Solving the 8-Puzzle Problem using Algorithms- A* and Greedy Best-First Search

Group-6
Anusree Roy 2011364042 Mashruba Akter Sumona 1913062042
Nusrat Jahan Pritu 2011920642

Introduction

Our problem was to create a puzzle solver that efficiently tackles diverse puzzles using multiple search algorithms, with customized heuristics for informed searches, and it should offer user-friendly input, algorithm selection, and performance comparison. For the puzzle, we chose the eight-puzzle problem that consists of a 3x3 grid with eight numbered tiles and one blank space. The goal is to rearrange the tiles to match the target or goal state by sliding one tile at a time into the blank space. In AI, the 8-puzzle problem serves as a benchmark for evaluating search algorithms, particularly those aimed at finding the optimal path to the goal state. Among the commonly used search algorithms, A* and Greedy Best-First Search are popular choices due to their effectiveness and efficiency. For this problem, we examine the implementation of the A* algorithm and the Greedy Best-First Search algorithm for solving the 8-puzzle problem. To guide the search, we also explore heuristic functions such as Manhattan Distance and Misplaced Tiles.

Problem Definition

The 8-puzzle consists of a grid where the tiles are labeled from 1 to 8, with the blank space represented by 0. The goal is to arrange the tiles in a configuration that is:
1 2 3 4 5 6 7 8 0
Given an arbitrary initial state, the challenge is to find the sequence of moves that will lead to the goal state.

State Representation

A state in the puzzle is represented as a list of integers, where each position in the list corresponds to a tile or a blank space. For example, the following list represents an initial puzzle configuration: 2, 8, 3, 1, 6, 4, 7, 0, 5 In this state, tile 2 is in the top-left corner, and the blank space (0) is in the middle of the bottom row.

Search Algorithms

A* Search

A* finds the optimal path to the goal by combining the cost from the start to a node (g(n)) and the estimated cost from the node to the goal (h(n)). The total cost for a node is: f(n) = g(n) + h(n)Where: g(n) is the cost to reach the current state from the initial state, h(n) is the heuristic estimate of the cost to reach the goal state from the current state. A* explores the node with the lowest total cost, prioritizing states that are estimated to be closer to the goal. It uses heuristics to guide the search.

Greedy Best-First Search

Greedy Best-First Search always expands the node that appears to be closest to the goal based on the heuristic function alone. The cost function for Greedy is: f(n) = h(n). This means that Greedy does not see the path's cost already taken (g(n)), focusing solely on minimizing the estimated distance to the goal.

Heuristic Functions

Manhattan Distance

The Manhattan Distance heuristic calculates the total number of horizontal and vertical moves required to get each tile to its correct position. For each tile, the Manhattan distance between its current and target positions is computed and summed. $h(n)=\sum(|x_{current}-x_{goal}|+|y_{current}-y_{goal}|)$

Misplaced Tiles

The Misplaced Tiles heuristic counts how many tiles are currently in the wrong position compared to the goal state. In 2, 8, 3, 1, 6, 4, 7, 0, 5 the misplaced tiles are 2, 8, 1, 6, 4, and 5. Therefore, the heuristic value is 6.

Implementation

We implemented the EightPuzzle class, which encapsulates the puzzle state, goal test, and heuristics. The Solver class implements A* and Greedy Best-First Search. Then there is user_interface.py for friendly user input and performance.py for comparing between the algorithms.

A *Implementation

The A* algorithm maintains a priority queue (min-heap) to store frontier nodes. Each node in the frontier has **f(n)**: Total cost (g + h), **g(n)**: Cost from the start, **State**: Current puzzle configuration. For each iteration, the node with the lowest f(n) is expanded., and the goal test is performed. If the goal state is reached, the solution path is reconstructed using a parent map. For each neighbor of the current state, the total cost (g(n) + h(n)) is calculated, and the neighbor is added to the frontier.

Greedy Best-First Search Implementation

The Greedy algorithm follows a similar structure to A*, but the priority queue only considers the heuristic value h(n). It selects the node that appears closest to the goal without considering the cost to reach that node (g(n)).

Challenges

One of the main challenges in solving the 8-puzzle problem is choosing an appropriate heuristic. While Manhattan Distance tends to guide the search more effectively than Misplaced Tiles, it is computationally more expensive. In contrast, Greedy Best-First Search, although faster, does not guarantee an optimal solution.

Performance

To compare the performance between two algorithms, 2 8 3 1 6 4 7 0 5 initial state was taken. In the case of A* with misplaced tiles, a solution is found after expanding several nodes. Time taken: 6.87 seconds. In the case of greedy, a solution is found but not guaranteed to be optimal. Time taken: 5.15 seconds. The results show that A* generally takes longer than Greedy, but it guarantees finding the optimal solution. Greedy is faster but may produce suboptimal solutions.

Conclusion

In this report, we implemented and compared two heuristic search algorithms: A* and Greedy Best-First Search, to solve the 8-puzzle problem. We also explored the effectiveness of two heuristics: Manhattan Distance and Misplaced Tiles. A* finds optimal solutions but takes longer, while Greedy is faster but can produce suboptimal results.