Faculty of Computing and Information Technology

University of the Punjab, Lahore

Artificial Intelligence Lab 4

27th August, 2024

Instructor: Qamar U Zaman

A* Search Algorithm and 8-Puzzle Problem

Objective:

Understand the A* (A-star) search algorithm and apply it to solve the 8-puzzle problem. This lab outlines the key steps and provides a code template without implementation logic.

---

1. *Introduction to A Search Algorithm**

A* is a search algorithm used to find the shortest path by minimizing a cost function. It uses:

- **g(n)**: Path cost from start to current node.
- **h(n)**: Estimated cost from current node to the goal.
- **f(n) = g(n) + h(n)**: The total estimated cost.

2. The 8-Puzzle Problem

The 8-puzzle consists of a 3x3 grid with tiles numbered 1 to 8 and one empty space. The goal is to rearrange the tiles by sliding them into the empty space until the goal configuration is reached.

---

3. Steps to Solve the 8-Puzzle Using A*

1. **Define the Problem:**
   - Start state, goal state, valid moves, and cost function `g(n)`.
   - Use a heuristic `h(n)` like Manhattan Distance or Misplaced Tiles.
2. **Priority Queue:**
   - Use an open list (priority queue) ordered by `f(n) = g(n) + h(n)` and a closed list for explored nodes.
3. **Expand Nodes:**
   - Expand the node with the lowest `f(n)` and generate children based on valid moves.
4. **Repeat:**
   - Continue until the goal is found or the open list is empty.
5. **Solution Trace:**
   - Trace back from the goal to get the solution path.

---

4. Heuristic Functions

- **Manhattan Distance**: Sum of the distances of each tile from its goal position.
- **Misplaced Tiles**: Number of tiles not in their correct positions.

5. Code Template

Here is a template for the A* implementation. You need to add the logic.

```
class PuzzleNode:
    def __init__(self, state, parent, move, g_cost, h_cost):
        # Initialize node with state, parent, move, g_cost, and h_cost
        pass

    def generate_children(self):
        # Generate possible child nodes by moving the empty tile
        pass

    def calculate_heuristic(self, goal_state):
        # Calculate heuristic based on the current state and goal
        pass

class AStarSolver:
    def __init__(self, start_state, goal_state):
        # Initialize the A* solver with start and goal states
        pass

    def solve(self):
        # Implement the A* algorithm to solve the puzzle
        pass

    def trace_solution(self, node):
        # Trace back from the goal to get the solution path
        pass

    def is_solvable(self, state):
        # Check if the puzzle state is solvable
        Pass
```

6. Lab Tasks:

1. Implement the A* algorithm for solving the 8-puzzle.
2. Choose and implement a heuristic function.
3. Test your solution with different start states.
4. Summarize your findings on the performance and heuristic impact.