import numpy as np

import random

def fitness(population, n):

clash=np.zeros(len(population))

for q,chromosome in enumerate(population):

cr = chromosome.tolist()

row\_col\_clash= sum([cr.count(col)-1 for col in cr])/2

clash[q]+=row\_col\_clash

cl=0

for i in range(len(chromosome)):

for j in range(len(chromosome)):

if ( i != j):

dx = abs(i-j)

dy = abs(chromosome[i] - chromosome[j])

if dx == dy:

cl+=1

clash[q]+=(cl//2)

return 28-clash

def select(population, fit):

fitness\_values=fitness(population,n)

probability=fitness\_values/sum(fitness\_values)

selected\_fitness=np.random.choice(fitness\_values,1,True,p=probability)

parent\_index=fitness\_values.tolist().index(selected\_fitness[0])

# a = [0,1,2,3,4]

# size = 1

# p = [.31, .29, 0.26, 0.14]

# '''take input: population and fit

# fit contains fitness values of each of the individuals

# in the population

# return: one individual randomly giving

# #more weight to ones which have high fitness score'''

return population[parent\_index]

def crossover(x, y):

n = len(x)

child = []

for i in range(n):

child.append(-1)

cross\_over\_point = random.randint(1, n - 1)

for i in range(cross\_over\_point):

child[i] = x[i]

for i in range(cross\_over\_point, n):

child[i]=y[i]

return child

# '''take input: 2 parents - x, y.

# Generate a random crossover point.

# Append first half of x with second

# half of y to create the child

# returns: a child chromosome'''

# return

def mutate(child):

index\_to\_mutate = random.randint(0,7)

gene\_to\_place = random.randint(1,8)

child[index\_to\_mutate]=gene\_to\_place

return child

# '''take input: a child

# mutates a random

# gene into another random gene

# returns: mutated child'''

#return

def GA(population, n, mutation\_threshold = 0.3):

generation\_fitness=[]

generations = 0

while 28 not in fitness(population,n) :

new\_population = []

for i in range(len(population)):

x=select(population,fitness)

y=select(population,fitness)

#select funtion

child = crossover(x,y)

if random.uniform(0.0,0.5) < mutation\_threshold:

child = mutate(child)

new\_population.append(child)

population = np.array(new\_population)

population\_fitness = fitness(population,n)

generation\_fitness.append(np.max(population\_fitness))

generation\_fitness.append(np.min(population\_fitness))

generation\_fitness.append(np.mean(population\_fitness))

generations+=1

#getting the best fitness from the last generation

best\_fitness = np.max(fitness(population))

#adding the best fitness value to a list to plot the evolution

generation\_fitness.append(best\_fitness)

#finding the index of the best fitness value

index = fitness(population).tolist().index(best\_fitness)

#using this index value to find result chromosome

individual= population[index]

#print

print(f'selected individual --> {individual} \n')

print(f'fitness --> {best\_fitness} \n')

print(f'Generations --> {generations} \n')

print(f'{generation\_fitness} \n')

return

# '''implement the pseudocode here by

# calling the necessary functions- Fitness,

# Selection, Crossover and Mutation

# print: the max fitness value and the

# chromosome that generated it which is ultimately

# the solution board'''

'''for 8 queen problem, n = 8'''

n = 8

'''start\_population denotes how many individuals/chromosomes are there

in the initial population n = 8'''

start\_population = 10

'''if you want you can set mutation\_threshold to a higher value,

to increase the chances of mutation'''

mutation\_threshold = 0.3

'''creating the population with random integers between 0 to 7 inclusive

for n = 8 queen problem'''

population = np.random.randint(0, n, (start\_population, n))

'''calling the GA function'''

GA(population, n, mutation\_threshold)