INDIAN INSTITUTE OF TECHNOLOGY
JAMMU

Moth Flame Optimisation Algorithm

Sahil Jindal
2016UEE0042

submitted to
Dr. Yamuna Prasad
Course : Natural language Processing

May 20,2020

Summary

I. Main Idea

The main inspiration of this optimizer is the navigation method of moths in nature called transverse orientation, where the moths move while maintaining a fixed angle to the light source.

II. Nature and Behavior of Moths

Moths fly in night by maintaining a fixed angle with respect to the moon, a very effective mechanism for travelling in a straight line for long distances. These insects are trapped in a useless/deadly spiral path around artificial lights. This paper mathematically models this behavior to perform optimization.

III. Moth Flame Optimization Summary

- In the proposed MFO algorithm, it is assumed that the candidate solutions are moths and the problems variables are the position of moths in the space.
- • This optimization algorithm works like particle swarm optimization and genetic algorithm.
- Since the Moth Flame Optimisation algorithm is a population-based algorithm, the set of moths is represented in a matrix M where n is the number of moths and d is the number of variables.

$$A = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & \cdots & m_{1,d} \\ m_{2,1} & m_{2,2} & \cdots & \cdots & m_{2,d} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m_{n,1} & m_{n,2} & \cdots & \cdots & m_{n,d} \end{bmatrix}$$

- For all the moths, we also assume that there is an array for storing the corresponding fitness values OM where n is the number of moths.

$$OM = \begin{bmatrix} OM_1 \\ OM_2 \\ \vdots \\ OM_n \end{bmatrix}$$

- Another key component in the proposed algorithm are flames.
- A matrix like the moth matrix is considered as F which has same dimension as M and has an array for storing corresponding fitness values in a matrix OF.

$$F = \begin{bmatrix} F_{1,1} & F_{1,2} & \dots & \dots & F_{1,d} \\ F_{2,1} & F_{2,2} & \dots & \dots & F_{2,d} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ F_{n,1} & F_{n,2} & \dots & \dots & F_{n,d} \end{bmatrix}$$

$$OF = \begin{bmatrix} OF_1 \\ OF_2 \\ \vdots \\ OF_n \end{bmatrix}$$

- Here the moths and the flames are both solutions, the only difference being the way they are iterated.
- The moths are actual search agents that move around the search space, whereas flames are the best position of moths that obtains so far.
- Hence, each moth searches around the flame and updates it if it finds a better solution. Thus, a moth never loses his best solution.
- Each moth is obliged to update its position using only one of the flames.
- The reason that why a specific flame is assigned to each moth is to prevent local optimum stagnation.
- If all the moths get attracted to a single flame, all of them converge to a point in the search spaces because they can only fly towards a flame and not outwards, requiring them to move around different flames, however, causes higher exploration of the search space and lower probability of local optima stagnation.
- A logarithmic spiral is chosen as the main update mechanism of moths, but any spiral can be chosen if the moth is the starting point and the flame is the ending point of the spiral and also the spiral should not leave the search space.
- A logarithmic spiral is defined as follows where Di indicates the distance of the i_{th} moth for the j_{th} flame, b is a constant for defining the shape of the logarithmic spiral, and t is a random number in [-1,1]. D is calculated as $|F_j - M_i|$ where M_i is the i_{th} moth, F_j is the j_{th} flame, and D_i is the distance of the i_{th} moth from the j_{th} flame.

$$S(M_i, F_j) = D_i \cdot e^{bt} \cdot \sin 2\pi t + F_j$$

- The t parameter in the spiral equation shows how much the next position of the moth should be close to the flame as $t = -1$ is the closest and $t = 1$ is the farthest position from the flame.
- The spiral equation allows a moth to fly around a flame and not necessarily in the space between them. To avoid the degradation of exploration, the following formula is used-

$$flame_no = round(N - l * \frac{N-1}{T})$$

- Here, I is the current number of iterations, N is maximum number of flames and T is the maximum number of iterations.

Complexity of algorithm is O(n)

IV. Algorithm

- The algorithm starts by initializing the population(x_i) with a random lower bound and upper bound.
- Sort x_i and set x_i as [x_i] matrix.
- Calculate the fitness function of each search agent.
- Calculate the best moth and the best fitness.
- Calculate the fitness function and tag the best position by the flames
- Update flame number, t and r.
- Calculate D for the corresponding moth.
- Update $M_{i,j}$ for the corresponding moth.
- If the termination criteria are satisfied, report the best position of the moth.

V. Code

```python
import numpy as np
import matplotlib.pyplot as plt

def F1(x): # fitness function
  return np.sum(np.power(x, 2), axis=1)

def MFO(nsa, dim, ub, lb, max_iter, fobj):

  # Initialize the positions of moths
  mothPos = np.random.uniform(low=lb, high=ub, size=(nsa, dim))

  convergenceCurve = np.zeros(shape=(max_iter))


  for iteration in range(max_iter):
    flameNo = int(np.ceil(nsa-(iteration+1)*((nsa-1)/max_iter)))
    mothPos = np.clip(mothPos, lb, ub)
    mothFit = fobj(mothPos)
    if iteration == 0:
      order = mothFit.argsort()
      mothFit = mothFit[order]
      mothPos = mothPos[order, :]
      bFlames = np.copy(mothPos)
      bFlamesFit = np.copy(mothFit)

    else:
      doublePop = np.vstack((bFlames, mothPos))
      doubleFit = np.hstack((bFlamesFit, mothFit))
      order = doubleFit.argsort()
      doubleFit = doubleFit[order]
      doublePop = doublePop[order, :]
      bFlames = doublePop[:nsa, :]
      bFlamesFit = doubleFit[:nsa]

    bFlameScore = bFlamesFit[0]
    bFlamesPos = bFlames[0, :]
    a = -1 + (iteration+1) * ((-1)/max_iter)
    distanceToFlames = np.abs(bFlames - mothPos)

    b = 1
    t = (a-1)*np.random.rand(nsa, dim) + 1
    temp1 = bFlames[:flameNo, :]
    temp2 = bFlames[flameNo-1, :]*np.ones(shape=(nsa-flameNo, dim))
    temp2 = np.vstack((temp1, temp2))
    mothPos = distanceToFlames*np.exp(b*t)*np.cos(t*2*np.pi) + temp2

    convergenceCurve[iteration] = bFlameScore
    if iteration %100 == 0:
        print("Iteration = ", iteration)
        print("Best Flame Score = ", bFlameScore)

  return bFlameScore, bFlamesPos, convergenceCurve

nsa = 10
max_iter = 1001

lb = -100
ub = 100
dim = 10

bFlameScore, bFlamesPos, convergenceCurve = MFO(nsa, dim, ub, lb,␣
 ↪max_iter, F1)

print("Final flames Score = ", bFlameScore)
print("Final flames position = ", bFlamesPos)
x = np.arange(0, max_iter, 1)
```

VI. Output

```
Iteration =  0
Best Flame Score =  25743.307095807824
Iteration =  100
Best Flame Score =  54.24248747378081
Iteration =  200
Best Flame Score =  5.195658575081754
Iteration =  300
Best Flame Score =  0.007602338536858343
Iteration =  400
Best Flame Score =  1.1778773993607353e-05
Iteration =  500
Best Flame Score =  2.3317591899468814e-07
Iteration =  600
Best Flame Score =  2.610348257728533e-09
Iteration =  700
Best Flame Score =  1.1019255825451095e-10
Iteration =  800
Best Flame Score =  1.0322801181574019e-10
Iteration =  900
Best Flame Score =  9.981675892074768e-11
Iteration =  1000
Best Flame Score =  9.981675886010315e-11
Final flames Score =  9.981675886010315e-11
Final flames position =  [ 2.61861220e-08 -9.98209190e-06  8.72711532e-09
-1.33211920e-07
 -1.23964067e-07  6.18037449e-08  2.31980556e-08  3.34705436e-07
  5.77463034e-08  1.44933083e-07]
```