SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJ19ITL504 **DATE: 15-10 2024**

COURSE NAME: Artificial Intelligence Laboratory **CLASS: TY-IT**

NAME: Anish Sharma **ROLL No: I011**

EXPERIMENT NO: 5

CO/LO: Apply various AI approaches to knowledge intensive problem solving, reasoning, planning and uncertainty.

AIM / OBJECTIVE: Implement Local Search algorithm: Hill Climbing search for a suitable problem

DESCRIPTION OF EXPERIMENT:

- Students should select an appropriate problem.
- Demonstrate local search algorithm.
- Apply modifications/variations for overcoming the challenges in the implemented solution.

EXPLANATION / SOLUTIONS (DESIGN):

Code:

```
import random


# Function to calculate the number of conflicts in the current state
def calculate_conflicts(state):
    conflicts = 0

    n = len(state)
```

```
    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


# Hill Climbing Algorithm
def hill_climbing(n):
    # Generate a random initial state (one queen in each row, random column)
    current_state = [random.randint(0, n - 1) for _ in range(n)]
    current_conflicts = calculate_conflicts(current_state)

    while True:
        # Generate neighboring states
        neighbors = []
        for row in range(n):
            for col in range(n):
                if current_state[row] != col:
                    neighbor = current_state[:]
                    neighbor[row] = col

neighbors.append(neighbor)
```

```
    # Find the neighbor with the fewest conflicts
    best_neighbor = None
    best_conflicts = current_conflicts

    for neighbor in neighbors:
        conflicts = calculate_conflicts(neighbor)
        if conflicts < best_conflicts:
            best_conflicts = conflicts
            best_neighbor = neighbor

    # If no better neighbor, return the current state (local optimum)
    if best_conflicts >= current_conflicts:
        return current_state, current_conflicts

    # Move to the best neighbor
    current_state = best_neighbor
    current_conflicts = best_conflicts

# Random Restart Hill Climbing to avoid local maxima
```

```
def random_restart_hill_climbing(n, max_restarts=100):
    for _ in range(max_restarts):
        solution, conflicts = hill_climbing(n)
        if conflicts == 0:
            return solution  # Found a solution with 0 conflicts
    return None  # No solution found after max_restarts


# Main Function
if __name__ == "__main__":
    n = 8  # For example, solving 8-Queens problem
    solution = random_restart_hill_climbing(n)

    if solution:
        print(f"Solution found for {n}-Queens: {solution}")
    else:
        print(f"No solution found for {n}-Queens problem within the restart limit.")
```

Output and traversal path:

```
Solution found for 8-Queens: [0, 6, 3, 5, 7, 1, 4, 2]
```