Summary of Implemented Bio-Inspired Optimization Algorithms

Páll Rúnarsson

May 7, 2025

Contents

Abstract

This document presents the working mechanisms of various bio-inspired optimization algorithms, providing detailed pseudocode for each. In the Optimization algorithm results section it includes benchmarking results for ten algorithms evaluated on optimization problems in 10, 20, and 30 dimensions, using 30 search agents over 100 iterations.

Introduction

This document provides a comprehensive summary of bio-inspired optimization algorithms implemented in the `AlgorithmMainScript.m` framework, developed by Páll Rúnarsson. These algorithms are inspired by natural systems and behaviors, leveraging principles such as swarm intelligence, evolutionary dynamics, and ecological interactions to address complex continuous and combinatorial optimization problems. For each algorithm, the document presents its biological inspiration, core mechanisms, detailed pseudocode, and typical application areas. Furthermore, the performance of ten selected algorithms is benchmarked on optimization problems in 10, 20, and 30 dimensions, using 30 search agents and 100 iterations. This unified presentation offers a valuable reference for comparison and analysis across a broad range of optimization tasks.

Script Name

`AlgorithmMainScript.m`

Author

Páll Rúnarsson

1 Bio-Inspired Optimization Algorithms

1.1 Grey Wolf Optimizer (GWO)

Description

The Grey Wolf Optimizer (GWO), proposed by Mirjalili et al. [10], is a nature-inspired metaheuristic algorithm modeled on the social hierarchy and hunting behavior of grey wolves (*Canis lupus*). GWO belongs to the class of swarm intelligence algorithms and is widely used for solving continuous optimization problems.

Main Idea

GWO mimics the leadership structure of a grey wolf pack, where:

- **Alpha** (α): best solution so far
- **Beta** (β): second-best solution
- **Delta** (δ): third-best solution
- **Omega** (ω): remaining search agents

The α, β, and δ guide the optimization, while ω wolves update their positions based on the leaders.

Hunting Phases

The GWO models three main phases:

1. **Encircling prey**: Wolves estimate and surround the prey (best solution) using:

$$\vec{D} = |\vec{C} \cdot \vec{X}_p - \vec{X}| \quad , \quad \vec{X}(t+1) = \vec{X}_p - \vec{A} \cdot \vec{D}$$

   where $\vec{A}$ and $\vec{C}$ are coefficient vectors, $\vec{X}_p$ is prey position, and $\vec{X}$ is the wolf's position.

2. **Hunting**: α, β, and δ share hunting knowledge; ω wolves update positions relative to these top wolves.

3. **Attacking prey (exploitation)**: As iterations progress, the control parameter a decreases linearly, reducing $|\vec{A}|$ and shifting from exploration to exploitation.

Exploration vs. Exploitation

When $|\vec{A}| > 1$, wolves explore the search space; when $|\vec{A}| < 1$, they exploit around the best solutions. This balance allows GWO to avoid local minima and converge efficiently.

Algorithm Outline

1. Initialize the grey wolf population randomly.
2. Evaluate fitness and assign α, β, δ.
3. Update positions using the three leaders.
4. Decrease a linearly and repeat until stopping criterion.

Applications

GWO has shown competitive performance on benchmark functions, engineering design problems, and real-world applications (e.g., spring design, beam design, pressure vessel design).

Pseudocode

1. Initialize a population of grey wolves X_i $(i = 1, 2, \ldots, n)$.
2. Evaluate the fitness $f(X_i)$ and identify the top three solutions:
   - α: best solution
   - β: second-best solution
   - δ: third-best solution
3. **Repeat until stopping criterion is met:**
   (a) For each wolf i:
   - Update coefficients:

$$\vec{A} = 2\vec{a} \cdot \vec{r} - \vec{a} \quad , \quad \vec{C} = 2 \cdot \vec{r}$$

     where $\vec{a}$ decreases linearly from 2 to 0; $\vec{r}$ is a random vector in $[0, 1]$.
   - Calculate distance vectors:

$$\vec{D}_\alpha = |\vec{C}_\alpha \cdot \vec{X}_\alpha - \vec{X}_i|, \quad \vec{D}_\beta = |\vec{C}_\beta \cdot \vec{X}_\beta - \vec{X}_i|, \quad \vec{D}_\delta = |\vec{C}_\delta \cdot \vec{X}_\delta - \vec{X}_i|$$

   - Update positions relative to leaders:

$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}_\alpha \cdot \vec{D}_\alpha, \quad \vec{X}_2 = \vec{X}_\beta - \vec{A}_\beta \cdot \vec{D}_\beta, \quad \vec{X}_3 = \vec{X}_\delta - \vec{A}_\delta \cdot \vec{D}_\delta$$

$$\vec{X}_i(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3}$$

(b) Evaluate the new fitness $f(X_i)$.
(c) Update α, β, δ if better solutions are found.

4. Return the best solution α.

1.2 Cuckoo Search

Description

The Cuckoo Search (CS) algorithm, proposed by Yang and Deb [18], is a nature-inspired metaheuristic that models how some cuckoo birds lay their eggs in the nests of other species. The algorithm also incorporates Lévy flights for random walk exploration, making it efficient for continuous optimization problems.

Main Idea

CS uses the following key mechanisms:

- **Cuckoo eggs (solutions):** Represent candidate solutions.
- **Host nests:** A population of solutions, some of which will be replaced.
- **Discovery rate (p_a):** Probability that a host discovers and abandons a cuckoo egg.

Cuckoos lay eggs in random nests; better solutions survive, while a fraction of nests are abandoned and rebuilt at each iteration.

Core Phases

Cuckoo Search consists of three main components:

1. **Laying eggs via Lévy flights:**
$$X_{\text{new}} = X_i + \alpha \cdot \text{Lévy}(\lambda)$$
where X_i is a current solution, α is the step size, and Lévy flights generate long jumps to explore the space.
2. **Host selection and replacement:** A randomly chosen nest is compared with the new solution; the better one is kept.
3. **Abandonment and randomization:** A fraction p_a of the worst nests are abandoned and replaced by new random solutions, introducing diversity.

Exploration vs. Exploitation

Lévy flights provide long jumps that enhance global exploration, while elitism (keeping the best solutions) drives local exploitation. The balance between random jumps and focused search makes CS effective at escaping local optima.

Algorithm Outline

1. Initialize a population of nests with random solutions.
2. Evaluate the fitness of each nest.
3. Repeat until stopping criterion:
   (a) Generate new solutions via Lévy flights.
   (b) Select random nests for potential replacement.
   (c) Abandon a fraction p_a of the worst nests and introduce new random solutions.
   (d) Update and track the best solution.
4. Return the best solution found.

Applications

Cuckoo Search has been applied successfully in feature selection, engineering design, scheduling, machine learning, and combinatorial optimization.

Pseudocode

1. Initialize a population of n host nests X_i $(i = 1, 2, ..., n)$.
2. Define the discovery rate $p_a \in [0, 1]$ (typically $p_a = 0.25$).
3. **Repeat until stopping criterion is met:**
   (a) Generate a new solution X_{new} for a cuckoo by Lévy flight:
   $$X_{\text{new}} = X_i + \alpha \cdot \text{Lévy}(\lambda)$$
   where α is the step size.
   (b) Randomly choose a nest j from the population.
   (c) **If** $f(X_{\text{new}}) < f(X_j)$, **then** replace X_j with X_{new}.
   (d) Abandon a fraction p_a of the worst nests and replace them with new random solutions.
   (e) Keep the current best solution (elitism).
4. Rank the nests and return the best solution found.

1.3 Particle Swarm Optimization (PSO)

Description

Particle Swarm Optimization (PSO), introduced by Kennedy and Eberhart [7], is a population-based metaheuristic where particles (solutions) fly through the search space, sharing information to find optima. PSO is one of the most widely used swarm intelligence algorithms.

Main Idea

PSO models the collective behavior of particles:

- **Particles:** Candidate solutions with positions and velocities.
- **Personal best (P_i):** Best position a particle has visited.
- **Global best (G):** Best position found by the entire swarm.

Particles adjust their movement by considering both their own experience and that of their neighbors.

Core Phases

The PSO algorithm consists of:

1. **Velocity update:**
   $$V_i \leftarrow wV_i + c_1 r_1 (P_i - X_i) + c_2 r_2 (G - X_i)$$
   where w is the inertia weight, c_1 and c_2 are cognitive and social coefficients, and r_1, r_2 are random numbers.
2. **Position update:**
   $$X_i \leftarrow X_i + V_i$$
3. **Personal and global best update:** Compare current fitness; update P_i and G if improvements are found.

Exploration vs. Exploitation

The inertia weight w balances exploration and exploitation:

- High $w \rightarrow$ more exploration.
- Low $w \rightarrow$ more exploitation.

Adjusting c_1 and c_2 tunes the influence of personal vs. social knowledge.

Algorithm Outline

1. Initialize particles with random positions and velocities.
2. Evaluate fitness and set initial P_i and G.
3. Repeat until stopping criterion:
   (a) Update velocity and position.
   (b) Evaluate fitness.
   (c) Update personal best P_i and global best G.
4. Return the global best solution G.

Applications

PSO has been widely applied in neural network training, feature selection, image processing, engineering design, control systems, and many other optimization tasks.

Pseudocode

1. Initialize a swarm of n particles with random positions X_i and velocities V_i.
2. Evaluate fitness $f(X_i)$ for each particle.
3. Set each particle's personal best $P_i = X_i$.
4. Identify the global best position G among all particles.
5. **Repeat until stopping criterion is met:**
   (a) For each particle i:
   - Update velocity:
   $$V_i \leftarrow wV_i + c_1 r_1 (P_i - X_i) + c_2 r_2 (G - X_i)$$
   where w is the inertia weight, c_1 and c_2 are cognitive and social coefficients, and $r_1, r_2 \sim U(0,1)$ are random numbers.
   - Update position:
   $$X_i \leftarrow X_i + V_i$$
   - Evaluate new fitness $f(X_i)$.
   - Update personal best:
   $$\text{if } f(X_i) < f(P_i), \quad P_i \leftarrow X_i$$
   (b) Update global best:
   $$G \leftarrow \arg\min\{f(P_i)\}$$
6. Return the global best solution G.

1.4 Differential Evolution (DE)

Description

Differential Evolution (DE), introduced by Storn and Price [15], is a simple and powerful evolutionary algorithm widely used for global optimization in continuous spaces. It works by combining population members using weighted differences to generate new candidate solutions.

Main Idea

DE operates on a population of candidate solutions:

- **Population vectors:** Represent candidate solutions.
- **Mutation:** Create new vectors by adding weighted differences between population members.
- **Crossover:** Mix components of mutated and current solutions.
- **Selection:** Choose between trial and current vectors based on fitness.

Core Phases

The DE algorithm consists of three main operations:

1. **Mutation:**
$$V_i = X_{r1} + F \cdot (X_{r2} - X_{r3})$$
where X_{r1}, X_{r2}, X_{r3} are distinct random vectors and $F \in [0, 2]$ is the mutation factor.

2. **Crossover:**
$$U_{ij} = \begin{cases} V_{ij}, & \text{if } rand_j \leq CR \text{ or } j = j_{rand} \\ X_{ij}, & \text{otherwise} \end{cases}$$
where CR is the crossover probability, and j_{rand} ensures at least one component comes from V_i.

3. **Selection:**
$$X_i(t+1) = \begin{cases} U_i, & \text{if } f(U_i) < f(X_i) \\ X_i, & \text{otherwise} \end{cases}$$

Exploration vs. Exploitation

Mutation and crossover promote exploration by introducing new solutions, while selection ensures exploitation by keeping the better individuals. The balance between F and CR controls the global vs. local search behavior.

Algorithm Outline

1. Initialize the population with random solutions.
2. Evaluate fitness of all solutions.
3. Repeat until stopping criterion:
   (a) Apply mutation to generate donor vectors.
   (b) Perform crossover to create trial vectors.
   (c) Apply selection to form the next generation.
4. Return the best solution found.

Applications

DE has been successfully applied in engineering design, neural network training, scheduling, control systems, and many continuous optimization problems.

1.5 Fruit Fly Optimization Algorithm (FOA)

Description

The Fruit Fly Optimization Algorithm (FOA), proposed by Pan [11], is a bio-inspired optimization method that mimics how fruit flies search for food sources. By combining random exploration and smell-based fitness evaluation, FOA efficiently solves continuous and discrete optimization problems.

Main Idea

FOA models the food-seeking behavior of fruit flies:

- **Smell-based search:** Fruit flies detect food from a distance using smell.
- **Vision-based refinement:** Once near food, they use vision to fine-tune their search.
- **Group behavior:** The swarm iteratively updates its best position and guides the next search.

Core Phases

The FOA algorithm consists of the following main steps:

1. **Random search:**
$$x_i = x_{\text{best}} + rand(), \quad y_i = y_{\text{best}} + rand()$$
where $rand()$ is a random value and $(x_{\text{best}}, y_{\text{best}})$ is the current best position.
2. **Smell concentration calculation:**
$$Dist_i = \sqrt{x_i^2 + y_i^2}, \quad S_i = \frac{1}{Dist_i}$$
3. **Fitness evaluation:** Evaluate the objective function:
$$f(S_i)$$
4. **Best position update:** Identify the position with the highest smell concentration (best fitness) and update $(x_{\text{best}}, y_{\text{best}})$.

Exploration vs. Exploitation

FOA balances exploration through random movement and exploitation by refining around the best-smelling position. This mechanism helps avoid local optima and improves convergence speed.

Algorithm Outline

1. Initialize the fruit fly swarm at random positions.
2. Calculate smell concentration and evaluate fitness.
3. Find the best fly and update the best-known position.
4. Repeat until stopping criterion:
   (a) Perform random search around the best-known position.
   (b) Calculate distances, smell concentration, and fitness.
   (c) Update the best position if a better one is found.
5. Return the best solution found.

Applications

FOA has been applied in function optimization, signal processing, image registration, neural network training, and engineering design problems.

Pseudocode

1. Initialize a fruit fly population at random positions (x_i, y_i).
2. Evaluate the smell concentration (fitness) at each position.
3. Find the best position $(x_{\text{best}}, y_{\text{best}})$ with the highest smell (best fitness).
4. **Repeat until stopping criterion is met:**
   (a) Move each fruit fly randomly:
   $$x_i \leftarrow x_{\text{best}} + \text{rand}() \quad , \quad y_i \leftarrow y_{\text{best}} + \text{rand}()$$
   where rand() generates a random number.
   (b) Compute distance to origin:
   $$\text{Dist}_i = \sqrt{x_i^2 + y_i^2}$$
   (c) Compute smell concentration (fitness):
   $$S_i = \frac{1}{\text{Dist}_i}$$
   (d) Evaluate the objective function $f(S_i)$.
   (e) Update the best position if a better solution is found.
5. Return the best solution found.

1.6 Whale Optimization Algorithm (WOA)

Description

The Whale Optimization Algorithm (WOA), proposed by Mirjalili and Lewis [9], is a swarm-based meta-heuristic that simulates how humpback whales encircle prey and hunt cooperatively using bubble-net feeding strategies. WOA is widely used for continuous optimization tasks.

Main Idea

WOA mimics two main whale behaviors:

- **Encircling prey:** Whales estimate the prey's position and move toward it.
- **Bubble-net attack:** Whales use a spiral-shaped path or shrinking circle to close in on prey.
- **Search for prey:** When no prey is identified, whales explore randomly.

Core Phases

The WOA algorithm operates in three main modes:

1. **Encircling prey:**
$$\vec{D} = |\vec{C} \cdot \vec{X}^* - \vec{X}|, \quad \vec{X}(t+1) = \vec{X}^* - \vec{A} \cdot \vec{D}$$
where $\vec{X}^*$ is the best solution, $\vec{A}$ and $\vec{C}$ are coefficient vectors.

2. **Bubble-net attacking:**
   - Shrinking encircling: decrease $\vec{A}$ over iterations.
   - Spiral update:
$$\vec{X}(t+1) = \vec{D}' \cdot e^{bl} \cdot \cos(2\pi l) + \vec{X}^*$$
where $\vec{D}' = |\vec{X}^* - \vec{X}|$, b is a constant, l is a random number.

3. **Search for prey:** When $|\vec{A}| \geq 1$, whales explore by moving toward a random whale:
$$\vec{X}(t+1) = \vec{X}_{rand} - \vec{A} \cdot |\vec{C} \cdot \vec{X}_{rand} - \vec{X}|$$

Exploration vs. Exploitation

The parameter $\vec{A}$ controls the balance:

- $|\vec{A}| < 1$: exploitation (focus on best solution).
- $|\vec{A}| \geq 1$: exploration (search the space).

A linear decrease of $\vec{A}$ over iterations improves convergence.

Algorithm Outline

1. Initialize whale positions randomly.
2. Evaluate fitness and identify the best solution.
3. Repeat until stopping criterion:
   (a) Update coefficient vectors $\vec{A}$ and $\vec{C}$.
   (b) Choose update strategy (encircling, spiral, or random search).
   (c) Update whale positions.
   (d) Evaluate fitness and update the best solution.
4. Return the best solution found.

Applications

WOA has been successfully applied in feature selection, energy system optimization, engineering design, image processing, and machine learning tasks.

Pseudocode

1. Initialize a population of whales X_i $(i = 1, 2, \ldots, n)$ with random positions.
2. Evaluate fitness $f(X_i)$ and determine the best solution X^*.
3. **Repeat until stopping criterion is met:**
   (a) For each whale i:
      i. Calculate coefficients:
      $$\vec{A} = 2\vec{a} \cdot \vec{r} - \vec{a} \quad , \quad \vec{C} = 2 \cdot \vec{r}$$
      where $\vec{a}$ decreases linearly from 2 to 0 and $\vec{r} \sim U(0, 1)$.
      ii. Update position:
      - If $|\vec{A}| < 1$ (exploitation):
      $$X_i(t+1) = X^* - \vec{A} \cdot |\vec{C} \cdot X^* - X_i(t)|$$
      - If $|\vec{A}| \geq 1$ (exploration):
      $$X_i(t+1) = X_{rand} - \vec{A} \cdot |\vec{C} \cdot X_{rand} - X_i(t)|$$
      where X_{rand} is a random whale.
      - Spiral bubble-net update (probabilistic switch):
      $$X_i(t+1) = D' \cdot e^{bl} \cdot \cos(2\pi l) + X^*$$
      where $D' = |X^* - X_i(t)|$, b is the spiral constant, and $l \sim [-1, 1]$.
      iii. Evaluate new fitness $f(X_i(t+1))$.
   (b) Update X^* if a better solution is found.
4. Return the best solution X^*.

1.7 Artificial Bee Colony (ABC)

Description

The Artificial Bee Colony (ABC) algorithm, proposed by Karaboga and Basturk [6], is a swarm intelligence algorithm that mimics how honey bees search for nectar sources, share information, and optimize their foraging strategies. ABC is widely used for both continuous and combinatorial optimization problems.

Main Idea

ABC divides bees into three types:

- **Employed bees:** Exploit specific food sources (solutions) and share information.
- **Onlooker bees:** Observe dances and choose food sources based on their quality.
- **Scout bees:** Search randomly for new food sources when current ones are exhausted.

Core Phases

The ABC algorithm has three main stages:

1. **Employed bee phase:**
$$V_{ij} = X_{ij} + \phi_{ij}(X_{ij} - X_{kj})$$
where $k \neq i$, and ϕ_{ij} is a random number in $[-1, 1]$.

2. **Onlooker bee phase:**
   - Select food sources probabilistically:
$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$$
   - Generate new solutions as in the employed bee phase.

3. **Scout bee phase:**
   - Replace abandoned food sources with new random solutions.

Exploration vs. Exploitation

- Employed and onlooker bees focus on exploitation by refining good solutions.
- Scout bees enhance exploration by introducing diversity when stagnation occurs.

Algorithm Outline

1. Initialize food sources (solutions) randomly.
2. Evaluate the fitness of each food source.
3. Repeat until stopping criterion:
   (a) Employed bee phase: generate new solutions and update fitness.
   (b) Onlooker bee phase: select and improve food sources.
   (c) Scout bee phase: replace abandoned sources.
   (d) Memorize the best solution found.
4. Return the best solution.

Applications

ABC has been applied to numerical optimization, clustering, image processing, neural network training, feature selection, and engineering design problems.

Pseudocode

1. Initialize a population of food sources (solutions) X_i $(i = 1, 2, \ldots, n)$.
2. Evaluate the fitness $f(X_i)$ for each food source.
3. **Repeat until stopping criterion is met:**
   (a) **Employed bee phase:**
      - For each employed bee:
$$V_{ij} = X_{ij} + \phi_{ij}(X_{ij} - X_{kj})$$
        where $k \neq i$ and $\phi_{ij} \sim U[-1, 1]$.

- Evaluate $f(V_i)$; use greedy selection between X_i and V_i.

(b) **Onlooker bee phase:**

- Calculate selection probabilities:

$$p_i = \frac{f(X_i)}{\sum_{j=1}^{n} f(X_j)}$$

- Select food sources probabilistically and generate new V_i as in employed bee phase.

(c) **Scout bee phase:**

- If a food source has not improved for a predefined limit, replace it with a random solution.

(d) Memorize the best solution found so far.

4. Return the best solution.

1.8 Ant Colony Optimization (ACO)

Description

Ant Colony Optimization (ACO), introduced by Dorigo et al. [3], is a nature-inspired metaheuristic originally designed for combinatorial optimization problems like the traveling salesman problem (TSP). It uses pheromone trails and probabilistic solution construction to guide the search.

Main Idea

ACO models the collective behavior of ants:

- **Artificial ants:** Build solutions probabilistically based on pheromone trails and heuristic information.
- **Pheromone trails:** Represent shared memory about good solutions.
- **Heuristic information:** Provides local problem-specific guidance (e.g., inverse of distance in TSP).

Core Phases

ACO consists of the following main steps:

1. **Solution construction:**

$$p_{ij}^k = \frac{\tau_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum_{l \in N_i^k} \tau_{il}^\alpha \cdot \eta_{il}^\beta}$$

where p_{ij}^k is the probability of ant k moving from node i to j, τ_{ij} is the pheromone value, η_{ij} is heuristic desirability, and α, β are parameters.

2. **Pheromone update:**

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \sum_k \Delta\tau_{ij}^k$$

where ρ is the evaporation rate, and $\Delta\tau_{ij}^k$ is the pheromone deposited by ant k.

3. **Optional local search:** Improve solutions using a problem-specific local optimizer.

Exploration vs. Exploitation

- Pheromone evaporation encourages exploration.
- Positive feedback (more pheromone on better paths) drives exploitation.
- Balancing α, β, and ρ controls search behavior.

Algorithm Outline

1. Initialize pheromone trails uniformly.
2. Repeat until stopping criterion:
   (a) Each ant constructs a solution probabilistically.
   (b) Evaluate the quality (fitness) of all solutions.
   (c) Update pheromone trails.
   (d) Optionally apply local search.
3. Return the best solution found.

Applications

ACO has been applied to routing, scheduling, vehicle routing, network design, subset selection, and continuous optimization extensions.

Pseudocode

1. Initialize pheromone trails τ_{ij} on all edges (i, j).
2. **Repeat until stopping criterion is met:**
   (a) For each ant k:
      i. Construct a solution (path) probabilistically:

$$p_{ij}^k = \frac{\tau_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum_{l \in N_i^k} \tau_{il}^\alpha \cdot \eta_{il}^\beta}$$

      where η_{ij} is heuristic desirability (e.g., inverse of distance), N_i^k is the neighborhood, α, β are parameters.
   (b) Evaluate the quality (fitness) of all ant solutions.
   (c) Update pheromone trails:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \sum_k \Delta\tau_{ij}^k$$

   where ρ is evaporation rate and $\Delta\tau_{ij}^k$ is the pheromone deposited by ant k.
   (d) Optionally apply pheromone limits or smoothing.
3. Return the best solution found.

1.9 Bat Algorithm

Description

The Bat Algorithm (BA), proposed by Yang [17], is a metaheuristic that simulates how bats use echolocation to detect prey, avoid obstacles, and navigate. BA balances global exploration and local exploitation using frequency tuning, loudness, and pulse rate.

Main Idea

BA models the behavior of bats using:

- **Frequency tuning:** Bats adjust frequency to control movement.
- **Loudness (A):** Controls the range and refinement of search.
- **Pulse rate (r):** Determines the likelihood of local search.

Each bat flies through the search space, adjusting its parameters adaptively.

Core Phases

The BA operates with three main steps:

1. **Frequency, velocity, and position update:**

$$f_i = f_{\min} + (f_{\max} - f_{\min}) \cdot rand()$$

$$V_i = V_i + (X_i - X^*) f_i, \quad X_i = X_i + V_i$$

2. **Local search:** If $rand() > r_i$, generate a local solution around the best:

$$X_{\text{new}} = X^* + \epsilon A_i$$

   where $\epsilon \sim U(-1, 1)$.

3. **Loudness and pulse rate update:**

$$A_i^{t+1} = \alpha A_i^t, \quad r_i^{t+1} = r_i^0 [1 - e^{-\gamma t}]$$

Exploration vs. Exploitation

- Large loudness and low pulse rate promote exploration.
- As loudness decreases and pulse rate increases, the algorithm shifts to exploitation.

The dynamic adjustment allows efficient convergence.

Algorithm Outline

1. Initialize bats with random positions, velocities, frequencies, loudness, and pulse rates.
2. Evaluate fitness and identify the best bat.
3. Repeat until stopping criterion:
   (a) Update frequency, velocity, and position.
   (b) Apply local search based on pulse rate.
   (c) Evaluate fitness and update best solution.
   (d) Update loudness and pulse rate.
4. Return the best solution.

Applications

BA has been used in engineering design, image processing, feature selection, data clustering, scheduling, and power system optimization.

Pseudocode

1. Initialize a population of bats with positions X_i, velocities V_i, frequencies f_i, pulse rates r_i, and loudness A_i.
2. Evaluate fitness $f(X_i)$ and find the best solution X^*.
3. **Repeat until stopping criterion is met:**
   (a) For each bat i:
      i. Update frequency:

$$f_i = f_{\min} + (f_{\max} - f_{\min}) \cdot rand()$$

ii. Update velocity and position:

$$V_i = V_i + (X_i - X^*)f_i \quad , \quad X_i = X_i + V_i$$

iii. If $rand() > r_i$, generate a local solution near the best:

$$X_{\text{new}} = X^* + \epsilon A_i$$

where $\epsilon \sim U[-1, 1]$.

iv. Evaluate fitness $f(X_i)$.

v. Update A_i and r_i:

$$A_i^{t+1} = \alpha A_i^t \quad , \quad r_i^{t+1} = r_i^0[1 - e^{-\gamma t}]$$

where α and γ control loudness and pulse rate.

(b) Update the global best solution X^* if necessary.

4. Return the best solution X^*.

1.10 Bacterial Foraging Optimization Algorithm (BFOA)

Description

The Bacterial Foraging Optimization Algorithm (BFOA), proposed by Passino [12], mimics how *E. coli* bacteria search for nutrients, avoid noxious substances, and communicate through swarming. BFOA is particularly effective for multimodal and noisy optimization problems.

Main Idea

BFOA models bacterial processes using:

- **Chemotaxis:** Movement by tumbling and swimming toward nutrients.
- **Swarming:** Attraction toward promising regions with other bacteria.
- **Reproduction:** Healthiest bacteria split; weakest are removed.
- **Elimination-dispersal:** Random relocation to explore new areas.

Core Phases

The BFOA has four main loops:

1. **Chemotaxis:**
   - Tumble:
   $$X_i(t+1) = X_i(t) + C_i \cdot \Delta$$
   where C_i is the step size and Δ is a random direction.
   - Swim: Continue tumbling in the same direction if fitness improves, up to N_s steps.
2. **Swarming:** Bacteria are attracted toward promising nutrient zones based on cell-to-cell signaling.
3. **Reproduction:** Sort bacteria by accumulated fitness; the healthier half splits, and the weaker half is eliminated.
4. **Elimination-dispersal:** With small probability P_{ed}, randomly relocate some bacteria to new positions.

Exploration vs. Exploitation

- Chemotaxis and elimination-dispersal promote exploration.
- Swarming and reproduction enhance exploitation near good solutions.

Algorithm Outline

1. Initialize a population of bacteria randomly.
2. Repeat for elimination-dispersal loops:
   (a) Repeat for reproduction loops:
      i. Perform chemotaxis (tumble and swim).
      ii. Evaluate fitness.
   (b) Apply reproduction.
3. Apply elimination-dispersal events.
4. Return the best bacterium (solution).

Applications

BFOA has been applied in control system tuning, function optimization, pattern recognition, antenna design, and bioinformatics.

Pseudocode

1. Initialize a population of bacteria with random positions X_i.
2. **Repeat for N_{re} reproduction loops:**
   (a) **Chemotaxis loop (N_c steps):**
      - For each bacterium i:
        i. Compute fitness $f(X_i)$.
        ii. Tumble:
        $$X_i = X_i + C_i \cdot \Delta$$
        where C_i is the step size and Δ is a random unit vector.
        iii. Swim: If fitness improves, continue tumbling in the same direction up to N_s steps.
   (b) **Reproduction:**
      - Sort bacteria by cumulative fitness.
      - Eliminate the least healthy half.
      - Duplicate the healthier half to maintain population size.
   (c) **Elimination-dispersal:**
      - With probability P_{ed}, randomly disperse some bacteria to new positions.
3. Return the best bacterium found.

1.11 Social Spider Optimization (SSO)

Description

The Social Spider Optimization (SSO) algorithm, proposed by Cuevas et al. [2], models the cooperative behavior and communication strategies observed in social spider colonies. SSO efficiently balances exploration and exploitation using gender-based roles and interactions.

Main Idea

SSO divides the population into two groups:

- **Female spiders:** Cooperatively move toward or away from neighbors based on attraction and repulsion.
- **Male spiders:** Divided into dominant and non-dominant males with distinct mating and movement roles.

Spiders communicate through a communal web, sharing information about fitness and position.

Core Phases

The SSO algorithm has three main processes:

1. **Female cooperative operator:**
    - Females move based on the weighted attraction or repulsion toward other spiders and the global best.
2. **Male cooperative operator:**
    - Dominant males are attracted to the nearest females.
    - Non-dominant males move toward the center of the male population.
3. **Mating operator:**
    - Dominant males mate with nearby females to generate offspring.
    - Offspring replace the worst spiders if they improve the population fitness.

Exploration vs. Exploitation

- Female movement and male cooperation enhance exploration.
- Mating and local interactions support exploitation around promising solutions.

Algorithm Outline

1. Initialize a population of spiders, divided by gender.
2. Evaluate fitness and assign weights.
3. Repeat until stopping criterion:
    (a) Apply female cooperative movement.
    (b) Apply male cooperative movement.
    (c) Perform mating and replace weak individuals if offspring are better.
    (d) Update fitness and weights.
4. Return the best solution found.

Applications

SSO has been applied in function optimization, feature selection, neural network training, power system design, and image processing.

Pseudocode

1. Initialize a population of spiders (solutions) X_i, divided into males and females.
2. Evaluate fitness $f(X_i)$ and assign weight (importance) w_i to each spider.
3. **Repeat until stopping criterion is met:**
   (a) **Female cooperative operator:**
   - Each female spider moves based on the weighted attraction or repulsion toward other spiders and the global best.

   (b) **Male cooperative operator:**
   - Male spiders are divided into dominant and non-dominant.
   - Dominant males are attracted to the nearest females.
   - Non-dominant males aggregate near the center of the male population.

   (c) **Mating operator:**
   - Dominant males mate with nearby females to generate new candidate solutions (offspring).
   - Replace the worst spiders with offspring if they have better fitness.

   (d) Evaluate fitness and update weights w_i.
4. Return the best solution found.

1.12 Symbiotic Organisms Search (SOS)

Description

The Symbiotic Organisms Search (SOS) algorithm, proposed by Cheng and Prayogo [1], models the mutualism, commensalism, and parasitism relationships among organisms to guide optimization. SOS is simple, parameter-light, and effective for various continuous optimization problems.

Main Idea

SOS simulates three biological interactions:

- **Mutualism:** Both organisms benefit from the interaction.
- **Commensalism:** One organism benefits, the other is unaffected.
- **Parasitism:** One organism benefits at the expense of another.

Organisms are candidate solutions that improve through repeated interactions.

Core Phases

The SOS algorithm consists of three interaction phases:

1. **Mutualism phase:**

$$X_i' = X_i + rand() \cdot (X^* - MV \cdot BFPatent1), \quad X_j' = X_j + rand() \cdot (X^* - MV \cdot BF_2)$$

   where MV is the mutual vector (average of X_i and X_j), BF_1 and BF_2 are benefit factors (1 or 2), and X^* is the best solution.

2. **Commensalism phase:**

$$X_i' = X_i + rand(-1, 1) \cdot (X^* - X_j)$$

3. **Parasitism phase:**
   - Create a parasite vector from X_i.
   - Replace X_j with the parasite if it has better fitness.

Exploration vs. Exploitation

- Mutualism and commensalism promote exploration.
- Parasitism emphasizes exploitation by replacing weak solutions.

Algorithm Outline

1. Initialize a population of organisms (solutions).
2. Evaluate fitness and identify the best solution.
3. Repeat until stopping criterion:
   (a) Apply mutualism between pairs of organisms.
   (b) Apply commensalism between pairs.
   (c) Apply parasitism to replace weak organisms.
   (d) Update the best solution if improved.
4. Return the best solution found.

Applications

SOS has been used in structural optimization, parameter estimation, power system optimization, feature selection, and engineering design.

Pseudocode

1. Initialize a population of organisms (solutions) X_i.
2. Evaluate fitness $f(X_i)$.
3. **Repeat until stopping criterion is met:**
   (a) **Mutualism phase:**
   - For each organism i, randomly select organism j.
   - Update both:

$$X_i' = X_i + rand() \cdot (X^* - MV \cdot BF_1) \quad , \quad X_j' = X_j + rand() \cdot (X^* - MV \cdot BF_2)$$

   where X^* is the best solution, $MV = \frac{X_i + X_j}{2}$, BF {1,2}.

   (b) **Commensalism phase:**
   - For each organism i, randomly select j.
   - Update:

$$X_i' = X_i + rand(-1, 1) \cdot (X^* - X_j)$$

   (c) **Parasitism phase:**
   - For each organism i, create a parasite vector P_i.
   - Randomly select j; if P_i is better, replace X_j.

   (d) Evaluate new fitness and update X^* if needed.
4. Return the best solution X^*.

1.13 Honey Badger Algorithm (HBA)

Description

The Honey Badger Algorithm (HBA), proposed by Hashim et al. [4], is a nature-inspired metaheuristic that simulates the foraging behavior of honey badgers, including digging and following prey. HBA effectively balances exploration and exploitation and is suitable for continuous optimization tasks.

Main Idea

HBA models two core honey badger strategies:

- **Digging phase:** The badger digs and searches around the best-known prey location.
- **Honey phase:** The badger actively follows the scent toward prey.

A density factor controls the balance between global exploration and local exploitation.

Core Phases

HBA consists of two main operations:

1. **Density factor calculation:**
$$F = e^{-t/MaxIter}$$
where t is the current iteration, controlling the search intensity.
2. **Position update:**
   - **Digging phase:**
$$X_i(t+1) = X^* + F \cdot (rand() \cdot X_i - rand() \cdot X^*)$$
   - **Honey phase (following scent):**
$$X_i(t+1) = X_i + F \cdot rand() \cdot (X^* - X_i)$$
   where X^* is the best-known solution.

Exploration vs. Exploitation

- Early iterations (large F) promote exploration.
- Later iterations (small F) focus on local exploitation around the best solution.

Algorithm Outline

1. Initialize honey badger population randomly.
2. Evaluate fitness and identify the best solution X^*.
3. Repeat until stopping criterion:
   (a) Calculate the density factor F.
   (b) Update positions using digging or honey phase equations.
   (c) Evaluate fitness and update the best solution.
4. Return the best solution.

Applications

HBA has been applied in feature selection, structural optimization, energy system optimization, machine learning model tuning, and image processing.

Pseudocode

1. Initialize a population of honey badgers (solutions) X_i.
2. Evaluate fitness $f(X_i)$.
3. Set the best solution X^*.
4. **Repeat until stopping criterion is met:**
   (a) Calculate density factor F and update scent concentration.
   (b) For each honey badger i:
   - **Digging phase:**
   $$X_i(t+1) = X^* + F \cdot (rand() \cdot X_i - rand() \cdot X^*)$$
   - **Honey phase (following scent):**
   $$X_i(t+1) = X_i + F \cdot rand() \cdot (X^* - X_i)$$
   (c) Update density factor F over iterations.
   (d) Evaluate new fitness $f(X_i)$ and update X^* if improved.
5. Return the best solution X^*.

1.14 Moth-Flame Optimization (MFO)

Description

The Moth-Flame Optimization (MFO) algorithm, proposed by Mirjalili [8], models how moths fly around flames in a logarithmic spiral pattern. It is an efficient population-based metaheuristic for solving continuous optimization problems.

Main Idea

MFO uses two key elements:

- **Moths:** Represent candidate solutions flying in the search space.
- **Flames:** Best-known solutions that guide moth movement.

Moths update their positions around flames using a logarithmic spiral path to balance exploration and exploitation.

Core Phases

MFO operates with these main steps:

1. **Spiral position update:**
   $$M_i(t+1) = S_i \cdot e^{b \cdot l} \cdot \cos(2\pi l) + F_j$$
   where S_i is the distance between moth i and flame j, b is a constant, and l is a random number in $[-1, 1]$.
2. **Flame number reduction:**
   $$Flame\_No = round(N - t \cdot \frac{N-1}{MaxIter})$$
   where N is the moth population size, t is the current iteration, and $MaxIter$ is the maximum iteration number.
3. **Flame update:** Sort the population based on fitness and assign the best solutions as flames.

Exploration vs. Exploitation

- Early iterations use more flames, promoting exploration.
- As iterations progress, fewer flames focus the search on exploitation.

Algorithm Outline

1. Initialize the moth population randomly.
2. Evaluate fitness and identify flames.
3. Repeat until stopping criterion:
   (a) Update the number of flames.
   (b) Update moth positions around flames using spiral equations.
   (c) Evaluate fitness and update flames.
4. Return the best solution.

Applications

MFO has been applied in feature selection, engineering design, power system optimization, scheduling, and machine learning tasks.

Pseudocode

1. Initialize a population of moths (solutions) M_i.
2. Evaluate fitness $f(M_i)$.
3. Sort flames (best solutions) F_j based on fitness.
4. **Repeat until stopping criterion is met:**
   (a) Update number of flames:

$$Flame\_No = round(N - t \cdot (N-1)/MaxIter)$$

   where N is population size, t is current iteration.
   (b) For each moth i:
   - Update position using a logarithmic spiral:

$$M_i(t+1) = S_i \cdot e^{b \cdot t} \cdot \cos(2\pi t) + F_j$$

   where S_i is distance to flame F_j, b controls spiral shape.
   (c) Evaluate fitness and update flames.
5. Return the best solution.

1.15 Elephant Herding Optimization (EHO)

Description

The Elephant Herding Optimization (EHO) algorithm, proposed by Wang et al. [16], simulates the social structure and movement patterns of elephant clans, including clan updating and separation mechanisms. EHO is designed for continuous optimization problems.

Main Idea

EHO models the collective behavior of elephants:

- **Clans:** Subgroups of the population, each with its own best member.
- **Clan updating:** Elephants within a clan adjust their positions toward the clan leader.
- **Separating operator:** The worst elephant is replaced by a random elephant to maintain diversity.

Core Phases

The EHO algorithm has two main operators:

1. **Clan updating operator:**

$$E_{i,j}(t+1) = C_j^* + \alpha \cdot (rand() \cdot (C_j^* - E_{i,j}(t)))$$

   where $E_{i,j}$ is the i-th elephant in clan j, C_j^* is the clan leader, and α is a scaling factor.

2. **Separating operator:**
   - Replace the worst elephant in each clan with a new randomly generated elephant.

Exploration vs. Exploitation

- Clan updating focuses on exploitation by moving elephants toward the best in the clan.
- Separation enhances exploration by introducing new random solutions.

Algorithm Outline

1. Initialize elephant population divided into clans.
2. Evaluate fitness and identify the best elephant in each clan.
3. Repeat until stopping criterion:
   (a) Apply clan updating operator to each clan.
   (b) Apply separating operator to replace the worst elephants.
   (c) Evaluate fitness and update clan bests.
4. Return the best solution from the population.

Applications

EHO has been applied in feature selection, machine learning, engineering design, scheduling, and power system optimization.

Pseudocode

1. Initialize a population of elephants divided into clans.
2. Evaluate fitness $f(E_i)$ for each elephant.
3. Identify the clan best C_j^* in each clan.
4. **Repeat until stopping criterion is met:**
   (a) **Clan updating operator:**

- For each clan, update member positions:

$$E_i(t+1) = C_j^* + \alpha \cdot (rand() \cdot (C_j^* - E_i(t)))$$

where α is a scaling factor.

(b) **Separating operator:**

- Replace the worst elephant in each clan with a random new elephant.

(c) Evaluate fitness and update clan best C_j^*.

5. Return the best elephant in the population.

1.16 Grasshopper Optimization Algorithm (GOA)

Description

The Grasshopper Optimization Algorithm (GOA), proposed by Saremi et al. [14], mimics the collective movement of grasshoppers, including attraction, repulsion, and social interaction forces, to balance exploration and exploitation in solving continuous optimization problems.

Main Idea

GOA models swarm dynamics using:

- **Grasshoppers:** Candidate solutions in the search space.
- **Social forces:** Attraction-repulsion functions that govern interactions between grasshoppers.
- **Global target:** The best-known solution guiding swarm movement.

Core Phases

The GOA algorithm has two main components:

1. **Social interaction model:**

$$S_i = \sum_{j=1, j\neq i}^{n} s(d_{ij}) \frac{X_j - X_i}{d_{ij}}$$

where d_{ij} is the distance between grasshopper i and j, and $s(.)$ is an attraction-repulsion function.

2. **Position update:**

$$X_i(t+1) = c \cdot S_i + T_g$$

where c decreases linearly over iterations and T_g is the global best solution.

Exploration vs. Exploitation

- Early iterations (large c) emphasize exploration.
- Late iterations (small c) enhance exploitation around the best solutions.

Algorithm Outline

1. Initialize grasshopper population randomly.
2. Evaluate fitness and identify the global best solution.
3. Repeat until stopping criterion:
   (a) Compute social interactions among grasshoppers.
   (b) Update positions toward the global target.
   (c) Evaluate fitness and update the global best solution.
4. Return the best solution.

Applications

GOA has been applied in feature selection, structural optimization, image segmentation, engineering design, and machine learning parameter tuning.

Pseudocode

1. Initialize a population of grasshoppers X_i.
2. Evaluate fitness $f(X_i)$.
3. Set the best solution X^*.
4. **Repeat until stopping criterion is met:**
   (a) For each grasshopper i:
   - Calculate social interaction:
   $$S_i = \sum_{j=1, j\neq i}^{n} s(d_{ij}) \cdot \frac{X_j - X_i}{d_{ij}}$$
   where d_{ij} is distance between X_i and X_j, and $s(.)$ is an attraction-repulsion function.
   - Update position:
   $$X_i(t+1) = c \cdot S_i + T_g$$
   where c decreases linearly (exploration $\rightarrow$ exploitation), and T_g is the target (best) position.

   (b) Evaluate fitness $f(X_i(t+1))$.
   (c) Update the global best X^* if necessary.
5. Return the best solution X^*.

1.17 Harris Hawks Optimization (HHO)

Description

The Harris Hawks Optimization (HHO) algorithm, proposed by Heidari et al. [5], simulates the dynamic and cooperative hunting behavior of Harris hawks when they surround and attack prey. HHO is designed for global optimization in continuous search spaces.

Main Idea

HHO uses the following strategies:

- **Exploration phase:** Hawks search randomly when prey escapes.
- **Exploitation phase:** Hawks surround and attack the prey using soft and hard besiege tactics.
- **Escape energy (E):** Controls the transition between exploration and exploitation.

Core Phases

The HHO algorithm has these main components:

1. **Escape energy calculation:**
   $$E = 2(1 - \frac{t}{T}) \cdot rand() - 1$$
   where t is the current iteration, T is the maximum iteration, and $rand()$ is a random number.
2. **Position update:**
   - **Exploration ($|E| \geq 1$):** Hawks explore randomly or based on other hawks.
   - **Exploitation ($|E| < 1$):** Hawks use soft besiege (gradual approach) or hard besiege (rapid dive) depending on E and the prey's escape probability.

Exploration vs. Exploitation

- The escape energy E dynamically controls the balance.
- As iterations progress, hawks shift from exploration to exploitation.

Algorithm Outline

1. Initialize hawk population randomly.
2. Evaluate fitness and identify the best solution (prey).
3. Repeat until stopping criterion:
   (a) Calculate escape energy E.
   (b) Choose exploration or exploitation strategy.
   (c) Update positions based on strategy.
   (d) Evaluate fitness and update the best solution.
4. Return the best solution.

Applications

HHO has been applied in engineering design, feature selection, image segmentation, parameter tuning, and energy system optimization.

Pseudocode

1. Initialize a population of hawks X_i with random positions.
2. Evaluate fitness $f(X_i)$ and identify the best solution X^*.
3. **Repeat until stopping criterion is met:**
   (a) For each hawk i:
   - Compute the escaping energy:

$$E = 2(1 - \frac{t}{T}) \cdot rand() - 1$$

     where t is the current iteration, T is max iterations.
   - If $|E| \geq 1$ (exploration), update position randomly.
   - If $|E| < 1$ (exploitation), choose between:
     - Soft siege: gradually approach the prey.
     - Hard siege: rapid attack on the prey.
   - Use different update equations depending on E and prey escape probability.

   (b) Evaluate fitness and update X^*.
4. Return the best solution X^*.

1.18 Coral Reefs Optimization (CRO)

Description

The Coral Reefs Optimization (CRO) algorithm, proposed by Salcedo-Sanz et al. [13], is a nature-inspired metaheuristic that simulates the processes of coral reproduction (sexual and asexual), larvae settlement, and depredation in a reef ecosystem. CRO has been successfully applied to combinatorial, continuous, and engineering optimization problems.

Main Idea

CRO models the dynamics of coral reefs:

- **Corals:** Represent candidate solutions.
- **Reef:** A grid structure where solutions compete for space.
- **Larvae:** New solutions produced through sexual/asexual reproduction.

Healthy solutions survive, reproduce, and occupy space, while weaker solutions are periodically removed to maintain diversity.

Key Phases

The CRO algorithm consists of three main phases:

1. **Reproduction phase**:
    - **Sexual reproduction (broadcast spawning):** Combine two corals using crossover.
    - **Sexual reproduction (brooding):** Apply mutation to a single coral.
    - **Asexual reproduction (budding):** Clone a single coral without modification.
2. **Larvae settlement**:
    - Try to place larvae into the reef grid.
    - If the larva has better fitness than an existing coral, it replaces the weak coral.
3. **Depredation**:
    - Randomly remove a fraction of the weakest corals to free space and maintain diversity.

Exploration vs. Exploitation

CRO balances exploration and exploitation by:

- Using sexual and asexual reproduction to explore new areas of the search space.
- Replacing weak solutions through settlement and depredation to intensify search near promising areas.

Algorithm Outline

1. Initialize the reef matrix with random corals (solutions).
2. Evaluate the fitness of all corals.
3. Repeat until stopping criterion:
    (a) Apply reproduction (broadcast spawning, brooding, budding) to generate larvae.
    (b) Attempt larvae settlement, replacing weak corals.
    (c) Perform depredation to remove a fraction of the worst corals.
    (d) Evaluate the updated reef.
4. Return the best coral (solution) found.

Applications

CRO has been applied successfully to diverse problems, including combinatorial optimization, feature selection, antenna design, power system optimization, and scheduling tasks.

Pseudocode

1. Initialize the coral reef matrix with randomly generated solutions (corals).
2. Evaluate fitness $f(C_i)$ of each coral.
3. **Repeat until stopping criterion is met:**
   (a) **Reproduction phase:**
   - Apply sexual reproduction (broadcast spawning and brooding) to generate larvae (new solutions).
   - Perform asexual reproduction (budding) for some corals.

   (b) **Larvae settlement:**
   - Try to place each larva in the reef; if it's better than the weakest coral, it replaces it.

   (c) **Depredation:**
   - Remove a fraction of the worst corals to increase diversity.

   (d) Evaluate fitness of the reef.
4. Return the best coral (solution) found.

Notes

This script provides a unified framework to run and compare multiple bio-inspired optimization algorithms using common test functions and parameter settings.

2 Benchmark Functions

In this section, several standard benchmark functions were used to evaluate the performance of 10 of the previously mentioned optimization algorithms. These functions represent a variety of optimization challenges, including unimodal and multimodal landscapes, convex and non-convex structures, and varying levels of d ficulty. For simplicity all of the functions have a global minima at $F_n = 0$ where n is the function number. They were evaluated with dimensions 10,20 and 30 using 30 search agents and 100 iterations.

2.1 F1: Sphere Function

$$F_1(x) = \sum_{i=1}^{n} x_i^2$$

This function has a global minimum at $x = 0$, where $F_1(x) = 0$.

2.2 F2: Schwefel 2.22 Function

$$F_2(x) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|$$

The global minimum is at $x = 0$, where $F_2(x) = 0$.

2.3 F3: Sum of Squares Function

$$F_3(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$$

This function has a global minimum at $x = 0$, where $F_3(x) = 0$.

2.4 F4: Maximum Absolute Value Function

$$F_4(x) = \max\left(|x_1|, |x_2|, ..., |x_n|\right)$$

It has a global minimum at $x = 0$ where $F_4 = 0$.

2.5 F5: Rosenbrock Function

$$F_5(x) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$$

The function has a global minimum at $x = (1, 1, \ldots, 1)$ Where $F_5 = 0$.

3 Optimization Algorithm Results

4 Optimization Results - 30D, 30 Search Agents, 100 Iterations

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
1	0.0133	0.0088	0.0032	6.71	1.58	5.01
2	0.0250	0.0082	0.0137	5.37	0.45	4.91
3	437.77	271.15	65.91	5.84	1.17	4.84
4	1.7696	0.8946	0.5472	4.95	0.20	4.76
5	31.28	1.93	28.89	5.34	0.16	5.13

Table 1: Grey Wolf Optimizer (GWO)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
1	28133	6726	18271	4.25	0.36	4.09
2	4.02e5	8.01e5	108.04	4.96	1.13	4.24
3	55570	12062	41356	4.45	0.29	4.16
4	82.36	7.31	68.01	4.08	0.10	3.95
5	7.33e7	3.92e7	2.36e7	4.65	0.53	4.26

Table 2: Particle Swarm Optimization (PSO)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
1	21413	5218.7	12044	4.78	0.75	3.92
2	1.72e6	4.59e6	108.51	4.73	1.47	4.03
3	47992	4796.6	40716	4.61	0.98	4.10
4	89.44	3.05	86.75	4.53	0.73	3.85
5	5.79e7	1.87e7	3.78e7	4.52	0.76	4.22

Table 3: Differential Evolution (DE)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
1	4001.1	843.32	2565.4	2.80	0.22	2.62
2	31.67	7.76	22.02	4.04	0.74	3.34
3	46230	8312.2	30852	2.92	0.09	2.77
4	74.84	6.13	62.00	2.88	0.18	2.72
5	8.41e6	2.86e6	4.76e6	3.89	0.15	3.73

Table 4: Artificial Bee Colony (ABC)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
1	1.14e-07	6.42e-10	1.13e-07	4.20	0.13	4.13
2	0.0184	5.46e-05	0.0183	4.51	0.86	3.94
3	3.50e-05	2.86e-07	3.45e-05	4.29	0.48	4.02
4	6.77e-05	5.70e-07	6.67e-05	4.20	0.09	4.05
5	28.68	0.076	28.51	4.27	0.10	4.15

Table 5: Fruit Fly Optimization Algorithm (FOA)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
1	1.26e-06	9.07e-07	6.13e-07	2.93	0.83	2.31
2	4.60e-05	6.48e-05	1.55e-06	3.53	0.06	3.47
3	93875	22224	50589	3.10	0.53	2.72
4	43.32	40.62	0.795	2.79	0.20	2.62
5	11.47	12.13	0.2299	3.94	0.35	3.71

Table 6: Whale Optimization Algorithm (WOA)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
1	2.85e-22	8.73e-22	1.30e-29	4.28	0.63	3.91
2	7.63e-13	1.44e-12	8.78e-15	4.00	0.03	3.94
3	2.73e-17	8.07e-17	2.06e-24	4.03	0.05	3.96
4	3.66e-12	1.05e-11	6.35e-16	3.74	0.06	3.66
5	0.4545	0.778	0.0036	4.23	0.39	3.96

Table 7: Harris Hawks Optimization (HHO)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
1	0.00134	0.0027	1.6e-11	3.99	0.57	3.62
2	0.00223	0.00464	1.42e-08	3.83	0.26	3.64
3	0.000161	0.000354	9.39e-09	3.82	0.17	3.63
4	1.32e-05	2.48e-05	4.33e-08	3.95	0.26	3.66
5	0.00227	0.00590	3.41e-09	3.85	0.12	3.66

Table 8: Grasshopper Optimization Algorithm (GOA)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
1	13851	3129.6	10159	4.22	0.42	3.84
2	405.91	989.31	59.33	4.21	0.24	3.98
3	72884	7368.3	61172	4.12	0.20	3.85
4	86.68	5.42	74.47	3.83	0.18	3.66
5	3.44e7	7.46e6	2.45e7	4.21	0.17	4.03

Table 9: Ant Colony Optimization (ACO)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
1	61164	6176	49843	4.04	0.56	3.79
2	2.20e6	2.94e6	13074	3.87	0.09	3.77
3	96551	26198	62441	4.19	0.35	3.88
4	81.26	1.75	78.62	3.85	0.15	3.66
5	1.84e8	2.17e7	1.57e8	4.47	0.55	3.96

Table 10: Coral Reefs Optimization (CRO)

5 20D, 30 Search Agents, 100 Iterations

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Grey Wolf Optimizer (GWO)						
1	0.0002419	0.0003396	2.34e-05	5.99	1.99	4.85
2	0.002995	0.001437	0.001555	5.05	0.26	4.73
3	16.37	15.00	2.36	4.83	0.14	4.66
4	0.2856	0.09712	0.1321	5.88	1.21	4.51
5	18.43	0.6845	17.39	5.05	0.66	4.57

Table 11: Grey Wolf Optimizer (GWO)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Particle Swarm Optimization (PSO)						
1	6200.8	1755.4	4285.2	3.65	0.20	3.56
2	94.16	96.38	36.51	3.56	0.06	3.50
3	22413	6240.7	13403	3.71	0.27	3.52
4	62.59	3.90	56.38	3.76	0.46	3.44
5	7.41e+06	4.92e+06	3.01e+06	3.87	0.09	3.77

Table 12: Particle Swarm Optimization (PSO)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Differential Evolution (DE)						
1	9837.9	2659.8	6381.8	4.72	1.01	3.90
2	73.41	28.03	43.33	3.96	0.07	3.86
3	16156	4666.1	7667.4	4.03	0.17	3.91
4	78.38	5.62	68.35	3.69	0.02	3.66
5	9.69e+06	5.99e+06	2.58e+06	4.00	0.06	3.93

Table 13: Differential Evolution (DE)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Artificial Bee Colony (ABC)						
1	288.34	83.00	152.77	3.44	1.37	2.21
2	4.56	1.31	2.89	2.26	0.08	2.10
3	16666	3745.5	10965	3.41	1.08	2.60
4	49.64	4.54	41.92	3.31	0.37	2.59
5	2.35e+05	1.11e+05	7.65e+04	4.54	0.51	3.74

Table 14: Artificial Bee Colony (ABC)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Fruit Fly Optimization Algorithm (FOA)						
1	7.14e-08	5.93e-10	7.04e-08	4.69	0.74	3.88
2	0.0119	3.76e-05	0.0119	4.48	0.48	4.03
3	1.00e-05	8.81e-08	9.84e-06	4.21	0.36	3.89
4	6.59e-05	5.30e-07	6.54e-05	4.06	0.08	3.98
5	18.80	0.0382	18.72	4.52	0.58	4.12

Table 15: Fruit Fly Optimization Algorithm (FOA)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Whale Optimization Algorithm (WOA)						
1	5.20e-06	8.21e-06	7.27e-07	2.49	0.16	2.28
2	2.54e-05	1.89e-05	2.65e-06	2.66	0.23	2.47
3	42434	8007.9	28434	2.71	0.06	2.62
4	36.65	32.54	0.78	2.61	0.06	2.52
5	7.57	8.59	0.21	3.75	0.65	2.77

Table 16: Whale Optimization Algorithm (WOA)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Harris Hawks Optimization (HHO)						
1	2.76e-24	6.09e-24	4.32e-29	3.63	0.22	3.50
2	4.94e-13	1.11e-12	2.38e-17	3.61	0.14	3.49
3	2.49e-16	5.32e-16	3.48e-24	3.64	0.06	3.58
4	3.44e-12	5.08e-12	1.48e-14	3.42	0.20	3.34
5	0.1450	0.2121	0.0007	3.85	0.18	3.63

Table 17: Harris Hawks Optimization (HHO)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Grasshopper Optimization Algorithm (GOA)						
1	3.44e-05	7.31e-05	1.18e-09	3.75	0.40	3.38
2	2.53e-05	5.68e-05	3.09e-09	4.16	1.07	3.48
3	0.0063	0.0198	5.86e-14	3.83	0.10	3.71
4	1.19e-04	1.47e-04	5.17e-09	3.80	0.09	3.72
5	5.47e-04	1.13e-03	2.07e-10	3.83	0.19	3.71

Table 18: Grasshopper Optimization Algorithm (GOA)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Ant Colony Optimization (ACO)						
1	1053.9	419.8	307.6	3.89	0.32	3.75
2	15.14	4.44	10.31	3.80	0.12	3.72
3	26781	4610.6	17693	3.85	0.09	3.76
4	68.79	6.18	60.66	3.62	0.11	3.53
5	6.61e+05	4.10e+05	2.01e+05	3.82	0.03	3.78

Table 19: Ant Colony Optimization (ACO)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Coral Reefs Optimization (CRO)						
1	33315	4937	22786	4.96	1.08	4.21
2	634.31	893.28	66.69	4.30	0.22	4.02
3	40512	10016	30920	4.45	0.11	4.34
4	72.22	5.85	62.05	4.08	0.04	4.03
5	7.49e+07	1.89e+07	4.93e+07	4.41	0.12	4.26

Table 20: Coral Reefs Optimization (CRO)

6 10D, 30 Search Agents, 100 Iterations

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Grey Wolf Optimizer (GWO)						
1	8.30e-07	1.55e-07	5.48e-07	2.60	0.45	2.05
2	3.76e-06	2.05e-06	9.95e-07	4.11	0.25	3.76
3	0.0056	0.0062	7.07e-05	4.25	0.28	3.87
4	0.0061	0.0095	0.0003	4.29	0.52	3.76
5	7.75	0.65	6.63	4.54	0.31	4.14

Table 21: Grey Wolf Optimizer (GWO)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Particle Swarm Optimization (PSO)						
1	256.39	185.43	115.88	3.55	0.55	3.13
2	6.92	3.08	2.77	3.13	0.09	3.06
3	2711.9	1669.3	640.85	3.32	0.08	3.16
4	16.94	3.70	10.35	3.11	0.03	3.07
5	54246	49283	763.28	3.36	0.09	3.27

Table 22: Particle Swarm Optimization (PSO)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Differential Evolution (DE)						
1	339.88	216.33	107.00	3.71	1.51	2.84
2	9.36	2.49	5.94	3.37	0.33	2.98
3	1178.00	506.03	710.81	3.38	0.11	3.27
4	19.60	5.24	8.52	3.55	0.26	3.25
5	11832.00	7790.60	3340.60	3.56	0.06	3.45

Table 23: Differential Evolution (DE)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Artificial Bee Colony (ABC)						
1	0.0584	0.0286	0.0109	3.25	0.26	2.78
2	0.0135	0.0038	0.0078	3.05	0.31	2.78
3	467.43	137.72	298.04	3.06	0.32	2.69
4	6.13	1.61	3.50	3.04	0.14	2.81
5	119.27	115.88	29.96	2.91	0.33	2.59

Table 24: Artificial Bee Colony (ABC)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Fruit Fly Optimization Algorithm (FOA)						
1	3.15e-08	3.01e-10	3.11e-08	4.11	0.61	3.78
2	0.0056	3.03e-05	0.0056	3.78	0.15	3.65
3	1.19e-06	1.67e-08	1.17e-06	4.25	0.46	3.79
4	6.15e-05	6.01e-07	6.06e-05	4.17	0.08	4.02
5	8.90	0.0084	8.89	4.17	0.23	3.88

Table 25: Fruit Fly Optimization Algorithm (FOA)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Whale Optimization Algorithm (WOA)						
1	3.15e-08	3.01e-10	3.11e-08	4.11	0.61	3.78
2	0.0056	3.03e-05	0.0056	3.78	0.15	3.65
3	1.19e-06	1.67e-08	1.17e-06	4.25	0.46	3.79
4	6.15e-05	6.01e-07	6.06e-05	4.17	0.08	4.02
5	8.90	0.0084	8.89	4.17	0.23	3.88

Table 26: Whale Optimization Algorithm (WOA)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Harris Hawks Optimization (HHO)						
1	1.24e-23	3.87e-23	4.56e-30	4.81	1.02	4.20
2	1.31e-13	2.49e-13	3.13e-16	5.84	0.76	4.54
3	2.01e-19	5.35e-19	5.24e-28	6.25	0.76	5.31
4	6.67e-13	9.62e-13	8.69e-16	5.18	0.31	4.92
5	0.12	0.12	0.0053	5.38	0.23	5.15

Table 27: Harris Hawks Optimization (HHO)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Grasshopper Optimization Algorithm (GOA)						
1	3.44e-05	7.31e-05	1.18e-08	7.37	2.92	4.36
2	4.72e-04	1.24e-03	2.83e-08	4.62	0.82	3.36
3	3.69e-05	5.78e-05	5.53e-08	4.51	0.60	3.80
4	6.71e-05	1.02e-04	1.30e-07	5.28	0.31	4.87
5	0.0019	0.0057	5.82e-07	6.13	1.51	4.23

Table 28: Grasshopper Optimization Algorithm (GOA)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Ant Colony Optimization (ACO)						
1	0.0534	0.0751	0.0098	4.32	1.21	3.36
2	0.0295	0.0194	0.0096	4.03	0.45	3.65
3	1168.6	750.9	383.3	5.01	0.50	4.60
4	1.83	0.61	0.94	5.55	0.65	4.56
5	88.59	106.49	11.80	5.30	1.16	3.33

Table 29: Ant Colony Optimization (ACO)

F_n	MeanBest	STDBest	BestBest	MeanTotalT [s]	STDTotalT [s]	BestTotalT [s]
Coral Reefs Optimization (CRO)						
1	11454	2919.3	7387	4.62	0.87	3.62
2	21.53	2.76	17.71	4.44	0.31	4.20
3	11030	3017.6	7132.9	6.27	1.79	4.66
4	55.79	8.53	37.46	3.92	0.50	3.27
5	1.09e+07	3.66e+06	2.54e+06	4.60	0.16	4.40

Table 30: Coral Reefs Optimization (CRO)

7 Results

The performance of the bio-inspired optimization algorithms was evaluated across benchmark functions F_1 to F_5 in dimensions 10, 20, and 30, using 30 search agents and 100 iterations. The results are summarized based on the mean best fitness (MeanBest), standard deviation of the best fitness (STDBest), and the best fitness achieved (BestBest). The following analysis identifies the four worst-performing algorithms and the best-performing algorithms across all dimensions, focusing on the MeanBest metric as the primary indicator of performance.

7.1 Worst-Performing Algorithms

The four algorithms that consistently exhibited the poorest performance, based on the highest MeanBest values across the benchmark functions and dimensions, are:

- **Coral Reefs Optimization (CRO)**: CRO showed the highest MeanBest values, particularly in 30D, with values such as 61164 for F_1, 2.20e6 for F_2, and 1.84e8 for F_5. In 20D and 10D, it continued to perform poorly, with MeanBest values like 33315 for F_1 (20D) and 11454 for F_1 (10D), indicating significant struggles in converging to optimal solutions.
- **Particle Swarm Optimization (PSO)**: PSO performed poorly, especially in higher dimensions, with MeanBest values of 28133 for F_1, 4.02e5 for F_2, and 7.33e7 for F_5 in 30D. In 20D and 10D, it also showed high MeanBest values, such as 6200.8 for F_1 (20D) and 256.39 for F_1 (10D), suggesting limited effectiveness across diverse functions.
- **Differential Evolution (DE)**: DE consistently ranked among the worst performers, with MeanBest values like 21413 for F_1, 1.72e6 for F_2, and 5.79e7 for F_5 in 30D. In lower dimensions, it maintained high MeanBest values, such as 9837.9 for F_1 (20D) and 339.88 for F_1 (10D), indicating challenges in achieving precise convergence.
- **Ant Colony Optimization (ACO)**: ACO also performed poorly, with MeanBest values of 13851 for F_1, 405.91 for F_2, and 3.44e7 for F_5 in 30D. In 20D and 10D, it showed similarly high MeanBest values, such as 1053.9 for F_1 (20D) and 0.0534 for F_1 (10D), reflecting difficulties in handling continuous optimization tasks effectively.

These algorithms struggled particularly with multimodal and high-dimensional functions, such as F_2 and F_5, where their MeanBest values were orders of magnitude higher than those of better-performing algorithms, indicating poor exploration and exploitation capabilities.

Best-Performing Algorithms

The algorithms that demonstrated the best performance, based on the lowest MeanBest values across the benchmark functions and dimensions, are:

- **Harris Hawks Optimization (HHO)**: HHO consistently achieved the lowest MeanBest values across all dimensions, with exceptional performance in 30D (e.g., 2.85e-22 for F_1, 7.63e-13 for F_2, and 0.4545 for F_5), 20D (e.g., 2.76e-24 for F_1, 4.94e-13 for F_2, and 0.1450 for F_5), and 10D (e.g., 1.24e-23 for

F_1, 1.31e-13 for F_2, and 0.12 for F_5). Its ability to balance exploration and exploitation, driven by dynamic escape energy, made it highly effective across all functions. *Speed*: HHO exhibits moderate computational speed, with MeanTotalT averaging 5.46s in 10D, 3.71s in 20D, and 3.91s in 30D. While slower than FOA in lower dimensions, its speed is competitive in higher dimensions, making it suitable for applications prioritizing accuracy with acceptable computational cost.

- **Grasshopper Optimization Algorithm (GOA)**: GOA also performed exceptionally well, particularly in 30D, with MeanBest values like 0.00134 for F_1, 0.00223 for F_2, and 0.00227 for F_5. In 20D and 10D, it maintained low MeanBest values, such as 3.44e-05 for F_1 (20D) and 3.44e-05 for F_1 (10D), demonstrating robust convergence through its social interaction model. *Speed*: GOA's speed is moderate, with MeanTotalT averaging 5.48s in 10D, 3.88s in 20D, and 3.89s in 30D. Its efficiency in higher dimensions makes it a strong choice for complex, high-dimensional optimization tasks.
- **Fruit Fly Optimization Algorithm (FOA)**: FOA showed strong performance, especially in higher dimensions, with MeanBest values of 1.14e-07 for F_1, 0.0184 for F_2, and 28.68 for F_5 in 30D. In 20D and 10D, it achieved values like 7.14e-08 for F_1 (20D) and 3.15e-08 for F_1 (10D), benefiting from its efficient smell-based search mechanism. *Speed*: FOA is among the fastest of the high-accuracy algorithms, with MeanTotalT averaging 4.36s in 10D, 4.14s in 20D, and 4.27s in 30D. Its simplicity ensures consistent speed, particularly for unimodal functions.
- **Grey Wolf Optimizer (GWO)**: GWO performed well, particularly in lower dimensions, with MeanBest values of 0.0133 for F_1, 0.0250 for F_2, and 31.28 for F_5 in 30D. In 20D and 10D, it achieved values like 0.0002419 for F_1 (20D) and 8.30e-07 for F_1 (10D), leveraging its hierarchical leadership structure for effective optimization. *Speed*: GWO is the fastest in 10D, with MeanTotalT averaging 2.77s, but significantly slower in 20D (5.78s) and 30D (5.99s), limiting its efficiency in high-dimensional problems.

These algorithms excelled in converging to near-optimal solutions, particularly on unimodal functions like F_1 and F_4, and showed competitive performance on the more challenging multimodal F_5. HHO and GOA stood out for their precision and stability, as evidenced by their low STDBest values, while FOA and GWO provided reliable performance across varying problem complexities. For the best balance of accuracy and speed, **HHO** and **GOA** are the top performers, with HHO offering unmatched accuracy and GOA providing robust accuracy with comparable speed, especially in 20D and 30D. **FOA** is the fastest among high-accuracy algorithms, ideal for simpler unimodal functions, while **GWO** is only competitive in 10D due to its speed but falters in higher dimensions.

Discussion

The superior performance of HHO, GOA, FOA, and GWO in terms of accuracy can be attributed to their effective mechanisms for balancing exploration and exploitation. HHO's dynamic escape energy and cooperative hunting strategies enabled adaptive solution refinement, achieving the lowest MeanBest values across all dimensions (e.g., 2.85e-22 for F_1 in 30D). GOA's social interaction model facilitated robust convergence, particularly in high-dimensional spaces (e.g., 0.00134 for F_1 in 30D). FOA's simplicity and smell-based search allowed rapid exploration, excelling in unimodal functions (e.g., 1.14e-07 for F_1 in 30D), though less effective on multimodal F_5. GWO's hierarchical structure provided stable guidance toward optima, performing well in lower dimensions (e.g., 8.30e-07 for F_1 in 10D) but struggling with F_5 in 30D. Regarding speed, HHO and GOA maintained moderate computational efficiency, with MeanTotalT averaging 3.71–3.91s (HHO) and 3.88–3.89s (GOA) in 20D and 30D, though slower in 10D (5.46s and 5.48s, respectively). FOA was consistently faster, averaging 4.14–4.36s across all dimensions, benefiting from its streamlined search mechanism. In contrast, GWO was the fastest in 10D (2.77s) but significantly slower in 20D (5.78s) and 30D (5.99s), limiting its scalability.

Conversely, CRO, PSO, DE, and ACO exhibited poor accuracy due to premature convergence and insufficient exploration, particularly in high-dimensional and multimodal landscapes (e.g., CRO's 61164 for F_1 in 30D). Their computational speeds were also suboptimal, with CRO averaging 4.96–6.27s and PSO/DE ranging from 3.11–4.72s, offering no significant advantage over the top performers. These findings highlight the importance of adaptive mechanisms for accuracy and computational efficiency for speed. HHO and GOA

emerged as the best algorithms for balancing high accuracy and reasonable speed, making them ideal for complex optimization tasks across dimensions. FOA excelled in speed for simpler unimodal problems, while GWO's efficiency was limited to low-dimensional scenarios, underscoring the need for scalable algorithms in high-dimensional optimization.

References

[1] Shu-Wei Cheng and Doddy Prayogo. Symbiotic organisms search: A new metaheuristic optimization algorithm. *Computers & Structures*, 139:98–112, 2014.

[2] Erik Cuevas, Mario Cienfuegos, Daniel Zaldívar, and Marco Pérez-Cisneros. A swarm optimization algorithm inspired in the behavior of social-spiders. *Expert Systems with Applications*, 40:6374–6384, 2013.

[3] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 26:29–41, 1996.

[4] Fadi Ali Hashim, Khaled Hussain, Essam Hassan Houssein, Mohamed S. Mabrouk, Walid Al-Atabany, and Seyedali Mirjalili. Honey badger algorithm. *Engineering Applications of Artificial Intelligence*, 97:104015, 2020.

[5] Ali Asghar Heidari, Seyedali Mirjalili, Hossam Faris, Ibrahim Aljarah, Majdi Mafarja, and Hu Chen. Harris hawks optimization: Algorithm and applications. *Future Generation Computer Systems*, 97:849–872, 2019.

[6] Dervis Karaboga and Bahriye Akay. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (abc) algorithm. *Journal of Global Optimization*, 39:459–471, 2009.

[7] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[8] Seyedali Mirjalili. Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. *Knowledge-Based Systems*, 89:228–249, 2015.

[9] Seyedali Mirjalili and Andrew Lewis. The whale optimization algorithm. *Advances in Engineering Software*, 95:51–67, 2016.

[10] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. Grey wolf optimizer. *Advances in Engineering Software*, 69:46–61, 2014.

[11] W. T. Pan. A new fruit fly optimization algorithm: Taking the financial distress model as an example. *Knowledge-Based Systems*, 26:69–74, 2012.

[12] Kevin M. Passino. Biomimicry of bacterial foraging for distributed optimization and control. *IEEE Control Systems Magazine*, 22:52–67, 2002.

[13] Sancho Salcedo-Sanz, Alejandro Pastor-Sánchez, Rosa Palacios, Juan García-Herrera, and Laureano Prieto. A coral reefs optimization algorithm for solving combinatorial optimization problems. *Expert Systems with Applications*, 41(16):7316–7324, 2014.

[14] Saeed Saremi, Seyedali Mirjalili, and Andrew Lewis. Grasshopper optimization algorithm: Theory and application. *Advances in Engineering Software*, 105:30–47, 2017.

[15] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[16] Gai-Ge Wang, Suash Deb, and Zhihua Cui. Elephant herding optimization. In *International Symposium on Computational and Business Intelligence (ISCBI)*, pages 1–5, 2015.

[17] Xin-She Yang. A new metaheuristic bat-inspired algorithm. *Nature Inspired Cooperative Strategies for Optimization*, pages 65–74, 2010.

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