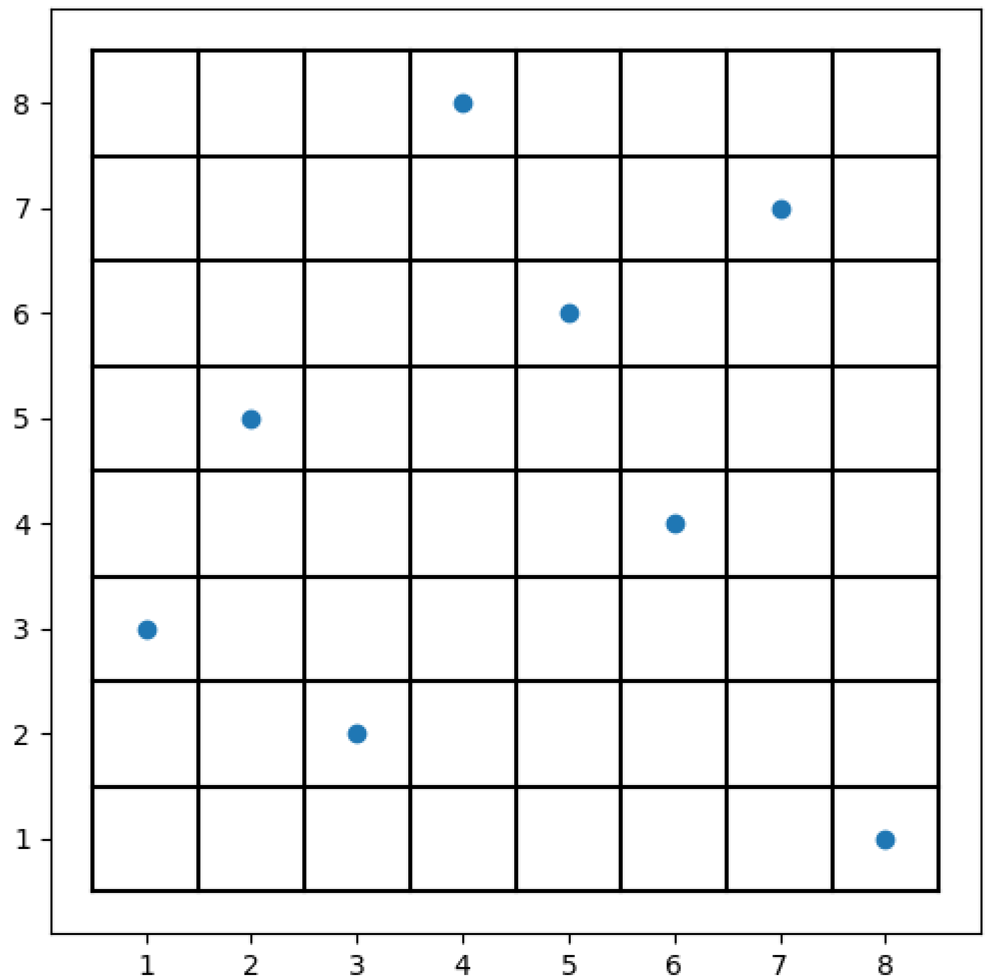
# CS480 Assignment -2

## Genetic Algorithm

Implement a Genetic algorithm to find a solution of the N-queens problem. Use the number of pairwise attacks as the objective function. Repeat the program 100 times for N = 8, N = 16, and N = 32 and show how many times you can find the solutions.

**Output for Queens = 8 Solution 1**

Each dot indicates the position of queens on 8X8 chess board for one of the possible results out of 92 distinct solutions which may or may not be found.



Code Output

```
C:\Users\Ashish\anaconda3\python.exe
C:\Users\Ashish\PycharmProjects\EightQueen\GeneticAlgorithmEightQueen.py

-----------------------------------------------------

Epoch 1

Best Solution: [2, 4, 2, 8, 1, 4, 7, 3, 3]

-----------------------------------------------------

Epoch 2

Best Solution: [2, 4, 2, 8, 1, 4, 7, 3, 3]

-----------------------------------------------------

Epoch 3

Best Solution: [2, 4, 2, 8, 1, 4, 7, 3, 3]

-----------------------------------------------------

Epoch 4

Best Solution: [2, 5, 2, 8, 1, 4, 7, 3, 1]

-----------------------------------------------------

Epoch 5

Best Solution: [2, 5, 2, 8, 1, 4, 7, 3, 1]

-----------------------------------------------------

Epoch 6

Best Solution: [2, 5, 2, 8, 1, 4, 7, 3, 1]

-----------------------------------------------------

Epoch 7

Best Solution: [2, 5, 2, 8, 1, 4, 7, 3, 1]
```

------------------------------------------------------

Epoch 8

Best Solution: [2, 5, 2, 8, 1, 4, 7, 3, 1]

------------------------------------------------------

Epoch 9

Best Solution: [2, 5, 2, 8, 1, 4, 7, 3, 1]

------------------------------------------------------

Epoch 10

Best Solution: [2, 5, 2, 8, 1, 4, 7, 3, 1]

------------------------------------------------------

Epoch 11

Best Solution: [2, 5, 2, 8, 1, 4, 7, 3, 1]

------------------------------------------------------

Epoch 12

Best Solution: [2, 5, 2, 8, 1, 4, 7, 3, 1]

------------------------------------------------------

Epoch 13

Best Solution: [2, 5, 2, 8, 1, 4, 7, 3, 1]

------------------------------------------------------

Epoch 14

Best Solution: [2, 5, 2, 8, 1, 4, 7, 3, 1]

------------------------------------------------------

Epoch 15

Best Solution: [2, 5, 2, 8, 1, 4, 7, 3, 1]

------------------------------------------------------

Epoch 16

Best Solution: [2, 5, 2, 8, 1, 4, 7, 3, 1]

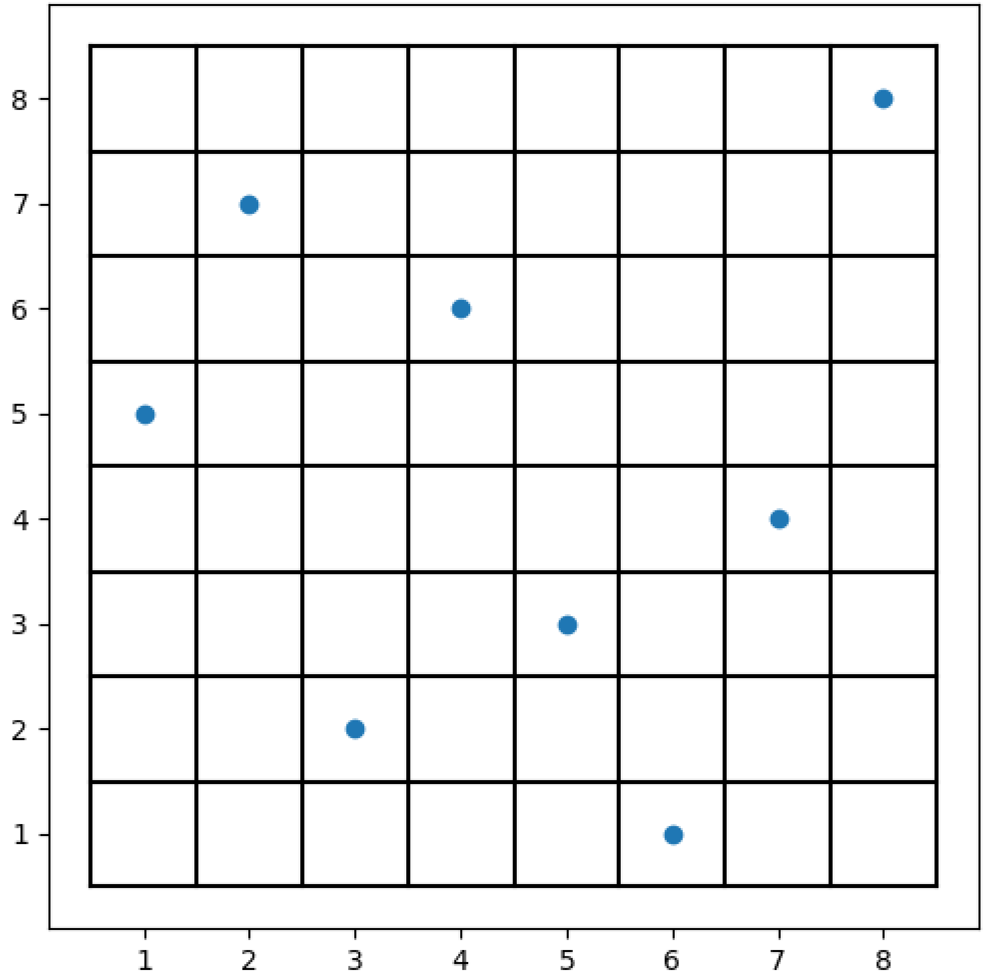
------------------------------------------------------

Epoch 17

Solution Found: [3, 5, 2, 8, 6, 4, 7, 1, 0]

**Output for Queens = 8 Solution 2**

Each dot indicates the position of queens




C:\Users\Ashish\anaconda3\python.exe
C:\Users\Ashish\PycharmProjects\EightQueen\GeneticAlgorithmEightQueen.py

\-------------------------------------------------------

Epoch 1

Best Solution: [2, 7, 1, 6, 4, 6, 1, 5, 3]

\-------------------------------------------------------

Epoch 2

Best Solution: [2, 7, 1, 6, 4, 6, 1, 5, 3]

\-------------------------------------------------------

Epoch 3

Best Solution: [2, 7, 1, 6, 3, 1, 4, 8, 2]

-----------------------------------------------------

Epoch 4

Best Solution: [2, 7, 1, 6, 3, 1, 4, 8, 2]

-----------------------------------------------------

Epoch 5

Best Solution: [2, 7, 1, 6, 3, 1, 4, 8, 2]

-----------------------------------------------------

Epoch 6

Best Solution: [2, 7, 1, 6, 3, 1, 4, 8, 2]

-----------------------------------------------------

Epoch 7

Best Solution: [2, 7, 1, 6, 3, 1, 4, 8, 2]

-----------------------------------------------------

Epoch 8

Best Solution: [2, 7, 1, 6, 3, 1, 4, 8, 2]

-----------------------------------------------------

Epoch 9

Best Solution: [2, 7, 2, 6, 3, 1, 4, 8, 1]

-----------------------------------------------------

Epoch 10

Solution Found: [5, 7, 2, 6, 3, 1, 4, 8, 0]

**Output for Queens = 32 Solution 1**

Each dot indicates the position of queens

Since the epoch values are very large, I am copying the last few

-----------------------------------------------------

Epoch 2609

Best Solution: [11, 22, 8, 17, 12, 32, 16, 27, 23, 18, 15, 28, 3, 5, 7, 9, 1, 29, 25, 19, 6, 24, 10, 21, 5, 26, 2, 14, 30, 13, 20, 4, 1]

-----------------------------------------------------

Epoch 2610

Best Solution: [11, 22, 8, 17, 12, 32, 16, 27, 23, 18, 15, 28, 3, 5, 7, 9, 1, 29, 25, 19, 6, 24, 10, 21, 5, 26, 2, 14, 30, 13, 20, 4, 1]

-----------------------------------------------------

Epoch 2611

Best Solution: [11, 22, 8, 17, 12, 32, 16, 27, 23, 18, 15, 28, 3, 5, 7, 9, 1, 29, 25, 19, 6, 24, 10, 21, 5, 26, 2, 14, 30, 13, 20, 4, 1]

-----------------------------------------------------

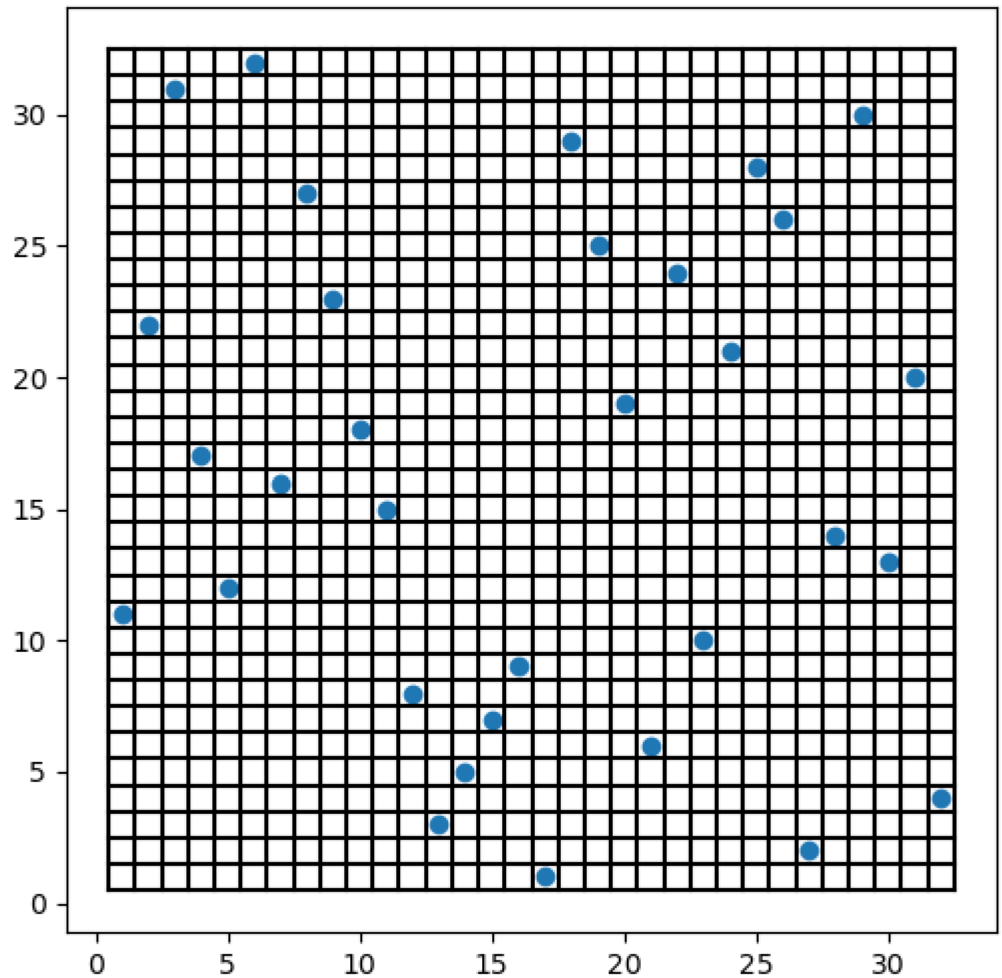
Epoch 2612

Best Solution: [11, 22, 8, 17, 12, 32, 16, 27, 23, 18, 15, 28, 3, 5, 7, 9, 1, 29, 25, 19, 6, 24, 10, 21, 5, 26, 2, 14, 30, 13, 20, 4, 1]

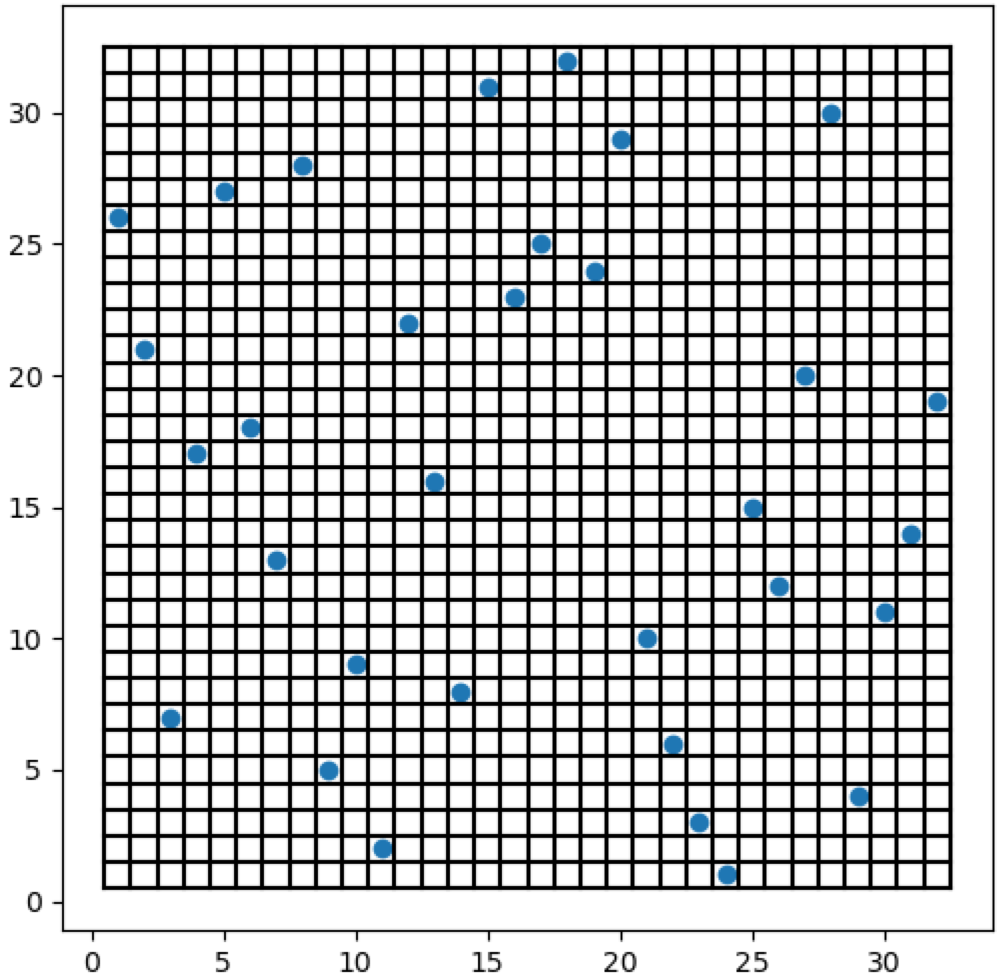
-----------------------------------------------------

Epoch 2613

Solution Found: [11, 22, 31, 17, 12, 32, 16, 27, 23, 18, 15, 8, 3, 5, 7, 9, 1, 29, 25, 19, 6, 24, 10, 21, 28, 26, 2, 14, 30, 13, 20, 4, 0]




**Output for Queens = 32 Solution 2**




Since the epoch values are very large, I am copying the last few

Epoch 735

Best Solution: [26, 21, 7, 4, 6, 29, 13, 28, 5, 9, 2, 22, 16, 8, 31, 23, 25, 32, 24, 17, 10, 6, 3, 1, 15, 12, 20, 30, 27, 11, 14, 19, 1]

------------------------------------------------------

Epoch 736

Best Solution: [26, 21, 7, 4, 6, 29, 13, 28, 5, 9, 2, 22, 16, 8, 31, 23, 25, 32, 24, 17, 10, 6, 3, 1, 15, 12, 20, 30, 27, 11, 14, 19, 1]

------------------------------------------------------

Epoch 737

Best Solution: [26, 21, 7, 4, 6, 29, 13, 28, 5, 9, 2, 22, 16, 8, 31, 23, 25, 32, 24, 17, 10, 6, 3, 1, 15, 12, 20, 30, 27, 11, 14, 19, 1]

------------------------------------------------------

Epoch 738

Best Solution: [26, 21, 7, 4, 6, 29, 13, 28, 5, 9, 2, 22, 16, 8, 31, 23, 25, 32, 24, 17, 10, 6, 3, 1, 15, 12, 20, 30, 27, 11, 14, 19, 1]

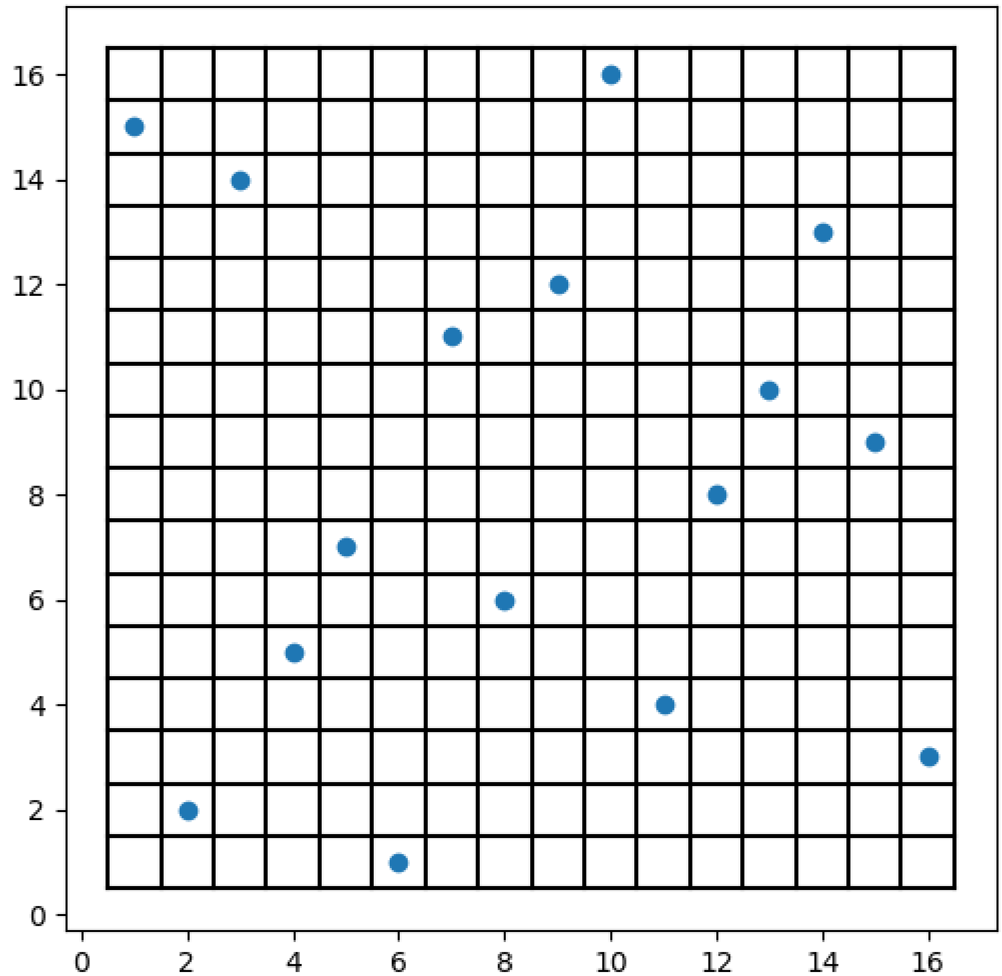
------------------------------------------------------

Epoch 739

Solution Found: [26, 21, 7, 17, 27, 18, 13, 28, 5, 9, 2, 22, 16, 8, 31, 23, 25, 32, 24, 29, 10, 6, 3, 1, 15, 12, 20, 30, 4, 11, 14, 19, 0]

**Output for Queens = 16 Solution 1**

Each dot indicates the position of queens




Since the epoch values are very large, I am copying the last few

Epoch 96

Best Solution: [15, 2, 10, 5, 7, 1, 11, 6, 14, 16, 4, 8, 4, 13, 9, 3, 1]

-----------------------------------------------------

Epoch 97

Best Solution: [15, 2, 10, 5, 7, 1, 11, 6, 14, 16, 4, 8, 4, 13, 9, 3, 1]

-----------------------------------------------------

Epoch 98

Best Solution:  [15, 2, 10, 5, 7, 1, 11, 6, 14, 16, 4, 8, 4, 13, 9, 3, 1]

-------------------------------------------------------

Epoch  99

Best Solution:  [15, 2, 10, 5, 7, 1, 11, 6, 14, 16, 4, 8, 4, 13, 9, 3, 1]

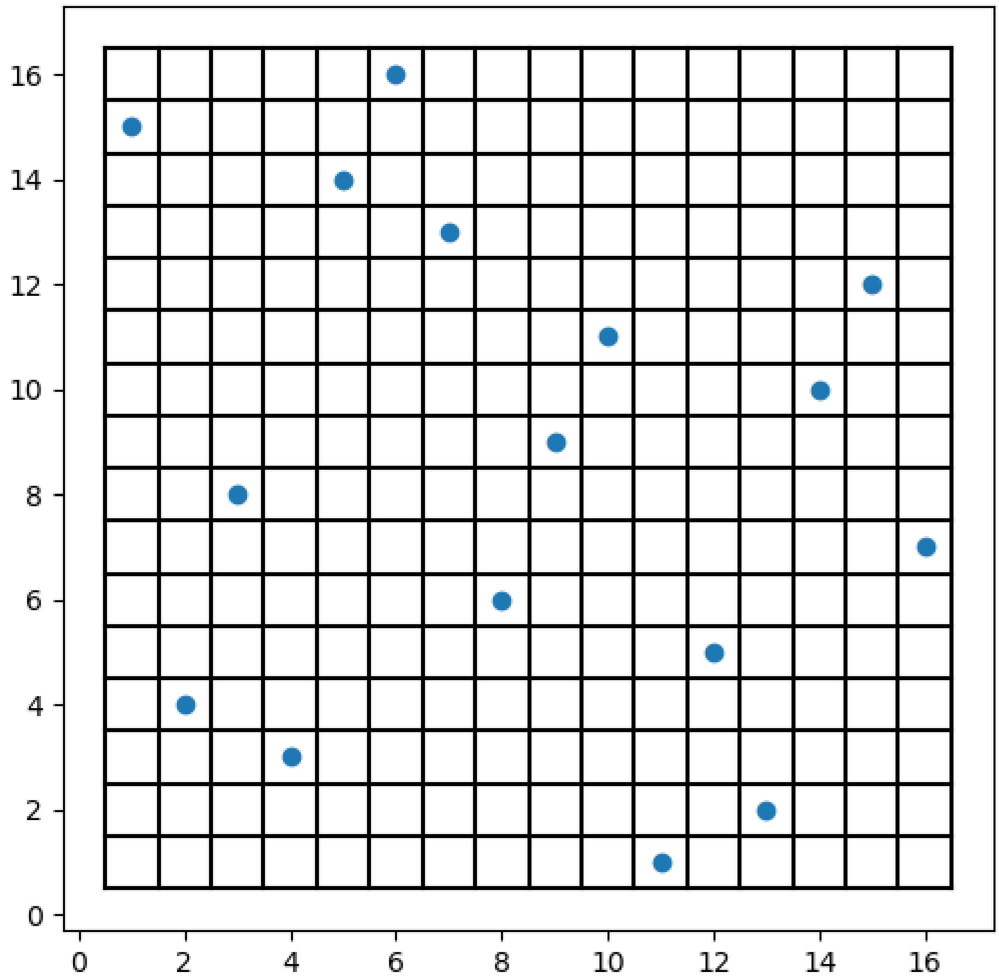
-------------------------------------------------------

Epoch  100

Solution Found:  [15, 2, 14, 5, 7, 1, 11, 6, 12, 16, 4, 8, 10, 13, 9, 3, 0]

**Output for Queens = 16 Solution 2**

Each dot indicates the position of queens




Since the epoch values are very large, I am copying the last few

Epoch 202

Best Solution: [15, 4, 8, 3, 14, 16, 13, 6, 1, 11, 15, 5, 2, 10, 12, 7, 1]

-----------------------------------------------------

Epoch 203

Best Solution: [15, 4, 8, 3, 14, 16, 13, 6, 1, 11, 15, 5, 2, 10, 12, 7, 1]

-----------------------------------------------------

Epoch 204

Best Solution: [15, 4, 8, 3, 14, 16, 13, 6, 1, 11, 15, 5, 2, 10, 12, 7, 1]

-------------------------------------------------------

Epoch 205

Solution Found: [15, 4, 8, 3, 14, 16, 13, 6, 9, 11, 1, 5, 2, 10, 12, 7, 0]

```python
import random
import matplotlib.pyplot as plt
import matplotlib
matplotlib.use('TkAgg')

n = 16 # Number of Queens
p = 100  # Number of Population

current_generation = []  # Current Generation
new_generation = []  # New Generation


def RandomPopulationGeneration(num_rows, num_queens):
    """
    :param num_rows: Rows of the board.
    :param num_queens: Number of queens for the problem.
    :return: List of the random board generated.
    """
    list_of_generation = []
    for i in range(num_rows):
        gene = []
        for j in range(num_queens):
            gene.append(random.randint(1, n))
        gene.append(0)
        list_of_generation.append(gene)
    return list_of_generation


def FitnessSurvival(population):
    """

    :param population: List of the population of the board generated.
    :return: Best fitting population after swap is made.
    """
    i = 0
    attacking = 0
    while i < len(population):
        j = 0
        attacking = 0
        while j < n:
            l = j + 1

            while l < n:
                if population[i][j] == population[i][l]:
                    attacking += 1
                if abs(j - l) == abs(population[i][j] - population[i][l]):
                    attacking += 1
                l += 1
            j += 1
        population[i][len(population[j]) - 1] = attacking
        i += 1

    for i in range(len(population)):
        minimum = i
        for j in range(i, len(population)):
            if population[j][n] < population[minimum][n]:
                minimum = j

        temp = population[i]
        population[i] = population[minimum]
        population[minimum] = temp
    return population


def CrossOver(list_of_generation):
    """
```

```python
    :param list_of_generation: List of the generations for crossover
    :return: Completed cross over  list
    """
    for i in range(0, len(list_of_generation), 2):
        z = 0
        new_child1 = []
        new_child2 = []
        while z < n:
            if (z < n // 2):
                new_child1.append(list_of_generation[i][z])
                new_child2.append(list_of_generation[i + 1][z])
            else:
                new_child1.append(list_of_generation[i + 1][z])
                new_child2.append(list_of_generation[i][z])
            z += 1
        new_child1.append(0)
        new_child2.append(0)
        list_of_generation.append(new_child1)
        list_of_generation.append(new_child2)
    return list_of_generation


def Mutation(list_of_generation):
    """

    :param list_of_generation: List for mutation function
    :return: Lis of mutated population.
    """
    list_of_mutation = []
    i = 0
    while i < p // 2:
        new_rand = random.randint(p // 2, p - 1)
        if new_rand not in list_of_mutation:
            list_of_mutation.append(new_rand)
            list_of_generation[new_rand][random.randint(0, n - 1)] = random.randint(1, n - 1)
            i += 1
    return list_of_generation


def ShowResults(response):
    """

    :param response: Plot the queens position using matplotblib
    :return: Show plot.
    """
    l = len(response)
    plt.figure(figsize=(6, 6))
    plt.scatter([x + 1 for x in range(l - 1)], response[:l - 1])
    for i in range(l):
        plt.plot([0.5, l - 0.5], [i + 0.5, i + 0.5], color="k")
        plt.plot([i + 0.5, i + 0.5], [0.5, l - 0.5], color="k")
    plt.show()


# Call the driver program.
current_generation = RandomPopulationGeneration(p, n)
current_generation = FitnessSurvival(current_generation)
epoch = 1
while True:
    print("-------------------------------------------------------")
    print("Epoch ", epoch)
    current_generation = current_generation[0:p // 2]
    new_generation = CrossOver(current_generation)
    new_generation = Mutation(new_generation)
    current_generation = new_generation
    current_generation = FitnessSurvival(current_generation)
    if current_generation[0][n] == 0:
```

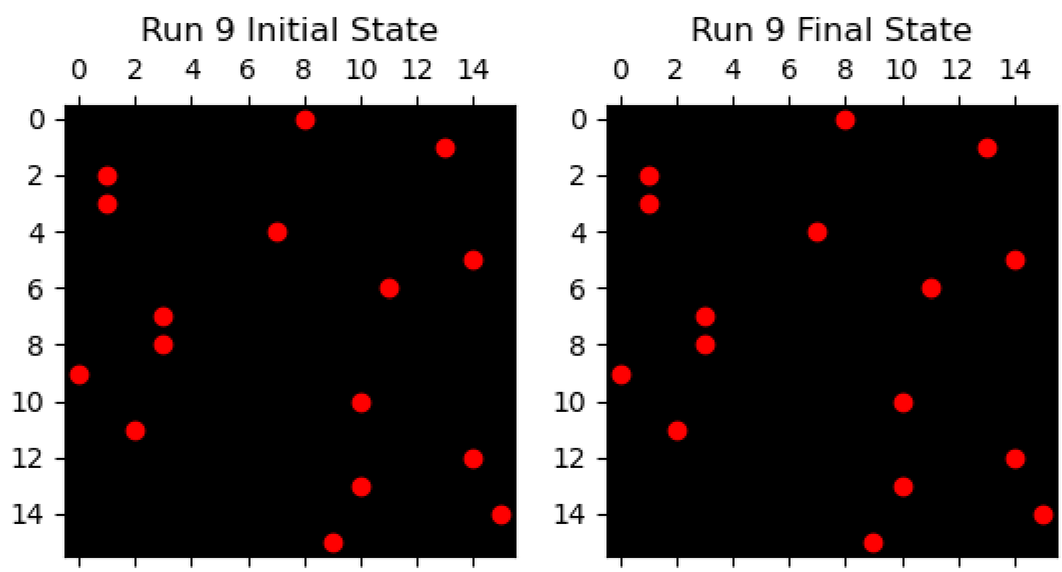
```
            print("Solution Found: ", current_generation[0])
            ShowResults(current_generation[0])
            break
        else:
            print("Best Solution: ", current_generation[0])
        epoch += 1
```

# CS480 Assignment -2

## Hill Climbing

Implement a Hill Climbing Search algorithm to find a solution of the N-queens problem from a random given position. Use the number of pairwise attacks as the objective function. Repeat the program 100 times for N = 8, N = 16, and N = 32 and show how many times you can find the solutions. Plot the initial state and the final state (not necessary the solution) of the first 10 times.

**Number of Queens = 16, output for first run**




C:\Users\Ashish\anaconda3\python.exe
C:\Users\Ashish\PycharmProjects\EightQueen\HillClimbingAlgorightmEightQueen.py

Run 1: Initial Attacks = 18, Final Attacks = 5

Run 2: Initial Attacks = 36, Final Attacks = 5

Run 3: Initial Attacks = 16, Final Attacks = 4

Run 4: Initial Attacks = 15, Final Attacks = 3

Run 5: Initial Attacks = 14, Final Attacks = 5

Run 6: Initial Attacks = 22, Final Attacks = 3

Run 7: Initial Attacks = 21, Final Attacks = 5

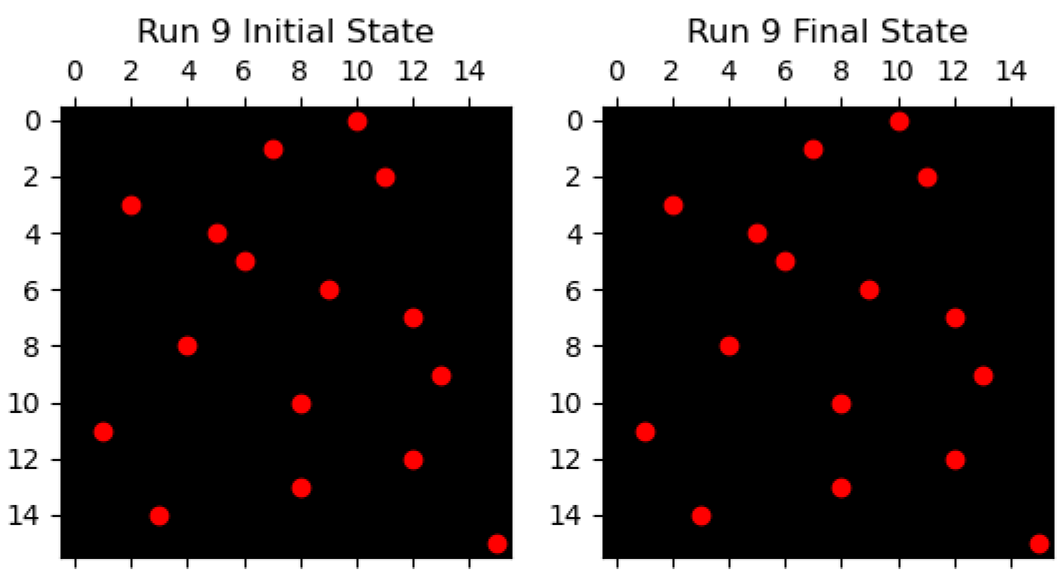
Run 8: Initial Attacks = 11, Final Attacks = 4

Run 9: Initial Attacks = 16, Final Attacks = 6

Run 10: Initial Attacks = 15, Final Attacks = 4

Total solutions found for N = 16: 0 out of 100 runs

**Number of Queens = 16, output for second run**




Run 1: Initial Attacks = 14, Final Attacks = 6

Run 2: Initial Attacks = 12, Final Attacks = 5

Run 3: Initial Attacks = 16, Final Attacks = 5

Run 4: Initial Attacks = 25, Final Attacks = 6

Run 5: Initial Attacks = 15, Final Attacks = 6

Run 6: Initial Attacks = 12, Final Attacks = 5

Run 7: Initial Attacks = 14, Final Attacks = 4

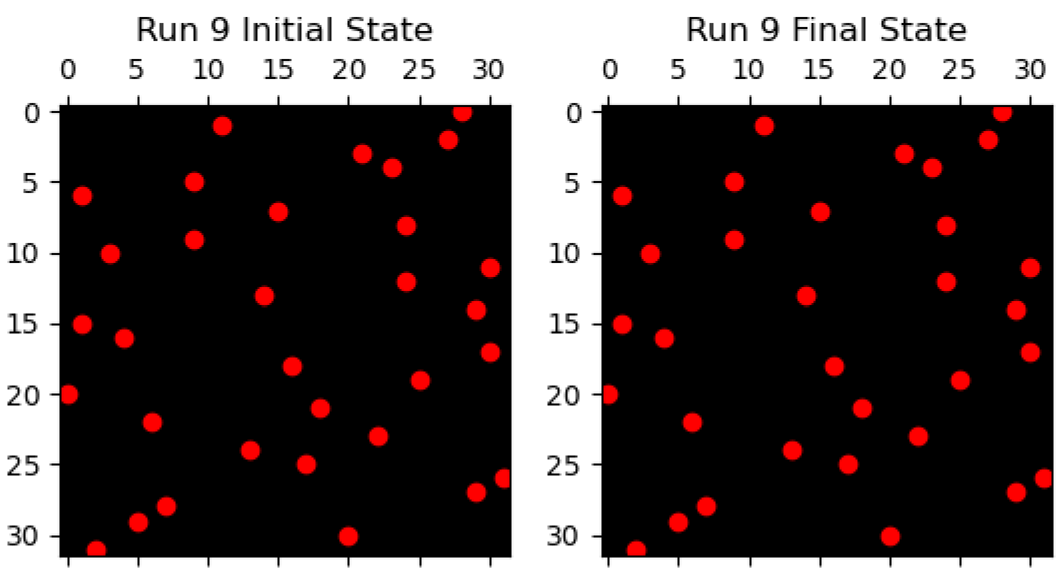
Run 8: Initial Attacks = 17, Final Attacks = 5

Run 9: Initial Attacks = 13, Final Attacks = 5

Run 10: Initial Attacks = 13, Final Attacks = 5

Total solutions found for N = 16: 0 out of 100 runs

**Number of Queens = 32, output for first run**




Run 1: Initial Attacks = 36, Final Attacks = 12

Run 2: Initial Attacks = 33, Final Attacks = 13

Run 3: Initial Attacks = 41, Final Attacks = 15

Run 4: Initial Attacks = 32, Final Attacks = 13

Run 5: Initial Attacks = 31, Final Attacks = 9

Run 6: Initial Attacks = 36, Final Attacks = 18

Run 7: Initial Attacks = 35, Final Attacks = 11

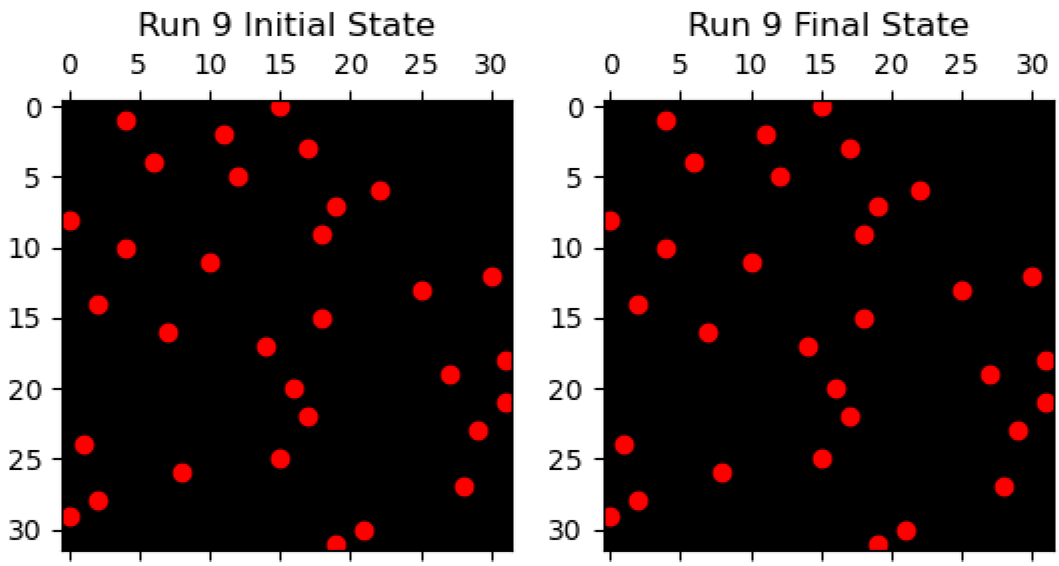
Run 8: Initial Attacks = 37, Final Attacks = 12

Run 9: Initial Attacks = 35, Final Attacks = 10

Run 10: Initial Attacks = 29, Final Attacks = 14

Total solutions found for N = 32: 0 out of 100 runs

**Number of Queens = 32, output for second run**




Run 1: Initial Attacks = 37, Final Attacks = 16

Run 2: Initial Attacks = 36, Final Attacks = 11

Run 3: Initial Attacks = 36, Final Attacks = 13

Run 4: Initial Attacks = 34, Final Attacks = 15

Run 5: Initial Attacks = 33, Final Attacks = 10

Run 6: Initial Attacks = 32, Final Attacks = 12

Run 7: Initial Attacks = 35, Final Attacks = 13

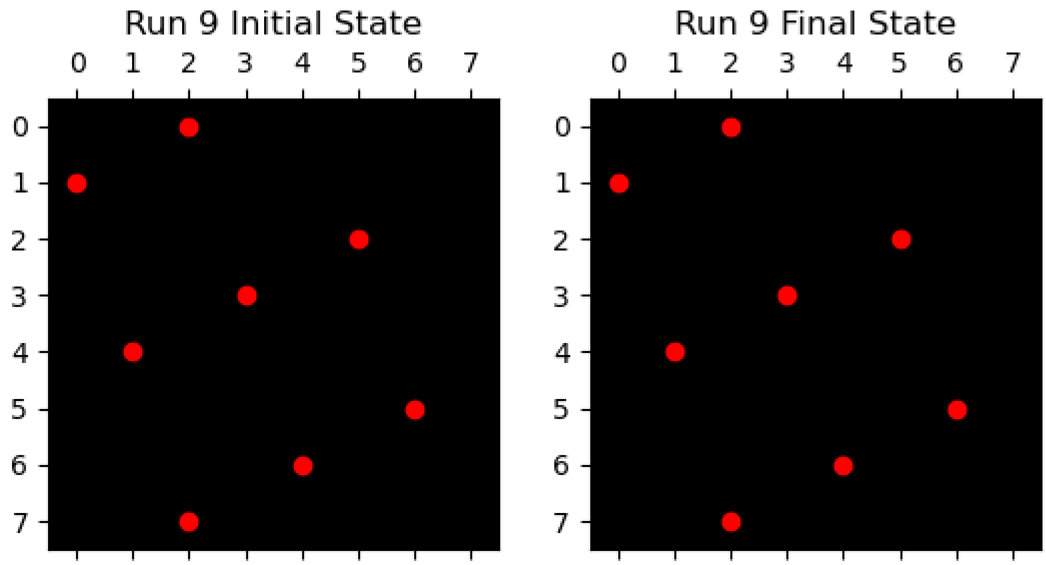
Run 8: Initial Attacks = 38, Final Attacks = 13

Run 9: Initial Attacks = 31, Final Attacks = 14

Run 10: Initial Attacks = 28, Final Attacks = 15

Total solutions found for N = 32: 0 out of 100 runs

**Number of Queens = 8, output for first  run**




Run 1: Initial Attacks = 11, Final Attacks = 2

Run 2: Initial Attacks = 6, Final Attacks = 1

Run 3: Initial Attacks = 4, Final Attacks = 1

Run 4: Initial Attacks = 7, Final Attacks = 1

Run 5: Initial Attacks = 11, Final Attacks = 1

Run 6: Initial Attacks = 11, Final Attacks = 2

Run 7: Initial Attacks = 11, Final Attacks = 2

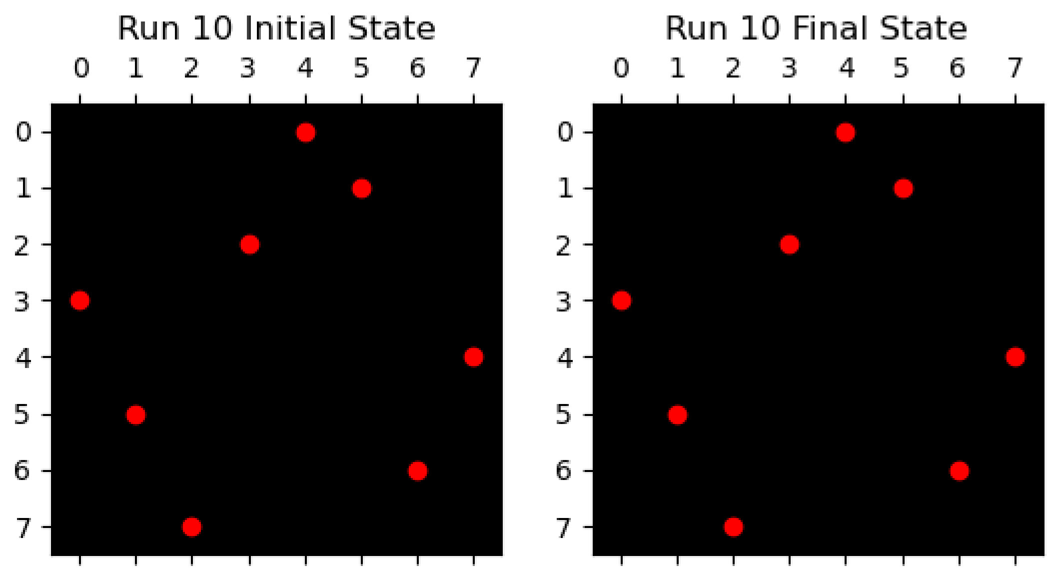
Run 8: Initial Attacks = 9, Final Attacks = 4

Run 9: Initial Attacks = 10, Final Attacks = 1

Run 10: Initial Attacks = 10, Final Attacks = 1

Total solutions found for N = 8: 7 out of 100 runs

**Number of Queens = 8, output for second  run**




Run 1: Initial Attacks = 7, Final Attacks = 1

Run 2: Initial Attacks = 9, Final Attacks = 1

Run 3: Initial Attacks = 4, Final Attacks = 0

Run 4: Initial Attacks = 6, Final Attacks = 1

Run 5: Initial Attacks = 8, Final Attacks = 2

Run 6: Initial Attacks = 14, Final Attacks = 1

Run 7: Initial Attacks = 7, Final Attacks = 1

Run 8: Initial Attacks = 6, Final Attacks = 2

Run 9: Initial Attacks = 7, Final Attacks = 2

Run 10: Initial Attacks = 6, Final Attacks = 2

Total solutions found for N = 8: 5 out of 100 runs

```python
import random
import numpy as np
import matplotlib.pyplot as plt


def GenerateRandomBoard(n):
    """
    :param n: Number queens for board
    :return: List of the random boards.
    """
    return [random.randint(0, n - 1) for _ in range(n)]


def ShowBoard(board, title1, title2):
    """

    :param board: Board with Queens
    :param title1: Initial State Plot
    :param title2: Final State Plot
    :return:
    """
    n = len(board)
    fig, (ax1, ax2) = plt.subplots(1, 2)

    # Plot initial state
    ax1.matshow(np.zeros((n, n)), cmap='gray')
    for i in range(n):
        ax1.plot(board[i], i, 'ro')
    ax1.set_title(title1)

    # Plot final state
    ax2.matshow(np.zeros((n, n)), cmap='gray')
    for i in range(n):
        ax2.plot(board[i], i, 'ro')
    ax2.set_title(title2)

    plt.show()


def GetQueenAttackCounts(board):
    """
    :param board: Any board state
    :return: Number of attacking queens
    """
    n = len(board)
    attacks = 0
    for i in range(n):
        for j in range(i + 1, n):
            if board[i] == board[j] or abs(board[i] - board[j]) == abs(j - i):
                attacks += 1
    return attacks


def HillClimbingAlgorithm(n, max_iterations=100):
    """
    :param n: Number of queens
    :param max_iterations: Number of times before the Algorithm ends
    :return:
    """
    solutions_found = 0

    for i in range(1, 101):  # Repeat the program 100 times
        board = GenerateRandomBoard(n)
        initial_attacks = GetQueenAttackCounts(board)
        current_attacks = initial_attacks
        iterations = 0
```

```python
        while current_attacks > 0 and iterations < max_iterations:
            neighbor = list(board)
            row = random.randint(0, n - 1)
            col = random.randint(0, n - 1)
            neighbor[row] = col
            neighbor_attacks = GetQueenAttackCounts(neighbor)

            if neighbor_attacks < current_attacks:
                board = neighbor
                current_attacks = neighbor_attacks

            iterations += 1

        if current_attacks == 0:
            solutions_found += 1

        if i <= 10:  # Plot initial and final states for the first 10 runs
            print(f"Run {i}: Initial Attacks = {initial_attacks}, Final Attacks =
{current_attacks}")
            ShowBoard(board, f"Run {i} Initial State", f"Run {i} Final State")

    return solutions_found


def main():
    """
    :return: Driver method.
    """
    N = 8
    solutions = HillClimbingAlgorithm(N)
    print(f"Total solutions found for N = {N}: {solutions} out of 100 runs")


if __name__ == "__main__":
    main()
```

How to execute the code.

Genetic Algorithm.

Chanage the paramerter n = 8,12,32 (number of queens)

Cd ~\EightQueen\GeneticAlgorithmEightQueen.py python3 GeneticAlgorithmEightQueen.py

Hill Climbing

Chanage the paramerter n = 8,12,32 (number of queens)

cd ~\EightQueen\HillClimbingAlgorightmEightQueen.py python3 HillClimbingAlgorightmEightQueen.py