14310166 Breandán Kerin 10/03/2019

AI – CS7IS2 – Assignment 1

Part 1 analysis – 10 points

1. Autograder marks.

Question	Mark	Reason Marks Lost
Question 1 – DFS	3/3	N/A
Question 2 – BFS	3/3	N/A
Question 3 – UCS	3/3	N/A
Question 4 – A* Search	3/3	N/A
Question 5 – Corners Representation	3/3	N/A
Question 6 – Corners Heuristic	3/3	N/A
Question 7 – Eating All Dots	4/4	N/A
Question 8 – Suboptimal Search	3/3	N/A
Total Mark Part 1	25/25	N/A

2. Algorithm Comparison.

Algorithm	Criteria 1:	Criteria 2:	Criteria 3: Optional
DFS	Memory Complexity	Time Complexity	Didn't Consider Criteria
BFS	Memory Complexity	Time Complexity	Didn't Consider Criteria
UCS	Memory Complexity	Time Complexity	Didn't Consider Criteria
A* Search	Memory Complexity	Time Complexity	Didn't Consider Criteria

3. Discussion: Analysis of the algorithms per criteria and trade-offs.

For the purposes of this report, it is required to compare different aspects or criteria of the algorithms implemented. To this end, the comparison criteria selected were i) Memory Complexity, which is measured by the amount of nodes expanded during the search phase; and ii) Time Complexity, which is the time taken to find a solution to the search problem. Time complexity is measured in seconds.

Note 1) A third and optional comparison criteria was not considered and hence omitted. Another possible criteria that could be used to help evaluate the algorithms is the total cost of the search in the event that each algorithm would find a separate cost for the solutions.

Note 2) The values shown below are the averages across three different individual runs of the experiments. Note, that the number of nodes expanded were rounded to the nearest whole number.

Note 3) Both the Manhattan and Euclidean heuristics were implemented and tested in this section for the A* algorithm. However the results displayed below only related to the values obtained when tested with the Manhattan Distance algorithm.

Time Complexity Measurements (s)				
Problem	**DFS time**	**BFS time**	**UCS time**	**A* Search time**
Tiny Maze	0.001	0.001	0.001	0.002
Medium Maze	0.011	0.015	0.039	0.03
Big Maze	0.024	0.034	0.156	0.128
Finding All Corners Tiny Corners	0.001	0.004	0.013	0.008

Finding All Corners Medium Corners	0.009	0.133	0.385	0.106
Eating All Dots testSearch	0.001	0.001	0.001	0.001

Memory Complexity Measurements (#Nodes Expanded)				
Problem	**DFS Nodes**	**BFS Nodes**	**UCS Nodes**	**A* Search Nodes**
Tiny Maze	15	15	15	14
Medium Maze	130	269	269	221
Big Maze	390	620	620	549
Finding All Corners Tiny Corners	51	252	252	160
Finding All Corners Medium Corners	221	106	106	806
Eating All Dots testSearch	7	14	14	12

As it can be seen from the above two tables on time and memory complexity of the different algorithms, DFS on average performs better than BFS, UCS and A*, except for the tiny maze where A* found the solution with less nodes expanded; albeit only 1 node less than DFS. When considering a comparison between DFS, A star, BFS and UCS, it would be expected that DFS could potentially outperform both of these models. This is due to the fact that DFS searches the depths of each sub-tree first, then continues onto the next sub-tree. And it would be expected that, depending on where pacman started, DFS would out-perform the breadth based models such as BFS and UCS. Hence, it is no surprise to see DFS outperform UCS, a star and BFS in both time and memory complexity. In a theoretical analysis of these results, it is known that the time and space complexity of there algorithms are as follows:

1. DFS
    a. Time – $O(b^d)$ Space – $O(bd)$
2. BFS
    a. Time – $O(b^d)$ Space – $O(b^d)$
3. UCS
    a. Time – $O(b^{C^*/E})$ Space – $O(b^{C^*/E})$
4. A star
    a. Time – $O(b^d)$ Space – $O(b^d)$

where b is the breadth of nodes in the level, d is the depth, C is the cost of the optimal Solution, E is max permissible cost for the next step*.

From the theoretical analysis of the algorithms, it can be seen that DFS will perform better with regards to space complexity, but would perform similarly to A Star, UCS and BFS in a worst case scenario for the time complexity. However, it tends to outperform the three other algorithms which would lead one to believe that this is due to a combination of i) the starting position of pacman, ii) the maze that has to be solved and iii) the method of searching the maze which leans in favour of DFS to find a solution with the least amount of nodes expanded.

Interestingly, in the ***Finding All Corners Medium Corners*** the two better solutions when memory availability is of upmost importance, BFS and UCS outperformed DFS. This again highlights how the solution to the problem at hand is determined by the problem – for example, the solution to this problem may have been in the second tier of nodes but on the extreme right of the graph/tree – this means that for DFS it would be required to search the subtrees to the left of the solution before the solution can be visited by the algorithm. This serves to highlight the importance of understanding the problem at hand; if possible, and then applying the most suitable algorithm to the problem task at hand.

It should be noted that A Star underperformed when compared to DFS, but performed better than both UCS and BFS for most tasks in memory complexity, and sometimes on time complexity. It would be interesting to compare total cost taken for the algorithms for this – as A* is an optimal search algorithm and if in the event that there were several possible ways to achieve a solution, A* would achieve the lowest total cost where it's not guaranteed with the other search algorithms.

Part 2 analysis – 10 points

1. Autograder marks.

Question	**Mark**	**Reason Marks Lost**
Question 1 – Reflex Agent	4/4	N/A
Question 2 – Minimax	5/5	N/A
Question 3 – Alpha-Beta Pruning	5/5	N/A
Question 4 – Expectimax	0/5	Didn't Implement
Question 5 – Evaluation Function	0/6	Didn't Implement
Total Mark Part 1	14/25	Q4 & Q5 not implemented

2. Algorithm Comparison.

Algorithm	**Criteria 1:**	**Criteria 2:**	**Criteria 3: Optional**
Question 1 – Reflex Agent	Average Score	Max Time	N/A
Question 2 – Minimax	Average Score	Max Time	N/A
Question 3 – Alpha-Beta Pruning	Average Score	Max Time	N/A
Question 4 – Expectimax	Didn't Implement	Didn't Implement	Didn't Implement
Question 5 – Evaluation Function	Didn't Implement	Didn't Implement	Didn't Implement

3. Discussion: Analysis of the algorithms per criteria and trade-offs.

		Reflex Agent	Minimax	Alpha-Beta Pruning
Trapped Classic	Win Rate	60%	60%	80%
	Average Time (s)	0.0044998624610901	0.0115000486373904	0.0138999992370605
Small Classic	Win Rate	60%	10%	10%
	Average Time (s)	0.1413999557495117	1.989900016784668	1.540900015930993
Open Classic	Win Rate	100%	Memory Error	Memory Error
	Average Time (s)	0.232099986076355	Memory Error	Memory Error

For the purposes of comparing the algorithms, the metrics chosen were the win rate; that is the percentage of wins the algorithm has achieved for N[1] iterative runs, and the average time to determine a solution. On average, it can be seen that the Reflex Agent outperforms both the minimax and alpha-beta pruning algorithms. One likely reason for this is that in the MiniMax algorithm when the death of pacman is unavoidable, i.e. the algorithm knows that there is little chance of pacman surviving longer, he will try to end the game as soon as possible in order to prevent obtaining a lower score for the run.

In order to obtain a better result for both the MiniMax and Alpha-Beta pruning algorithms, a better evaluation function should be implemented: Note this algorithms was originally intended to be completed but due to having the requirement of implementing the expectimax function – which hasn't been covered, this piece of implementation has been omitted. It should be noted for *OpenClassic* for both minimax and alpha-beta pruning, the pacman game had ran out of memory. With this, I assume that this is due to pacman becoming stuck in a local area unable to escape given the certain circumstances of the game, leading to large amounts of nodes to be expanded without a solution out of the local area. I suspect that if the better evaluation function for question 5 was implemented, this would eliminate this issue.

Despite this, it is clear from the results that the performance of the algorithms is problem dependant similar to that in the single agent search problems described above. This can be seen that the performance of alpha-beta pruning outperformed both minimax and reflex agents by 20%, albeit it took a little longer to run to completion compared to the reflex agent system: The alpha-beta pruning system took an extra 0.0094001367759704 seconds when compared to the reflex agent.

Similarly, this can also be seen for the tests run on the small classic map, when the reflex agent outperformed both minimax and alpha-beta pruning in both average time taken to complete the tests and the average win rate for the 10 tests ran.

[1] For the purposes of this report N is 10 – i.e. the tests were run 10 time iteratively to generate average performance statistics.