AI
LAB 3

Implement Hill Climbing search algorithm to solve N-Queens problem

```
# Hill Climbing Algorithm for N-Queens Problem
# Date: 5 November 2024
# College: BMSCE

import random

# ----- Configuration -----
N = 8  # You can change this to any N

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

# ----- Heuristic Function -----
def compute_conflicts(state):
    """Count number of pairs of queens attacking each other."""
    conflicts = 0
    for i in range(N):
        for j in range(i + 1, N):
            if state[i] == state[j] or abs(state[i] - state[j]) ==
abs(i - j):
                conflicts += 1
    return conflicts

# ----- Generate all neighbors -----
def get_neighbors(state):
    """Generate all possible neighbors by moving one queen in its
column."""
    neighbors = []
    for col in range(N):
        for row in range(N):
            if state[col] != row:
                new_state = state[:]
                new_state[col] = row
                neighbors.append(new_state)
    return neighbors

# ----- Hill Climbing Algorithm -----
def hill_climbing(max_restarts=1000):
```

```
    """Solve N-Queens using Hill Climbing with random restarts."""
    for restart in range(max_restarts):
        current = random_state()
        current_conflicts = compute_conflicts(current)

        while True:
            neighbors = get_neighbors(current)
            neighbor_conflicts = [compute_conflicts(n) for n in
neighbors]
            min_conflicts = min(neighbor_conflicts)

            # Find the best neighbor
            best_neighbor =
neighbors[neighbor_conflicts.index(min_conflicts)]

            # If no improvement, restart
            if min_conflicts >= current_conflicts:
                break

            # Move to the better neighbor
            current = best_neighbor
            current_conflicts = min_conflicts

        # Check if solution found
        if current_conflicts == 0:
            return current, restart

    return None, max_restarts
def print_board(state):
    """Display the chessboard."""
    for i in range(N):
        row = ""
        for j in range(N):
            row += " Q " if state[j] == i else " . "
        print(row)
    print("\n")

solution, restarts = hill_climbing()

print("Hill Climbing Search for N-Queens Problem")
print("-----------------------------------------")
print(f"Board Size: {N} x {N}")

if solution:
    print(f"✅ Solution Found after {restarts} random restarts!\n")
    print_board(solution)
else:
    print("⚠ No solution found after maximum restarts.")
```

OUTPUT:

```
Hill Climbing Search for N-Queens Problem
----------------------------------------
Board Size: 8 x 8
✅ Solution Found after 1 random restarts!

. . . . Q . . .
. . . . . . Q .
Q . . . . . . .
. . . Q . . . .
. Q . . . . . .
. . . . . . . Q
. . . . . Q . .
. . Q . . . . .
```

```
Hill Climbing Search for 4-Queens Problem
----------------------------------------
Board Size: 4 x 4
✅ Solution Found after 5 random restarts!

. Q . .
. . . Q
Q . . .
. . Q .
```