14.11.24

Date / /
Page

Cuckoo Search (CS)

Purpose :- This is a nature optimized inspired optimization algorithm based on the brood parasitism of some cuckoo species. This involves laying eggs in the nests of other birds, leading to the optimization of survival strategies. CS uses levy flights to generate new solutions promoting global search capabilities and avoiding local minima.

Parameters :-

1. Number of nests (n_nests) : Controls how many candidate solutions (nests) are maintained throughout the algorithm.

2. Probability of Discovery (Pa) : Controls how often a cuckoo's egg is discovered by the host bird.

3. Number of Iterations (n_itee) : Defines how many times the algorithm will run to evolve the population of nests.

4. Levy Flight Parameter (beta) : This controls the step size distribution during levy flights.

5. Dimension of the problem (dim) : Defines the number of variables or decision parameters in the problem.

Date
Page

Applications of Cuckoo Search :-

- Engineering Design
- Machine learning
- Data Mining
- Control Systems
- Robotics
- Scheduling

Algorithm (Optimizes the Rastrigin function)

1. Define the objective function : func(x)
2. Initialise parameters :
   - n_nests : Number of nests (candidate solutions)
   - pa : Probability of discovery (how often eggs are discovered by birds)
   - n_iter : Number of iterations
   - beta : levy flight parameter (controls step size distribution)
   - dim : no. of decision variables
3. Initialise nests with random solutions in search space
   Nests = Initialise random population (size n_nests, dimension dim)
4. Evaluate fitness of each nest :
   Fitness[i] = func(Nests[i]) for all i from 1 to n_nests
5. Set the best solution as the initial best
   - Best_nest = Nests[best_index]
   - Best_fitness = Fitness[best_index]
6. For each iteration from ~~to~~ 1 to n_iter :
   a. Generate new solutions using levy flights :
      for each nest i in nests :

Step = levy_flight(beta, dim)
New_nest[i] = Nests[i] + step * (Nests[i] - Best_nests)
New_nest[i] = Clip new nest within search space bounds
Evaluate fitness of the new nest:
New_fitness = func(New_nest[i])
If New_fitness < Fitness[i], replace Nests[i] with New_nest[i]

b. Abandon the worst nests with probability pa:
For each nest i in nests:
if rand() < pa:
- Replace Nests[i] with new random solution
Evaluate fitness: Fitness[i] = func(Nests[i])

c. Update the best solution
Best_fitness = min(Fitness)
Best_nest = Nests[best index]

7. Output the best solution & its fitness
Return Best_nest, Best_fitness.

Output :-
Best_nest = [1.8805 0.10537]
Best_fitness = 1.3661853769607717

14-11-24