Faculty of Computing and Information Technology

University of the Punjab,

Lahore

Artificial Intelligence Lab 5

17 October, 2024

Instructor: Qamar U Zaman

1. Greedy Best-First Search (GBFS)

Introduction to Greedy Best-First Search (GBFS)

Greedy Best-First Search (GBFS) is a heuristic search algorithm that expands the node that appears to be closest to the goal according to a heuristic function, typically **h(n)**.

- **h(n)**: The estimated cost from the current node to the goal (heuristic function).
- In GBFS, the algorithm selects the node with the smallest **h(n)**, focusing only on which node appears to be the closest to the goal.

Problem: The 8-Puzzle Problem

The 8-puzzle consists of a 3x3 grid with numbered tiles from 1 to 8 and one empty space. The objective is to slide the tiles around until they match the goal configuration. The puzzle looks like this:

Start State:

```
1  2  3
4  0  5
6  7  8
```

Goal State:

```
1  2  3
4  5  6
7  8  0
```

- **Start State**: The initial configuration of the puzzle.
- **Goal State**: The target configuration to reach by sliding the tiles into the empty space (represented by `0`).

Heuristic: Use the **Manhattan Distance** as the heuristic: $h(n)=|x1-x2|+|y1-y2|$

where (x1,y1) is the current node, and (x2,y2)(is the goal node.

Code Template:

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

    def generate_children(self):
        # Generate possible child nodes by moving in 4 directions (up, down,
left, right)
        pass

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

class GreedyBestFirstSearch:
    def __init__(self, start_state, goal_state):
        # Initialize the search with start and goal states
        pass

    def solve(self):
        # Implement GBFS to find the goal
        pass

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

Lab Tasks:

1. Implement the **Greedy Best-First Search** algorithm for the 2D grid.
2. Use the **Manhattan Distance** heuristic.
3. Find the path from the start position (0,0) to the goal position (4,4).

---

2. Minimax Algorithm

Introduction to Minimax Algorithm

The Minimax algorithm is used to minimize the possible loss in a worst-case scenario and is typically applied in two-player, zero-sum games.

- **Maximizer**: Tries to get the highest score.
- **Minimizer**: Tries to get the lowest score.
- **Terminal States**: End states of the game (e.g., win, lose, draw).

Code Template:

```
class Minimax:
    def __init__(self, game_state):
        # Initialize with the current game state
        pass

    def is_terminal(self, state):
        # Check if the game has reached a terminal state (win/lose/draw)
        pass

    def utility(self, state):
        # Return the utility value of the terminal state
        pass

    def minimax(self, state, depth, maximizing_player):
        # Implement the Minimax algorithm
        pass

    def best_move(self, state):
        # Determine the best move using Minimax
        pass
```

Lab Tasks:

1. Implement the Minimax algorithm for a two-player game (e.g., Tic-Tac-Toe).
2. Test the algorithm for different board configurations.