McGill University

COMP 424: Artificial Intelligence

Felix Simard (260865674)

February 8th, 2021

Assignment 1

**Question 1: Six-Puzzle**

1. Show the solution path (i.e., the sequence of puzzle states from the initial to the goal state) found by each of the following algorithms, assuming transitions have unit cost. You must ensure that puzzle states that have been explored are not added to the search queue. Given multiple states to explore that are otherwise equivalent in priority, the algorithm should prefer the state that involves moving a lower-numbered piece.

Breadth First Search (BFS)

A picture containing text

Description automatically generated

Uniform Cost Search (UCS)

Note: UCS with a cost of 1 ends up being the same as running BFS.

Text

Description automatically generated with medium confidence

Depth First Search (DFS)

Since this algorithm visits a lot more states the others, we will represent the board configurations as a list reading the board values from top-left to top-right, bottom-left to bottom-right.

For example: our initial state:

Text

Description automatically generated with medium confidence would transalte to the list: [1, 4, 2, 5, 3, 0]

[1, 4, 0, 5, 3, 2] -> [1, 0, 4, 5, 3, 2] -> [0, 1, 4, 5, 3, 2] -> [5, 1, 4, 0, 3, 2] -> [5, 1, 4, 3, 0, 2] -> [5, 0, 4, 3, 1, 2] -> [5, 4, 0, 3, 1, 2] -> [5, 4, 2, 3, 1, 0] -> [5, 4, 2, 3, 0, 1] -> [5, 4, 2, 0, 3, 1] -> [0, 4, 2, 5, 3, 1] -> [4, 0, 2, 5, 3, 1] -> [4, 2, 0, 5, 3, 1] -> [4, 2, 1, 5, 3, 0] -> [4, 2, 1, 5, 0, 3] -> [4, 0, 1, 5, 2, 3] -> [4, 1, 0, 5, 2, 3] -> [4, 1, 3, 5, 2, 0] -> [4, 1, 3, 5, 0, 2] -> [4, 0, 3, 5, 1, 2] -> [0, 4, 3, 5, 1, 2] -> [5, 4, 3, 0, 1, 2] -> [5, 4, 3, 1, 0, 2] -> [5, 4, 3, 1, 2, 0] -> [5, 4, 0, 1, 2, 3] -> [5, 0, 4, 1, 2, 3] -> [5, 2, 4, 1, 0, 3] -> [5, 2, 4, 0, 1, 3] -> [0, 2, 4, 5, 1, 3] -> [2, 0, 4, 5, 1, 3] -> [2, 1, 4, 5, 0, 3] -> [2, 1, 4, 5, 3, 0] -> [2, 1, 0, 5, 3, 4] -> [2, 0, 1, 5, 3, 4] -> [0, 2, 1, 5, 3, 4] -> [5, 2, 1, 0, 3, 4] -> [5, 2, 1, 3, 0, 4] -> [5, 0, 1, 3, 2, 4] -> [0, 5, 1, 3, 2, 4] -> [3, 5, 1, 0, 2, 4] -> [3, 5, 1, 2, 0, 4] -> [3, 5, 1, 2, 4, 0] -> [3, 5, 0, 2, 4, 1] -> [3, 0, 5, 2, 4, 1] -> [0, 3, 5, 2, 4, 1] -> [2, 3, 5, 0, 4, 1] -> [2, 3, 5, 4, 0, 1] -> [2, 3, 5, 4, 1, 0] -> [2, 3, 0, 4, 1, 5] -> [2, 0, 3, 4, 1, 5] -> [2, 1, 3, 4, 0, 5] -> [2, 1, 3, 0, 4, 5] -> [0, 1, 3, 2, 4, 5] -> [1, 0, 3, 2, 4, 5] -> [1, 3, 0, 2, 4, 5] -> [1, 3, 5, 2, 4, 0] -> [1, 3, 5, 2, 0, 4] -> [1, 3, 5, 0, 2, 4] -> [0, 3, 5, 1, 2, 4] -> [3, 0, 5, 1, 2, 4] -> [3, 2, 5, 1, 0, 4] -> [3, 2, 5, 0, 1, 4] -> [0, 2, 5, 3, 1, 4] -> [2, 0, 5, 3, 1, 4] -> [2, 1, 5, 3, 0, 4] -> [2, 1, 5, 3, 4, 0] -> [2, 1, 0, 3, 4, 5] -> [2, 0, 1, 3, 4, 5] -> [0, 2, 1, 3, 4, 5] -> [3, 2, 1, 0, 4, 5] -> [3, 2, 1, 4, 0, 5] -> [3, 0, 1, 4, 2, 5] -> [3, 1, 0, 4, 2, 5] -> [3, 1, 5, 4, 2, 0] -> [3, 1, 5, 4, 0, 2] -> [3, 0, 5, 4, 1, 2] -> [0, 3, 5, 4, 1, 2] -> [4, 3, 5, 0, 1, 2] -> [4, 3, 5, 1, 0, 2] -> [4, 3, 5, 1, 2, 0] -> [4, 3, 0, 1, 2, 5] -> [4, 0, 3, 1, 2, 5] -> [4, 2, 3, 1, 0, 5] -> [4, 2, 3, 1, 5, 0] -> [4, 2, 0, 1, 5, 3] -> [4, 0, 2, 1, 5, 3] -> [4, 5, 2, 1, 0, 3] -> [4, 5, 2, 0, 1, 3] -> [0, 5, 2, 4, 1, 3] -> [5, 0, 2, 4, 1, 3] -> [5, 1, 2, 4, 0, 3] -> [5, 1, 2, 4, 3, 0] -> [5, 1, 0, 4, 3, 2] -> [5, 0, 1, 4, 3, 2] -> [5, 3, 1, 4, 0, 2] -> [5, 3, 1, 4, 2, 0] -> [5, 3, 0, 4, 2, 1] -> [5, 0, 3, 4, 2, 1] -> [5, 2, 3, 4, 0, 1] -> [5, 2, 3, 0, 4, 1] -> [0, 2, 3, 5, 4, 1] -> [2, 0, 3, 5, 4, 1] -> [2, 4, 3, 5, 0, 1] -> [2, 4, 3, 0, 5, 1] -> [0, 4, 3, 2, 5, 1] -> [4, 0, 3, 2, 5, 1] -> [4, 3, 0, 2, 5, 1] -> [4, 3, 1, 2, 5, 0] -> [4, 3, 1, 2, 0, 5] -> [4, 0, 1, 2, 3, 5] -> [4, 1, 0, 2, 3, 5] -> [4, 1, 5, 2, 3, 0] -> [4, 1, 5, 2, 0, 3] -> [4, 0, 5, 2, 1, 3] -> [0, 4, 5, 2, 1, 3] -> [2, 4, 5, 0, 1, 3] -> [2, 4, 5, 1, 0, 3] -> [2, 4, 5, 1, 3, 0] -> [2, 4, 0, 1, 3, 5] -> [2, 0, 4, 1, 3, 5] -> [0, 2, 4, 1, 3, 5] -> [1, 2, 4, 0, 3, 5] -> [1, 2, 4, 3, 0, 5] -> [1, 0, 4, 3, 2, 5] -> [1, 4, 0, 3, 2, 5] -> [1, 4, 5, 3, 2, 0] -> [1, 4, 5, 3, 0, 2] -> [1, 0, 5, 3, 4, 2] -> [1, 5, 0, 3, 4, 2] -> [1, 5, 2, 3, 4, 0] -> [1, 5, 2, 3, 0, 4] -> [1, 5, 2, 0, 3, 4] -> [0, 5, 2, 1, 3, 4] -> [5, 0, 2, 1, 3, 4] -> [5, 3, 2, 1, 0, 4] -> [5, 3, 2, 0, 1, 4] -> [0, 3, 2, 5, 1, 4] -> [3, 0, 2, 5, 1, 4] -> [3, 1, 2, 5, 0, 4] -> [3, 1, 2, 5, 4, 0] -> [3, 1, 0, 5, 4, 2] -> [3, 0, 1, 5, 4, 2] -> [3, 4, 1, 5, 0, 2] -> [3, 4, 1, 5, 2, 0] -> [3, 4, 0, 5, 2, 1] -> [3, 0, 4, 5, 2, 1] -> [3, 2, 4, 5, 0, 1] -> [3, 2, 4, 0, 5, 1] -> [0, 2, 4, 3, 5, 1] -> [2, 0, 4, 3, 5, 1] -> [2, 5, 4, 3, 0, 1] -> [2, 5, 4, 0, 3, 1] -> [0, 5, 4, 2, 3, 1] -> [5, 0, 4, 2, 3, 1] -> [5, 3, 4, 2, 0, 1] -> [5, 3, 4, 2, 1, 0] -> [5, 3, 0, 2, 1, 4] -> [5, 0, 3, 2, 1, 4] -> [5, 1, 3, 2, 0, 4] -> [5, 1, 3, 0, 2, 4] -> [0, 1, 3, 5, 2, 4] -> [1, 0, 3, 5, 2, 4] -> [1, 2, 3, 5, 0, 4] -> [1, 2, 3, 5, 4, 0] -> [1, 2, 0, 5, 4, 3] -> [1, 0, 2, 5, 4, 3] -> [0, 1, 2, 5, 4, 3] -> Goal state reached in 167 moves

Iterative Deepening Search

The solution path is also basically identical to BFS since we increment by a depth of 1 at each iteration of depth-limited search.

Text

Description automatically generated

1. Suppose now that transitions have differing costs. In particular, the cost of a transition is equal to the number of the piece that is moved (e.g., moving the “4” costs 4). If we employ the Manhattan distance heuristic for the original unit cost version of the eight-puzzle presented in class (Lecture 4, slide 11, h2), would this heuristic still be an admissible heuristic for A\* search in the new variant? Justify your answer.

Let the cost of a transition be equal to the number of the piece that is moved.

We defined an “admissible heuristic” for A\* search as being an optimistic heuristic which always gives an underestimate of the true cost for all paths from n to a goal state.

As seen in the lecture slides, the Manhattan distance heuristic estimates the true cost of getting to the goal state by summing the distances of each tile from the current position to where they should end up being in the goal state.

Hence, by using a unit cost version, a tile that is “far” from its desired end position, should proportionally contribute to a larger heuristic value.

Now, with our cost variant approach, we could have the following scenario:

* Higher-numbered tiles could be very close to their desired goal position, but instead of proportionally reflecting it in the heuristic cost calculation (ie: little added cost since close to desired location), they would make the heuristic value larger. Even so, in the worst case, high-numbered tiles far away will incur a way larger heuristic value.

As a result, our condition for having an “admissible heuristic” could fail in some board configurations, meaning where our heuristic is not optimistic and overshoots the true cost. In that case, this heuristic would not be an admissible heuristic for A\* search in the new variant.

1. Design an admissible heuristic that dominates the heuristic from part b, under the same cost scheme as part b.

An admissible heuristic which would dominate the heuristic from part b:

Heuristic: Number of tiles out of row + Number of tiles out of column

Why? Since this heuristic does not take into account the transition cost to develop its heuristic value. And, as mentioned in the reference below, it is admissible “since every tile that is out of column or out of row must be moved at least once and every tile that is both out of column and out of row must be moved at least twice.” resulting in a guaranteed optimistic heuristic value.

Reference: <https://cse.iitk.ac.in/users/cs365/2009/ppt/13jan_Aman.pdf>

**Question 2: Search Algorithms**