Cuckoo Search Optimization (CSO) Via Levy Flights for heart disease prediction

Cuckoo search is an optimization algorithm developed by Xin-she Yang and Suash Deb. It was inspired by the obligate brood parasitism of some cuckoo species by laying their eggs in the nests of other host birds (of other species).

Algorithm:

```
Objective function:
                    $f(x), X = (x_1, x_2, x_3, \dots, x_d)$
Generate an initial population of $n$ host nests;
While ($t$ < MaxGeneration) or (stop criterion)
    Get a cuckoo's nest $i$ randomly and replace its solution by performing Lévy flights;
    Evaluate its quality/fitness $F_i$
    Choose a nest $j$ among $n$ randomly;
    if ($F_i > F_j$),
        Replace $j$ by the new solution;
    end if
    A fraction ($P_a$) of the worse nests are abandoned and new ones are built;
    Keep the best solutions/nests;
    Rank the solutions/nests and find the current best;
    Pass the current best solutions to the next generation;
end while
```

- An important advantage of this algorithm is its simplicity.
- In fact, compared with other population- or agent-based metaheuristic algorithms such as particle swarm optimization and harmony search, there is essentially only a single parameter P_a in CS (apart from the population size n).
- Therefore, it is very easy to be implemented.

Requirements:

Numpy

matplotlib

Once the installation is finished (download or cloning), go the cso folder and follow the below simple guidelines to execute CSO effectively (either write the code in command line or in a python editor).

from cso import CSO

Next, a fitness function (or cost function) is required. I have included four different fitness functions for example purposes namely fitness_1, fitness_2, fitness_3, and fitness_4.

Fitness-1 (Himmelblau's Function)

Minimize: $f(x) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$

Optimum solution: $x = 3$; $y = 2$

Fitness-2 (Booth's Function)

Minimize: $f(x) = (x + 2y - 7)^2 + (2x + y - 5)^2$

Optimum solution: $x = 1$; $y = 3$

Fitness-3 (Beale's Function)

Minimize: $f(x) = (1.5 - x - xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$

Optimum solution: $x = 3$; $y = 0.5$

Fitness-4

Maximize: $f(x) = 2xy + 2x - x^2 - 2y^2$

Optimum solution: $x = 2$; $y = 1$

Fitness-5 (Bivariate Michaelwicz function)

Minimize: $f(x) = -\sin(x)(\sin^{2m}(x^2/\pi)) - \sin(y)(\sin^{2m}(2y^2/\pi))$

When the bound is (x,y) ∈ (0,5) x (0,5) and m=10 [1]

Optimum solution: x = 2.20319 ; y = 1.57049

```
>>> from fitness import fitness_1, fitness_2, fitness_3, fitness_4, fitness_5
```

Now, if you want, you can provide bound values for all the particles (not mandatory) and optimize (minimize or maximize) the fitness function using CSO:

NOTE: a bool variable min=True (default value) for MINIMIZATION PROBLEM and min=False for MAXIMIZATION PROBLEM

```
>>> CSO(fitness=fitness_5, bound=[(0,5),(0,5)]).execute()
```

You will see the following similar output (there can be other minima as well):

```
OPTIMUM SOLUTION
  > [2.2028966, 1.5707915]
OPTIMUM FITNESS
  > -1.8013034
```

When fitness_4 is used, observe that min=False since it is a Maximization problem.

```
>>> CSO(fitness=fitness_4, bound=[(-4,4),(-4,4)], min=False).execute()
```

You will see the following similar output:

```
OPTIMUM SOLUTION
  > [2.0, 1.0]
OPTIMUM FITNESS
  > 2.0
```

Incase you want to print the fitness value for each iteration, set verbose=True (here Tmax=50 is the maximum iteration)

```
>>> CSO(fitness=fitness_2, Tmax=50, verbose=True).execute()
```

You will see the following similar output:

```
Iteration:    0 | best global fitness (cost): 4.2060194
Iteration:    1 | best global fitness (cost): 4.2060194
```

```
Iteration:    2 | best global fitness (cost): 4.2060194
Iteration:    3 | best global fitness (cost): 4.2060194
Iteration:    4 | best global fitness (cost): 4.2060194
Iteration:    5 | best global fitness (cost): 3.0228358
Iteration:    6 | best global fitness (cost): 2.0454478
Iteration:    7 | best global fitness (cost): 1.647782
Iteration:    8 | best global fitness (cost): 0.2005788
Iteration:    9 | best global fitness (cost): 0.1981048
Iteration:   10 | best global fitness (cost): 0.1981048
Iteration:   11 | best global fitness (cost): 0.1981048
Iteration:   12 | best global fitness (cost): 0.1981048
Iteration:   13 | best global fitness (cost): 0.1981048
Iteration:   14 | best global fitness (cost): 0.1981048
Iteration:   15 | best global fitness (cost): 0.1981048
Iteration:   16 | best global fitness (cost): 0.1981048
Iteration:   17 | best global fitness (cost): 0.1981048
Iteration:   18 | best global fitness (cost): 0.0547217
Iteration:   19 | best global fitness (cost): 0.0367725
Iteration:   20 | best global fitness (cost): 0.0367725
Iteration:   21 | best global fitness (cost): 0.0367725
Iteration:   22 | best global fitness (cost): 0.0367725
Iteration:   23 | best global fitness (cost): 0.0367725
Iteration:   24 | best global fitness (cost): 0.0367725
Iteration:   25 | best global fitness (cost): 0.0367725
Iteration:   26 | best global fitness (cost): 0.0367725
Iteration:   27 | best global fitness (cost): 0.0367725
Iteration:   28 | best global fitness (cost): 0.0367725
Iteration:   29 | best global fitness (cost): 0.0367725
Iteration:   30 | best global fitness (cost): 0.0367725
Iteration:   31 | best global fitness (cost): 0.0367725
```

```
Iteration:   32 | best global fitness (cost): 0.0367725
Iteration:   33 | best global fitness (cost): 0.0367725
Iteration:   34 | best global fitness (cost): 0.0367725
Iteration:   35 | best global fitness (cost): 0.0367725
Iteration:   36 | best global fitness (cost): 0.0367725
Iteration:   37 | best global fitness (cost): 0.0196146
Iteration:   38 | best global fitness (cost): 0.0087851
Iteration:   39 | best global fitness (cost): 0.0087851
Iteration:   40 | best global fitness (cost): 0.0087851
Iteration:   41 | best global fitness (cost): 0.0087851
Iteration:   42 | best global fitness (cost): 0.0087851
Iteration:   43 | best global fitness (cost): 0.0087851
Iteration:   44 | best global fitness (cost): 0.0087851
Iteration:   45 | best global fitness (cost): 0.0087851
Iteration:   46 | best global fitness (cost): 0.0068151
Iteration:   47 | best global fitness (cost): 0.0068151
Iteration:   48 | best global fitness (cost): 0.0068151
Iteration:   49 | best global fitness (cost): 0.0068151
OPTIMUM SOLUTION
  > [0.9965802, 3.0395979]
OPTIMUM FITNESS
  > 0.0068151
```

Now, in case you want to plot the fitness value for each iteration, then set plot=True (here Tmax=50 is the maximum iteration)

```
>>> CSO(fitness=fitness_2, Tmax=50, plot=True).execute()
```

You will see the following similar output:

```
OPTIMUM SOLUTION
  > [0.9965802, 3.0395979]


OPTIMUM FITNESS
  > 0.0068151
```

References:

1. X. Yang and Suash Deb, "Cuckoo Search via Lévy flights," 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC), 2009, pp. 210-214, doi: 10.1109/NABIC.2009.5393690.

2. I. Fister Jr., D. Fister, and I. Fister, "A comprehensive review of cuckoo search: variants and hybrids," International Journal of Mathematical Modelling and Numerical Optimisation, vol. 4, no. 4, pp. 387–409, 2013. View at: Publisher Site | Google Scholar

3. I. Fister Jr., X.-S. Yang, D. Fister, and I. Fister, "Cuckoo search: a brief literature review," in Cuckoo Search and Firefly Algorithm, X.-S. Yang, Ed., vol. 516 of Studies in Computational Intelligence, pp. 49–62, 2014. View at: Publisher Site | Google Scholar

4. R. B. Payne, M. D. Sorenson, and K. Klitz, The Cuckoos, Oxford University Press, 2005.

5.http://www.scientificcomputing.com/news-DA-Novel-Cuckoo-Search-Algorithm-Beats-Particle-Swarm-Optimization-060110.aspx.

6. http://en.wikipedia.org/wiki/Cuckoo_search.

7. Liu, X., & Fu, H. (2014). PSO-based support vector machine with cuckoo search technique for clinical disease diagnoses. The Scientific World Journal, 2014.