# Lab 5

## Cuckoo Search Optimization

17/10/25 LAB - V

CUCKOO SEARCH OPTIMIZATION

Pseudocode:

Initialize value is set to no. of schedules n, probability $\in (0,1)$ and maximum iterations MaxIter

Iteration counter $t = 0$

For i=1 to n do

  generate an initial schedule $S_i$ randomly

  evaluate the fitness func^n $f(S_i)$ = total completion time of schedule $S_i$

End for

while $t <$ MaxIter do

  Generate a new schedule $S_i'$ from $S_i$ using Levy Flight (small random changes in task order)

  Evaluate the fitness $f(S_i')$

  Randomly select a schedule $S_j$ among n schedules

  If $f(S_i') < f(S_j)$ then

    Replace $S_j$ with new schedule $S_i'$

  End if

  Abandon a fraction Pa of worst schedules and generate new schedules randomly

  Keep the best schedules found so far

  Rank all schedules by fitness & update the current best

  Increment iteration $t = t+1$

End while

Output the best schedule S_best

Output:

Iteration 1 : Best Fitness = 229

Iteration 2 : Best Fitness = 228

⋮

Iteration 100 : Best Fitness = 211

Best schedule : [6, 1, 3, 5, 0, 4, 7, 2]

Best Fitness : 211

Code:

```
import numpy as np
import math
print('Shreya Raj 1BM23CS317')
def objective_function(x):
    return np.sum(x**2)


def initialize_nests(num_nests, dim, lower_bound, upper_bound):
    return np.random.uniform(lower_bound, upper_bound, size=(num_nests, dim))


def levy_flight(Lambda, size):
    sigma = (math.gamma(1 + Lambda) * math.sin(math.pi * Lambda / 2) /
             (math.gamma((1 + Lambda) / 2) * Lambda * 2**((Lambda - 1) / 2))) ** (1 / Lambda)
    u = np.random.randn(*size) * sigma
    v = np.random.randn(*size)
    step = u / np.abs(v) ** (1 / Lambda)
    return step


def cuckoo_search(num_nests=25, dim=2, lower_bound=-10, upper_bound=10,
                  pa=0.25, max_iter=100):

    nests = initialize_nests(num_nests, dim, lower_bound, upper_bound)
    fitness = np.apply_along_axis(objective_function, 1, nests)

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

```
for t in range(max_iter):
    new_nests = nests + 0.01 * levy_flight(1.5, nests.shape) * (nests - best_nest)
    new_nests = np.clip(new_nests, lower_bound, upper_bound)

    new_fitness = np.apply_along_axis(objective_function, 1, new_nests)

    mask = new_fitness < fitness
    nests[mask] = new_nests[mask]
    fitness[mask] = new_fitness[mask]

    rand = np.random.rand(num_nests, dim)
    new_nests = np.where(rand > pa, nests,
                         initialize_nests(num_nests, dim, lower_bound, upper_bound))

    new_fitness = np.apply_along_axis(objective_function, 1, new_nests)
    mask = new_fitness < fitness
    nests[mask] = new_nests[mask]
    fitness[mask] = new_fitness[mask]

    if np.min(fitness) < best_fitness:
        best_nest = nests[np.argmin(fitness)].copy()
        best_fitness = np.min(fitness)

    print(f"Iteration {t+1}/{max_iter} | Best Fitness: {best_fitness:.6f}")

return best_nest, best_fitness
```

```
best_solution, best_value = cuckoo_search()

print("\nBest solution found:", best_solution)

print("Best fitness value:", best_value)
```

Output:

```
Shreya Raj 1BM23CS317
Iteration 1/100 | Best Fitness: 14.471185
Iteration 2/100 | Best Fitness: 14.471185
Iteration 3/100 | Best Fitness: 14.471185
Iteration 4/100 | Best Fitness: 0.894298
Iteration 5/100 | Best Fitness: 0.894298
Iteration 6/100 | Best Fitness: 0.894298
Iteration 7/100 | Best Fitness: 0.894298
Iteration 8/100 | Best Fitness: 0.894298
Iteration 9/100 | Best Fitness: 0.564269
Iteration 10/100 | Best Fitness: 0.564269
Iteration 11/100 | Best Fitness: 0.564269
Iteration 12/100 | Best Fitness: 0.457079
Iteration 13/100 | Best Fitness: 0.457079
Iteration 14/100 | Best Fitness: 0.457079
Iteration 15/100 | Best Fitness: 0.457079
Iteration 16/100 | Best Fitness: 0.457079
Iteration 17/100 | Best Fitness: 0.457079
Iteration 18/100 | Best Fitness: 0.457079
Iteration 19/100 | Best Fitness: 0.457079
Iteration 20/100 | Best Fitness: 0.457079
Iteration 21/100 | Best Fitness: 0.457079
Iteration 22/100 | Best Fitness: 0.457079
Iteration 23/100 | Best Fitness: 0.457079
Iteration 24/100 | Best Fitness: 0.457079
Iteration 25/100 | Best Fitness: 0.457079
Iteration 26/100 | Best Fitness: 0.457079
Iteration 27/100 | Best Fitness: 0.457079
```

```
Iteration 80/100 | Best Fitness: 0.000133
Iteration 81/100 | Best Fitness: 0.000133
Iteration 82/100 | Best Fitness: 0.000133
Iteration 83/100 | Best Fitness: 0.000133
Iteration 84/100 | Best Fitness: 0.000133
Iteration 85/100 | Best Fitness: 0.000133
Iteration 86/100 | Best Fitness: 0.000133
Iteration 87/100 | Best Fitness: 0.000133
Iteration 88/100 | Best Fitness: 0.000133
Iteration 89/100 | Best Fitness: 0.000133
Iteration 90/100 | Best Fitness: 0.000133
Iteration 91/100 | Best Fitness: 0.000133
Iteration 92/100 | Best Fitness: 0.000133
Iteration 93/100 | Best Fitness: 0.000133
Iteration 94/100 | Best Fitness: 0.000133
Iteration 95/100 | Best Fitness: 0.000133
Iteration 96/100 | Best Fitness: 0.000133
Iteration 97/100 | Best Fitness: 0.000133
Iteration 98/100 | Best Fitness: 0.000133
Iteration 99/100 | Best Fitness: 0.000133
Iteration 100/100 | Best Fitness: 0.000133

Best solution found: [-0.01150773  0.00076445]
Best fitness value: 0.0001330122491796386
```