8-Puzzle Problem Solution Using A* Algorithm

Introduction

The 8-puzzle problem is a classic search problem involving a 3x3 grid with eight numbered tiles and one blank space. The objective is to rearrange the tiles from an initial configuration to a goal configuration by sliding the tiles into the blank space, using the least number of moves possible. This report presents a solution to the 8-puzzle problem using the A* search algorithm, which utilizes both the Hamming and Manhattan distance heuristics to guide the search.

Problem Description

The 8-puzzle consists of a 3x3 grid containing eight numbered tiles (1 to 8) and one blank space. The goal is to reach a specific target configuration, typically the tiles arranged in ascending order with the blank space in the bottom-right corner. The problem is computationally challenging due to the large number of possible states (9! = 362,880), and efficient algorithms are required to find the solution within a reasonable time.

Solution Approach

The solution employs the A* search algorithm, which combines the cost to reach the current state (g) and a heuristic estimate of the cost to reach the goal (h). The algorithm uses two heuristics:

- Hamming Distance: Counts the number of tiles out of place compared to the goal state.
- Manhattan Distance: Sums the vertical and horizontal distances each tile is from its goal position.

The A* algorithm maintains a priority queue to explore states in order of their estimated total cost (f = g + h). The search continues until the goal state is reached or all states have been explored.

8-Puzzle Problem Solution Using A* Algorithm

Implementation Details

The implementation includes the following key components:

- State Representation: A structure `State` stores the board configuration, the position of the blank space, and the cost incurred to reach the state.
- Priority Queue: A priority queue is used to manage the states during the search, ordered by their total cost (g + h).
- Heuristic Functions: The functions `calculateHammingPriority` and `calculateManhattanPriority` calculate the respective heuristics for a given state.

Key Functions:

1. `initializeState(int initial[N][N])`: Initializes the state with the given board configuration.
2. `isGoalState(State* state)`: Checks if the current state matches the goal configuration.
3. `calculateHammingPriority(State* state)`: Computes the Hamming distance for the current state.
4. `calculateManhattanPriority(State* state)`: Computes the Manhattan distance for the current state.
5. `aStarSearch(State* initialState)`: The main function that implements the A* search algorithm.

Results and Testing

The solution was tested with an initial state configuration:

```c
int initial[N][N] = {
    {1, 2, 3},
    {4, 0, 5},
```

```
    {6, 7, 8}
};
```

The algorithm successfully found the solution path to the goal state, demonstrating the effectiveness of the A* algorithm with the chosen heuristics. The Hamming and Manhattan priorities for the initial state were calculated, guiding the search process efficiently.

Conclusion

The A* algorithm provides an efficient solution to the 8-puzzle problem by leveraging heuristic functions to estimate the cost to reach the goal. The use of Hamming and Manhattan distances as heuristics ensures that the search is guided towards the most promising states, reducing the number of explored states. Future work could explore alternative heuristics or optimization techniques to further enhance the performance.