Solving the N-Queens Problem Using Hill Climbing Algorithm

1. Problem Description

The N-Queens problem aims to place N queens on an N×N chessboard such that no two queens threaten each other. This means no two queens share the same row, column, or diagonal. In this project, the board size is fixed to N = 5.

2. State Representation

Each state is represented as a list of integers where the index represents the column and the value represents the row of the queen in that column.

Example:
[0, 2, 4, 1, 3]

This representation ensures that only one queen exists in each column.

3. Initial State

The algorithm starts with a random initial state where one queen is placed in each column. This initial configuration may contain conflicts.

4. Heuristic Function

The heuristic function used in Hill Climbing is the number of conflicting queen pairs. The objective is to minimize this value until it reaches zero, which represents a valid solution.

5. Hill Climbing Algorithm

At each step, the algorithm evaluates neighboring states generated by moving a queen within its column. The neighbor with the lowest number of conflicts is selected as the new current state. The process continues until no better neighbor is found.

6. Python Code

```python
import random

# Board size
N = 5

# Count conflicts between queens
def conflicts(state):
    count = 0
    for i in range(len(state)):
        for j in range(i + 1, len(state)):
```

```python
            if state[i] == state[j] or abs(state[i] - state[j]) ==
abs(i - j):
                count += 1
    return count

# Generate a random initial state
def random_state():
    return [random.randint(0, N - 1) for _ in range(N)]

# Hill Climbing function
def hill_climbing():
    current = random_state()
    current_conflicts = conflicts(current)

    while True:
        neighbors = []

        for col in range(N):
            for row in range(N):
                if row != current[col]:
                    neighbor = current.copy()
                    neighbor[col] = row
                    neighbors.append(neighbor)

        next_state = current
        next_conflicts = current_conflicts

        for state in neighbors:
            c = conflicts(state)
            if c < next_conflicts:
                next_state = state
                next_conflicts = c

        if next_conflicts >= current_conflicts:
            return current

        current = next_state
        current_conflicts = next_conflicts

# Run Hill Climbing
solution = hill_climbing()
print("Solution:", solution)
print("Conflicts:", conflicts(solution))
```

Example Output

Solution: [4, 2, 0, 3, 1]

Conflicts: 0

Each number represents the row position of the queen in the corresponding column.

7. Expected Behavior

Hill Climbing is fast and memory efficient, but it may get stuck in local optima. In some runs, the algorithm may not reach a conflict-free solution without random restarts.

8. Conclusion

The Hill Climbing algorithm demonstrates how local search techniques can efficiently handle constraint satisfaction problems like N-Queens. Although it does not guarantee an optimal solution, it provides fast approximate solutions and complements global search algorithms.