EXP #4 : Cuckoo Search Algorithm

Code:

```
import numpy as np
import math
import random

def levy_flight(d):
    beta = 1.5
    sigma = (np.power(math.gamma(1 + beta), 1/beta) * np.sin(np.pi * beta / 2)) /
np.power(math.gamma((1 + beta) / 2), 1/beta)
    u = np.random.randn(d) * sigma
    v = np.random.randn(d)
    step = u / np.power(np.abs(v), 1/beta)
    return step

def cuckoo_search(fobj, dim, n, pa, itermax):
    """
    Cuckoo Search algorithm implementation.

    Args:
        fobj: Objective function to be minimized.
        dim: Number of dimensions.
        n: Population size.
        pa: Probability of using Lévy flight.
        itermax: Maximum number of iterations.

    Returns:
        best_sol: Best solution found.
        best_fitness: Fitness of the best solution.
    """

    # Initialize the population
    X = np.random.rand(n, dim)

    # Iterate until the maximum number of iterations
    for t in range(itermax):
        # Generate new solutions using Lévy flights
        for i in range(n):
            if np.random.rand() < pa:
                X[i] = X[i] + levy_flight(dim)

        # Evaluate the fitness of each solution
```

```
        fitness = np.array([fobj(x) for x in X])

        # Rank the solutions
        best_idx = np.argmin(fitness)
        best_sol = X[best_idx]

        # Replace worst solutions with new ones
        worst_idx = np.argmax(fitness)
        X[worst_idx] = best_sol + np.random.rand(dim)

    return best_sol, fitness[best_idx]

def gear_train_objective_function(x):
    N1, N2, N3, module = x

    # Calculate gear ratios
    GR1 = N2 / N1
    GR2 = N3 / N2

    # Calculate center distance
    CD = (N1 + N2) * module / 2 + (N2 + N3) * module / 2

    # Calculate cost, weight, noise, and space (simplified examples)
    cost = N1**2 + N2**2 + N3**2 + CD**2  # Replace with actual cost model
    weight = module**2 * (N1 + N2 + N3)  # Replace with actual weight calculation
    noise = 1 / module  # Simplified noise model, higher module, lower noise
    space = CD  # Center distance as a measure of space

    # Calculate objective function value
    obj_value = w1 * cost + w2 * weight + w3 * noise + w4 * space

    # Apply constraints
    if not (min_teeth <= N1 <= max_teeth and
            min_teeth <= N2 <= max_teeth and
            min_teeth <= N3 <= max_teeth and
            min_center_distance <= CD <= max_center_distance and
            abs(GR1 * GR2 - desired_gear_ratio) <= tolerance):
        obj_value += penalty_factor

    return obj_value

# Define problem parameters
dim = 4  # Number of design variables (N1, N2, N3, module)
n = 20  # Population size
```

```
pa = 0.25  # Probability of using Lévy flight
itermax = 100  # Maximum number of iterations
desired_gear_ratio = 10
min_teeth = 12
max_teeth = 60
min_center_distance = 50
max_center_distance = 200
w1, w2, w3, w4 = 0.2, 0.3, 0.4, 0.1  # Weights for objective function components
penalty_factor = 1000  # Penalty for violating constraints

# Run Cuckoo Search
best_sol, best_fitness = cuckoo_search(gear_train_objective_function, dim, n, pa, itermax)

print("Best gear train design:", best_sol)
print("Best fitness (objective function value):", best_fitness)
```

Output:

```
Best gear train design: [ -6.21109808  14.32747582 -23.87848038 -34.24269025]
Best fitness (objective function value): -4258.694535700955
```