UCS1504 - Artificial Intelligence Lab Name: *Krithika Swaminathan*
AY: *2022-23* Roll No.: *205001057*

Exercise 5 – Genetic algorithm: Solving 8 Queens' Problem

Date: 22/09/2022

Aim:

To place 8 queens in a chessboard of 8x8 such that no queen is under attack from any other queen in horizontal, vertical and diagonal directions, using the Genetic Algorithm.

For the given problem description do the following.

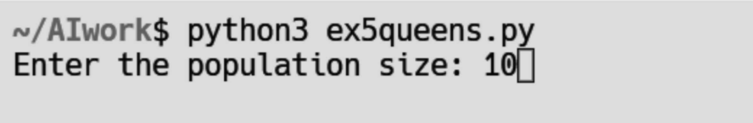
1. Find the suitable fitness function to solve the 8 queens' problem.
   Hint: non-attacking pairs of queens in horizontal, vertical and diagonal
2. Implement Genetic algorithm to find any one safe configuration.
3. Analyze the time complexity (no. of evolutions) by changing the K value.

Code:

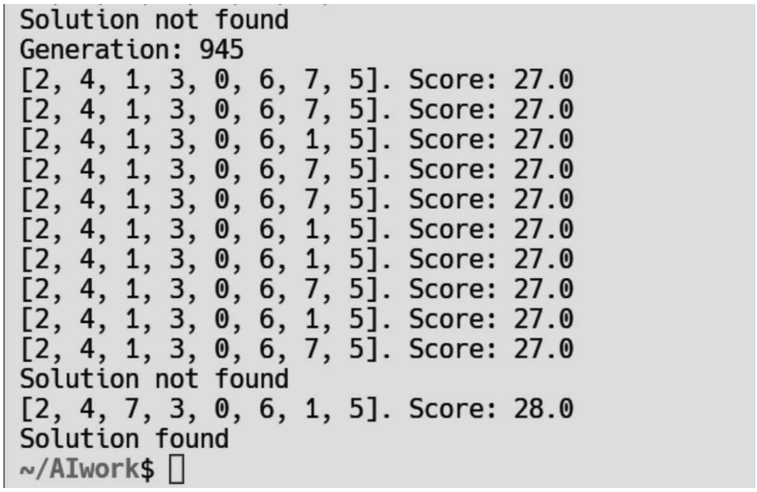
```
#Constants, experiment parameters
# k = 10
NUM_QUEENS = 8
POPULATION_SIZE = 10
MIXING_NUMBER = 2
MUTATION_RATE = 0.05

def fitness_score(seq):
    score = 0

    for row in range(NUM_QUEENS):
        col = seq[row]

        for other_row in range(NUM_QUEENS):

            #queens cannot pair with itself
            if other_row == row:
                continue
            if seq[other_row] == col:
                continue
            if other_row + seq[other_row] == row + col:
                continue
            if other_row - seq[other_row] == row - col:
                continue
            #score++ if every pair of queens are non-attacking.
            score += 1

    #divide by 2 as pairs of queens are commutative
    return score/2

import random
from scipy import special as sc

def selection(population):
    parents = []

    for ind in population:
        #select parents with probability proportional to their fitness score
        if random.randrange(sc.comb(NUM_QUEENS, 2)*2) < fitness_score(ind):
            parents.append(ind)


    return parents

import itertools

def crossover(parents):

    #random indexes to to cross states with
    cross_points = random.sample(range(NUM_QUEENS), MIXING_NUMBER - 1)
    offsprings = []

    #all permutations of parents
    permutations = list(itertools.permutations(parents, MIXING_NUMBER))

    for perm in permutations:
        offspring = []

        #track starting index of sublist
        start_pt = 0

        for parent_idx, cross_point in enumerate(cross_points): #doesn't account for last parent

            #sublist of parent to be crossed
            parent_part = perm[parent_idx][start_pt:cross_point]
            offspring.append(parent_part)

            #update index pointer
            start_pt = cross_point

        #last parent
        last_parent = perm[-1]
        parent_part = last_parent[cross_point:]
        offspring.append(parent_part)

        #flatten the list since append works kinda differently
        offsprings.append(list(itertools.chain(*offspring)))

    return offsprings

def mutate(seq):
  for row in range(len(seq)):
    if random.random() < MUTATION_RATE:
      seq[row] = random.randrange(NUM_QUEENS)

  return seq

def print_found_goal(population, to_print=True):
    for ind in population:
        score = fitness_score(ind)
        if to_print:
            print(f'{ind}. Score: {score}')
        if score == sc.comb(NUM_QUEENS, 2):
            if to_print:
                print('Solution found')
            return True

    if to_print:
        print('Solution not found')
    return False

def evolution(population):

    #select individuals to become parents
    parents = selection(population)

    #recombination. Create new offsprings
    offsprings = crossover(parents)
    #mutation
    offsprings = list(map(mutate, offsprings))

    #introduce top-scoring individuals from previous generation and keep top fitness individuals
    new_gen = offsprings

    for ind in population:
        new_gen.append(ind)

    new_gen = sorted(new_gen, key=lambda ind: fitness_score(ind),
reverse=True)[:POPULATION_SIZE]

    return new_gen

def generate_population():
    population = []

    for individual in range(POPULATION_SIZE):
        new = [random.randrange(NUM_QUEENS) for idx in range(NUM_QUEENS)]
        population.append(new)

    return population

#Running the experiment

generation = 0

k = int(input("Enter the population size: "))
POPULATION_SIZE = k

#generate random population
population = generate_population()

while not print_found_goal(population):
    print(f'Generation: {generation}')
    print_found_goal(population)
    population = evolution(population)
    generation += 1
```

Output:

k = 10

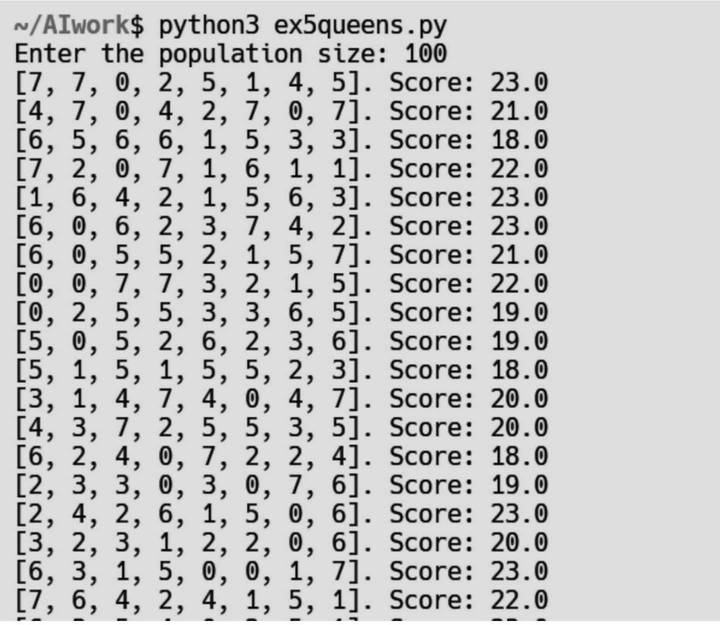


```
~/AIwork$ python3 ex5queens.py
Enter the population size: 10□
```

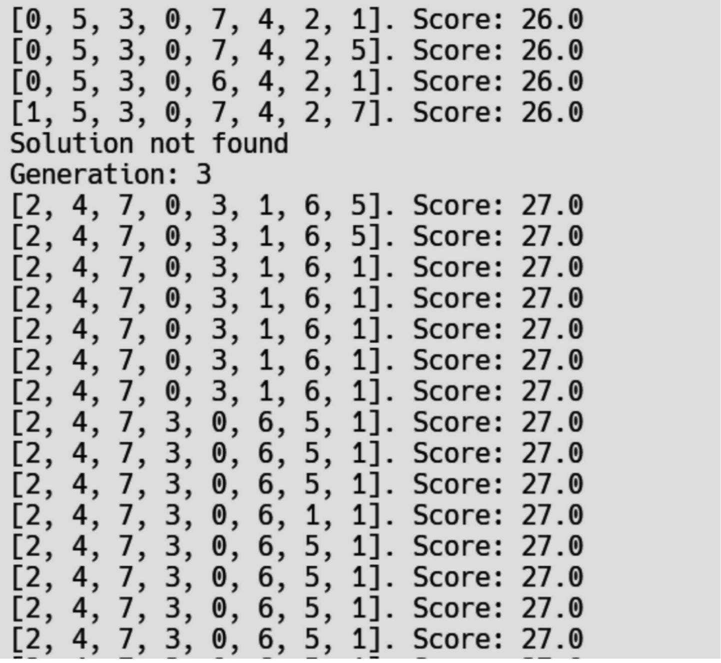


```
Solution not found
Generation: 945
[2, 4, 1, 3, 0, 6, 7, 5]. Score: 27.0
[2, 4, 1, 3, 0, 6, 7, 5]. Score: 27.0
[2, 4, 1, 3, 0, 6, 1, 5]. Score: 27.0
[2, 4, 1, 3, 0, 6, 7, 5]. Score: 27.0
[2, 4, 1, 3, 0, 6, 7, 5]. Score: 27.0
[2, 4, 1, 3, 0, 6, 1, 5]. Score: 27.0
[2, 4, 1, 3, 0, 6, 1, 5]. Score: 27.0
[2, 4, 1, 3, 0, 6, 7, 5]. Score: 27.0
[2, 4, 1, 3, 0, 6, 1, 5]. Score: 27.0
[2, 4, 1, 3, 0, 6, 7, 5]. Score: 27.0
Solution not found
[2, 4, 7, 3, 0, 6, 1, 5]. Score: 28.0
Solution found
~/AIwork$ □
```

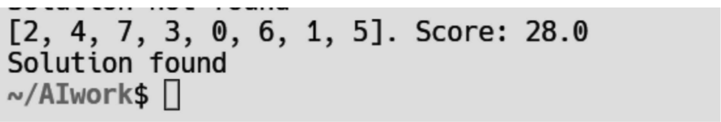
k = 100



```
~/AIwork$ python3 ex5queens.py
Enter the population size: 100
[7, 7, 0, 2, 5, 1, 4, 5]. Score: 23.0
[4, 7, 0, 4, 2, 7, 0, 7]. Score: 21.0
[6, 5, 6, 6, 1, 5, 3, 3]. Score: 18.0
[7, 2, 0, 7, 1, 6, 1, 1]. Score: 22.0
[1, 6, 4, 2, 1, 5, 6, 3]. Score: 23.0
[6, 0, 6, 2, 3, 7, 4, 2]. Score: 23.0
[6, 0, 5, 5, 2, 1, 5, 7]. Score: 21.0
[0, 0, 7, 7, 3, 2, 1, 5]. Score: 22.0
[0, 2, 5, 5, 3, 3, 6, 5]. Score: 19.0
[5, 0, 5, 2, 6, 2, 3, 6]. Score: 19.0
[5, 1, 5, 1, 5, 5, 2, 3]. Score: 18.0
[3, 1, 4, 7, 4, 0, 4, 7]. Score: 20.0
[4, 3, 7, 2, 5, 5, 3, 5]. Score: 20.0
[6, 2, 4, 0, 7, 2, 2, 4]. Score: 18.0
[2, 3, 3, 0, 3, 0, 7, 6]. Score: 19.0
[2, 4, 2, 6, 1, 5, 0, 6]. Score: 23.0
[3, 2, 3, 1, 2, 2, 0, 6]. Score: 20.0
[6, 3, 1, 5, 0, 0, 1, 7]. Score: 23.0
[7, 6, 4, 2, 4, 1, 5, 1]. Score: 22.0
```



```
[0, 5, 3, 0, 7, 4, 2, 1]. Score: 26.0
[0, 5, 3, 0, 7, 4, 2, 5]. Score: 26.0
[0, 5, 3, 0, 6, 4, 2, 1]. Score: 26.0
[1, 5, 3, 0, 7, 4, 2, 7]. Score: 26.0
Solution not found
Generation: 3
[2, 4, 7, 0, 3, 1, 6, 5]. Score: 27.0
[2, 4, 7, 0, 3, 1, 6, 5]. Score: 27.0
[2, 4, 7, 0, 3, 1, 6, 1]. Score: 27.0
[2, 4, 7, 0, 3, 1, 6, 1]. Score: 27.0
[2, 4, 7, 0, 3, 1, 6, 1]. Score: 27.0
[2, 4, 7, 0, 3, 1, 6, 1]. Score: 27.0
[2, 4, 7, 0, 3, 1, 6, 1]. Score: 27.0
[2, 4, 7, 3, 0, 6, 5, 1]. Score: 27.0
[2, 4, 7, 3, 0, 6, 5, 1]. Score: 27.0
[2, 4, 7, 3, 0, 6, 5, 1]. Score: 27.0
[2, 4, 7, 3, 0, 6, 1, 1]. Score: 27.0
[2, 4, 7, 3, 0, 6, 5, 1]. Score: 27.0
[2, 4, 7, 3, 0, 6, 5, 1]. Score: 27.0
[2, 4, 7, 3, 0, 6, 5, 1]. Score: 27.0
[2, 4, 7, 3, 0, 6, 5, 1]. Score: 27.0
[2, 4, 7, 3, 0, 6, 5, 1]. Score: 27.0
```



```
Solution not found
[2, 4, 7, 3, 0, 6, 1, 5]. Score: 28.0
Solution found
~/AIwork$ 
```