def probability(n, fitness):

return fitness(n) / desired\_fitness

def select(population, probabilities):

populationWithProbability = zip(population, probabilities)

total = sum(y for x, y in populationWithProbability)

least\_value = random.uniform((total/4), total)

upto = 0

for chromosome, probability in zip(population, probabilities):

if upto + probability > least\_value:

return chromosome

upto += probability

def crossover(x, y):

index = random.randint(0,8)

right = y[index:8]

left = x[:index]

left.extend(right)

return left

def fitness(population):

diagonal\_collisions = 0

length = len(population)

left\_diagonal = [0] \* 2\*length

right\_diagonal = [0] \* 2\*length

for i in range(2\*length-1):

tracker = 0

if left\_diagonal[i] > 1:

tracker += left\_diagonal[i]-1

if right\_diagonal[i] > 1:

tracker += right\_diagonal[i]-1

diagonal\_collisions += tracker / (length-abs(i-length+1))

for i in range(length):

left\_diagonal[i + population[i] - 1] += 1

right\_diagonal[len(population) - i + population[i] - 2] += 1

row\_collisions = sum([population.count(i)-1 for i in population])/2

return int(desired\_fitness - (row\_collisions + diagonal\_collisions))

def mutate(child):

length = len(child)

index = random.randint(0, length - 1)

mutated\_value = random.randint(1, length)

child[index] = mutated\_value

return child

def genetic\_algorithm(population, fitness, mutation\_threshold):

new\_population = []

probabilities = []

for n in population:

probabilities.append(probability(n, fitness))

for i in range(len(population)):

x = select(population, probabilities)

y = select(population, probabilities)

child = crossover(x, y)

if random.random() < mutation\_threshold:

child = mutate(child)

new\_population.append(child)

if fitness(child) == desired\_fitness:

break

return new\_population

def print\_result(population,desired\_fitness,n,generation):

for i in population:

if fitness(i) == desired\_fitness:

print ("...done. \n\nresult ", i," found in Solved in {} generations.".format(generation-1))

import random

n = 8

start\_population = 10

mutation\_threshold = 0.3

desired\_fitness = (n\*(n-1))/2

generation = 1

population = []

for i in range(10):

population.append([ random.randint(1, n) for i in range(n) ])

print("Goal Fit for current population: ", int(desired\_fitness),"\n")

print("Problem dimension:",n,"x",n)

print("Population size: ",start\_population)

print ("\nrunning...\n")

while not desired\_fitness in [fitness(i) for i in population]:

population = genetic\_algorithm(population, fitness, mutation\_threshold)

generation += 1

print\_result(population,desired\_fitness,n,generation)