**References Mentioned:**

**[1]https://www.geeksforgeeks.org/8-puzzle-problem-in-ai/**

**[2]Figure 1 source:** [**https://www.wikihow.com/Solve-8-Puzzle**](https://www.wikihow.com/Solve-8-Puzzle)

**[4] Russell, S. J., & Norvig, P. (1995). Artificial Intelligence: A Modern Approach**

**[5] CS205 slides By Professor Keogh**

# Introduction

Sliding-tile puzzles are a classic AI benchmark in which numbered tiles are slid into an empty space to reach a goal configuration. The 8-Puzzle consists of a 3×3 board containing eight tiles (numbered 1–8) and one blank. From any arrangement, the blank may move Up, Down, Left, or Right, swapping places with an adjacent tile. The objective is to transform a given start state into the goal state by a sequence of such moves. Figure 1 illustrates a sample puzzle in its goal state configuration.

This report implements three search strategies—

* 1. Uniform-Cost Search.
  2. A\* with the Misplaced-Tile heuristic.
  3. A\* with the Manhattan-Distance heuristic

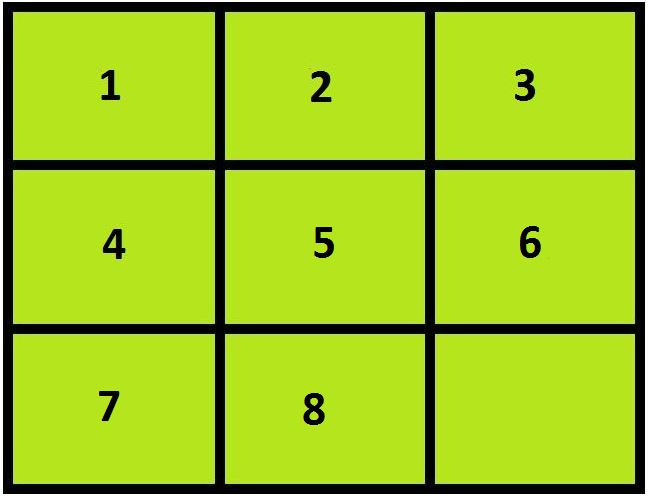
****

Figure 1: Image of an 8 Puzzle Problem

The eight-puzzle game provides a very intuitive way to understand the theory behind how to model a Search problem.

This report clearly outlines the resources used to solve this 8-Puzzle problem.

In order to have an efficient strategy the problem needs to be formulated clearly

A problem can be defined formally by five components [4]:

Initial State, Action, Transition Model, Path Cost, Goal test.

By efficiently formulating the above components, an algorithm can be designed to reach an optimal solution if possible.

Three different path cost strategies are analyzen namely-

1. Uniform cost Search –
   * Uniform Cost search is an Uninformed Search Strategy.
2. A\* Search with Misplaced Tile Heuristic
3. A\* Search with Manhattan distance Heuristics

# Problem Formulation

For all the 3 search strategies the components are defined as below

1. Initial State: Represented using a list carrying 8 numbers and 0 representing the empty position-For example [1,2,3,4,5,6,7,0,8]
2. Action: What actions can be performed on the state.
   1. UP – Move the empty tile up
   2. Down – Move the empty tile down
   3. Right – Move the empty tile right
   4. Left – Move the empty tile left
3. Transition Model: This showcases how the actions transform a state.

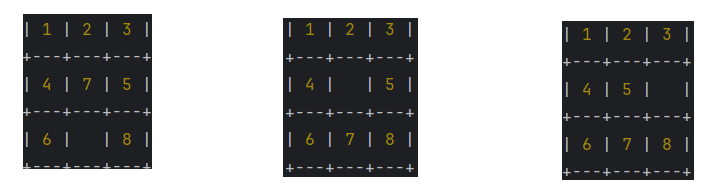


Figure 2: The 3 sequential images show transitions UP and Right

1. Path Cost:
   1. Uniform Cost Search: Uniform Cost Search (UCS) is a best-first graph-search algorithm that always expands the frontier node with the lowest cumulative path cost, guaranteeing that the first time a goal is dequeued the solution is optimal. It maintains a priority queue (min-heap) keyed by each path’s total cost and an explored set to avoid revisiting states.
   2. A\* with Misplaced Tiles Heuristic: Misplaced Tile Heuristic in A\* computes *h(n)* as the number of tiles out of place compared to the goal configuration (ignoring the blank), giving a simple admissible estimate of remaining moves.
   3. A\* with Manhattan Distance Heuristic: Manhattan Distance Heuristic in A\* estimates ℎ(𝑛) as the sum of the absolute row-and-column distances each tile is from its goal position, yielding a tighter admissible bound than just counting misplaced tiles. It is both admissible and consistent, ensuring A\*’s optimality and completeness with non-negative step costs. By reflecting actual minimum moves, it typically expands far fewer nodes than the simpler misplaced-tile heuristic.

*f(n) = g(n) + h(n)*

*g(n) = Cost incurred till reaching the current state*

1. Goal State: In this case the goal state is an sequential alignment of the 8 puzzle problem illustrated in figure3. In the implementation this state is represented using a list. Ex- [1,2,3,4,5,6,7,8,0]

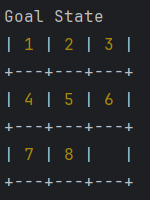


Figure 3: Goal State of 8-Puzzle problem

# Analysis of Algorithms

Parameters of Comparison [5]-

1. Completeness: Is the technique guaranteed to find an answer (if there is one).
2. Optimality: Is the technique guaranteed to find the best answer (if there is more than one)
3. Time: How long does it take to find a solution
4. Space: How much memory is required to find a solution

The below analysis was done considering the test initial cases provided in the github code. Building on this following analysis is reported-

1. Completeness and Optimality-

UCS, A\* with Misplaced-Tile, and A\* with Manhattan, all strategies solved every test puzzle, confirming they are complete and return the optimal solution length in every case.

1. Time-

Figure4 illustrates the plot of runtime against solution depth (on a log scale).

* UCS grows most steeply: from microseconds at depth 0 to over ten seconds by depth 24.
* A\* with Misplaced-Tile cuts the USC run time roughly in half at higher depths.
* A\* with Manhattan distance runs nearly ten times faster than UCS at depth 24 and stays under a tenth of a second even on the hardest puzzles.

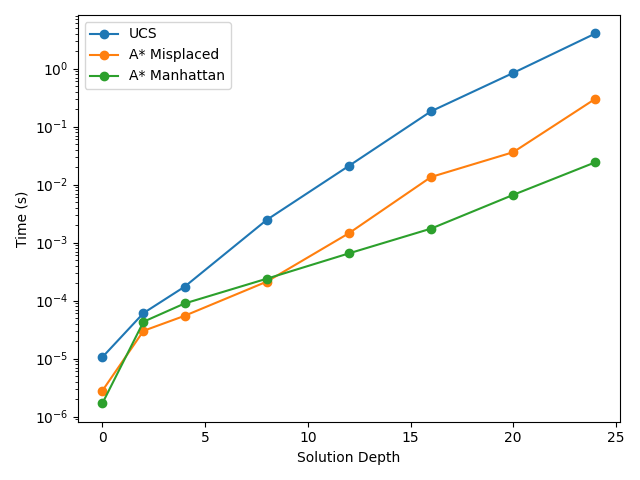


Figure 4: Relation between Solution Depth and Time

1. Space:

Figure 5 Illustrates the plot of Nodes with respect to the Depth.

* UCS expands from a single node at depth 0 to well over 10⁵ nodes at depth 24.
* The Misplaced-Tile heuristic reduces expansions by about an order of magnitude in the mid-range with respect to USC and A\* Manhattan
* Manhattan distance delivers another ten-fold reduction, expanding only on the order of 10³ nodes at the same depth.

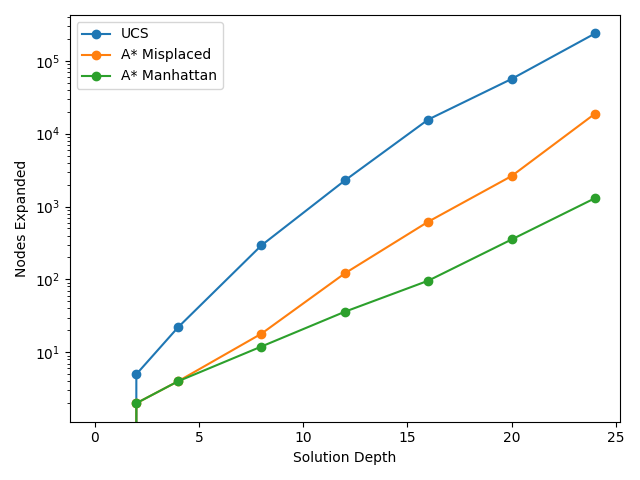


Figure 5: Nodes Expanded vs Depth of Solution

Figure 6 Illustrates maximum frontier size (which approximates space complexity). This relationship follows the same pattern as that of Nodes Expanded to Solution Depth.

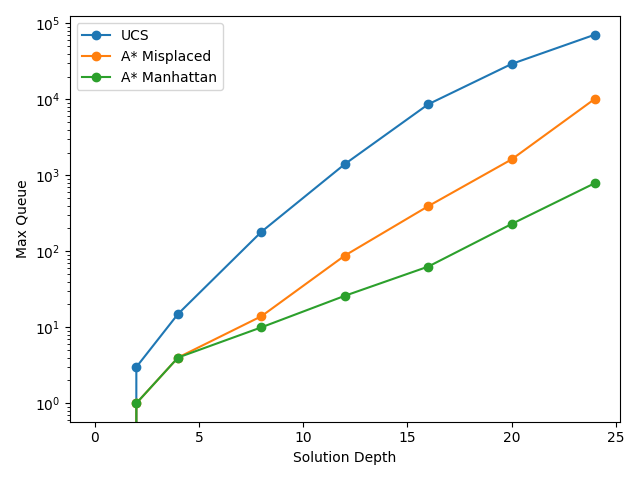


Figure 6: Relationship between Solution Depth and Max States in the Queue

##### Summary of Analysis-

In summary, while all methods guarantee completeness and optimality, A\* with Manhattan distance overwhelmingly outperforms the others in both time and space as puzzle difficulty grows, making it the strategy of choice for larger or more complex sliding-tile problems.

**References**

* + List any textbooks (e.g., Russell & Norvig), papers (Korf 1990), and online tutorials you consulted.