Ahsanullah University of Science and Technology

Department of Computer Science and Engineering

Term Assignment: 2

Genetic Algorithm

CSE 4108

Artificial Intelligence

Submitted By:

Anika Tanzim 16.02.04.072

Date of Submission: **12 September 2020**

Q: Implement a Genetic algorithm in python or prolog.

Answer:

Genetic algorithms:

A genetic algorithm is a variant of a stochastic beam search. It is commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover, and selection. GA model reproduction involves two parents. A successor is generated by combining two states. Parent states are chosen from a set after rating with an objective function. The function is called 'fitness function'. 'Crossover' and 'Mutation' are also accommodated.

Input:

We take the **8-queens problem** to demonstrate the working of the Genetic Algorithm strategy. A state is represented as an eight-digit positive integer (with 1, 2, 3, ..., 8 only). The value is an array of integers that represents the respective row and the index represents the column number. The column number is a 0-based index whereas the row number is between 1 to 8 which is 1 base index. The following array represents the second queen is in the 4th row.

1	4	2	3	7	3	2	6

Here, we take a few states of the 8-queens problem as the **initial population**.

```
intl_sts = [[3, 2, 7, 5, 2, 4, 1, 1], [6, 1, 5, 7, 4, 3, 8, 1], [1, 2, 3, 4, 5, 6, 7, 8],
        [8, 7, 6, 5, 4, 3, 2, 1], [1, 8, 2, 7, 3, 6, 4, 5], [4, 5, 3, 6, 2, 7, 1, 8],
        [2, 4, 6, 8, 1, 3, 5, 7], [1, 5, 2, 6, 3, 7, 4, 8], [8, 4, 7, 3, 6, 2, 5, 1]]
```

Major Steps of Processing:

1. At first, we have to set a threshold value for the formation of parent generation, and also set a target value of the fitness function.
2. There is a **fitness function** that determines how fit an individual is (the ability of an individual to compete with other individuals). Here, for 8 queens problems, the fitness function is the total number of non-attacking queens for each state.
3. The process begins with the **selection** of a set of individuals which is called a **Population**. **Parents** will be generated from the population according to their fitness function such that fitness value should be greater than the threshold value.
4. After selection, two parents are chosen randomly from population[]. And for each pair of parents to be mated, a **crossover** is performed within the genes. To do the crossover, a

crammer point is chosen randomly. And they exchange the genes from the crammer point. By that, a new generation is obtained consisting of the ‘**offsprings**’.

5. The offsprings will be tested. If their fitness value crosses or equal to the target value, the process terminates. Otherwise, the mutation will be begun.
6. In certain new offspring formed, some of their genes can be subjected to a **mutation** with a low random probability. To perform this, a **mutation_acces**s point is generated randomly. And at that point, two offsprings exchange their genes. And again, the offsprings will be tested to check if the target value is reached or not.
7. The formation of a new population and evaluation of the individuals is carried out until an individual with the target fitness is found. So, if the target is not found, the process goes back to step 4.

Source Code:

```python
import random

found= False
visited = []

def fitness(intl_sts):
  queeninfo = []

  for i in range(8):
     t = []
     t.insert(0, i + 1)
     t.insert(1, intl_sts[i])
     t.insert(2, i + 1)
     t = tuple(t)
     queeninfo.append(t)

  cnt = 0
  for i in range(8):
     currentQueen = queeninfo[i][0]
     currentQueenRow = queeninfo[i][1]
     currentQueenColumn = queeninfo[i][2]
     for j in range(8):
        rowDif = abs(currentQueenRow - queeninfo[j][1])
        colDif = abs(currentQueenColumn - queeninfo[j][2])

        if (queeninfo[j][0] == currentQueen):
           continue
        elif (rowDif == colDif):
           cnt += 1
        elif (currentQueenRow == queeninfo[j][1]):
           cnt += 1
        elif (currentQueenColumn == queeninfo[j][2]):
           cnt += 1
  cnt /= 2
  return (28 - int(cnt))
```

```
def initial_population(intl_sts):
  for sts in intl_sts:
      fit = fitness(sts)
      if fit > threshold:
         population.append(sts)

  return len(population)


def crossover():
  #choosing different parent
  random.seed()
  p1 = random.randint(0, len(population) - 1)
  while True:
      p2 = random.randint(0, len(population) - 1)
      if p2==p1:
         continue
      break

  #crossing over
  crammer_point = random.randint(0, 7)
  new1 = []
  new2 = []
  for i in range(crammer_point):
      new1.append(population[p1][i])
      new2.append(population[p2][i])
  for i in range(crammer_point,8):
      new1.append(population[p2][i])
      new2.append(population[p1][i])

  print("new generation: " + str(new1) + "\tfitness:\t" +str(fitness(new1)))
  print("new generation: " + str(new2) + "\tfitness:\t" +str(fitness(new2)))

  population.append(new1)
  population.append(new2)

  # offspring test

  for i in range(n,len(population)):
      at = fitness(population[i])
      if (at >= target):
         return population[i]


def new_genration(population):
  while not found:
      crossover()

      #mutation
      random.seed()
      mutatuion_access = random.randint(0, 7)
      population[len(population)-1][mutatuion_access],population[len(population)-2][mutatuion_access] =
population[len(population)-2][mutatuion_access],population[len(population)-1][mutatuion_access]
```

```
    print("new generation: " + str(population[len(population)-1]) + "\tfitness:\t" +
str(fitness(population[len(population)-1])))
    print("new generation: " + str(population[len(population)-2]) + "\tfitness:\t" +
str(fitness(population[len(population)-2])))

    population.append(population[len(population)-1])
    population.append(population[len(population)-2])

    # offspring test
    for i in range(n,len(population)):
        at = fitness(population[i])
        if (at >= target):
            return population[i]


# main code

intl_sts = [[3, 2, 7, 5, 2, 4, 1, 1], [6, 1, 5, 7, 4, 3, 8, 1], [1, 2, 3, 4, 5, 6, 7, 8], [8, 7, 6, 5, 4, 3, 2, 1],
            [1, 8, 2, 7, 3, 6, 4, 5], [4, 5, 3, 6, 2, 7, 1, 8], [2, 4, 6, 8, 1, 3, 5, 7],
            [1, 5, 2, 6, 3, 7, 4, 8], [8, 4, 7, 3, 6, 2, 5, 1]]
population=[]

while True:
  threshold = int(input('Enter threshold value: '))
  if threshold<0 or threshold > 28:
    print('Threshold value should be 0 - 28')
    continue
  break
while True:
  target = int(input('Enter target fitness value: '))
  if target > 28:
    print('Minimum fitness value can not exceed 28')
    continue
  break

print("Intial population: ")
n= initial_population(intl_sts)
for p in population:
  print(p)

result = new_genration(population)
print("final population\t" + str(result) +"\tfitness\t" + str(fitness(result)))
```

Output:

```
Enter threshold value: 23
Enter target fitness value: 26
Intial population:
[1, 8, 2, 7, 3, 6, 4, 5]
[4, 5, 3, 6, 2, 7, 1, 8]
[2, 4, 6, 8, 1, 3, 5, 7]
[1, 5, 2, 6, 3, 7, 4, 8]
[8, 4, 7, 3, 6, 2, 5, 1]
new generation: [1, 5, 2, 6, 3, 3, 5, 7]  fitness:    24
new generation: [2, 4, 6, 8, 1, 7, 4, 8]  fitness:    24
new generation: [2, 4, 6, 8, 3, 7, 4, 8]  fitness:    25
new generation: [1, 5, 2, 6, 1, 3, 5, 7]  fitness:    25
new generation: [2, 4, 6, 8, 3, 7, 4, 8]  fitness:    25
new generation: [2, 4, 6, 8, 3, 7, 4, 8]  fitness:    25
new generation: [2, 4, 6, 8, 3, 7, 4, 8]  fitness:    25
new generation: [2, 4, 6, 8, 3, 7, 4, 8]  fitness:    25
new generation: [2, 4, 6, 8, 3, 7, 4, 8]  fitness:    25
new generation: [2, 4, 6, 8, 1, 3, 5, 7]  fitness:    25
new generation: [2, 4, 6, 8, 1, 3, 5, 8]  fitness:    24
new generation: [2, 4, 6, 8, 3, 7, 4, 7]  fitness:    25
new generation: [2, 4, 6, 8, 3, 7, 4, 8]  fitness:    25
new generation: [2, 4, 6, 8, 3, 7, 4, 8]  fitness:    25
new generation: [2, 4, 6, 8, 3, 7, 4, 8]  fitness:    25
new generation: [2, 4, 6, 8, 3, 7, 4, 8]  fitness:    25
new generation: [2, 4, 6, 8, 1, 3, 5, 8]  fitness:    24
new generation: [2, 4, 6, 8, 3, 7, 4, 8]  fitness:    25
new generation: [2, 4, 6, 8, 3, 7, 5, 8]  fitness:    24
new generation: [2, 4, 6, 8, 1, 3, 4, 8]  fitness:    23
new generation: [2, 4, 6, 6, 3, 7, 4, 8]  fitness:    24
new generation: [1, 5, 2, 8, 3, 7, 4, 8]  fitness:    26
new generation: [1, 5, 2, 8, 3, 7, 4, 8]  fitness:    26
new generation: [2, 4, 6, 6, 3, 7, 4, 8]  fitness:    24
final population   [1, 5, 2, 8, 3, 7, 4, 8]     fitness:    26

Process finished with exit code 0
```