[TDDC17] Lab 2 Report

	Student 1	**Student 2**
First Name	Jarrett Shan Wei	Yong Loong
Last Name	Yeo	Ang
LiU-ID	Jarye821	Yonan994
Personal No.	19960207T533	19960506T275

1. In the vacuum cleaner domain in part 1, what were the states and actions? What is the branching factor?

Each square of the grid is a state. The initial state (1,1), which is the starting point for the agent. The goal state when the agent is on the grid square with dirt, where the agent suck the dirt up. In a given state, the agent can suck or move (up, down, left or right). The branching factor is 4

2. What is the difference between Breadth First Search and Uniform Cost Search in a domain where the cost of each action is 1?

Uniform Cost Search prioritise the action with the lowest cost while Breadth First Search always use the “nearest action”. Since the cost of each action is 1 for this case, both search methods will be the same to find the first goal. However, the difference is that after the first goal is found, Uniform Cost Search continues to examine all other nodes at the goal’s depth to find a lower cost solution while Breadth First Search will stop.

3. Suppose that h1 and h2 are admissible heuristics (used in for example A*). Which of the following are also admissible?

An admissible heuristic never overestimates the cost of reaching the goal from n.

If we assume C* is the optimal cost for a given path. If both h1 and h2 are admissible, then they are both less than or equal to C*.

$C^* > 0$

$h1 > 0$

$h2 > 0$

h1 = C* - y, y >= 0

h2 = C* - z, z >= 0

a) (h1+h2)/2

The average of h1 and h2 is also less than or equal to C* since the average of two smaller numbers is also a smaller number.

Proof:

(h1 + h2) / 2 = (2C* - y - z) / 2 = C* - y/2 - z/2 <= C*

Thus **(h1+h2)/2** is admissible.

b) 2h1

Two times of a smaller number does not have to be smaller than a larger number.

Proof:

Let C* = 10 and h1 = 6

2*h1 = 12 > C*

Thus **2h1** is not admissible.

c) max (h1,h2)

The larger number of the two smaller numbers is still smaller.

Proof:

Assuming h1 <= h2 (h1 and h2 can swap in variables, not affecting results), we get

max(h1, h2) = h2 = C* - z <= C*

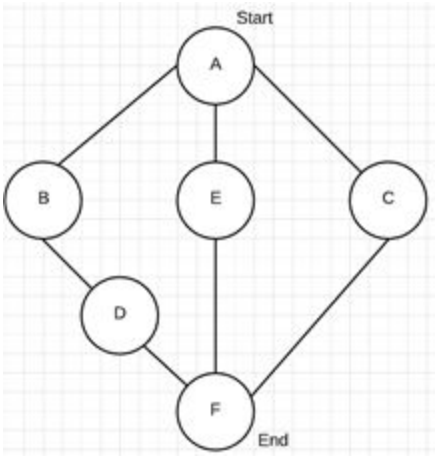
Thus **max(h1, h2)** is admissible.

4. If one would use A* to search for a path to one specific square in the vacuum domain, what could the heuristic (h) be? The cost function (g)? Is it an admissible heuristic?

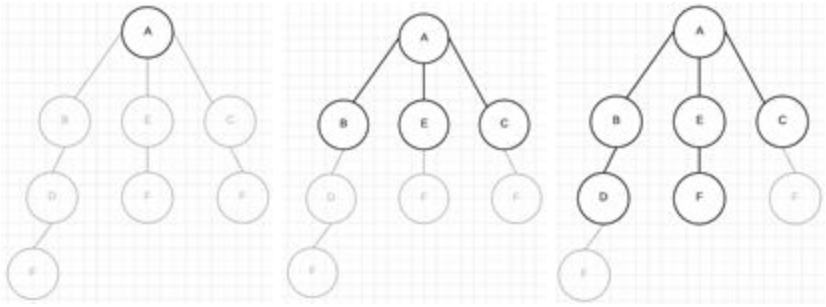
The heuristic h(n) could be the Manhattan distance from the current node to the goal node. The cost function g(n) could be the total number of steps taken to reach the current position node from where it started. With the above, it can be concluded as an admissible heuristic.

5. Draw and explain. Choose your three favorite search algorithms and apply them to any problem domain (it might be a good idea to use a domain where you can identify a good heuristic function). Draw the search tree for them, and explain how they proceed in the searching. Also include the memory usage. You can attach a hand-made drawing.

For the following graph example, we want to find the shortest path to go from A to F.



<u>Breadth-First Search (BFS)</u>
It explores the tree level by level and always expands to the shallowest node in the frontier.



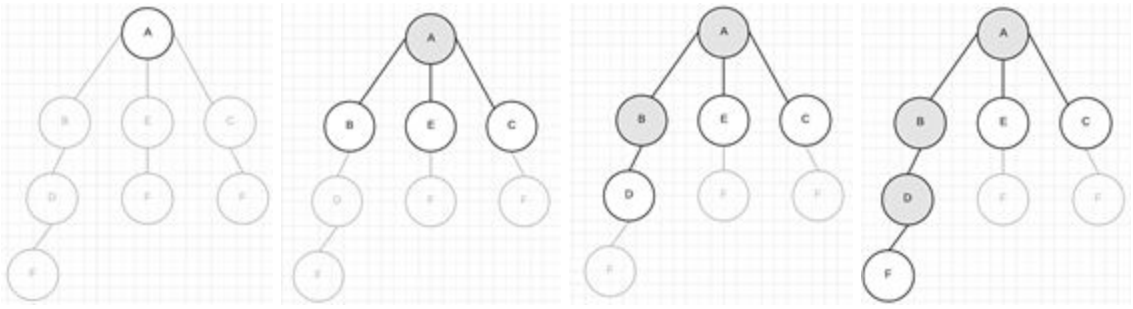
	Frontier	Explored
Init	A	
Pop		A
Add Children	B,E,C	
Pop	E,C	A,B
Add Children	E,C,D	
Pop	C,D	A,B,E

Add Children	C,D,F	
Pop	D,F	A,B,E,C
Pop	F	A,B,E,C,D
Pop		A,B,E,C,D,F

For each child of the expanded node, BFS checks if this child is already explored or in the frontier. If not, it checks if this child state is the goal state. If that is true, it returns the solution. In this case the algorithm stops before the green square as shown on the table above since F is found. For memory wise, every node is stored either in the frontier or explored list. The branching factor *b* is 1 <= *b* <= 3 and the goal depth *d* is 2. The space complexity is $O(b^d) \approx O(3^2)$.

Depth-First Search (DFS)

It explores the tree branch by branch, expanding the first child node, continuously going deeper until a goal is found or a node has no children.

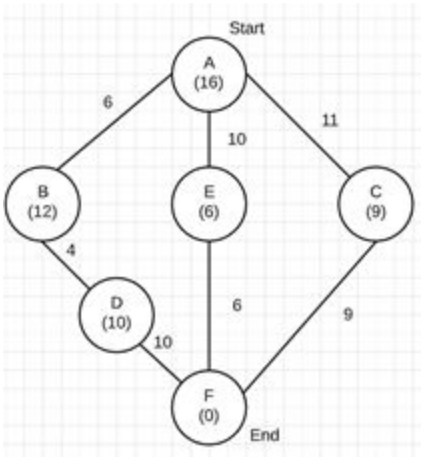


	Frontier	Explored
Init	A	
Pop		A
Add Children	B,E,C	
Pop	E,C	A,B
Add Children	D,E,C	
Pop	E,C	A,B,D
Add Children	F,E,C	
Pop	E,C	A,B,D,F

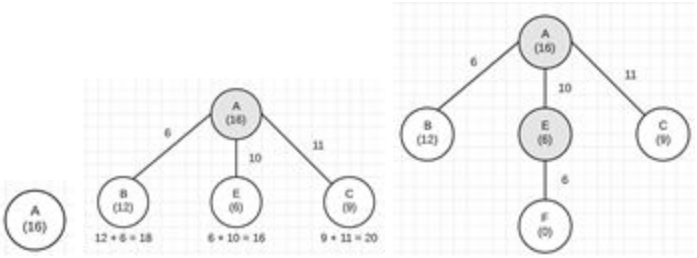
From above, we can see that DFS is not optimal as the solution returned is not the shortest path. The path cost is 3 branches instead of the 2 on the branch A-E-F. The space complexity is equal to $O(bd) \approx O(3 * 2)$, which is less than BFS.

A* Search

This algorithm is based on informed search. A good heuristic *h(n)* is the straight line between the goal node and the current node. The cost function *g(n)* is the total cost from the start node to the current node.



The function *f(n) = h(n) + g(n)* after an expansion for each child node to always select the cheapest path.



	Frontier	Explored
Init	A	
Pop		A
Add Children	E,B,C	

Pop	B,C	A,E
Add Children	F,B,C	
Pop	B,C	A,E,F

For the worst case scenario, the space complexity is $O(b^d)$.

6. Look at all the offline search algorithms presented in chapter 3 plus A* search. Are they complete? Are they optimal? Explain why!

Breadth-First Search (BFS)

It is complete. If the shallowest goal node is at some finite depth d, BFS is guaranteed to find it after searching all shallower nodes if there is a solution, provided the branching factor is finite.

It is optimal. The shallowest goal node may not necessarily be optimal, but it will be optimal if the path cost is a non-decreasing function of the depth of the node, i.e. each action has the same cost, as that will be the path of minimal depth.

Depth-First Search (DFS)

It is complete for the Graph Search version in finite state spaces. It avoids repeated states and redundant paths and will eventually expand every node. However, it is not complete for the Tree Search version as it may loop infinitely on one branch.

Both versions are not optimal as they will always return the first solution found, even if a lower cost solution exists.

Uniform-Cost Search (UCS)

It is complete since UCS is based on BFS, but utilising priority queue instead of FIFO queue and expand to the lowest path cost node.

It is optimal. Unlike BFS, it is optimal even if the step-cost between nodes varies since it takes the current path cost into consideration when expanding.

A* Search

It is complete in finite space. It will eventually reach a contour equal to the path of the cost to the goal state, similar to UCS.

It is optimal only if the heuristic given is admissible and consistent.

Depth-Limited Depth-First Search

It is not complete. It is based on DFS, but with a search limit to avoid infinite-path problem. If all possible solutions are deeper than the limit, no solution will be found.

It is not optimal just like DFS, having the first solution not necessary being the lowest cost solution or when the depth-limit is greater than the depth of the optimal solution.

Iterative-Deepening Depth-First Search

It is complete when the branching factor is finite since it will explore every level fully and not explore one branch infinitely.

It is optimal when the path cost is a non-decreasing function of the depth of the node since that will be the path of minimal depth.

7. Assume that you had to go back and do Lab 1/Task 2 once more (if you did not use search already). Remember that the agent did not have perfect knowledge of the environment but had to explore it incrementally. Which of the search algorithms you have learned would be most suited in this situation to guide the agent's execution? What would you search for? Give an example.

We used Depth-First Search (DFS) for node search and Breadth-First Search (BFS) for path search for both our lab 1 task 1 and task 2. Given that the agent has no knowledge of the environment when it starts, it would not be possible to use heuristic search and an uninformed search should be used. We used DFS to ensure the Agent takes the furthest path possible from the starting position, and backtrack only when it has finished visiting the deepest nodes. This way we can minimise the number of actions to take. Our agent also runs BFS every time it wants to travel from one node to another to ensure the shortest path possible.

For the final part where the agent is supposed to return to home position after cleaning up all the dirt, since the environment is fully discovered, we could use a heuristic algorithm such as A* Search to take the shortest path back to the home position.