Artificial Bee Colony

March 30, 2021

[16]:

```
from inspect import signature
import numpy as np
import random
import pandas as pd
from statistics import median
```

[3]:

```
class Colony:

    def __init__(self, popSize, optimizationFunction, parameterConstraints,␣
 ↪iterations):

        self.popSize = popSize
        self.optimizationFunction = optimizationFunction
        self.parameterConstraints = parameterConstraints
        self.functionParameters = signature(optimizationFunction).parameters
        self.halfPop = int(popSize / 2) if popSize % 2 == 0 else int((popSize +␣
 ↪1) / 2)
        initialParameters = [[np.random.uniform(parameterConstraints[param][0],␣
 ↪parameterConstraints[param][1],1)[0] for param in self.functionParameters]␣
 ↪for i in range(self.halfPop)]
        # Need to update this one in case there are more parameters
        self.workerBees =␣
 ↪[WorkerBee(initialParameters[i],optimizationFunction(initialParameters[i][0],initialParamet
 ↪for i in range(self.halfPop)]
        self.onlookerBees = []
        self.limit = (popSize * len(self.functionParameters)) / 2
        self.iterations = iterations

    def RunSimulation(self):

        iteration = 0
        while iteration < self.iterations:
            self.GetNeighborValue()
            self.GetProbabilityVector()
            self.GetCurrentBestValue()
            self.CheckAbondonedBees()
            iteration += 1
        return self.bestValue

    def CheckAbondonedBees(self):

        for bee in self.workerBees:

            if bee.currentLimit == 6:
                parameters = [np.random.uniform(self.
↪parameterConstraints[param][0], self.parameterConstraints[param][1],1)[0]␣
↪for param in self.functionParameters]
                bee = WorkerBee(parameters,self.
↪optimizationFunction(parameters[0],parameters[1]))

    def GetNeighborValue(self):

        for bee in self.workerBees:

            parameterIndex = random.randint(0, len(self.functionParameters) - 1)

            beeIndex = self.GetComparisonBeeIndex(parameterIndex, bee)
            bee.updatedParameters[parameterIndex] = self.
↪GetUpdatedValue(parameterIndex, bee, beeIndex)
            newFitness = 1 / (1 + self.optimizationFunction(bee.
↪updatedParameters[0], bee.updatedParameters[1]))
            if newFitness > bee.fitnessValue:
                bee.fitnessValue = newFitness
                bee.currentParameters = bee.updatedParameters
            else:
                bee
                bee.currentLimit += 1


    def GetComparisonBeeIndex(self, parameterIndex, workerBee):

        while True:
            index = random.randint(0, len(self.workerBees) - 1)
            if self.workerBees.index(workerBee) != index:
                return index

    def GetUpdatedValue(self, parameterIndex, workerBee, beeIndex):

        ## Need to udpate this one as well
        while True:
            updatedValue = workerBee.currentParameters[parameterIndex] + (np.
↪random.uniform(-1,1,1)[0] * (workerBee.currentParameters[parameterIndex] -␣
↪self.workerBees[beeIndex].currentParameters[parameterIndex]))
            if -5 <= updatedValue <= 5:
                return updatedValue

    def GetProbabilityVector(self):

        probabilities = np.array([workerBee.fitnessValue for workerBee in self.
↪workerBees])
        probabilities = (probabilities / sum(probabilities)).cumsum()

        for i in range(self.halfPop):
            value = np.random.uniform(0,1,1)[0]
            parameterIndex = random.randint(0, len(self.functionParameters) - 1)
            beeIndex = probabilities.argsort()[probabilities > value][0]
            self.onlookerBees.append(self.workerBees[beeIndex])
            bee = self.onlookerBees[i]
            bee.updatedParameters[parameterIndex] = self.
↪GetUpdatedValue(parameterIndex, bee, beeIndex)
            newFitness = 1 / (1 + self.optimizationFunction(bee.
↪updatedParameters[0], bee.updatedParameters[1]))
            bee.fitnessValue = newFitness
            bee.currentParameters = bee.updatedParameters

    def GetCurrentBestValue(self):

        fitnessValues = np.array([bee.fitnessValue for bee in self.
↪onlookerBees])
        maxIndex = fitnessValues[fitnessValues.argsort()[len(fitnessValues) -␣
↪1]]
        self.bestValue = list(self.onlookerBees[int(maxIndex)].
↪currentParameters)
        self.onlookerBees = []
```

[4]:

```
class WorkerBee:

    def __init__(self, initialParameters, initialFunctionValue):
        self.currentLimit = 0
        self.currentParameters = initialParameters
        self.functionValue = initialFunctionValue
        self.fitnessValue = 1 / (1 + self.functionValue)
        self.updatedParameters = list(self.currentParameters)
```

[5]:

```
class OnlookerBee:

    def __init__(self, workerBee):
        self.workerBee = workerBee
        self.fitnessValue = workerBee.fitnessValue
        self.currentParameters = list(workerBee.currentParameters)
        self.updatedParameters = list(workerBee.currentParameters)
```

[11]:

```
func = lambda x, y: (x ** 2 + y ** 2)
```

[6]:

```
funcConstraints = {'x': (-5, 5), 'y': (-5, 5)}
```

[9]:

```
populationSizes = [6, 20, 50, 100]
iterations = [10, 100, 1000]
```

[12]:

```
simulations = []
for pop in populationSizes:
    for iteration in iterations:
        newList = []
        for i in range(100):
            newList.append(Colony(pop, func, funcConstraints, iteration).
 ↪RunSimulation())
        simulations.append(list(newList))
```

[13]:

```
functionValues = [[func(sim[0], sim[1]) for sim in values] for values in␣
 ↪simulations]
```

[21]:

```
functionAvgs = [sum(values) / len(values) for values in functionValues]
functionMedian = [median(values) for values in functionValues]
functionBest = [min(values) for values in functionValues]
functionWorst = [max(values) for values in functionValues]
```

[27]:

```
params = {'Pop Size':[], 'Iterations':[], 'Average':[], 'Median':[], 'Best':[],␣
 ↪'Worst':[] }
counter = 0
for pop in populationSizes:
    for iteration in iterations:
        print(pop, iteration)
        params['Pop Size'].append(pop)
        params['Iterations'].append(iteration)
        params['Average'].append(functionAvgs[counter])
        params['Median'].append(functionMedian[counter])
        params['Best'].append(functionBest[counter])
        params['Worst'].append(functionWorst[counter])
        counter += 1

dfFinalValues = pd.DataFrame(params)
```

```
6 10
6 100
6 1000
20 10
```

```
20 100
20 1000
50 10
50 100
50 1000
100 10
100 100
100 1000
```

```
[ ]: dfFinalValues.sort_values('Median').to_excel(r'C:
      ↪\Users\Crique\Desktop\Montclair\Optimization\Mini Project 2\RunValues.xlsx',␣
      ↪'RunValues')
```

```
[28]: dfFinalValues.sort_values('Median')
```

```
[28]:     Pop Size  Iterations    Average    Median      Best      Worst
      8         50        1000   5.760590  2.739866  0.015276  40.403542
      10       100         100   5.039346  2.775214  0.089167  19.896095
      7         50         100   5.069313  2.892166  0.045277  24.524966
      11       100        1000   6.219872  3.172488  0.027147  44.318755
      6         50          10   5.653799  3.216454  0.014587  24.948187
      4         20         100   6.006169  3.536928  0.003823  31.715841
      9        100          10   5.247839  3.817527  0.039891  26.458109
      3         20          10   7.319009  3.892097  0.003264  32.510191
      5         20        1000   7.110129  5.406135  0.001738  26.950626
      0          6          10   9.867372  8.101616  0.030834  38.745527
      1          6         100  11.042840  8.910618  0.165666  33.154316
      2          6        1000  11.395756  9.219872  0.195069  41.528791
```