National University of Computer & Emerging Sciences

AL2002 – Artificial Intelligence – Lab (Spring 2025)

BSCS-6B

Lab Work 4 (Hill Climbing Search, Simulated Annealing, Local Beam Search)

Lab Instructor	Momna Javaid
Department	Computer Science

Instructions:

1. You also have to submit .ipynb file.
2. Comments in the code explaining chunks of the code are important.
3. Plagiarism is strictly prohibited, 0 marks would be given to students who cheat.

Lab Tasks:

Task 1:

A robot needs to find an optimal path from its starting position to a target position in a **2D grid** with obstacles. The goal is to reach the target **while minimizing the cost** of movement, considering constraints such as grid boundaries and obstacles.

Problem Definition

- The **environment** is a **grid-based map** where each cell has a cost associated with movement.
- The **robot** can move **up, down, left, or right** but **cannot pass through obstacles**.
- The **goal** is to find the shortest path (minimum cost) from the start position to the target.
- The **heuristic function (h)** is the **Manhattan distance** from the current position to the goal:

$$h(x,y) = |xgoal - x| + |ygoal - y|$$

- **Constraints:**
  - The robot **cannot move diagonally**.
  - The robot **cannot move outside the grid boundaries**.
  - The robot **cannot move into an obstacle cell**.
  - The robot always chooses the neighbor with the lowest heuristic value (greedy choice).

Possible Outcomes

1. **Success**: The robot reaches the goal efficiently.
2. **Failure (Local Maximum)**: The robot gets stuck in a state where no neighbor has a better heuristic.

Solution to Local Maxima:

- **Random Restart Hill Climbing:** Restart from a new random position.
- **Simulated Annealing:** Occasionally accept worse moves to escape local maxima.

Algorithm:

1. Initialize the Current State
2. Repeat Until the Goal is Reached or No Better Move Exists:
   a. **Select the neighbor with the lowest heuristic value (`best_h`).**

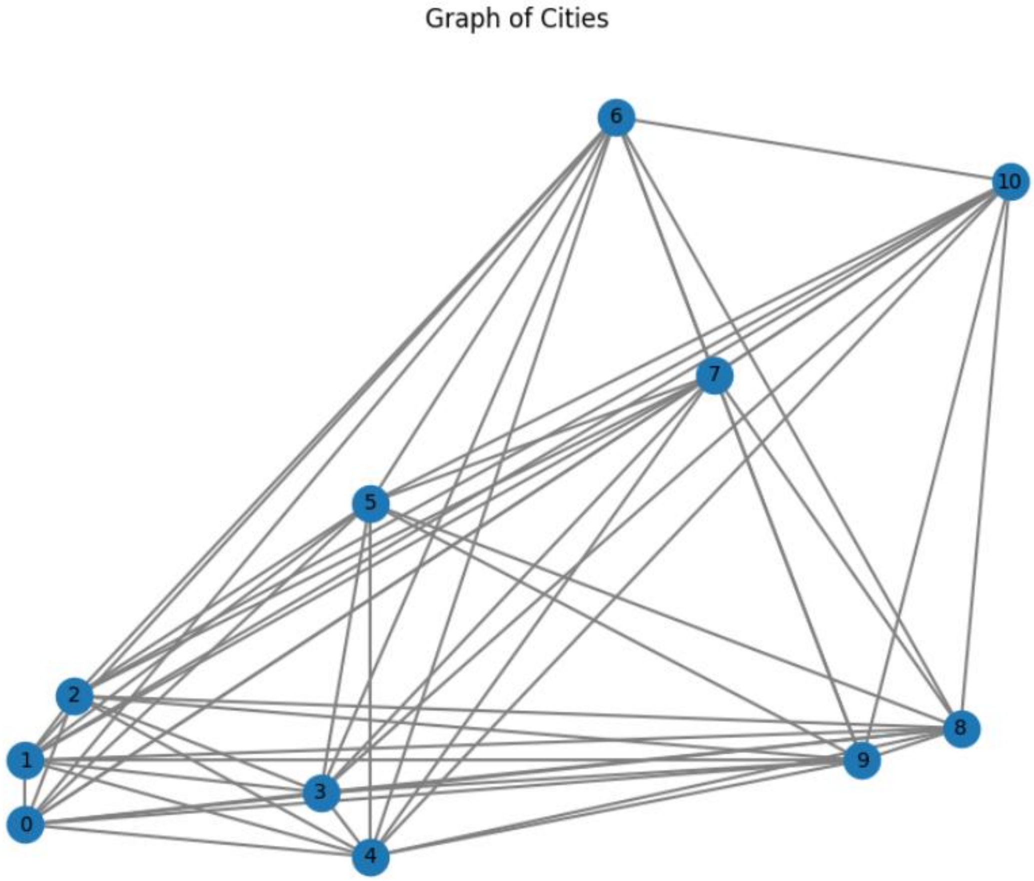
   i. If `best_h < h(current)`, move to that neighbor.
   ii. Otherwise, **stop (local maximum reached)**.

3. Check Termination Conditions:
   a. If `current == goal`, **return success**.
   b. If no valid move improves `h(current)`, **return failure** (local maximum).

Calculate Time complexity and Space Complexity.

Task 2:

A salesman must visit **N cities** exactly **once** and return to the starting city while minimizing the total travel distance. The goal is to find a near-optimal route using **Simulated Annealing (SA)**.



Problem Definition

- **Given**: A set of **N cities** and a **distance matrix** D[i][j]D[i][j]D[i][j] representing the travel cost between city iii and city jjj.
- **Objective**: Find the shortest path that **visits each city exactly once** and returns to the starting city.
- **Solution Representation**: A **permutation** of cities (e.g., **[A → B → C → D → A]**).
- **Evaluation Function**: The total path length

$$E(s) = \sum_{1=1}^{N} D[s_i][s_{i+1}]$$

- **Neighbor Function**: A new path is generated by **swapping** two cities in the current path.

Constraints

1. **Each city must be visited exactly once**.
2. **The solution must be a valid cycle** (returning to the starting city).
3. **Swaps must maintain feasibility** (no repeated cities).
4. **The algorithm must accept worse solutions occasionally** to escape local optima.
5. **Annealing schedule controls temperature decay** over time.

Formulas Used

1. **Energy Function (Cost Function)**

$$E(s) = \sum_{1=1}^{N} D[s_i][s_{i+1}]$$

- **Lower energy (cost)** means a better solution.
- **Goal**: Minimize E(s).

2. Acceptance Probability (Metropolis Criterion)

A worse solution may be accepted with probability:

$$P = e^{\frac{E(s_{current}) - E(s_{new})}{T}}$$

where:

- $E(s_{current})$ = Energy of the current solution.
- $E(s_{new})$ = Energy of the new solution.
- T = Temperature (controls randomness).

3. Temperature Decay Schedule

The temperature gradually **decreases** using:

$$T = T_0 . \alpha$$

where:

- T_0 = Initial temperature.
- α = Cooling rate (e.g., 0.95).
- k = Current iteration number.

Algorithm for Simulated Annealing (TSP)

Input:

- Distance matrix D[i][j]
- Number of cities N
- Initial temperature T_0
- Cooling rate α
- Iteration limit

Algorithm:

Steps:

1. **Initialize:**
   - Start with a random tour s.
   - Compute initial energy E(s).
   - Set $T=T_0$
2. **Repeat Until Stopping Condition is Met:**
   - Generate a new solution s_{new} by swapping two cities.
   - Compute the energy difference:
$$\Delta E = E(s_{new}) - E(s_{current})$$
   - If $\Delta E<0$, accept s_{new} (better solution).
   - If $\Delta E>0$, accept s_{new} with probability
$$P = e^{-\Delta E/T}$$
   - Update temperature:
$$T = T_0 . \alpha$$
3. Stop if Temperature is Too Low or Max Iterations Reached.
4. Return the Best Found Tour.

Calculate time complexity and space complexity and find complete path

Task 3:

A **mountaineer** wants to reach the **highest peak** from a given starting point in a **mountain terrain grid**. The mountaineer can move **up, down, left, or right**, but they always choose the neighboring position with the **highest elevation**. The goal is to **maximize elevation** while avoiding obstacles and local maxima.

Problem Definition

- The **terrain** is represented as a **2D elevation grid** M[x][y], where M[i][j] denotes the height at position (i,j).
- The **starting position** is $S(x_s,y_s)$).
- The **goal is to reach the highest elevation point**.
- The **evaluation function (heuristic)** is the elevation value: h(x,y)=M[x][y]
- The **mountaineer always moves to the neighbor with the highest elevation**.

 The mountaineer can only move to a valid neighbor:

- **Up** (x,y+1)
- **Down** (x,y−1)
- **Left** (x−1,y)
- **Right** (x+1,y)

 Boundary Constraint: The movement must be within the terrain grid.
 Obstacle Constraint: Some cells contain obstacles (e.g., cliffs) and are not accessible.

 Local Maximum Constraint: If no neighboring cell has a higher elevation, the search terminates (even if the global highest peak is elsewhere).

Only include moves that **are within bounds** and **not obstacles**.

Steps:

1. **Initialize**
   - Set current = (x_s, y_s).
   - Compute initial **elevation**: h(current)=M[x_s][y_s].
2. **Repeat Until No Higher Elevation Neighbor Exists:**
   - Get **valid neighbors** of current.
   - Compute their elevation values h(x,y).
   - Select the **neighbor with the highest elevation**.
   - If **no neighbor has a higher elevation**, **terminate (local maximum reached)**.
   - Otherwise, **move to the best neighbor** and repeat.
3. **Return the Reached Peak.**

Calculate time complexity and space complexity and complete path.