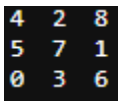
Giuliano Cancilla

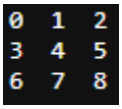
Professor Phillips

CSCI 4350 Intro to AI

10/3/2023



(Figure 1: Scrambled State)



(Figure 2: Goal Configuration)

The 8 Puzzle Problem

The 8-puzzle problem presents a game grid, which can vary in size, although we use a 3x3 grid in this experiment (see Figure 1). Visualize each number as a tile snugly fitted so that all sides of the tiles touch one another. In our case, the 0 represents an empty tile that adjacent tiles can move to, exchanging places with the empty tile (0). You cannot pick up and move other pieces; you can only move tiles adjacent to the empty one. Figure 1 represents a scrambled board where a series of shifts may eventually lead it to what is known as a Goal Configuration (Figure 2). The goal is to correctly slide the tiles to reach the final state shown in Figure 2. There are specific rules for tile movement: wrapping is not allowed, meaning a tile on the leftmost column cannot wrap around to the rightmost column and vice versa; similarly, moving down on the bottom row or up on the top row is not valid.

My Code, And How it Solves The 8-Puzzle Problem

The method that I implement in solving the 8-Puzzle Problem is called A* search. A* search is known as an informed search algorithm. This search algorithm explores different states to find the best outcome. Understanding the distinction between uninformed and informed searches is crucial for assessing time complexity, memory usage, and solution optimality. An uninformed search can be thought of as a brute force technique that doesn't use any inferences

on which state to choose next. Uninformed search merely goes through all possible states till the Goal Configuration is found. On the other hand, Informed search allows you compare a value called a heuristic, which gives an indication of which state to choose next. For the 8-puzzle problem, an example heuristic involves counting the number of tiles in the current state that do not match the goal state. Figure 1 contains the maximum number of displaced tiles, because no number is in the correct slot for which the Goal Configuration (Figure 2) calls for. If we also consider the depth of the current node, we can estimate how many moves are needed to reach the Goal Configuration. When comparing two different moves, a state with a lower displaced tile heuristic is favored because it represents fewer tiles that need to be moved to reach the Goal Configuration.

My code initializes a random board and attempts to find the most efficient sequence of valid moves to reach the Goal Configuration. Each node in the algorithm contains a heuristic value, a path to the previous node, the current 8-puzzle configuration, the node's depth, and a unique ID. First, we get our random board and our goal state. Then, we store the initial Node into a list called the Frontier and check to see if that Node is the goal state. If it is not the goal state, we generate all the possible valid states made by performing valid moves on the current board. We then put those into the frontier in the form of Nodes which hold various properties defined above. Next, we remove the checked Node from the Frontier and add it to the Closed List. The Frontier is then sorted by lowest heuristic value and if there exists a tie we use the newest node that was generated (the one that has a lower unique ID). The lowest heuristic value will be chosen as the next Node to evaluate, and the process repeats itself until a goal state is achieved. When, the goal state is achieved the number of nodes expanded, branching factor (how many children on average does each node have), the depth of the solution (how many nodes down did

we have to go to get the solution), the number of nodes that were held in memory, and the steps the algorithm took to get to the goal state are printed.

The process of sorting the frontier by using different heuristic values is the most important thing about A* search. The accuracy of your heuristic determines a more optimal path to reach the goal. My program allows you to try different heuristics. Option 0 represents no heuristics, effectively making A* search an uninformed search. Option 1 is the displaced tiles heuristic which tells us how many tiles are out of order in the current state. Option 2 uses a heuristic called the Manhattan Block Distance. The Manhattan Block Distance is the distance between two points on a grid through making right angles, like walking from block to block. Option 3 is a heuristic that I designed myself, which is the average of the displaced tiles and the Manhattan distance. I chose to do the average of the Manhattan distance and displaced tiles because the Manhattan distance is a good mapping for how to get from point A to point B, but the displaced tiles heuristic tells me how far off I am from my destination.

The experiment I ran was on 100 instances of random and different, solvable 8-puzzle configurations for each different heuristic. None represents when we set the heuristic to 0. The only thing considered is the depth of the nodes in this case. This is the same as an uninformed search, namely a Breadth First Search (BFS). This explores nodes level by level. Then, we have the Displaced Tiles heuristic, the Manhattan Block Distance, and my personal heuristic the average of Displaced and Manhattan Block Distance. I have coined the term “Manhattanplaced” heuristic in the scope of this experiment, but I am sure I am not the only one to have thought of that. The meaning behind testing different heuristics is to see how the different estimations of what path should be taken effects how quickly and optimally the search is performed.

Results of Different Heuristics

Key
V= Total Nodes Visited/Expanded
N=Max Number of Nodes Stored in Memory
d= Depth
b= Branching Factor

Abbreviations
None = No Heuristic
Displaced = Displaced Tiles Heuristic
MBD = Manhattan Block Distance
Manhattanplaced = Average of Manhattan Block Distance + Displaced Tiles

Heuristic	Minimum	Maximum	Std Dev	Mean	Median
None (V)	16	176624	54896.40593	53355.06061	38575
None (N)	32	353248	109792.8119	106710.1212	77150
None (d)	4	27	6.456732451	17.75757576	20
None (B)	1.33431	1.5	0.032246879	1.363437273	1.35493
Displaced (V)	3	59469	13220.80046	7692.94	1603
Displaced (N)	6	118938	26441.60091	15385.88	3206
Displaced (d)	3	27	6.047488835	17.44	18
Displaced (B)	1.31818	1.66667	0.051005525	1.3595306	1.342795
MBD (V)	3	7771	1389.777778	826.3	329.5
MBD (N)	6	15542	2779.555556	1652.6	659
MBD (d)	3	27	6.047488835	17.44	18
MBD (B)	1.31716	1.66667	0.056920796	1.3645876	1.341735
Manhattanplaced (V)	3	17546	3408.707403	1917.68	561
Manhattanplaced (N)	6	35092	6817.414806	3835.36	1122
Manhattanplaced (d)	3	27	6.047488835	17.44	18

Manhattanplaced (B)	1.31833	1.66667	0.058330665	1.3636103	1.34025

Results Analysis

One thing I need to note about the results is that the None heuristic took so long to run that I didn't even make it to 100 puzzles solved. It ran 32 puzzles within about a 6-hour span of time. My implementation didn't do much as far as specialized conditions specific to the 8-Puzzle Problem. The implementation that I went with is a more general implementation that can be used at a basic level to work with a different variety of problems. One thing I could have done differently was create a check that checked if a current puzzle was solvable in the current configuration. If the puzzle wasn't solvable we could knock off a lot of time by not exploring any further down that subsection of the tree.

n terms of heuristic performance for the 8-Puzzle problem, the Manhattan Block Distance (MBD) heuristic stands out as the top choice. If we examine the metrics of Total Nodes Visited/Expanded (V) and Max Number of Nodes in Memory (N), we observe that the mean values for both V and N with MBD are significantly lower than those for the Manhattanplaced and Displaced heuristics. On average, MBD explored 826.3 nodes, while Manhattanplaced explored 1917.68, Displaced explored 7692.94, and None explored 53355.06. This efficiency speaks volumes about MBD's effectiveness, being over twice as efficient as my implementation and significantly more efficient than the other two. The less nodes you visit, the less time you take to find the solution and the less memory used.

MBD is also consistent, if you look at the standard deviation for V and N it is less than 3000 for both, whereas the Displaced heuristic's standard deviation was stdV=13220.8 and stdN = 26441.6, showing that it was led astray from the correct path much more often than that of MBD.

The heuristic I selected performed well, ranking second in terms of consistency. It maintained a lower standard deviation for nodes explored and memory used than Displaced and None, staying under 7000, while the other two exceeded 10,000 for average V and N by far which demonstrates that they have more inconsistencies than my heuristic.

Overall, the best heuristic to use in the case of the 8-Puzzle problem is the Manhattan Block Distance heuristic. It outdoes every other heuristic in this chart in every meaningful aspect. The depth of the solution and the branching factor don't show too much as far as efficiency sake because they should be the same for every heuristic. However, in terms of V and N Manhattan Block Distance outperforms all other heuristics.

Work Cited

https://www.geeksforgeeks.org/check-instance-8-puzzle-solvable/#

This source shows you how to tell if an 8-puzzle is unsolvable. This is beneficial because it allows us to pass our algorithm valid puzzles to solve, so it doesn't run forever.

https://docs.python.org/3/tutorial/classes.html

Python documentation. This helped me specifically to refresh my memory on certain syntax and ways to declare things. Especially helpful while making my Node class. Also helped with sorting.

https://www.youtube.com/watch?v=vP5TkF0xJgI

GeeksforGeeks A* youtube video to help study A* algorithm.

https://www.youtube.com/watch?v=Mb1srg1ON60

Simplilearn A* algorithm video

https://www.programiz.com/python-programming/set#:~:text=In%20Python%2C%20we%20create%20sets,or%20dictionaries%20as%20its%20elements.

Programiz python sets tutorial

chat.openai.com

ChatGPT to help me with some of the bash scripting to pull data from files. Also helped me format some of the functions in Excel to properly display the results. Also used it to verify the functions used in excel gave me the correct numbers.