Lab-4

classmate
Date
Page

Cuckoo Search Algorithm

Algorithm:-

Input:

n: No of host nests (population size)

Pa: Fraction of worse nests to be abandoned

Max Iterations: Max no of iterations

f(x): Objective fn to minimize

Dimension: Dimensionality of the problem

Bounds: Lower and upper limits of the search space

Initialize:

1) Generate an initial population of n random host nests x_i (for $i = 1, 2, \ldots, n$)

2) Evaluate the fitness $f(x_i)$ for each nest. Determine the current best solution x^* with the best fitness $f(x^*)$

While the current iteration $t <$ Max Iterations:

Step 1: Perform Levy flight for randomly selected nest

Select a random nest x_i

Generate a new solution x' using Levy flight:

$$x' = x_i + \lambda \cdot L(x_i - x^*)$$

where L is the Levy flight step, λ is the step size

Clip x' within bounds, if necessary

Step 2 :- Evaluate the fitness $f(x')$ of the new solution.

If $f(x') < f(x_j)$, replace a randomly chosen nest x_j with x'

Step 3 :- Abandon worse nests

Identify Pa-n worst nests and replace them with new random solutions

Step 4 :- Update the current best solution.
Identify and retain the nest with best fitness and while

Post-process results:
Output the best solution x* and its fitness for

Program:-

```
import numpy as np
def objective_function(x):
    return sum(x**2)

def levy_flight(lambda, dimension, best, current):
    beta = 1.5
    sigma = (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, dimension)
    V = np.random.normal(0,1, dimension)
    step = u/abs(V)**(1/beta)
    step_size = step*(current - best)
    return current + step_size * lambda

def cuckoo_search(n, pa, max_iteration,
    dimension, lower_bound, upper_bound):
    nests = np.random.uniform(lower_bound, upper_bound,
        (n, dimension))
    fitness = np.array([objective_function(nest) for
        nest in nests])
```

```
best_solution = nests[np.argmin(fitness)]
best_fitness = min(fitness)

for iteration in range(max_iterations):
    cuckoo_index = np.random.randint(0, n)
    cuckoo = levy_flight(0.01, dimension,
      best_solution, nests[cuckoo_index])
    cuckoo = np.clip(cuckoo, lower_bound, upper_bound)
    cuckoo_fitness = objective_function(cuckoo)

    random_nest_index = np.random.randint(0, n)
    if cuckoo_fitness < fitness[random_nest_index]:
        nests[random_nest_index] = new_nest
        fitness[random_nest_index] = cuckoo_fitness

    worst_nest_indices = np.argsort(fitness)[-int(pa*n):]
    for worst_nest_index in worst_nest_indices:
        new_nest = np.random.uniform(lower_bound,
          upper_bound, dimension)
        nests[worst_best_index] = new_nest
        fitness[worst_nest_index] = objective_function(new_nest)

    current_best_index = np.argmin(fitness)
    if fitness[current_best_index] < best_fitness:
        best_solution = nests[current_index]
        best_fitness = fitness[current_best_index]
return best_solution, best_fitness
```

```
if __name__ == "__main__":
    n = 25
    pa = 0.25
    max_iterations = 100
    dimension = 5
    lower_bound = -10
    upper_bound = 10
    best_solution, best_fitness = cuckoo_search(n,
        pa, max_iterations, dimension, lower_bound,
            upper_bound)
    print(best_solution, best_fitness)
```

Output:

Best solution: [-0.68 1.34 2.188 -2.21 -0.8]

Best fitness = 12.6010