Allen Jake Polintan

Algorithm	Time Complexity	Space Complexity	Complete	Optimal
BFS	$O(B^{d+1})$	$O(B^{d+1})$	Yes	Yes
UCS	$O(\mathbf{b}^{(1+C/e)})$ (e is the minimum cost and C is the cost)	$O(\mathbf{b}^{(1+C/\varepsilon)})$	Yes	Yes
DFS	$O(B^{N})$ (n is the depth)	O(bn)	Yes (on finite graphs)	No
DLS	$O(B^{L})$ (L is limit of the depth)	$O(Bl)$(l is the limit of the depth)	No	No
IDS	$O(B^{d})$ (d is the depth)	$O(Bd\ )$	Yes	Yes
A*	$O(B^{d})$ (d is the depth)	$O(B^{d})$	Yes	Yes

Running Mission Complete:

Def mission_complete(state):

return s.loc = “battery” and s.sample_extracted == True and s.charged == True

I ran the following function

```
s1 = RoverState()
    result1 = breadth_first_search(s1, action_list, mission_complete)
    result1 = depth_first_search(s1, action_list, mission_complete)
    result1 = depth_limited_search(s1, action_list, mission_complete,10)
```

Mission Complete Goal

```
def mission_complete(state) :
    return state.charged == True and state.loc == "battery" and
state.sample_extracted == True
```

Results of BFS AND DFS Implementation

Breath First Search: 65 States

Depth First Search: 44 States

Depth Limited Search: 44 States

Problem Decomposition for Breath First Search

```
  def moveToSampleTest(s):
        return s.loc == "sample"
    def removeSampleTest(s):
        return s.sample_extracted == True
    def returnToChargerTest(s):
        return s.loc == "battery" and s.charged == True
```

```
print(s2)
    result2 = breadth_first_search(s2, action_list, moveToSampleTest)
    result3 = breadth_first_search(result2[0], action_list,
removeSampleTest)
    result4 = breadth_first_search(result3[0], action_list,
returnToChargerTest)
```

I ran the following to see how many states are created if I separated mission complete to multiple functions.
As stated before Breath First Search w/o separating it to multiple problems will result in 61 states

Result of Problem Decomposition

The amount of states generated by separating the big problem to a small problem lead the following calculation:
Result 2: 4 states
Result 3: 16 states
Result 4: 27 states
Overall, **47 states** were generated

If I were to compare the number of states generated, problem decomposition generated less states than running one big breath search on the whole mission complete function.

Rover Problem

Running the following code

```
 s1 = map_state(g=1,h=1,location="8,8")
    result = a_star(s1, sld, reachedGoal)
    result = a_star(s1, h1, reachedGoal)
```

Led to the following in terms of generating states

Results (Starting Location 8,8)

A* : 61 states were generated

Uniform Cost Search: 61 states were generated

(**Starting Location 4,4)**
A*: 41 states were generated
UCS 61 states were generated

(Starting Location 3,6)
A*:27 states were generated
UCS: 48 states were generated

Question 5

1) The engineering advances that helped lead to Deep Blue's success after Deep Blue 1's failure included redesigning the chess chip to involve 8000 features instead of 6400 features. These features included better hardware repetition detection, specialized move generation nodes, and efficiency improvements. These lead to improvements to have things such as a large search capability, ability to use non-uniform search, and having a complex evaluation system. Other advances included doubling the number of chess chips and adding debugging features for Deep Blue which included tests and visulizations.

The advances that can be transferred to other problems include the debugging features that were added, search speed improvements, and hardware repetition detection. Having a parallel system that can allow parallel search can also be applied to other problems as well.

The advance specifically related to chess was Deep Blue's advanced evaluation system. Although other problems can have evaluation systems, Deep Blue's evaluation system was unique to chess. This system analyzed thousands of different patterns related to chess. Advances in the evaluation system included better detection of piece mobility, king safety, and rooks on files. The extended book is another advance specifically used for chess. The extended book is a tool that allows Grandmaster game database to influence Deep Blue's play. This is useful if there is no information for the opening section of the game and the algorithm would be on its own. The extended book gives more points towards moves made by Grandmasters. These moves consider factors such as how often the moves were played, the results of the moves, and various annotations made on the moves. The chess chip also included end game databases that guided the algorithm when there were only 5 or 6 chess pieces left.

2) In the reading we learned that Alpha Zero was able to beat Stockfish while using only 1/100 of computing time compared to Stockfish. The computing time difference is due to the amount of searches. Alpha Zero searched only 80 thousand positions per second in a game like chess while Stockfish searches 70 million positions per second when they played against each other. The reason why Alpha Zero is able to beat Stockfish because the variations of the nodes that are evaluated are more focused towards the most successful, or "most promising" nodes for the the game. Alpha Zero uses a deep neural network to accomplish this. Alpha Zero uses only one neural network that is "updated continually, rather than waiting for an iteration to complete" and

reused hyper parameters. This means that alpha zero is self-learning and calculates the best moves and probabilities based off plays made before it.

Alpha Zero essentially mastered the openings and moves involved in chess. Compared to AlphaGo Zero, Alpha Zero optimizes the expected outcome.