Experiment 5

Aim: Implement the Hill Climbing algorithm in Python.

Theory: Hill Climbing is a heuristic optimization algorithm used for solving computationally hard problems. It iteratively improves a solution based on a local search strategy. Starting from an initial state, it evaluates neighboring solutions and moves to a better neighbor if found. The algorithm stops when no better neighbor exists, potentially leading to a local optimum.

Applications of Hill Climbing include:

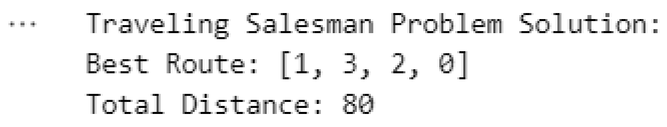
- Solving optimization problems like TSP and N-Queens
- Feature selection in machine learning
- Game playing algorithms

Hill Climbing can suffer from local maxima, plateaus, and ridges, which may require modifications like random restarts or simulated annealing to overcome.

Code:

```
import random
from typing import List, Callable, Tuple, Any

class HillClimbing:
    @staticmethod
    def solve(
        initial_state: Any,
        objective_function: Callable[[Any], float],
        generate_neighbors: Callable[[Any], List[Any]],
        max_iterations: int = 100
    ) -> Tuple[Any, float]:
        current_state = initial_state
        current_score = objective_function(current_state)

        for _ in range(max_iterations):
            neighbors = generate_neighbors(current_state)
            best_neighbor = max(neighbors, key=objective_function)
            best_neighbor_score = objective_function(best_neighbor)

            if best_neighbor_score <= current_score:
                break

            current_state = best_neighbor
            current_score = best_neighbor_score

        return current_state, current_score

# Example: Solving the Traveling Salesman Problem
def solve_traveling_salesman():
    distance_matrix = [
        [0, 10, 15, 20],
        [10, 0, 35, 25],
        [15, 35, 0, 30],
        [20, 25, 30, 0]
    ]

    def generate_neighbors(route: List[int]) -> List[List[int]]:
        neighbors = []
        for i in range(len(route)):
            for j in range(i+1, len(route)):
                neighbor = route.copy()
                neighbor[i], neighbor[j] = neighbor[j], neighbor[i]
                neighbors.append(neighbor)
        return neighbors

    def calculate_total_distance(route: List[int]) -> float:
        total_distance = 0
        for i in range(len(route) - 1):
            total_distance += distance_matrix[route[i]][route[i+1]]
        total_distance += distance_matrix[route[-1]][route[0]]
        return -total_distance
    initial_route = list(range(len(distance_matrix)))
    random.shuffle(initial_route)
    best_route, best_distance = HillClimbing.solve(
        initial_route,
        calculate_total_distance,
        generate_neighbors
    )

    print("Traveling Salesman Problem Solution:")
    print("Best Route:", best_route)
    print("Total Distance:", -best_distance)

def main():
    solve_traveling_salesman()

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

Output:



```
Traveling Salesman Problem Solution:
Best Route: [1, 3, 2, 0]
Total Distance: 80
```

Conclusion: The Hill Climbing algorithm efficiently solves optimization problems like the TSP by iteratively improving the solution. However, it may get trapped in local optima, highlighting the importance of enhancements such as random restarts or simulated annealing to achieve better results in complex scenarios.