Alan Huang
Dr. Daisy Tang
CS 4200

CS 4200 Project 1 Report

Approach:

The 8 puzzler A* search algorithm is very easy to calculate as an in-class exercise but to implement it into a program that is a different story. Luckily this project is a very popular task for many schools, so there are many YouTube tutorial videos, and websites teach people how to implement this 8 puzzler A* search algorithm to program. For the puzzlerSolver class, it was pretty similar to my CS3310 best-first search algorithm project. I just use the priority queue to holds the nodes and remove the lowest cost node each cycle. The lowest cost node will then be checked to see if the puzzler is already solved, if not then the algorithm will start to create the next possible move (nodes).

The hardest parts to design the node class was to figure out algorithms to count misplaced tiles, manhattan distance, and if the puzzle is solvable. After watching lots of videos and checking from websites I was able to find a few useful algorithms to implement all the functions I need inside the node. After some testing, I was able to solve the 8 puzzler problems by my program. Just when I thought the program is finished, I noticed that my program is extremely slow to solve problems that are more than 10 steps. I started to check on google to find a possible solution to fix my program's efficiency. I realized that I need to find a way to keep track of all of the visited nodes, so I don’t create duplicated nodes to wasted time.

The best solution I found was to use HashSet to track all of my visited nodes and use hashMap to remove duplicated nodes since HashMap doesn’t allow duplicate keys. To use HashSet and HashMap the node class will need a custom hashCode() and equals() methods to compare each node’s key. I was surprised to found out that most of the java ide like eclipse can automatically generate hashCode() and equals() methods for me, so I don’t need to figure out if the code was correct or not. After use HashSet, and HashMap my program was 10 times faster than before. I am glad that this program allows me to learned how to use HashSet, and HashMap because

the efficiency gain in my program was amazing. This project really helped me improve my programming knowledge a lot.

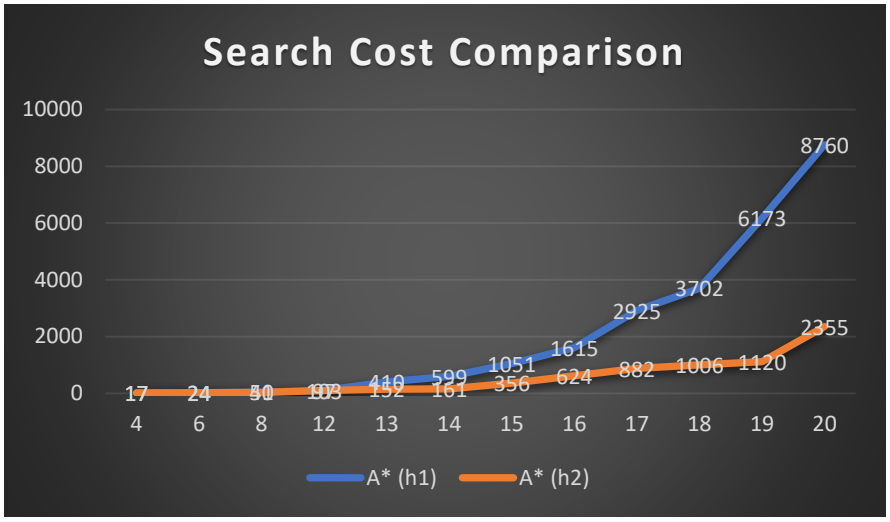
Comparison of two heuristics:

After I made the compare table, we can easily see that the Manhattan distance algorithm used fewer steps to reach goal states. I think the reason why Manhattan distance performance better was because hamming distance doesn't update the total misplaced tile numbers every movement The hamming distance couldn't detective if two misplaced tiles switch position, so there is less information provided to the A* search to approach to goal states. On the other hand, Manhattan distance updated the total steps to reach the goal state every time a tile moved, so there is more detailed information given to the A* search. The more variable information that we can provident to program, the accurate the program can predictive the solution. The Manhattan distance is probably the most efficient heuristics to solve the 8 puzzle problem.

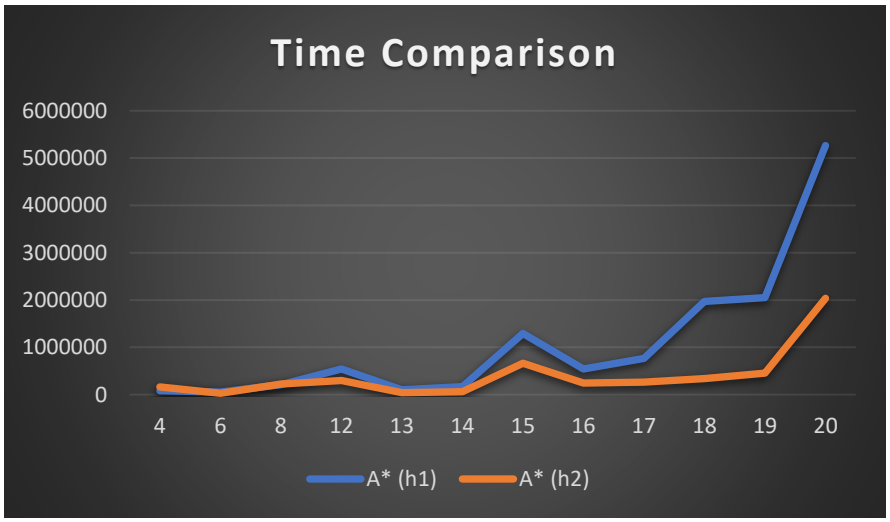
Table: Hamming distance compare to Manhattan distance

	Search Cost				
d	Count	A*(h1)	A*(h2)	A*(h1) Time (ns)	A*(h2) Time (ns)
4	11	17	15	84100	164890
6	1	24	24	53900	28200
8	9	51	40	209911	225778
12	10	103	97	543340	297820
13	1	410	152	101300	41800
14	5	599	161	172440	64100
15	4	1051	356	1293775	663475
16	12	1615	624	542166	240767
17	2	2925	882	763050	262400
18	15	3702	1006	1971209	336390
19	8	6173	1120	2047860	453380
20	22	8760	2355	5263820	2035460

Search Cost Comparison



Time Comparison



Finding:

I use the information I collected from the table to generate the time and depths comparison graph. From the graph, we can see that for 20 steps problem Manhattan distance created 2355 nodes to get to the goal state, and the hamming distance created 8760 nodes to get to the goal state. The hamming distance created almost 4 times of nodes to solve the same steps problem. From the time comparison graph, we can see hamming distance used twice the amount of time to solve the 20 steps 8 puzzler problem. From both of the comparison graphs, we can conclude that Manhattan distance is the ultimate solution algorithm to solve 8 puzzler problems.