

**#Genetic\_Algorithm\_Number\_Problem and Genetic\_Algorithm\_8QueensProblem**

**ABUBAKAR ASIF**

**#abubakarasif3111@gmail.com**

**#https://github.com/Abubakar3111**

**#https://www.linkedin.com/in/abubakar-asif-b3b94021a/**

**Artificial Intelligence**

**SUBMITTED TO:DR SHOAIB AZMAT**

**#Genetic\_Algorithm\_Number\_Problem:**

from random import randint,random

def individual(length,imin,imax):

return [randint(imin, imax) for i in range(length)]

def population(count,length,imin,imax):

return [individual(length, imin, imax) for i in range(count)]

def fitness(individual,target):

total=sum(individual)

return abs(total-target)

def graded(population,target):

pf1=[fitness(i, target) for i in population]

af=sum(pf1)/(len(population)\*1.0)

return af

def evolve(p,target,r=0.2,c=0.05,m=0.01):

fil=[(fitness(i, target),i) for i in p ]

spi=[i for (f,i) in sorted(fil)]

r1=int(len(p)\*r)

#spi=[t[1] for t in sp]

parents=spi[:r1]

for i in spi[r1:]:

if c>random():

parents.append(i)

pn=[]

while len(pn) <len(p):

i1=randint(0,len(parents)-1)

i2=randint(0,len(parents)-1)

if i1==i2:

continue

p1=parents[i1]

p2=parents[i2]

p3=p1[0:5]+p2[5:10]

p4=p2[0:5]+p1[5:10]

pn.append(p3)

pn.append(p4)

for i in pn:

if random()<m:

dtm=randint(0,len(i)-1)

i[dtm]=randint(min(i), max(i))

return pn

a=population(40,10,1,100)

i=0

while i<10:

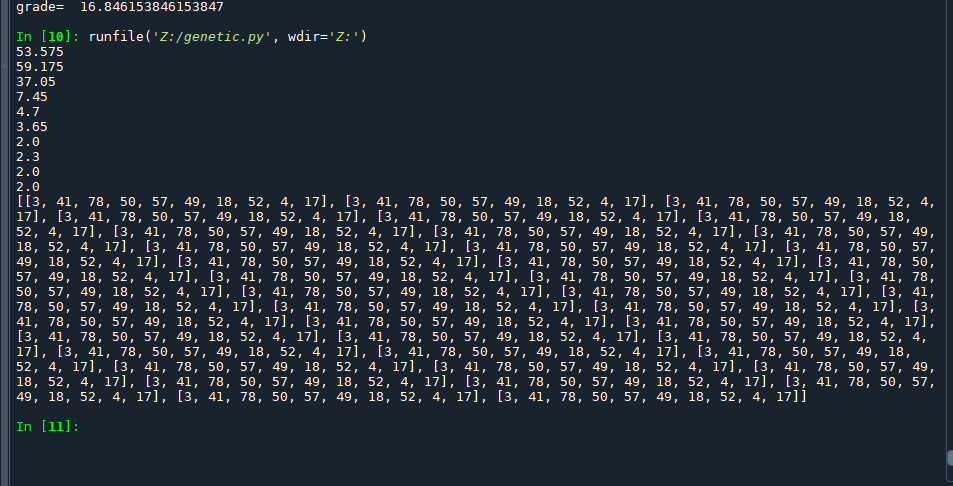
z=evolve(a,371)

a=z

print(graded(a,371))

i+=1

print(a)



**#Genetic\_Algorithm\_8QueensProblem:**

#Function to determine the fitness for 8-Queen problem

from random import randint,random

#Function to randomly generate Individual

def individual(length,imin,imax):

return [randint(imin,imax) for i in range(length)]

def fitness(individual,target=0):

attacks=0

for i in range(len(individual)):

for j in range(len(individual)):

iin=individual[i]

jin=individual[j]

#Check if column number is same

if i == j:

continue

#Check if row number is same

if iin==jin:

attacks +=1

continue

#Check if queens are in same diagonal

if iin-i == jin-j or iin+i == jin+j:

attacks +=1

#Return number of attacking pairs

return attacks

#Function to randomly generate a population of individuals

def population(count,length,imin,imax):

return [individual(length, imin, imax) for i in range(count)]

#Function to determine the average fitness of a population

def graded(population,target):

pfl=[fitness(i, target) for i in population]

return sum(pfl)/(len(population)\*1.0)

#Helper function to cross two parents together

def crossover(parents):

children=[]

while len(children) <len(parents):

rand1=randint(0,len(parents)-1)

rand2=randint(0,len(parents)-1)

if rand1 == rand2:

continue

p1=parents[rand1]

p2=parents[rand2]

half=int(len(p1)/2)

p1\_h=p1[:half]

p2\_h=p2[half:]

c1=p1\_h+p2\_h

c2=p2\_h+p1\_h

children.append(c1)

children.append(c2)

return children

#Function to mutate the individual after crossover

def mutate(pn,m):

for i in pn:

if random() < m:

dtm = randint(0,len(i)-1)

i[dtm]= randint(min(i),max(i))

return pn

#Function to create the next generation of individuals

def evolve(p,target,r=0.2,c=0.05,m=0.01):

#pick fittest

fil=[(fitness(i,target),i) for i in p]

#sort the list based on fitness

sp=sorted(fil)

spi=[i for (f,i) in sp]

#sp = [i for (f,i) in sorted(fil)]

r1 =int(len(p)\*r)

#spi= [ i[1] for i in sp]

parents=spi

# parents=spi[:r1] # for i in range(rl)]

# for i in spi[r1:]:

# #To make 5% Probability

# if random() < c:

# parents.append(i)

# pn=crossover(parents)

pn=crossover(parents)

mpn=mutate(pn,m)

return mpn

#Main Genetic Algorithm function

defGeneticAlgorithm(target):

for i in range(100):

p=population(25, 8, 1,8 )

p=evolve(p,target)

print("grade= ",graded(p, target))

#To find the solution for target value 50 of sum of list

#GeneticAlgorithm(50)

#To find the solution to 8-Queen Problem

GeneticAlgorithm(0)

