CS 4346 - Project 2 Analysis Report

By

Zohair A Khan (zak21@txstate.edu)

- ## The Problem Description:

So the primarily objective of this project is to use the A* Search algorithm in an 8-puzzle game to find the solution for the game. An 8-puzzle game has 9 tiles that are capable of sliding. The board is a 3x3 grid that has a 0 or a blank space, the user is permitted to shift a tile or number to the blank space. The goal of the game is to get to the end goal in a specific order of tiles in the grid. The shifting tile have rules set to make sure the validity of the game is in check, said rules are such as that tile can ONLY swap with the blank space and that the corresponding tile can only move in the cardinal direction of the empty space (North, South, West or East) and that they cannot move diagonally. The primary goal of the problem is to use different heuristic functions to check the most optimal method in winning the game i.e reaching the goal node. Different heuristic functions work in different ways and some take longer steps to finish the game. The task of the project was to create three different heuristic functions that would work in different ways in the A* Search algorithm. One of the three heuristic functions that we need to use are one of the ones discussed in the previous lectures; misplaced tiles, Euclidean distance and Manhattan distance. We were also assigned to create our own custom made heuristic function. In addition to this, we were tasked to find the execution time, the number of nodes generated, the number of nodes expanded, depth of the tree, effective branching factor b* and the total path.

● The Domain:

The domain of the project/assignment are AI models, A* Search algorithms and the heuristic functions.

● Methodologies:

The methodologies for the project/assignment are the A* Search algorithms, execution time, the number of nodes generated, the number of nodes expanded, depth of the tree, effective branching factor b* and the total path.

● Source Code Implementation:

Starting from the beginning in the cpp file, the source code entails an x and y validity check followed by the Manhattan distance heuristic function that we were tasked to implement for one of the three functions we learned in class. This is simply using the x and y coordinates and calculating the distance. Brad Hughes heuristic function is the next one, a custom heuristic function that calculates distances by adding all the rows and then adding all the columns. I am unfamiliar with the intricacies for his heuristic so I will not go over it too much but all I know is that it took him SEVERAL hours of calculating by hand in pen and paper. Following Brad's heuristic function is Zohair Khan heuristic function which is based on The Permutation Inversion Count heuristic. This distance heuristic function uses the number of moves required to estimate and solve an 8-puzzle problem. It works by counting the number of inversions in the state of the puzzle. First we need to talk about what an inversion is, an inversion takes place when a tile with a higher number appears before a tile with a lower number. So to calculate the number of inversions that are taking place, we must first convert the board in a 1D array where it is simply 0, 1, 2, 3, 4, 5, 6, 7, 8 (0 is blank). We then have to simply iterate over all the possible pairs of tiles that are in the array and then count for the number of inversions. If a pair of tiles (let's say a and b) is an inversion and that a appears BEFORE b in the array, then a is greater than b. The reasoning for this heuristic is that the inversion numbers in the present state provides a lower bound on the number of moves that are required to solve the puzzle i.e if we know that the present state has x number of inversion, we will that it will take at least x number of moves to solve the puzzle. After this are the vectors that generate the successors. Since the tiles only move in the cardinal directions, they are generating successors of only North, South, East and West. There is an F sort for the node as well after this and a function call that checks the successor and updates the heuristic. Now comes the main, which starts with the initial state of 1 and 2 with an empty board. The main is the A* Algorithm and it follows the instructions of the pseudo code. It starts with an open node that only contains the initial node. There is a while loop statement that states that if Open is empty then the program shuts down, but if it is not then it goes through a loop to check and find the lowest f ` value. Once the lowest one is found that node is called Best node and taken out of Open and placed in Closed which is simply the CLOSED.push_back(BESTNODE);. After this is accomplished the program will close but we have to generate its successors as well (of the best node). This starts on line 342 where the best node is pointing to the successors, and computes the algorithm of g(successor) = g(bestnode) + cost (distance) from best node to successor) and f ` (successor) = g(successor) + h(successor). All of this was beautifully implemented by Brad Hughes in the header file as a void function of setHeur. Once this was done, we created OLD which is simply just the successor in the same node as any node on open. We quickly realized that we do not have to throw the successor away and that we could simply just overwrite the old successor with old thus removing/throwing it away. The final part of the cpp is simply the cout statements that give detail on the nodes generated, tree depth, nodes expanded and the branching factor. The header file was handled primarily by Brad Hughes so my expertise on the matter is a bit weaker but I will try my best to explain it. So he created a class called PuzzleNode which handles EVERYTHING in the node, from best node to the parents to the successor etc followed by various void functions corresponding to them.

• Source Code

The Source code is attached to the file. There are 2 cpp files, one is called Project2_Manhattan.cpp and the other is Project2_Zohair.cpp. The Project2_Manhattan.cpp runs the program with the Manhattan Distance heuristic implemented, while the Project2_Zohair.cpp runs with The Permutation Inversion Count heuristic function.

• A Copy of the program Run

In the folder are screenshots of the successful runs of the project. The Manhattan Distance is using the Manhattan Distance heuristic function and the following results. The Permutation Inversion Count is using the Permutation Inversion Count heuristic function and its results.

• Analysis of the program

I personally added the Permutation Inversion Count, which is a modified version of a heuristic function that I wrote the c++ code for from several researches and other algorithms. The main in the cpp file was also something I handled primarily which contains the A* Search algorithm.

- ## Tabulation of results

Heuristic	ET	NG	NE	D	b*	TP
Manhattan	0.685283 ms	17	6	6	2.83333	

Heuristic	ET	NG	NE	D	b*	TP
Permutate Inversion	0.331296	17	6	6	2.83333	

- ## Analysis of the results

Since the TP was not generated, I will not include that in the result. The ET shows that the permutation inversion ran shorter and provided the same results as the manhattan distance which means that, over time, it is actually exceeding the manhattan distance by two times. This means that the Permutation Inversion method is TWICE as fast as the Manhattan distance thus resulting in an overall faster and efficient heuristic function.

- ## Conclusion

I learned a great deal about optimization in this project. Taking a general algorithm (Manhattan), then comparing and experimenting with the one that we had to create ourselves was definitely a fun challenge. I personally find a great feeling of joy in knowing that I wrote something that is faster and more efficient than the general universal heuristic function. I also learned a lot about class and vectors from my partner Brad Hughes and it was an absolute pleasure working on this project with him.

- ## Team Members Contributions

Brad Hughes developed a pseudocode as groundworks for the project since Zohair Khan is weaker on the instruction side. Once he helped Zohair understand the goal, he worked on the header file and created the class function for the node. Brad also worked on the Manhattan distance heuristic function and his own heuristic function as well. Zohair Khan wrote his own heuristic function and the vector implementation of the generation of the successors in accordance with their cardinal directions. Zohair Khan also worked heavily on the main loop and the overall A* Search algorithm implementation in the main loop. He did ask Brad Hughes for some guidance and advice but overall did the majority of the main loop and the key features of the algorithm.

● References

I didn’t use too many outside references except the following ones provided. Majority of the instructions were clear and the only major documentation/reference I used were the slides from SCP#5.

Sonawane, Ajinkya. “Solving 8-Puzzle Using a* Algorithm.” *Medium*, Good Audience, 24 June 2020, https://blog.goodaudience.com/solving-8-puzzle-using-a-algorithm-7b509c331288

This source was used primarily to learn about the algorithm more and that’s it. It was used for general understanding and rule set of how the A* Search algorithm works with the 8-puzzle game.

YuChen, Ding. “Looking into K-Puzzle Heuristics.” *Medium*, The Startup, 2 July 2020, https://medium.com/swlh/looking-into-k-puzzle-heuristics-6189318eaca2.

There was not anything used from this source but it was a good inspiration because the blog contains different approaches and styles of heuristics, so it was definitely interesting to see how people came up with different heuristics. Again, nothing from this website was used for our code or this paper.