WEEK 4:

Question 1

Solving the Symmetric Traveling Salesman Problem (sTSP) Using Variants of Hill Climbing Algorithms

Objective

The objective of this study is to analyze, implement, and compare multiple Hill Climbing algorithms for solving the Symmetric Traveling Salesman Problem (sTSP). The sTSP requires finding the shortest possible route that visits each city exactly once and returns to the starting point, with symmetric distances between cities.

This research aims to evaluate the strengths and weaknesses of different Hill Climbing strategies in optimizing tour length, computational efficiency, and convergence behavior. Specifically, we will examine:

- Simple Hill Climbing
- Stochastic Hill Climbing
- Steepest Ascent Hill Climbing

By conducting experiments on benchmark datasets, we will assess the effectiveness of each algorithm and provide insights into their suitability for solving the sTSP.

Scope of Work

1. **Data Processing**
    - Parse and preprocess TSP datasets from **TSPLIB95**.
    - Extract city coordinates and compute the corresponding distance matrix.
2. **Algorithm Implementation**
    - Implement **Simple Hill Climbing**, **Stochastic Hill Climbing**, and **Steepest Ascent Hill Climbing** for solving TSP.
    - Ensure efficient solution representation and neighborhood search strategies.
3. **Experimental Evaluation**
    - Run the algorithms on benchmark datasets:
        - rd100.tsp
        - eil101.tsp
        - a280.tsp
        - d198.tsp
        - ch150.tsp
    - Compare performance based on:
        - **Solution quality** (total tour distance)
        - **Convergence behavior** (iterations to reach a solution)
        - **Computational time**
4. **Comparative Analysis & Insights**
    - Identify which algorithm produces the best results for different datasets.
    - Analyze trade-offs between solution quality and runtime.
    - Discuss algorithmic limitations and potential improvements.

Deliverables

1. Implementation Code

- Python scripts for parsing TSP datasets and computing distance matrices.
- Implementations of all selected Hill Climbing algorithms.
- Execution of the algorithms on benchmark datasets.

2. Experimental Results

- A comparative table summarizing the tour cost (total distance) for each algorithm and dataset.
- Visualizations (e.g., tour plots, convergence graphs) for performance evaluation.
- Observations on solution quality, convergence, and computational efficiency.

3. Research Report

- Explanation of methodology and algorithmic strategies.
- Comparative analysis of the algorithms, including best/worst-performing cases.

EXPLORE ABOUT Random Restart Hill Climbing

Question 2

****

Objective: This work aims to apply different Hill Climbing algorithms—simple Hill Climbing, Stochastic Hill Climbing, and Steepest Ascent Hill Climbing—to solve a Job Scheduling Optimization problem. The goal is to minimize the total processing time of jobs on multiple machines by efficiently assigning jobs to machines while respecting the constraints.

Deliverables Required:

1. **Python Implementation:**
    - A Python implementation of Simple Hill Climbing, Stochastic Hill Climbing, and Steepest Ascent Hill Climbing for solving the job scheduling problem.
    - The implementation should include:
        - Generation of random initial schedules.
        - Calculation of the total processing time based on job assignments.
        - Performance comparison across different job sizes.
2. **Heuristic Function:**
    - An explanation of the heuristic function used to evaluate the quality of job schedules.
    - The heuristic function should calculate the total processing time or cost for each schedule.
3. **Algorithm Comparison:**
    - A detailed comparison of the performance of the three algorithms:
        - Metrics to include: total processing time, execution time, and success rate (i.e., finding an optimal solution or near-optimal solution).
        - Comparison should be made across varying numbers of jobs (e.g., 10, 50, 100,200, 500). And number of machine (ex: 10, 100, 100,1000, 10000, 100000)
4. **Graphs and Results:**
    - Graphs or tables showing the performance of each algorithm in terms of total processing time and execution time for different job sets.
    - A discussion of the strengths and weaknesses of each algorithm based on the results.

****

Question 3

N-Queens Problem Using Local Search Methods

The **N-Queens Problem** is a classical constraint satisfaction problem where the goal is to place **N queens on an N×N chessboard** such that no two queens attack each other. This means:

- No two queens can be in the same row.
- No two queens can be in the same column.

- No two queens can be on the same diagonal.

In this exercise, we will **solve the N-Queens Problem** using three **Hill Climbing** variants:

1. **Simple Hill Climbing** – Moves to a better neighboring state if one exists, stopping at local optima.
2. **Stochastic Hill Climbing** – Chooses a random better move instead of always picking the best one.
3. **Steepest Ascent Hill Climbing** – Evaluates all possible moves and chooses the best improvement.

Tasks to Perform

For **N = 4, 8, 16, 32, 64**, perform the following:

1. **Generate an Initial Board:** Start with a random configuration where some queens may be attacking each other.
2. **Apply Each Hill Climbing Algorithm:**
    - Implement **Simple Hill Climbing**, **Stochastic Hill Climbing**, and **Steepest Ascent Hill Climbing**.
    - Track the number of iterations taken to find a valid solution (or detect failure due to local optima).
    - Compare performance across different values of **N**.
    - Add, additional performance metrics such as the **number of iterations**.
3. Add, additional performance metrics such as the **number of iterations**.

4. **Analyze and Compare:**
    - Measure the number of restarts needed if a local optima is reached.
    - Compare success rates, execution times, and efficiency of each algorithm.

Deliverables

1. **Code Implementation** for the three Hill Climbing approaches.
2. **Performance Analysis** in the form of tables and graphs comparing iterations, success rate, and efficiency.
3. **Discussion & Conclusion** on which algorithm performs best and why.
4. **Observations** on how increasing N affects each method's ability to find a solution.