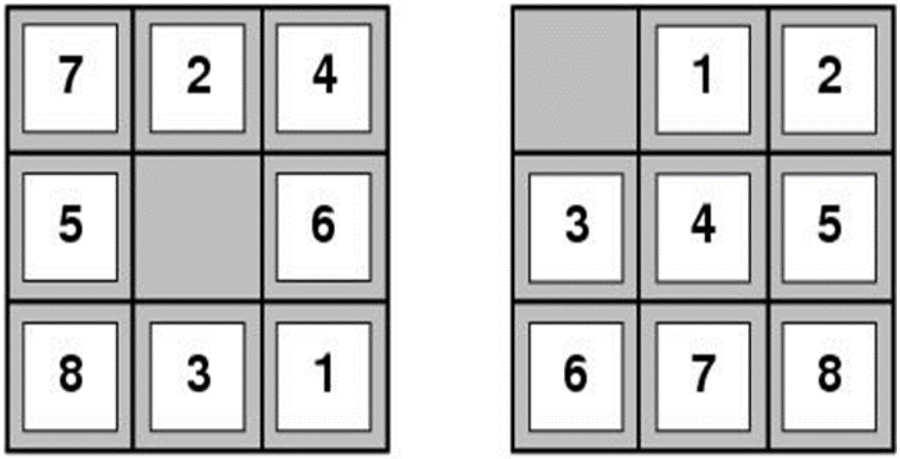
Brandon Goode



The 8 puzzle problem is a problem that involves taking a board filled with 8 tiles(each tile being a number from 1 through 8) and a blank space with each tile starting in some random position and moving them into such an order that the blank space is in the top left position of the board, and the board is then filled in sequentially from left to right. This process is displayed in the figure above. This is a difficult problem to solve due to the large number of possible configurations that the board has; specifically, the board can be configured into 181,440 unique states.

The way we can solve this problem efficiently is by implementing a Shortest Path algorithm known as the A* algorithm. The A* algorithm works very similarly to Dijkstra's Algorithm with one exception, instead of using just the cumulative path cost, G(n), to formulate which path is the shortest, the A* algorithm also utilizes a concept called a heuristic as well. A heuristic is essentially an estimate that tells you how close you are, in any given state, to the goal state. There are many ways to calculate heuristics, and they depend largely on the specific problem you are trying to solve, but the goal of the heuristic is to get as close to the true solution cost as possible without overestimating the solution cost. If the heuristic used is set to 0, the algorithm performs the same as Dijkstra's Algorithm. By using a heuristic that is lower than the true solution cost, the heuristic is classified as admissible. Admissible heuristics have the benefit of getting the optimal path every time, whereas inadmissible heuristics(caused by overestimating the cost of the true solution cost) will not always give you the shortest possible path. Setting the heuristic higher than the true solution cost causes the algorithm to perform more like a greedy search algorithm.

The A* algorithm consists of two structures to hold nodes: the closed list and the open list. The open list is an area for nodes that have been newly generated and have not been expanded yet, and the closed list is an area for the nodes that have already been expanded from the open list to be placed to ensure they do not get expanded more than once. The algorithm starts by taking the starting node and calculating its F(n). The F(n) is the cumulative path cost, G(n), plus the heuristic value, H(n), to become the new path cost. The F(n) value is the value that determines which node has a lower cost. The F(n) is then set as the value for that node and the

node is placed into the open list. The open list is a sorted structure and therefore causes the node with the smallest F(n) to be at the front of the list. The first node of the open list is then checked to see if it exists in the closed list already, if it does, nothing happens; however, if it does not, the node then uses a successor function to generate all the possible states that could succeed the current state. After the node's children have been generated, they are all placed into the open list, and the current node is then placed into the closed list. The algorithm continues by expanding the first node in the open list every time until the node contains the goal state.

My A* algorithm implementation for the 8 puzzle problems starts with taking the root node(the starting state the board is in) and placing it into a priority queue data structure(the open list). After the open list has its first value, it gets popped off and there is a check performed to see if that node is already in the open list, if it doesn't then it continues, if it does then it pops the next element off of the open list to check. The node that gets popped off from the open list is then expanded into up to four separate states: a state with the blank moved up, down, left, and right from the position it is currently at. Every node that gets expanded causes the value of V(total number of nodes expanded) to increase by one. The nodes get expanded in a loop that terminates once the goal state is found. After the goal state is found, the N, b, and d values are calculated. N is the number of nodes stored in memory(the size of the open list plus the size of the closed list), d is the depth of the solution found, and b is the branching factor calculated by N = b^d.

I used four different heuristics when running my program. The first heuristic I used was a heuristic of 0. This provided no additional functionality to the A* algorithm and effectively turned it into Dijkstra's algorithm. The second heuristic I used was the Tile Displacement heuristic. The Tile Displacement heuristic is calculated by finding the number of tiles that are not in the correct position in any given state and adding them together. The sum becomes the H(n) for that node. The third heuristic I used is the Manhattan Distance heuristic. This heuristic is calculated by looking at every tile and calculating how many tiles it would take to move that tile into its correct position. This is done for every tile and the sum of all the distances becomes H(n) for that node. The final heuristic I used was a combination of the Tile Displacement heuristic and the Manhattan distance heuristic. I calculated both heuristics for every node and then added them together to try and get as close as possible to the true solution cost.

I ran my program 100 times with 100 unique starting states for each heuristic to analyze which heuristics had the best and worst performances. I analyzed the V(total nodes expanded), N(number of nodes in memory), d(depth of solution), and b(branching factor) by taking all the data acquired from running the program 100 times and calculating the minimum, median, mean, maximum, and standard deviation of each factor for each heuristic.

Brandon Goode

The V value changed dramatically once adding a heuristic. Every statistic improved once adding any type of heuristic to the algorithm. No heuristic ended up having the worst results with a minimum of 18 nodes expanded and a maximum of 165,827 nodes expanded, whereas the Novel heuristic(adding Tile Displacement and Manhattan Distance together) generated only 5 nodes in the best case and 7729 in the worst case.

V				
Stats	None	Tile Displacement	Manhattan Distance	Novel
Minimum	18	5	5	5
Median	7326	319	241	137
Mean	22982	1690	995	560
Maximum	165827	30464	10404	7729
Standard Deviation	33388	3878	1925	1096

The number of nodes stored in memory behaved in a much similar fashion as the number of nodes expanded. No heuristic had significantly higher statistics than any of the other heuristics with each heuristic providing gradually better and better performance. The no heuristic approach caused the minimum number of nodes stored to triple from the three approaches using an actual heuristic. Even with the Tile Displacement heuristic, the mean number of nodes stored was more than ten times less than no heuristic.

N				
Stats	None	Tile Displacement	Manhattan Distance	Novel
Minimum	40	13	13	13
Median	14564	674	473	279
Mean	39670	3425	1958	1083
Maximum	211078	59171	21204	15376
Standard Deviation	52461	7662	3745	2123

Brandon Goode

The d value's statistics show some different results than the results for the V and N values. The no heuristic, Tile Displacement, and Manhattan Distance all have the exact same statistics, whereas the Novel heuristic has slightly different values. The reason for this is because the first 3 heuristics are admissible and therefore will always give the optimal path. Using these will always cause the algorithm to dive to the exact depth of the optimal solution and then stop. The Novel heuristic is inadmissible because when adding the Tile Displacement and Manhattan Distance heuristics together it sometimes becomes slightly higher than the true solution cost and this is demonstrated in these statistics. The novel heuristics does not perform significantly worse than the other three, but it does perform slightly worse by sometimes diving deeper than it needs to in order to find the solution.

d				
Stats	None	Tile Displacement	Manhattan Distance	Novel
Minimum	3	3	3	3
Median	15	15	15	15
Mean	14.66	14.66	14.66	14.94
Maximum	26	26	26	30
Standard Deviation	5.38839	5.3883901	5.388390063	5.63686

The statistics for the branching factor show similar results to the V and N values. The lack of a heuristic hurt the performance pretty significantly with a branching factor of 3.42 while the use of a heuristic had a worst-case branching factor of 2.35. The statistics for the data when using a heuristic was pretty similar with the maximum values being the same and the minimum values only differing by at most .15.

b				
Stats	None	Tile Displacement	Manhattan Distance	Novel
Minimum	1.6	1.504	1.4060	1.3503
Median	1.88	1.5409	1.5093	1.4654
Mean	1.95	1.57	1.5483	1.5047
Maximum	3.42	2.3513	2.3513	2.3513
Standard Deviation	0.27	0.1113	0.1384	0.1439