CMSC170$_E xer3$

John Byron Garin

September 2023

1 2 3 4 8 0 7 6 5			
	DFS	**BFS**	**A***
Path cost:	62331	5	5
Explored states:	78213	45	6
3 0 2 6 5 1 4 7 8			
	DFS	**BFS**	**A***
Path cost:		21	21
Explored states:		65196	955
8 7 6 5 4 3 2 1 0			
	DFS	**BFS**	**A***
Path cost:			30
Explored states:			18048
Additional test case for DFS (no video)			
2 3 0 1 5 6 4 7 8			
	DFS	**BFS**	**A***
Path cost:	6	6	6
Explored states:	7	90	7

Figure 1: Exercise 3 Test Cases

1 BFS, DFS, and A* Algorithm

Breadth Search First (BFS) is renowned for the optimality of its solutions, which is achieved by fully investigating all nodes at each depth level before moving on. However, due to the memory-intensive nature of keeping all explored nodes, its complexity might rise in complicated puzzle setups. It is clear from the test cases above that when problem difficulty increases, BFS's path cost and explored states likewise increase.

Depth Search First DFS, in comparison, may uncover solutions that are not the shortest path and does not ensure optimality. It is clear from the first three test cases that the DFS algorithm consistently generates

high explored states and high path costs. While it may be speedy for less complex puzzles (i.e Additional Test Case in Figure 1), it may thoroughly investigate nodes in a linear direction before backtracking, which may sometimes be deeper than expected depending on the puzzle complexity. For this reason, test cases 3 0 2 6 5 1 4 7 8 and 8 7 6 5 4 3 2 0 are unable to even give results for path cost and explored states. Although, the extra test case 2 3 0 1 5 6 4 7 8 is a nice illustration of when its optimality is better than BFS, but it can still be significant, especially for complicated problems.

A* excels in both efficiency and optimality. It ensures the shortest path solution and incorporates aspects of both BFS and DFS. According to the aforementioned test cases, while keeping optimality, A* frequently beats BFS on complicated puzzles or even DFS on easy ones. Since it gives priority to nodes with lower route costs, using the Manhattan algorithm, it usually has a more manageable space complexity and increases the likelihood of reaching the desired pattern.

Additional Test Case 1			
2 3 0 1 5 6 4 7 8			
	DFS	**BFS**	**A***
Path cost:	6	6	6
Explored states:	7	90	7
Time (seconds):	0.006458759308	0.02581501007	0.004483938217
Space (bytes):	19760	93840	19312
Additional Test Case 2			
1 2 3 4 0 5 7 8 6			
	DFS	**BFS**	**A***
Path cost:	28	2	2
Explored states:	29	9	3
Time (seconds):	0.01034188271	0.004175186157	0.003071784973
Space (bytes):	51824	17552	17424
Additional Test Case 3			
1 2 3 4 8 0 7 6 5			
	DFS	**BFS**	**A***
Path cost:	62331	5	5
Explored states:	78213	45	6
Time (seconds):	2814.325648	0.02329993248	0.004433631897
Space (bytes):	70314752	54512	19280

Figure 2: Additional Test Cases

In Figure 2, additional test cases were used to strengthen the claims more regarding the said algorithms. In terms of time complexity, among the three (3) additional test cases, the A* algorithm offers the least amount of time in seconds. When an algorithm produces a small amount of time, it means that it is able to complete the computation quickly. In this context, A* algorithm is able to finish the task quickly among the other algorithms.

Moving on to the memory space occupied, among the three (3) additional test cases, the A* algorithm still offers the least amount of memory space occupied. This is due to its way of prioritizing and managing the nodes it explores. A* algorithm incorporates a heuristic function that estimates the cost from a node to the goal which helps the program to prioritize nodes that are more likely to lead to the optimal solution. When an algorithm like A* occupies little memory space, it means that the algorithm requires fewer resources to store data and other information while it's running, promoting efficiency for its memory, and scalability to handle other more complex puzzle combinations.

A* algorithm is both the best choice for its time complexity and memory space occupied which then

followed by either DFS or BFS depending on which of the two reach the goal first, which can be known by its number of explored states. The lesser the explored states, the faster the algorithm to reach the goal puzzle combination and the less memory it will occupy. Although BFS is designed for its optimality of its solutions, there are still some cases where the DFS happens to find the solution first than BFS.

Furthermore, the selection of an algorithm is also influenced by certain conditions and limitations. Based from the points claimed earlier from the recorded path costs, explored states, time complexity, and memory space occupied, the most appropriate strategy for resolving the 8-puzzle game is A*. Once more, it is evident from the test cases from Figure 1 that the A* algorithm is the most practical method for solving the most complicated input, which is 8 7 6 5 4 3 2 1, the test case that even DFS and BFS cannot solve due to the non-terminating program. DFS, on the other hand, would be the least appropriate since, in contrast to BFS, it does not always guarantee the shortest path. Most of the test cases also show that it may discover less-than-optimal solutions. Notice how among all the test cases from Figure 1 and Figure 2, BFS and A* algorithm always comes up with the same path cost due to its guaranteed algorithm of finding the most optimal solution, while there were test cases that the DFS algorithm offers a different and also a larger amount of path cost.

2 Informed and Uninformed search strategies

Uninformed search strategies work by depending just on the given puzzle structure. Without considering any patterns or qualities that may help quickly identify the answer, they evenly traverse the area, frequently widening all viable pathways until one is discovered. BFS and DFS are two instances of this kind of search. Their effectiveness, however, varies, with BFS ensuring optimality but perhaps experiencing high time and space complexity, especially in vast search areas, and DFS only being sometimes dependable on easy puzzles.

As opposed to this, informed search strategies make use of heuristics values, which calculate the amount still needed to achieve the goal. These algorithms prioritize nodes based on path costs, focusing first on more promising paths that are closer to the goal. Notably, when an accurate heuristic is applied, A* search, an informed search strategy, ensures optimality, just like the BFS algorithm. Because A* algorithm investigate fewer states and save memory, these kind of searches are frequently more time and space efficient when compared to uninformed search.