Name – Yash Gandhi BE IT Batch A 2017140014

Aim:
To apply genetic algorithm for a given problem

Problem Statement: Nqueens problem using genetic algorithm

Task 2: Implement the chosen research paper by using the same components mentioned in the paper. Also, use other types of components (other encoding/ selection/crossover/mutation schemes) and obtain a comparative analysis of the same.

Tool/Language:
Python

Report Summary: Nqueens was implemented using two approaches and the corresponding results were observed. In one approach repetition in the chromosome was allowed and in the second approach repetition was eliminated and more randomness was introduced.

Algorithms :
Encoding scheme:
Initialize a random population of chromosome of length 1000.
Every chromosome is represented as a vector of length N, which is a random permutation (with repetition) of (1, 2, 3… N).
N-tuple (c1, c2, c3… cN), where ci represents the position of the queen to be in ith column and cth row.

Selection:
Selecting two chromosomes X, Y based on their probability (fitness value/max_fitness).
Select the one with highest probability.

Crossover:
Crossover(X,Y)
C=RandomInteger(0,n-1)
New_chromosome=X[0:C]+Y[C:N]

Mutation:
Mutation(X)

C=RandomInteger(0,n-1)
M=RandomInteger(1,n)
X[C]=M

Fitness function algorithm:
Function fitness (chromosome) {

Max_Fitness=n*(n-1)/2 // total number of conflicts possible
Horizontal_collision= Total number of repetitions of each gene in the chromosome
t1 = 0; //number of repetitive queens in one diagonal while seen from left corner
t2 = 0; //number of repetitive queens in one diagonal while seen from right corner
size = length (chromosome);
for i= 1 to size:
 f1 (i) = (chromosome (i)-i);
 f2 (i) = ((1+size)-chromosome (i)-i);
end
f1=sort (f1);
f2=sort (f2);
for i=2 to size:
 if(f1 (i) == f1 (i-1)) //checks whether two Queens are in same diagonals seeing from left corner or not
 t1 = t1+1;
 end
 if(f2 (i) == f2(i-1)) //checks whether two Queens are in same diagonals seeing from right corner or not
 t2 = t2+1;
 end
end
fitness_value = t1 + t2;
return Maxfitness- fitness_value

Genetic Algorithm:

Initiate population of 100 chromosome
While(Maxfitness not in population)
  Selection
  Crossover
  Mutation

Results:

For n=7 Maxfitness required = 21
Starts with Generation 1 with population of 100, Each Chromosom with the fitness value

```
    print()
    print_board(board)
```

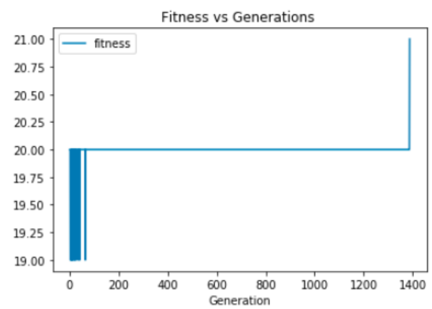
```
Enter Number of Queens: 7
=== Generation 1 ===
Chromosome = [7, 1, 4, 4, 3, 5, 7],  Fitness = 18
Chromosome = [7, 7, 4, 5, 1, 4, 2],  Fitness = 18
Chromosome = [6, 3, 2, 6, 7, 4, 5],  Fitness = 18
Chromosome = [7, 4, 5, 2, 5, 3, 2],  Fitness = 18
Chromosome = [7, 7, 3, 6, 5, 2, 6],  Fitness = 18
Chromosome = [2, 2, 4, 6, 3, 2, 4],  Fitness = 16
Chromosome = [2, 4, 5, 2, 1, 7, 6],  Fitness = 18
Chromosome = [6, 7, 7, 7, 4, 5, 3],  Fitness = 16
Chromosome = [2, 7, 7, 7, 7, 3, 2],  Fitness = 13
Chromosome = [6, 5, 5, 4, 7, 5, 1],  Fitness = 17
Chromosome = [6, 3, 2, 6, 1, 1, 7],  Fitness = 18
Chromosome = [2, 5, 3, 6, 1, 7, 1],  Fitness = 19
Chromosome = [2, 4, 7, 4, 1, 4, 5],  Fitness = 17
Chromosome = [3, 2, 4, 4, 4, 5, 1],  Fitness = 17
Chromosome = [3, 6, 1, 7, 3, 3, 2],  Fitness = 17
Chromosome = [7, 7, 2, 5, 2, 2, 5],  Fitness = 15
Chromosome = [3, 1, 6, 7, 7, 2, 6],  Fitness = 18
Chromosome = [3, 4, 2, 4, 5, 6, 4],  Fitness = 17
```

Chromosome with highest fitness value 21 in the 1388 generation

```
Chromosome = [5, 5, 6, 4, 1, 3, 7],  Fitness = 19
Chromosome = [2, 5, 6, 4, 1, 3, 7],  Fitness = 20
Chromosome = [2, 5, 6, 4, 1, 3, 7],  Fitness = 20
Chromosome = [2, 5, 6, 4, 4, 3, 7],  Fitness = 19
Chromosome = [2, 5, 6, 4, 1, 3, 7],  Fitness = 20
Chromosome = [2, 5, 7, 4, 1, 3, 6],  Fitness = 21

Maximum Fitness = 21
Solved in Generation 1388!

One of the solutions:
Chromosome = [2, 5, 7, 4, 1, 3, 6],  Fitness = 21

x x Q x x x x
x x x x x x Q
x Q x x x x x
x x x Q x x x
x x x x x Q x
Q x x x x x x
x x x x Q x x
```

Fitness vs Generation Graph:

```
In [6]: import matplotlib.pyplot as plot

        df.plot.line(y='fitness',x='Generation',title="Fitness vs Generations ");

        plot.show(block=True);
```



Improvement in the Algorithm:
Population Generation:
Eliminating Repetitve values in the cromosome to reduce horizontal collisions

Crossover:
Crossover(X,Y)
C=RandomInteger(0,n-1)
M= RandomInteger(0,n-1)
A= X[C:M]
B= Genes from Y not present in X
New_chromosome=A+B

Mutation:
Mutation(X)
C=RandomInteger(0,n-1)
M=RandomInteger(0,n-1)
Swap(X[C],X[M])

Results for new Approach:
N=7

```
#print(df)
```

```
Enter Number of Queens: 7
=== Generation 1 ===
Chromosome = [1, 2, 3, 4, 5, 6, 7],  Fitness = 20
Chromosome = [7, 1, 2, 3, 4, 5, 6],  Fitness = 20
Chromosome = [1, 5, 2, 3, 4, 6, 7],  Fitness = 20
Chromosome = [1, 2, 3, 4, 5, 6, 7],  Fitness = 20
Chromosome = [1, 2, 3, 4, 5, 6, 7],  Fitness = 20
Chromosome = [4, 6, 1, 2, 3, 5, 7],  Fitness = 20
Chromosome = [2, 1, 3, 4, 5, 6, 7],  Fitness = 19
Chromosome = [5, 1, 2, 3, 4, 6, 7],  Fitness = 20
Chromosome = [1, 7, 2, 3, 4, 5, 6],  Fitness = 20
Chromosome = [6, 1, 2, 3, 4, 5, 7],  Fitness = 20
Chromosome = [1, 2, 3, 4, 5, 6, 7],  Fitness = 20
Chromosome = [4, 2, 6, 7, 1, 3, 5],  Fitness = 20
Chromosome = [1, 2, 3, 4, 5, 6, 7],  Fitness = 20
Chromosome = [1, 2, 3, 4, 5, 6, 7],  Fitness = 20
Chromosome = [2, 1, 3, 4, 5, 6, 7],  Fitness = 19
Chromosome = [1, 2, 3, 4, 5, 6, 7],  Fitness = 20
Chromosome = [2, 1, 3, 4, 5, 6, 7],  Fitness = 19
Chromosome = [5, 6, 1, 2, 3, 4, 7],  Fitness = 19
```

Solution in 224 generations

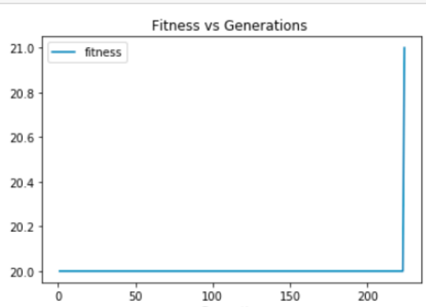
```
#print(df)
```

```
Chromosome = [1, 2, 3, 4, 5, 6, 7],  Fitness = 20
Chromosome = [3, 4, 5, 1, 2, 6, 7],  Fitness = 20
Chromosome = [1, 2, 3, 4, 5, 6, 7],  Fitness = 20
Chromosome = [1, 2, 3, 4, 5, 6, 7],  Fitness = 20
Chromosome = [4, 1, 5, 2, 6, 3, 7],  Fitness = 21

Maximum Fitness = 21
Solved in Generation 224!

One of the solutions:
Chromosome = [4, 1, 5, 2, 6, 3, 7],  Fitness = 21

x x x x x x Q
x x x x Q x x
x x Q x x x x
Q x x x x x x
x x x x x Q x
x x x Q x x x
x Q x x x x x
```

```
In [12]: print(df)
```

Fitness vs Generation graph

```
In [14]: import matplotlib.pyplot as plot

         df.plot.line(y='fitness',x='Generation',title="Fitness vs Generations ");

         plot.show(block=True);
```



Comparison of the two variants used commenting on which performed better for your problem:

	Old Approach ***Repetition of digits allowed***	***New Approach wih Crossover done using Changing initial chromosome,crossover and mutation***
Generations for N=7	*1388*	*244*

Hence avoiding the repetition of positions in a chromosome and changing the crossover and mutation function by introducing more randomness decreases the number of generations to find the best solution.

Code:

```
import pandas as pd
import random

def random_chromosome(nq): #making random chromosomes
    return [ random.randint(1, nq) for _ in range(nq) ]

def fitness(chromosome):
    horizontal_collisions = sum([chromosome.count(queen)-1 for queen in chromosome])/2
    diagonal_collisions = 0

    n = len(chromosome)
    left_diagonal = [0] * 2*n
    right_diagonal = [0] * 2*n
    for i in range(n):
        left_diagonal[i + chromosome[i] - 1] += 1
        right_diagonal[len(chromosome) - i + chromosome[i] - 2] += 1

    diagonal_collisions = 0
    for i in range(2*n-1):
        counter = 0
        if left_diagonal[i] > 1:
            counter += left_diagonal[i]-1
        if right_diagonal[i] > 1:
            counter += right_diagonal[i]-1
        diagonal_collisions += counter / (n-abs(i-n+1))

    return int(maxFitness - (horizontal_collisions + diagonal_collisions)) #28-(2+3)=23

def probability(chromosome, fitness):
    return fitness(chromosome) / maxFitness

def random_pick(population, probabilities):
    populationWithProbabilty = zip(population, probabilities)
    total = sum(w for c, w in populationWithProbabilty)
    r = random.uniform(0, total)
    upto = 0
    for c, w in zip(population, probabilities):
        if upto + w >= r:
            return c
        upto += w
    assert False, "Shouldn't get here"

def reproduce(x, y): #doing cross_over between two chromosomes
    n = len(x)
    c = random.randint(0, n - 1)
    return x[0:c] + y[c:n]

def mutate(x):  #randomly changing the value of a random index of a chromosome
    n = len(x)
    c = random.randint(0, n - 1)
    m = random.randint(1, n)
    x[c] = m
```

```
    return x

def genetic_queen(population, fitness):
    mutation_probability = 0.03
    new_population = []
    probabilities = [probability(n, fitness) for n in population]
    for i in range(len(population)):
        x = random_pick(population, probabilities) #best chromosome 1
        y = random_pick(population, probabilities) #best chromosome 2
        child = reproduce(x, y) #creating two new chromosomes from the best 2 chromosomes
        if random.random() < mutation_probability:
            child = mutate(child)
        print_chromosome(child)
        new_population.append(child)
        if fitness(child) == maxFitness: break
    return new_population

def print_chromosome(chrom):
    print("Chromosome = {},  Fitness = {}"
        .format(str(chrom), fitness(chrom)))

if __name__ == "__main__":
    nq = int(input("Enter Number of Queens: ")) #say N = 8
    maxFitness = (nq*(nq-1))/2  # 8*7/2 = 28
    population = [random_chromosome(nq) for _ in range(100)]
    df = pd.DataFrame(columns = ['Generation', 'fitness'])

    generation = 1

    while not maxFitness in [fitness(chrom) for chrom in population]:
        print("=== Generation {} ===".format(generation))
        population = genetic_queen(population, fitness)
        print("")
        print("Maximum Fitness = {}".format(max([fitness(n) for n in population])))
        df = df.append({'Generation' : generation, 'fitness' :format(max([fitness(n) for n in population])) },
                ignore_index = True)
        generation += 1
    chrom_out = []
    print("Solved in Generation {}!".format(generation-1))
    for chrom in population:
        if fitness(chrom) == maxFitness:
            print("");
            print("One of the solutions: ")
            chrom_out = chrom
            print_chromosome(chrom)

    board = []

    for x in range(nq):
        board.append(["x"] * nq)

    for i in range(nq):
        board[nq-chrom_out[i]][i]="Q"


    def print_board(board):
        for row in board:
            print (" ".join(row))

    print()
    print_board(board)
```

Changes:

```
import random
import pandas as pd
def random_chromosome(nq): #making random chromosomes
    clist = list(range(1, nq+1))  # the cast to list is optional in Python 2
    random.shuffle(clist)
    return [ clist.pop() for _ in range(nq) ]


def fitness(chromosome):
    horizontal_collisions = sum([chromosome.count(queen)-1 for queen in chromosome])/2
    diagonal_collisions = 0

    n = len(chromosome)
    left_diagonal = [0] * 2*n
    right_diagonal = [0] * 2*n
    for i in range(n):
        left_diagonal[i + chromosome[i] - 1] += 1
```

```
        right_diagonal[len(chromosome) - i + chromosome[i] - 2] += 1

    diagonal_collisions = 0
    for i in range(2*n-1):
        counter = 0
        if left_diagonal[i] > 1:
            counter += left_diagonal[i]-1
        if right_diagonal[i] > 1:
            counter += right_diagonal[i]-1
        diagonal_collisions += counter / (n-abs(i-n+1))

    return int(maxFitness - (horizontal_collisions + diagonal_collisions)) #28-(2+3)=23

def probability(chromosome, fitness):
    return fitness(chromosome) / maxFitness

def random_pick(population, probabilities):
    populationWithProbabilty = zip(population, probabilities)
    total = sum(w for c, w in populationWithProbabilty)
    r = random.uniform(0, total)
    upto = 0
    for c, w in zip(population, probabilities):
        if upto + w >= r:
            return c
        upto += w
    assert False, "Shouldn't get here"

def reproduce(x, y): #doing cross_over between two chromosomes
    n = len(x)
    c = random.randint(0, n - 1)
    m = random.randint(0, n - 1)
    lista=x[c:m]
    a = set(lista)
    b = set(y)
    d=list(b-a)
    flist=lista+d
    return  flist

def mutate(x):  #randomly changing the value of a random index of a chromosome
    n = len(x)
    c = random.randint(0, n-1)
    m = random.randint(0, n-1)
    p=x[c]
    x[c] = x[m]
    x[m]=p
    return x

def genetic_queen(population, fitness):
    mutation_probability = 0.03
    new_population = []
    probabilities = [probability(n, fitness) for n in population]
    for i in range(len(population)):
        x = random_pick(population, probabilities) #best chromosome 1
        y = random_pick(population, probabilities) #best chromosome 2
        child = reproduce(x, y) #creating two new chromosomes from the best 2 chromosomes
        if random.random() < mutation_probability:
            child = mutate(child)
        print_chromosome(child)
        new_population.append(child)
        if fitness(child) == maxFitness: break
    return new_population

def print_chromosome(chrom):
    print("Chromosome = {},  Fitness = {}"
        .format(str(chrom), fitness(chrom)))

if __name__ == "__main__":
    nq = int(input("Enter Number of Queens: ")) #say N = 8
    maxFitness = (nq*(nq-1))/2  # 8*7/2 = 28
    population = [random_chromosome(nq) for _ in range(100)]
    df = pd.DataFrame(columns = ['Generation', 'fitness'])

    generation = 1

    while not maxFitness in [fitness(chrom) for chrom in population]:
        print("=== Generation {} ===".format(generation))
        population = genetic_queen(population, fitness)
        print("")
        print("Maximum Fitness = {}".format(max([fitness(n) for n in population])))
        df = df.append({'Generation' : generation, 'fitness' :format(max([fitness(n) for n in population])) },
```

```
            ignore_index = True)
    generation += 1
chrom_out = []
print("Solved in Generation {}!".format(generation-1))
for chrom in population:
    if fitness(chrom) == maxFitness:
        print("");
        print("One of the solutions: ")
        chrom_out = chrom
        print_chromosome(chrom)

board = []

for x in range(nq):
    board.append(["x"] * nq)

for i in range(nq):
    board[nq-chrom_out[i]][i]="Q"


def print_board(board):
    for row in board:
        print (" ".join(row))

print()
print_board(board)
#print(df)
```

Conclusion:

Genetic algorithm was used to solve the nqueens problem. Selection, Crossover, Mutation and fitness function strategies were used and improved to find the best solution in minimal time. Hence we conclude that we can improve the algorithm by introducing more randomness and eliminating the repetition in chromosomes.