

Program 5

Problem statement: Cuckoo Search (CS)

Algorithm:

1/11/2023 CUCKOO SEARCH ALGORITHM

- ① Each cuckoo lays one egg at a time and drops it in a randomly chosen nest.
- ② If the host nest with high quality eggs will be swapped over to the next generation.
- ③ No. of host nests = fixed $P = \text{prob of cuckoo egg being discovered}$ $P \in (0, 1) \rightarrow$ either raise it or abandon the egg or find a new nest.

Objective function $f(x) = x = (x_1, \dots, x_d)^T$

Generate initial population of n host nests x_i

while ($t < \text{Max generation}$) or (stop criterion)
[get a cuckoo randomly
 generate a solution by Levy flight
 Evaluate its solution quality or objective value f_j
 choose a nest among n ($x_{1:j}$) randomly
 if ($f_{\text{new}} < f_j$)
 Replace j by this new solution;
 end
] A (portion) of worse nests are abandoned
new nests/solutions are built/generated (local random walk)
keep best solution
~~Rank solutions and find current best~~
My task: $t \leftarrow t + 1$
end while

Lévy Flight

$$x_i^{t+1} = x_i^t + \lambda u(\lambda)$$

row
 column ϵ_{step} step
 solution x_3^t Lévy exponent

$$\log(\lambda) = g - \lambda \quad 1 < \lambda < 3$$

$u, v \sim \text{normal distribution}$

$$s = \frac{u}{|v|^\gamma \lambda} \quad \leftarrow \text{step length}$$

$$u \sim N(0, G^2 u)$$

$$v \sim N(0, G^2 v)$$

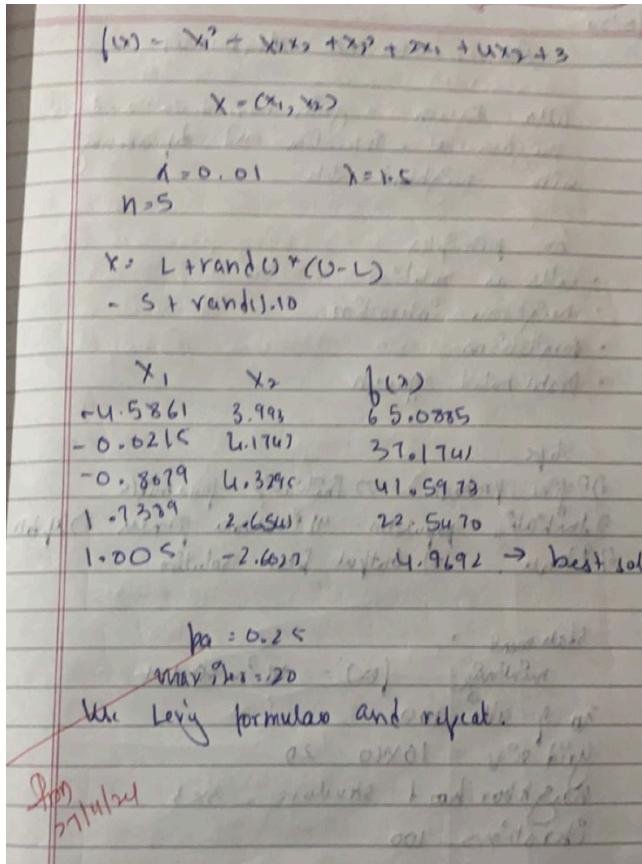
$$G_u = \sqrt{\frac{\Gamma(1+\alpha)}{\Gamma(1+\alpha/2) \times \alpha/2^{\alpha-1/2}}}$$

$$G^2 v = 1$$

local random walk random solution

$$x_i^{t+1} = x_i^t + h(p_a - \epsilon) \sum_j (x_j^t - x_i^t)$$

1 summing parameter
 heavy side function



Code:

```

import numpy as np
def objective_function(x):
    return np.sum(x**2)
class CuckooSearch:
    def __init__(self, objective_function, dim, num_nests, max_iter, pa=0.25, lb=-5, ub=5):
        self.objective_function = objective_function
        self.dim = dim
        self.num_nests = num_nests
        self.max_iter = max_iter
        self.pa = pa
        self.lb = lb
        self.ub = ub
        self.nests = np.random.uniform(self.lb, self.ub, (self.num_nests, self.dim))
        self.fitness = np.array([self.objective_function(nest) for nest in self.nests])
        self.best_nest = self.nests[np.argmin(self.fitness)]
        self.best_fitness = np.min(self.fitness)
    def levy_flight(self):
        step = np.random.normal(0, 1, self.dim) * np.random.uniform(0, 1)**(1/1.5)
        return step
    def generate_new_solution(self, nest):
        step = self.levy_flight()
        new_nest = nest + step

```

```

new_nest = np.clip(new_nest, self.lb, self.ub)
return new_nest
def abandon_worst_nests(self):
    num_worst = int(self.pa * self.num_nests)
    worst_indices = np.argsort(self.fitness)[-num_worst:]
    for i in worst_indices:
        new_nest = np.random.uniform(self.lb, self.ub, self.dim)
        self.nests[i] = new_nest
        self.fitness[i] = self.objective_function(new_nest)
def run(self):
    for t in range(self.max_iter):
        for i in range(self.num_nests):
            new_nest = self.generate_new_solution(self.nests[i])
            new_fitness = self.objective_function(new_nest)
            if new_fitness < self.fitness[i]:
                self.nests[i] = new_nest
                self.fitness[i] = new_fitness
            if new_fitness < self.best_fitness:
                self.best_nest = new_nest
                self.best_fitness = new_fitness
        self.abandon_worst_nests()
        print(f'Iteration {t+1}/{self.max_iter}: Best Fitness = {self.best_fitness}')
    return self.best_nest, self.best_fitness
dim = 2
num_nests = 30
max_iter = 10
pa = 0.25
lb = -5
ub = 5
cs = CuckooSearch(objective_function, dim, num_nests, max_iter, pa, lb, ub)
best_nest, best_fitness = cs.run()
print(f"\nBest Nest: {best_nest}")
print(f"Best Fitness: {best_fitness}")

```

```

import numpy as np
def objective_function(x):
    return np.sum(x**2)
class CuckooSearch:
    def __init__(self, objective_function, dim, num_nests, max_iter, pa=0.25, lb=-5, ub=5):
        self.objective_function = objective_function
        self.dim = dim
        self.num_nests = num_nests
        self.max_iter = max_iter
        self.pa = pa
        self.lb = lb
        self.ub = ub
        self.nests = np.random.uniform(self.lb, self.ub, (self.num_nests, self.dim))
        self.fitness = np.array([self.objective_function(nest) for nest in self.nests])
        self.best_nest = self.nests[np.argmin(self.fitness)]
        self.best_fitness = np.min(self.fitness)
    def levy_flight(self):
        step = np.random.normal(0, 1, self.dim) * np.random.uniform(0, 1)**(1/1.5)
        return step
    def generate_new_solution(self, nest):
        step = self.levy_flight()
        new_nest = nest + step
        new_nest = np.clip(new_nest, self.lb, self.ub)
        return new_nest
    def abandon_worst_nests(self):
        num_worst = int(self.pa * self.num_nests)
        worst_indices = np.argsort(self.fitness)[-num_worst:]
        for i in worst_indices:
            new_nest = np.random.uniform(self.lb, self.ub, self.dim)
            self.nests[i] = new_nest
            self.fitness[i] = self.objective_function(new_nest)
    def run(self):
        for t in range(self.max_iter):
            for i in range(self.num_nests):
                new_nest = self.generate_new_solution(self.nests[i])
                new_fitness = self.objective_function(new_nest)
                if new_fitness < self.fitness[i]:
                    self.nests[i] = new_nest
                    self.fitness[i] = new_fitness
                if new_fitness < self.best_fitness:
                    self.best_nest = new_nest
                    self.best_fitness = new_fitness
            self.abandon_worst_nests()
            print(f"Iteration {t+1}/{self.max_iter}: Best Fitness = {self.best_fitness}")
        return self.best_nest, self.best_fitness
dim = 2
num_nests = 30
max_iter = 10
pa = 0.25
lb = -5
ub = 5
cs = CuckooSearch(objective_function, dim, num_nests, max_iter, pa, lb, ub)
best_nest, best_fitness = cs.run()
print(f"\nBest Nest: {best_nest}")
print(f"Best Fitness: {best_fitness}")

```

Output:

```
⤵ Iteration 1/10: Best Fitness = 1.1321338482886207  
Iteration 2/10: Best Fitness = 1.1321338482886207  
Iteration 3/10: Best Fitness = 0.9581521162910784  
Iteration 4/10: Best Fitness = 0.9581521162910784  
Iteration 5/10: Best Fitness = 0.4939211503841384  
Iteration 6/10: Best Fitness = 0.026043904239381056  
Iteration 7/10: Best Fitness = 0.008822631651070133  
Iteration 8/10: Best Fitness = 0.003456235504629316  
Iteration 9/10: Best Fitness = 0.003456235504629316  
Iteration 10/10: Best Fitness = 0.003456235504629316  
  
Best Nest: [-0.01878645 -0.05570731]  
Best Fitness: 0.003456235504629316
```