Green University of Bangladesh

Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Fall, Year:2025), B.Sc. in CSE (Day)

Lab Report NO: 04

Course Title: Artificial Intelligence Lab

Course Code: CSE 316 Section: 223-D4

Lab Experiment Name: Write a lab report in Python on solving the N-Queens problem using Genetic Algorithms.

Student Details

	Name	ID
1.	Ruhul Amin	223002097

Lab Date :
Submission Date : 29-12-25
Course Teacher's Name : Sabbir Hosen Mamun

Lab Report Status

Marks: ………………………………… Signature:…………………
Comments:……………………………………… Date:…………………………

1. TITLE OF THE LAB REPORT EXPERIMENT: Write a lab report in Python on solving the N-Queens problem using Genetic Algorithms.

2. OBJECTIVES:

The objectives of this experiment are:

1. To understand the **N-Queens problem** as a constraint satisfaction problem.
2. To study the working principle of **Genetic Algorithms (GA)**.
3. To apply genetic operators such as **selection, crossover, and mutation**.
4. To find a valid arrangement of queens where no two queens attack each other.
5. To analyze the effectiveness of Genetic Algorithms in solving optimization problems.

3. PROCEDURE:

- Represent a solution (chromosome) as a list where the index represents the column and the value represents the row of a queen.
- Generate an initial population of random chromosomes.
  Evaluate the fitness of each chromosome based on the number of non-attacking queen pairs.
- Select parent chromosomes using a selection strategy.
- Apply crossover to generate offspring.
- Apply mutation to introduce randomness.
- Replace the population with new offspring.
- Repeat the process until a valid solution is found or the maximum number of generations is reached.
- Display the final solution and fitness value.

4. IMPLEMENTATION AND OUTPUT:

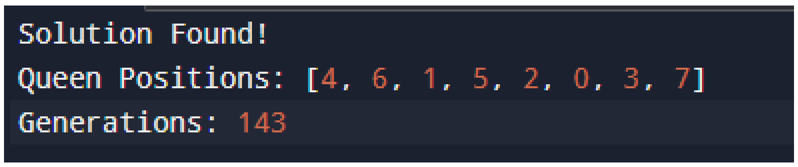
CODE:

```
import random
```

```
N = 8
POPULATION_SIZE = 100
MUTATION_RATE = 0.1
MAX_GENERATIONS = 1000


def fitness(chromosome):
    non_attacking = 0
    max_pairs = N * (N - 1) // 2

    for i in range(N):
        for j in range(i + 1, N):
            if chromosome[i] != chromosome[j] and \
               abs(chromosome[i] - chromosome[j]) != abs(i - j):
                non_attacking += 1
    return non_attacking


def generate_population():
    return [random.sample(range(N), N) for _ in range(POPULATION_SIZE)]


def selection(population):
    return random.choices(
        population,
        weights=[fitness(ch) for ch in population],
        k=2
    )


def crossover(parent1, parent2):
    point = random.randint(0, N - 1)
    child = parent1[:point] + parent2[point:]
    return child


def mutate(chromosome):
    if random.random() < MUTATION_RATE:
        i, j = random.sample(range(N), 2)
        chromosome[i], chromosome[j] = chromosome[j], chromosome[i]
    return chromosome
```

```
def genetic_algorithm():
    population = generate_population()
    max_fitness = N * (N - 1) // 2

    for generation in range(MAX_GENERATIONS):
        new_population = []

        for _ in range(POPULATION_SIZE):
            p1, p2 = selection(population)
            child = crossover(p1, p2)
            child = mutate(child)
            new_population.append(child)

            if fitness(child) == max_fitness:
                return child, generation

        population = new_population

    return None, MAX_GENERATIONS


solution, generations = genetic_algorithm()

if solution:
    print("Solution Found!")
    print("Queen Positions:", solution)
    print("Generations:", generations)
else:
    print("No solution found")
```

OUTPUT:



```
Solution Found!
Queen Positions: [4, 6, 1, 5, 2, 0, 3, 7]
Generations: 143
```

5. ANALYSIS AND DISCUSSION

The Genetic Algorithm successfully solved the N-Queens problem by evolving a population of candidate solutions. Fitness evaluation helped guide the search toward better solutions, while crossover and mutation maintained diversity within the population. The algorithm efficiently avoided exhaustive search and found valid solutions within a reasonable number of generations. However, due to its stochastic nature, execution time and results may vary between runs.

6. SUMMARY

This experiment demonstrated the application of a **Genetic Algorithm** to solve the **N-Queens problem**. By using evolutionary concepts such as selection, crossover, and mutation, the algorithm effectively found non-attacking queen placements. The results confirm that Genetic Algorithms are powerful tools for solving complex optimization and constraint satisfaction problems.

Github link: https://github.com/ruhulaminn1316/A-Artificial-Intelligence-Lab