Sanjana Shetty
5E

Cuckoo Search Algorithm

```
import numpy as np
import math

# Objective function for 1D (x^2)
def objective_function_1d(x):
    return x[0]**2  # x is a 1D array, even though we just care about the
first element

# Lévy Flight to generate new solutions
def levy_flight(num_dim, beta=1.5):
    sigma_u = (np.math.gamma(1 + beta) * np.sin(np.pi * beta / 2) /
               np.math.gamma((1 + beta) / 2) * beta * (2 ** ((beta - 1) /
2)))**(1 / beta)
    u = np.random.normal(0, sigma_u, num_dim)  # Lévy-distributed steps
    v = np.random.normal(0, 1, num_dim)
    return u / np.abs(v) ** (1 / beta)

# Cuckoo Search Algorithm for 1D
def cuckoo_search_1d(num_iterations, num_nests, pa=0.25):
    num_dim = 1  # 1D problem
    nests = np.random.rand(num_nests, num_dim) * 10 - 5  # Random
initialization within [-5, 5]
    fitness = np.apply_along_axis(objective_function_1d, 1, nests)  #
Evaluate initial fitness
    best_nest = nests[np.argmin(fitness)]
    best_fitness = np.min(fitness)

    for _ in range(num_iterations):
        for i in range(num_nests):
            new_nest = nests[i] + levy_flight(num_dim)  # Generate new
solution using Lévy flight
            new_fitness = objective_function_1d(new_nest)
```

```
            if new_fitness < fitness[i]:  # Replace if new solution is
better
                nests[i] = new_nest
                fitness[i] = new_fitness

        # Abandon the worst nests
        worst_nests = np.argsort(fitness)[-int(pa * num_nests):]
        for j in worst_nests:
            nests[j] = np.random.rand(num_dim) * 10 - 5  # Randomly
initialize new nests
            fitness[j] = objective_function_1d(nests[j])

        # Update best solution found so far
        current_best_idx = np.argmin(fitness)
        current_best_fitness = fitness[current_best_idx]
        if current_best_fitness < best_fitness:
            best_fitness = current_best_fitness
            best_nest = nests[current_best_idx]

    return best_nest, best_fitness  # Return the best solution and its
fitness

# Run the cuckoo search on the 1D problem
best_solution, best_fitness = cuckoo_search_1d(num_iterations=1000,
num_nests=25)
print(f"Best solution found: {best_solution} with objective value:
{best_fitness}")
```

OUTPUT:

```
<ipython-input-2-766b3d6e0184>:11: DeprecationWarning: `np.math` is a deprecated alias for
  np.math.gamma((1 + beta) / 2) * beta * (2 ** ((beta - 1) / 2)))**(1 / beta)
Best solution found: [-0.0001197] with objective value: 1.4328760925515824e-08
```