**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:

# 