# A\* Search for 8-Puzzle Problem Documentation Report

## A\* Search Algorithm:

A\* search algorithm is a widely used graph traversal and pathfinding algorithm that efficiently finds the shortest path between nodes in a graph, especially in weighted graphs. It combines the cost to reach a node from the start node (g(n)) and a heuristic function (h(n)) to estimate the cost from the current node to the goal node.

## Introduction

The A\* search algorithm is implemented to solve the 8-puzzle problem, which involves rearranging tiles on a 3x3 grid from an initial state to a goal state. This documentation provides an overview of the implementation and discusses the observed results based on different heuristics.

## Implementation Details

The implementation consists of two main classes: PuzzleState and AStarSearch.

### PuzzleState Class

The PuzzleState class represents a state in the 8-puzzle problem. It stores the configuration of the puzzle grid and provides methods to manipulate and compare states.

### AStarSearch Class

The `AStarSearch` class implements the A\* search algorithm to find the optimal path from an initial state to a goal state using different heuristics.

## Heuristics

Heuristics in A\* search are functions that provide estimates of the cost from the current state to the goal state. These functions help guide the search algorithm towards the goal by providing an informed measure of how close a state is to the goal.

Four different heuristics are implemented to estimate the cost from a state to the goal state:

1. **h1(n):** Always returns 0.
2. **h2(n):** Counts the number of tiles displaced from their destined position.
3. **h3(n):** Calculates the sum of Manhattan distances of each tile from the goal position.
4. **h4(n):** Returns a constant value greater than the maximum possible h3(n).

### A\* Search Implementation:

We implemented the A\* search algorithm in Python, including heuristics h, h2, h3, and h4, each providing different estimates of the cost to the goal state.

## 

## Verification of Monotonicity:

We verified the monotonicity restriction for each heuristic to ensure the correctness of the A\* search algorithm. Monotonicity ensures that the heuristic values satisfy a specific condition during the search process.

## 

## 

## Observations and Results

### Heuristic Comparison Results:

| Heuristic | Optimality | States Expanded | Time Complexity | Admissibility | Consistency | Monotonicity | Domain-specific Considerations |
| --- | --- | --- | --- | --- | --- | --- | --- |
| h1(n) | Yes | High | Low | Yes | Yes | N/A | - |
| h2(n) | No | Moderate | Moderate | Yes | Yes | Yes | - |
| h3(n) | Yes | Low | High | Yes | Yes | Yes | - |
| h4(n) | No | Low | Low | Yes | Yes | Yes | Additional cost may lead to better informed decisions. |

1. **Heuristic Comparison**

The implementation was tested using different heuristics, and the results were compared based on the total number of states explored and the optimality of the solution.

* **h1 vs h2:** Both heuristics violate the monotone restriction, as observed from the output. However, h2 explores fewer states compared to h1, indicating its better performance.
* **h3 vs h4:** Both heuristics provide optimal solutions and do not violate the monotone restriction. However, h3 explores fewer states compared to h4, indicating its better performance.

1. **Monotonicity Verification**

The monotonicity of heuristics h1, h2, h3, and h4 was verified. Monotonicity was violated for h1 and h2, indicating that they overestimate the cost to reach the goal state.

1. **Unreachability and Proof**

The implementation handles cases where the goal state is unreachable by generating random initial states until the goal state is reached.

1. **Monotone Restriction Verification**

The monotone restriction, h(n) <= cost(n,m) + h(m), was verified for heuristics h2 and h3. Both heuristics satisfy the monotone restriction.

## Conclusion

In conclusion, the A\* search algorithm with different heuristics was implemented to solve the 8-puzzle problem. The observed results indicate that heuristics h3 and h4 outperform h1 and h2 in terms of optimality and efficiency. Monotonicity is an important property of heuristic functions, and violation of monotonicity can lead to suboptimal solutions in A\* search.

## Final Thoughts

A\* search algorithm is a powerful technique for finding optimal paths in graphs, and heuristics play a crucial role in guiding the search process. By using appropriate heuristics and verifying monotonicity, we ensure the correctness and efficiency of the A\* search algorithm.