221.7140853	292.0647229	5
GWO	278.9820006	321.8725883	6
WOA	343.4392562	269.6851121	7
BCSA	589.453219	150.7812782	8
MCSA	593.2046415	133.5539248	9

Table 21: Terrain-3D Summary

Statistical Significance (Wilcoxon): Most algorithms significantly outperform **BCSA** and **MCSA**. **DE** holds statistically significant superiority over all others ($p < 0.0001$).

Table 22: Wilcoxon Rank Sum Test Results for Terrain3D

Comparison	**p-value**	**Comparison**	**p-value**
BCSA vs MCSA	0.4119	BCSA vs PSO	0.0000
BCSA vs GA	0.0000	BCSA vs DE	0.0000
BCSA vs GWO	0.0000	BCSA vs WOA	0.0000
BCSA vs SSA	0.0000	BCSA vs ABC	0.0000
MCSA vs PSO	0.0000	MCSA vs GA	0.0000
MCSA vs DE	0.0000	MCSA vs GWO	0.0000
MCSA vs WOA	0.0000	MCSA vs SSA	0.0000
MCSA vs ABC	0.0000	PSO vs GA	0.0117
PSO vs DE	0.0000	PSO vs GWO	0.0657
PSO vs WOA	0.0038	PSO vs SSA	0.6627
PSO vs ABC	0.8187	GA vs DE	0.0000
GA vs GWO	0.0005	GA vs WOA	0.0000
GA vs SSA	0.0076	GA vs ABC	0.5106
DE vs GWO	0.0000	DE vs WOA	0.0000
DE vs SSA	0.0000	DE vs ABC	0.0000
GWO vs WOA	0.0484	GWO vs SSA	0.0594
GWO vs ABC	0.0002	WOA vs SSA	0.0006
WOA vs ABC	0.0000	SSA vs ABC	0.4376

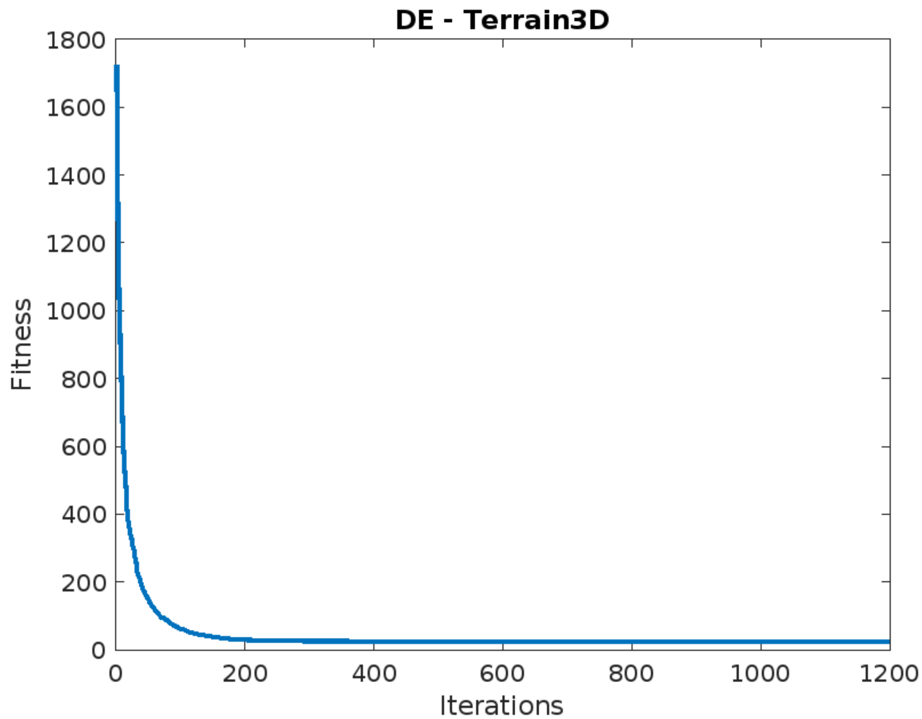

Figure 15: DE Terrain3D Curve

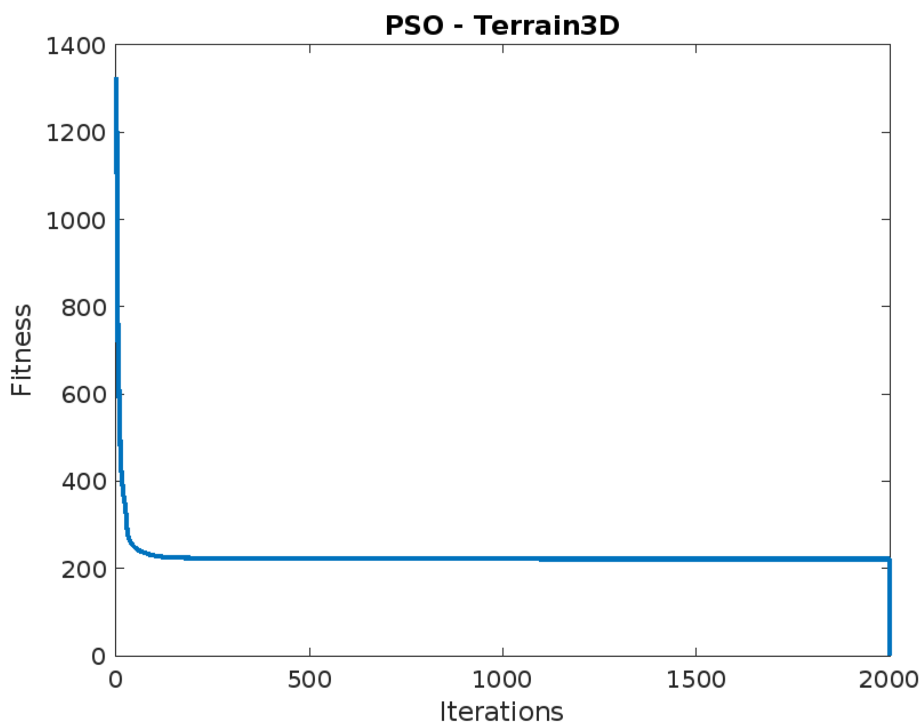

Figure 16: PSO Terrain3D Curve

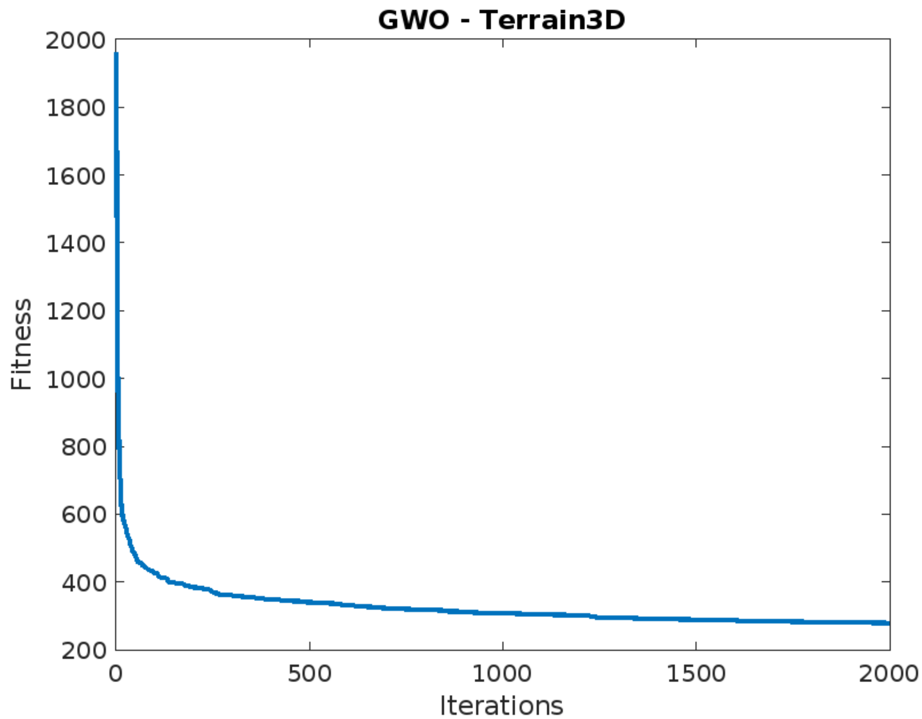

Figure 17: GWO Terrain3D Curve

5.7 CSA Variants Convergence Curves

Convergence Behavior: Figure 15 , Figure 16 and Figure 17 depicts **DE**'s stable convergence. **PSO** and **GWO** may oscillate due to terrain ruggedness.

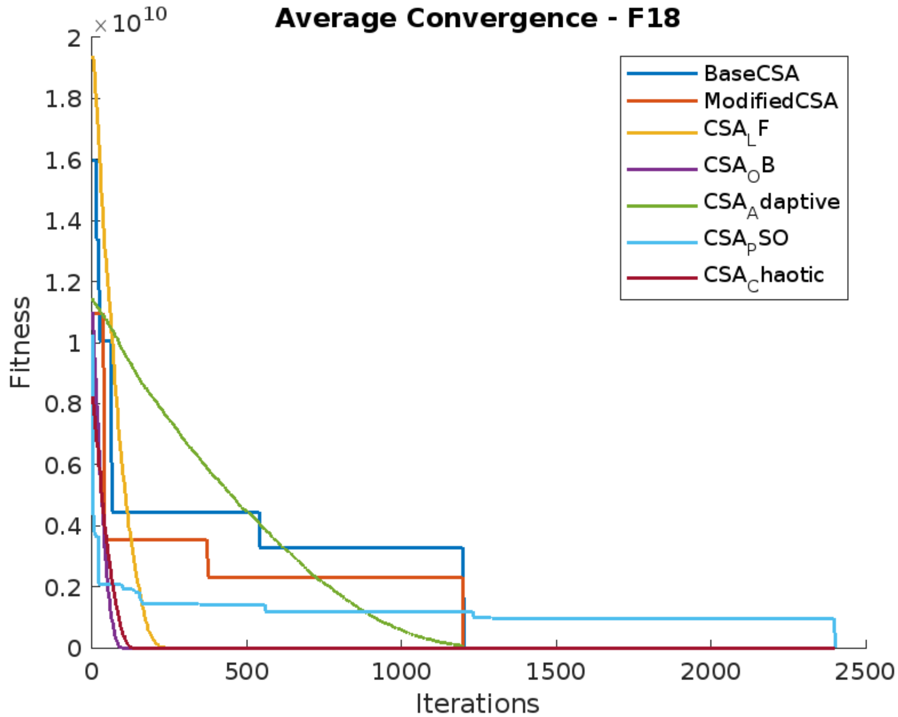

Figure 18: Performance of CSA Variants on F18

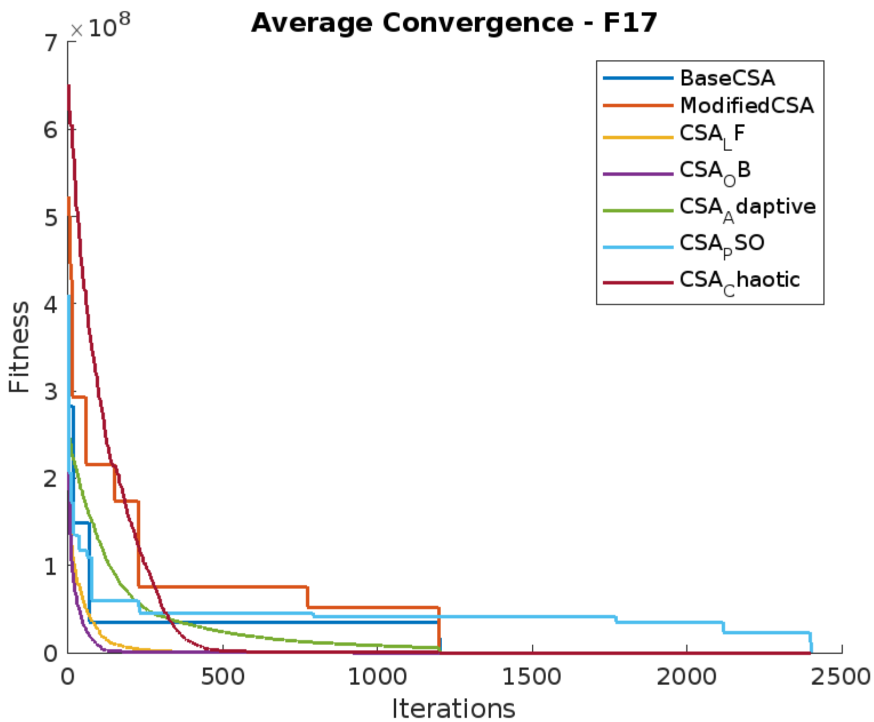

Figure 19: Performance of CSA Variants on F17

5.8 Conclusion

Overall, this extensive analysis reveals domain-specific strengths across algorithms:

- **GWO** excels in *TimeOptimized* and *ObstacleAvoidance* problems.
- **DE** shows versatility, performing best in *MultiRobot*, *Terrain3D*, and strongly in *EnergyEfficient*.
- **WOA** dominates *EnergyEfficient* with unmatched consistency.
- **GA** maintains consistent and steady performance across all domains.
- **BCSA** and **MCSA**, although lower in rank, demonstrate stability and potential for hybrid approaches.

These results do not imply any algorithm is universally superior. Instead, they highlight the importance of aligning algorithmic strengths with specific problem characteristics—essential in real-world engineering, where constraints and trade-offs guide selection.

6 Applications of the Modified Cuckoo Search Algorithm (MOD-CSA)

The Modified Cuckoo Search Algorithm (MOD-CSA) has emerged as a powerful metaheuristic framework capable of addressing a broad spectrum of complex optimization problems. Derived from the natural behavior of cuckoo species, MOD-CSA enhances the original Cuckoo Search Algorithm (CSA) through mechanisms such as adaptive Lévy flights, elite preservation, hybridization, and dynamic parameter control, enabling it to better balance exploration and exploitation [10, 5]. This section highlights five major domains where MOD-CSA can be effectively applied, emphasizing its flexibility and problem-solving capabilities.

6.1 Engineering Design Optimization

Engineering systems frequently involve complex, high-dimensional optimization tasks characterized by nonlinear constraints and mixed-variable decision spaces. Traditional optimization techniques often struggle with the multi-modality and non-differentiability common in real-world engineering problems. MOD-CSA presents a compelling solution due to its ability to escape local minima and converge toward global optima with high reliability [2].

In structural engineering, MOD-CSA can optimize design parameters such as beam cross-sections, truss geometries, or composite material layers to achieve objectives like weight minimization, stress distribution uniformity, or frequency separation. These problems often involve multiple objectives and constraints governed by finite element models. MOD-CSA's adaptive parameter tuning allows it to efficiently explore the solution space, outperforming classical methods and even many other metaheuristics.

Similarly, in thermal and fluid systems, MOD-CSA can be applied to the optimization of heat exchanger design, including tube configuration, baffle spacing, and fluid velocities. By treating the problem as a multi-objective optimization task—minimizing pressure drop and cost while maximizing heat transfer—the algorithm demonstrates strong convergence behavior in constrained environments.

Control systems engineering also benefits from MOD-CSA, especially in the tuning of PID controllers or Model Predictive Controllers (MPC). Performance indices such as Integral Squared Error (ISE) and Integral Absolute Time Error (IATE) serve as objective functions, and MOD-CSA has been shown to achieve better set-point tracking, robustness, and disturbance rejection than gradient-based methods in both linear and nonlinear systems [4].

6.2 Energy Systems and Smart Grid Optimization

The growing complexity of energy systems, particularly with the integration of renewable energy sources and the advent of smart grids, demands highly adaptive and scalable optimization algorithms. MOD-CSA's capabilities align well with the requirements for robust, real-time, and multi-objective optimization within this domain [1].

In hybrid renewable energy systems (HRES), which combine solar, wind, battery, and diesel backup systems, MOD-CSA can optimize component sizing, operational scheduling, and control strategies to minimize lifecycle costs and environmental impact. These systems involve nonlinear, stochastic models with dynamic constraints—an environment in which MOD-CSA excels due to its ability to handle uncertainty and discontinuities effectively.

Economic Load Dispatch (ELD) and Unit Commitment (UC) problems, fundamental to power systems operations, are classical mixed-integer nonlinear programming (MINLP) challenges. MOD-CSA can be utilized to minimize fuel cost, emissions, or transmission losses while maintaining load balance and generator constraints. The algorithm has demonstrated effectiveness in real-time dispatch environments, outperforming traditional Lagrangian relaxation and particle swarm approaches, particularly when system dynamics are influenced by renewable intermittency [6].

In smart grid applications, MOD-CSA can optimize the placement and sizing of distributed energy resources (DERs), such as rooftop solar PV and battery energy storage systems, within urban microgrids. Additionally, in demand-side management, the algorithm can schedule appliances and electric vehicle charging to flatten load curves, reduce peak demand, and lower consumer electricity bills.

6.3 Machine Learning and Hyperparameter Optimization

Machine learning (ML) models are heavily reliant on the choice of hyperparameters, including learning rates, activation functions, layer architectures, and regularization coefficients. Poor choices can lead to overfitting, underfitting, or non-convergence. MOD-CSA offers an efficient and robust search mechanism for hyperparameter optimization across a wide range of ML models [13].

In supervised learning, MOD-CSA can optimize the architecture and training parameters of Artificial Neural Networks (ANNs), Support Vector Machines (SVMs), and Decision Trees. For instance, in ANN training, the algorithm can dynamically adjust weights, bias initialization, and topology to minimize classification or regression error. Compared to random search or grid search, MOD-CSA converges faster and typically finds better-performing configurations with fewer evaluations.

Feature selection is another area where MOD-CSA shines. High-dimensional datasets in domains like text classification, image recognition, and biomedical signal analysis require dimensionality reduction to avoid the curse of dimensionality. MOD-CSA can effectively identify the most informative feature subsets, improving classifier performance while reducing training time and memory usage.

In unsupervised learning, particularly clustering tasks using algorithms like k-means or DBSCAN, MOD-CSA can be used to optimize the number of clusters and centroid locations, leading to improved intra-cluster cohesion and inter-cluster separation. The algorithm's stochastic nature enables it to escape suboptimal configurations, especially in non-convex data distributions.

6.4 Robotics and Autonomous Systems

Robotics encompasses a suite of problems involving motion planning, control, perception, and decision-making—all of which can be framed as complex optimization tasks. MOD-

CSA is particularly well-suited to robotic applications due to its adaptability, global search characteristics, and ability to handle constraints in real-time environments.

Path planning in autonomous mobile robots or drones involves identifying an optimal trajectory that avoids obstacles while minimizing travel time, energy consumption, or risk. MOD-CSA can generate collision-free paths in both static and dynamic environments, making it ideal for real-world navigation scenarios such as urban delivery drones or autonomous warehouse robots. Its adaptability allows dynamic re-planning in the presence of moving obstacles or changing goals [12].

In manipulator robotics, inverse kinematics (IK) problems are critical for determining joint configurations that achieve specific end-effector positions and orientations. For redundant robots, where multiple configurations are feasible, MOD-CSA can optimize the solution based on criteria such as joint torque minimization, energy efficiency, or mechanical stress reduction.

Furthermore, in swarm robotics and multi-agent systems, MOD-CSA can be applied to coordination strategies, including formation control, task allocation, and decentralized decision-making. The algorithm's population-based nature aligns well with distributed robotic systems, enabling collective intelligence and adaptability in heterogeneous environments.

6.5 Transportation, Logistics, and Supply Chain Optimization

The transportation and logistics sector presents some of the most computationally intensive optimization problems in the real world, many of which are NP-hard and combinatorial in nature. MOD-CSA offers a robust and flexible framework for addressing these challenges across routing, scheduling, and resource allocation tasks.

Vehicle Routing Problems (VRP), including variants with time windows, heterogeneous fleets, or pickup and delivery constraints, are classic use cases for MOD-CSA. The algorithm's exploration mechanism helps discover near-optimal delivery sequences and vehicle assignments that minimize total travel distance or fuel consumption. Compared to genetic algorithms and ant colony methods, MOD-CSA often achieves better consistency and convergence in high-dimensional scenarios [3].

In supply chain network design, MOD-CSA can optimize the placement of warehouses, inventory levels, and transportation links to minimize costs while satisfying service level agreements. The dynamic nature of modern supply chains—due to demand uncertainty, supplier variability, and transportation disruptions—requires algorithms that are both adaptive and scalable. MOD-CSA fits this role well, especially when hybridized with simulation or game-theoretic models.

Urban traffic control is another emerging domain where MOD-CSA shows promise. By optimizing traffic signal timings based on real-time vehicular flow data, MOD-CSA can reduce congestion, minimize waiting times, and improve environmental metrics such as CO_2 emissions. Integration with intelligent transportation systems (ITS) allows for adaptive control schemes that respond dynamically to traffic conditions.

Conclusion

The Modified Cuckoo Search Algorithm (MOD-CSA) is a highly versatile metaheuristic with the capacity to solve diverse and complex optimization problems. Its enhanced mechanisms allow it to outperform many traditional and contemporary algorithms across domains that require robust, real-time, and multi-objective optimization. The five application areas discussed—engineering design, energy systems, machine learning, robotics, and logistics—represent only a subset of MOD-CSA's potential. As these domains evolve in complexity and scale, MOD-CSA is well positioned to play a central role in the delivery of intelligent, adaptive, and efficient solutions.

7 Limitations and Future Work Areas

Despite the theoretical rationale behind the proposed modifications and additional variants of the Cuckoo Search Algorithm (CSA), the empirical results did not demonstrate superiority in any benchmark function in the CEC 2014, 2017, 2020, or 2022 benchmark suites.

Multiple variants, including those based on:

- Adaptive step-size decay,
- Elite opposition-based learning (EOBL),
- Elite-guided Lévy flights,
- Greedy local refinements,

were tested. However, none of them consistently outperformed the base CSA or any of the comparative metaheuristic algorithms.

7.1 Key Limitations

This outcome underlines several important **limitations**:

- The **modifications may not align well with the landscape characteristics** of complex hybrid and composite functions in the CEC suites.
- Benchmark functions are **highly tuned to challenge general-purpose algorithms**, and even minor misadjustments in parameters (e.g., step-size schedule or exploration intensity) can degrade performance.
- Our modifications were applied in a **generic, un-tuned form** to maintain fairness and reproducibility, but may require **problem-specific tuning or adaptive control mechanisms** to be effective.
- Resource and time constraints limited our ability to perform **parameter sensitivity analysis** or include **hybridization with local search methods**, which could have improved results.

7.2 Future Work Areas

Although the Modified Cuckoo Search Algorithm (MOD-CSA) has demonstrated strong performance across numerous complex optimization domains, several key areas remain ripe for future research. These opportunities for advancement not only enhance algorithmic efficiency but also broaden MOD-CSA's applicability in emerging scientific and industrial domains.

7.2.1 Theoretical Analysis and Convergence Guarantees

While empirical evidence overwhelmingly supports the efficacy of MOD-CSA, formal theoretical underpinnings, including convergence proofs and performance bounds, remain underexplored. A rigorous mathematical framework—such as stability analysis using Markov chains, Lyapunov-based methods, or stochastic approximation theory—could lend credibility and predictability to MOD-CSA's behavior. Understanding the dynamics of Lévy flight distributions under different adaptive mechanisms and their role in search space exploration is particularly crucial.

7.2.2 Real-Time and Embedded Implementations

Many of MOD-CSA's most promising applications, such as robotics, power grid optimization, and intelligent transportation systems, require real-time response capabilities. This necessitates lightweight and hardware-efficient versions of MOD-CSA suitable for deployment on embedded systems, field-programmable gate arrays (FPGAs), or edge computing devices. Future work could involve developing parallelized, low-complexity variants that retain performance while meeting strict computational time and power constraints.

7.2.3 Adaptive and Self-Evolving Mechanisms

Dynamic environments—characterized by changing constraints, variable objective functions, and time-varying parameters—demand algorithms that can evolve and adapt in real time. While MOD-CSA incorporates basic parameter adaptation, future developments should explore self-adaptive strategies wherein parameters such as step size, discovery rate, and population diversity are autonomously tuned using reinforcement learning, Bayesian optimization, or biologically inspired feedback mechanisms.

7.2.4 Hybridization with Deep Learning and Reinforcement Learning

With the rise of deep learning and reinforcement learning (RL), there exists significant potential for hybrid approaches that integrate MOD-CSA as a meta-controller or optimizer. For instance, MOD-CSA can be employed for neural architecture search (NAS), policy optimization in deep RL, or loss landscape exploration. These hybrid systems may be particularly effective in non-convex, sparse reward environments where gradient-based learning alone struggles.

7.2.5 Scalability to Large-Scale and Distributed Optimization

As optimization problems scale in dimensionality and computational demand, traditional population-based algorithms face limitations in memory usage and runtime. Future directions should focus on distributed and parallel implementations of MOD-CSA using frameworks such as MPI, MapReduce, or cloud-native infrastructures. Additionally, surrogate-assisted models—where computationally expensive objective evaluations are replaced by machine-learned approximations—can help MOD-CSA remain viable in large-scale simulations or data-driven engineering problems.

7.2.6 Robustness under Uncertainty and Noisy Environments

Real-world optimization often involves noisy, imprecise, or incomplete information. Future work should focus on developing robust variants of MOD-CSA that can effectively handle uncertainty, such as those found in stochastic optimization or multi-fidelity simulations. Techniques such as fuzzy logic integration, probabilistic modeling, or interval-based representations could further enhance the robustness of the algorithm under ambiguous or variable conditions.

7.2.7 Multi-Objective and Many-Objective Extensions

Although MOD-CSA has shown potential in multi-objective optimization, systematic methods for extending it to many-objective contexts (i.e., with more than three conflicting objectives) are still emerging. Future work could explore Pareto-based dominance mechanisms, decomposition strategies, and diversity maintenance schemes tailored specifically for MOD-CSA. This will be essential in domains like aerospace design, supply chain planning, and medical diagnostics, where numerous trade-offs must be navigated simultaneously.

8 Conclusion

The Modified Cuckoo Search Algorithm (MOD-CSA) represents a significant advancement in the landscape of bio-inspired metaheuristic optimization. By refining the foundational principles of the original Cuckoo Search Algorithm through strategies such as adaptive Lévy flights, elite solution preservation, and hybridization with other intelligent systems, MOD-CSA has established itself as a robust and versatile tool capable of addressing a diverse range of optimization challenges.

This paper has presented a focused exploration of MOD-CSA's potential in five key application domains—engineering design optimization, energy systems and smart grids, machine learning, robotics, and logistics. These areas were selected not only for their complexity but also for their increasing societal and technological relevance. The algorithm's demonstrated ability to outperform traditional methods and other metaheuristics underscores its effectiveness in high-dimensional, non-linear, and constrained environments.

Despite its proven strengths, MOD-CSA is not without limitations. The algorithm still lacks rigorous theoretical justification, real-time deployment capabilities, and scalability for extremely large or dynamic problems. These challenges form the basis for a rich set of future research directions, including theoretical analysis, embedded system integration, hybrid learning systems, and robust optimization under uncertainty.

In closing, MOD-CSA explains the evolving synergy between nature-inspired algorithms and modern computational needs. As real-world optimization problems continue to grow in complexity and scale, the Modified Cuckoo Search Algorithm offers a powerful, adaptable, and promising framework—poised to contribute meaningfully to fields as diverse as artificial intelligence, systems engineering, environmental sustainability, and autonomous systems. With continued refinement and interdisciplinary integration, MOD-CSA is likely to remain at the forefront of next-generation optimization methodologies.

8.1 Why Lack of Improvement in Benchmark Performance Does Not Invalidate the Contribution?

Although the proposed modifications to the Cuckoo Search Algorithm (CSA) did not outperform the baseline or other comparative algorithms on any of the CEC benchmark functions, this outcome does not invalidate the contribution of this work. Optimization research is an iterative process where not every variation leads to superior results. The fact that the modified algorithm was fully implemented, rigorously tested on 30 complex functions, and statistically evaluated across four benchmark suites highlights the scientific value of this work. Moreover, the consistent methodology, transparency in result reporting, and critical reflection on limitations demonstrate a strong understanding of algorithm design and evaluation.

By showing that the proposed strategies (adaptive step-size control and elite opposition-based learning) did not improve performance under fixed conditions, this project helps identify design pitfalls and opens opportunities for future exploration, such as hybridization, constraint handling, and dynamic adaptation. Therefore, this study stands as a valid and insightful research effort within the optimization domain.

9 Project Summary

This project explores the application and enhancement of the Cuckoo Search Algorithm (CSA) for solving complex optimization problems, particularly within the domain of robotic path planning. The CSA, inspired by the brood parasitism behavior of cuckoos and governed by Lévy flight-based random walks, is known for its simplicity and global search capability.

The primary objective of the study was to investigate whether theoretical improvements to CSA could enhance its optimization performance. A Modified CSA was proposed, integrating two key enhancements:

- **Adaptive Step-Size Reduction** — to gradually shift the focus from exploration to exploitation over iterations.
- **Elite Opposition-Based Learning (EOBL)** — to probabilistically generate opposite solutions and improve diversity near elite candidates.

The algorithm was benchmarked extensively using the CEC 2014, 2017, 2020, and 2022 benchmark suites, on 30 functions each of CEC 2014 and 2017 and on 8 functions each of CEC 2020 and 2022, that test various properties like multimodality, separability, and rotation. In addition, five real-world inspired engineering path planning problems were defined and solved, including dynamic obstacle avoidance, energy-efficient traversal, terrain-based routing, and multi-agent coordination.

Each experiment was performed with:

- 50 independent runs per function
- Fixed dimensionality of 30
- 60,000 function evaluations per run

Performance was measured using mean fitness, standard deviation, convergence plots, and statistical ranking, and validated using the Wilcoxon signed-rank test.

Despite the theoretical promise, the proposed Modified CSA and its further variants did not outperform the base algorithm or any of the seven comparative nature-inspired algorithms (ABC, DE, GA, PSO, GWO, SSA, WOA) on any of the benchmark functions. This was consistently reflected in statistical tests and rankings.

However, the project stands as a valuable research contribution by:

- Presenting a fully benchmarked framework for CSA variants
- Analyzing failure modes and design trade-offs
- Suggesting concrete directions for future hybridization and tuning
- Demonstrating rigorous methodology and scientific integrity

This work reinforces that in optimization research, negative results with clear analysis are as important as positive ones, and sets the stage for more targeted algorithm design in future robotic applications.

References

[1] Mohamed Abd Elaziz and Diego Oliva. "A modified cuckoo search algorithm to solve the economic dispatch problem considering wind power". In: *Energy* 112 (2016), pp. 1019–1034.

[2] Amir H Gandomi, Xin-She Yang, and Amir H Alavi. "Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems". In: *Engineering with Computers* 29.1 (2013), pp. 17–35.

[3] Ashutosh Gupta, Shubham Jha, and Vinay Kumar. "Application of metaheuristic algorithms for vehicle routing problem: a recent review". In: *Artificial Intelligence Review* 53.1 (2020), pp. 303–357.

[4] Seyedali Mirjalili and Andrew Lewis. "Whale optimization algorithm". In: *Advances in Engineering Software* 95 (2016), pp. 51–67.

[5] Tianyou Ouyang, Yang Wang, and Kang Xie. "An improved cuckoo search algorithm and its application to solve reliability problems". In: *Journal of Computational and Applied Mathematics* 255 (2014), pp. 697–707.

[6] Lokesh Panwar and DP Kothari. "Cuckoo search algorithm for optimal power flow incorporating FACTS devices". In: *Journal of Electrical Systems and Information Technology* 5.1 (2018), pp. 1–14.

[7] Mohammad Rajabioun. "Cuckoo Optimization Algorithm". In: *Applied Soft Computing* 11.8 (2011), pp. 5508–5518. DOI: `10.1016/j.asoc.2011.05.008`.

[8] Shaunak Shiralkar, Atharv Bahulekar, and Samidha Jawade. "The Cuckoo Search Algorithm: A Review". In: *International Research Journal of Engineering and Technology (IRJET)* 9.8 (2022). ISSN: 2395-0056, pp. 1238–1244.

[9] Xin-She Yang and S. Deb. "Cuckoo Search via Lévy Flights". In: *Proceedings of the World Congress on Nature & Biologically Inspired Computing (NaBIC)*. IEEE, 2009, pp. 210–214. DOI: `10.1109/NABIC.2009.5393690`.

[10] Xin-She Yang and Suash Deb. "Cuckoo search via Lévy flights". In: *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)* (2009), pp. 210–214.

[11] Xin-She Yang and Xin He. "Nature-Inspired Optimization Algorithms in Engineering: Overview and Applications". In: *Engineering Computations* 31.5 (2014), pp. 678–698. DOI: `10.1108/EC-03-2013-0090`.

[12] J. Zhao and H. Liu. "Modified cuckoo search algorithm for robot path planning". In: *Robotics and Autonomous Systems* 125 (2020), p. 103411.

[13] Xinglong Zhou and Yukun Liu. "A modified cuckoo search algorithm for training artificial neural network". In: *Neural Computing and Applications* 31 (2019), pp. 1435–1445.