TDDC17 - Artificial Intelligence

Lab 2 Report

Group Members: Alim Amanzholov (aliam864), Thibaut Vagner (thiva474)

Part 1. Search

We implemented a graph search that can be used as a BFS or DFS depending on how we use the frontier. In particular, if we use the frontier as a queue (i.e in a FIFO order), our graph search will search all the nodes on one level before proceeding to the next level of depth (i.e in a BFS fashion). If we use the frontier as a stack (i.e in a LIFO order), our graph search will become a DFS algorithm. To identify whether we should use the frontier as a queue or stack we used the insertFront boolean. In particular, BFS sets insertFront as true, which makes the frontier act like a queue and DFS sets the insertFront as false, which makes the frontier act like a stack.

Part 2. Theory

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

Our states contain x and y position and also whether there is dirt in that position or not.

Actions are going up, down, right and left and also sucking the dirt. Actions allow us to switch from one state to the other.

The branching factor is the number of possible actions and in our case, there are 5 possible actions, which means the branching factor is 5.

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

When the cost of each action is 1, both BFS and UCS are optimal and there is no difference between these two algorithms.

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

a) (h1+h2)/2

b) 2h1

c) max (h1,h2)

B is not admissible heuristic. For example, if h1 is 19 and the cost function is 20, 2h1 will be 38 which overestimates the cost function.

C is admissible heuristic because either h1 or h2 would be chosen and both are admissible

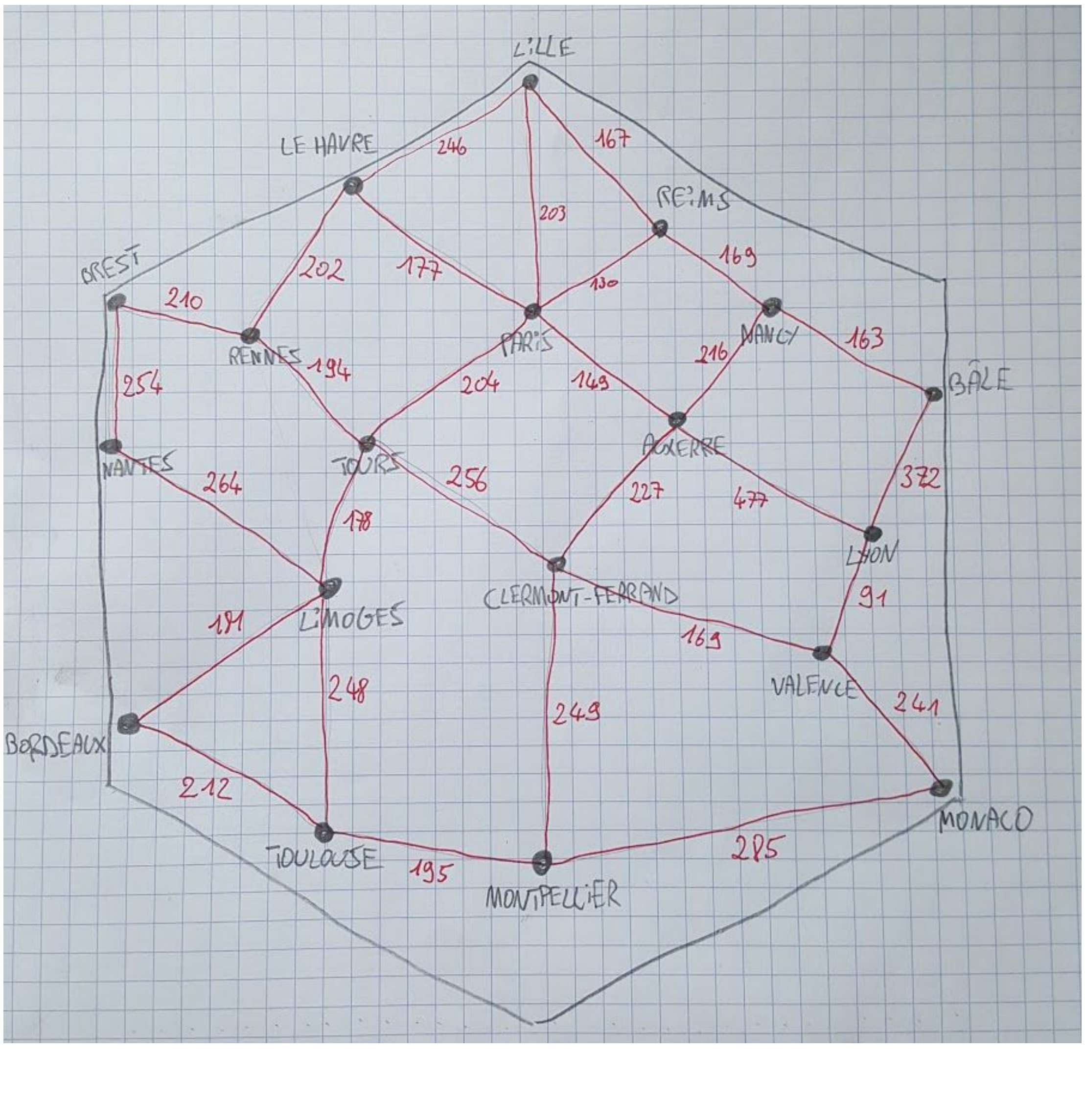
A is admissible heuristic because the average of two admissible heuristics will also be admissible since it cannot overestimate the cost function.

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) can be the length of straight line up or down until we reach the y position of goal and then left or right until we reach the x position of the goal. The cost function is the actual length of the path from the current node to the goal. The heuristic (h) is admissible since it never overestimates the cost function (g) because the heuristic is the shortest possible path from the node to the goal.

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 memory usage. You can attach a hand-made drawing.

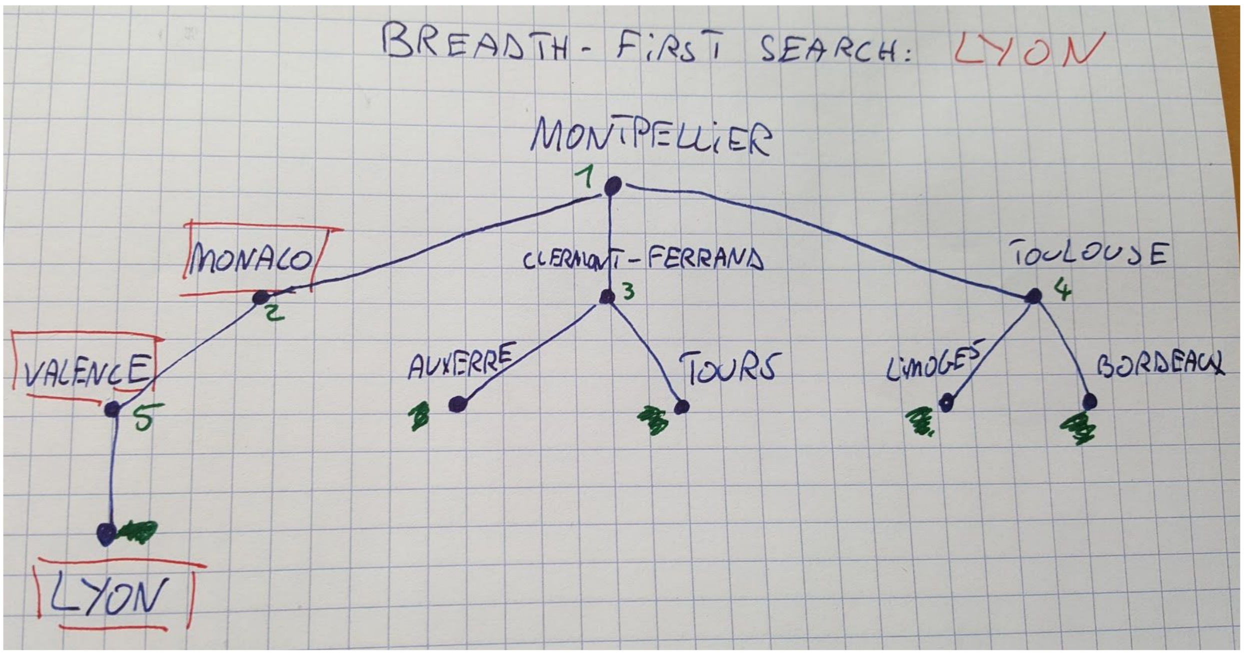
Our three favorite search algorithms are BFS, DFS, Uniform search.



BFS

In this case, the starting node is called “montpellier” and the goal node one is “lyon” . At each iteration, we we’ll visit all children nodes of the current nodes. At the beginning, the current one is montpellier. So, in that case, we will go to monaco first. If this node contains a child called “lyon” the algorithm stops and returns the path. But Monaco has only one child called “valence”, so we continue to other child node of “montpellier”. Then, we’re going to clermont-ferrand and checking if it contains the goal node. And finally, we visit toulouse. None of those nodes contains the goal node, so we’re going now one level deeper. We’re now at “valence” node, and “lyon” is a child of “valence”, so the algorithm stops and return path : [goto(monaco), goto(valence), goto(lyon)].

Assume that lyon wouldn’t be a child of valence, we would have to continue the same logic : visiting all neighbor nodes and checking, and go one deeper level if the goal node isn’t reach.



	Frontier (FIFO queue)	Explored
INIT	[montpellier]	
POP	montpellier	[montpellier]
Add Children	[monaco,Clermont-ferrand,toulouse]	
POP	monaco	[montpellier,monaco]
Add Children	[Clermont-ferrand,toulouse,Valence]	

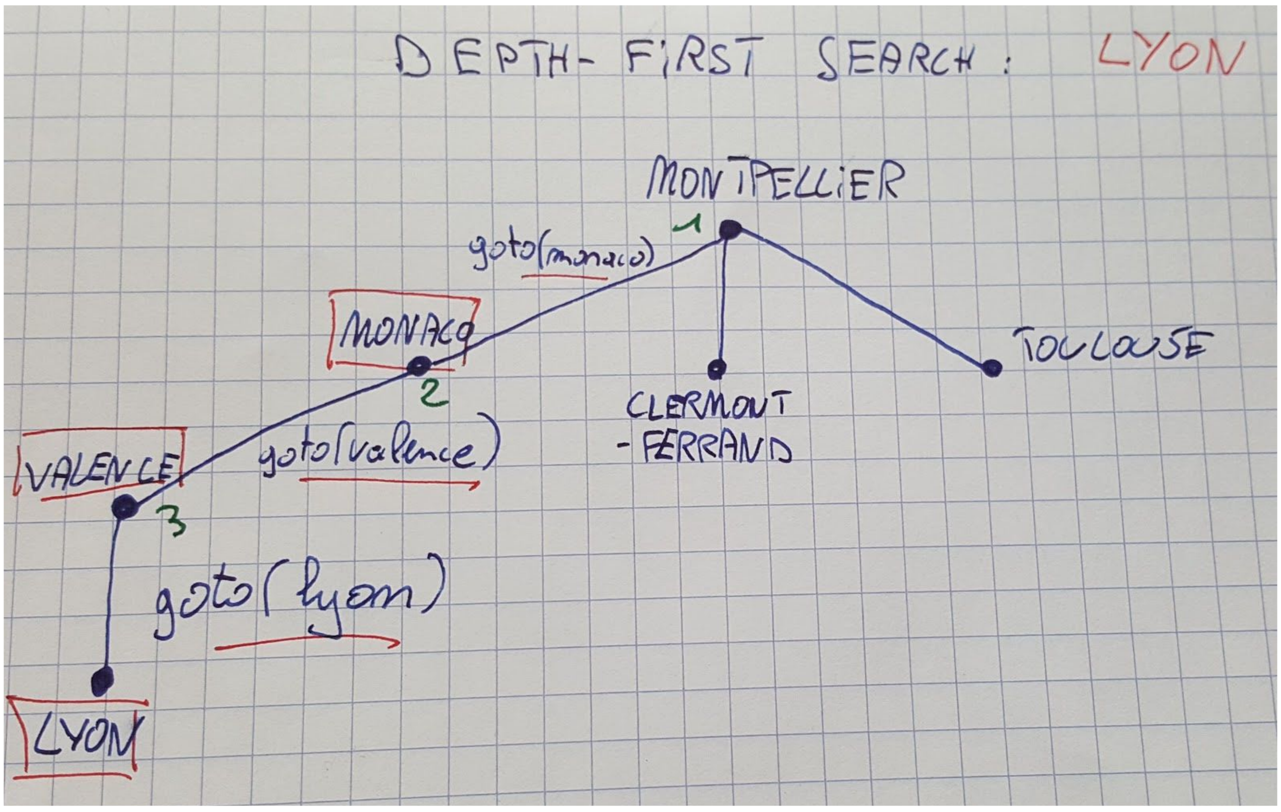
POP	Clermont-ferrand	[montpellier,monaco, Clermont-ferrand]
Add Children	[toulouse,Valence,auxerre, tours]	
POP	toulouse	[montpellier,monaco, Clermont-ferrand, toulouse]
Add Children	[Valence,auxerre, tours, limoges, bordeaux]	
POP	valence	[montpellier,monaco, Clermont-ferrand, toulouse, valence]
Add Children	[auxerre, tours, limoges, bordeaux]	
Goal Node	Lyon	

Solution :: [goto(monaco), goto(valence), goto(lyon)]

DFS

The depth-first search algorithm will also traverse graph. The difference with the Breadth-first search is that instead of visiting the all nodes closest to “current node”, it follow a path as far as we can go before we backtrack. In our case, we will choose monaco, then we continue because that node has at least one child. We’re now in valence node, we continue again. And the algorithm stops because the current node is the goal node : lyon.

Assume that lyon wouldn’t be a child of valence, we would have to backtrack to montpellier (because monaco as only one child) and we would go to clermont-ferrand.

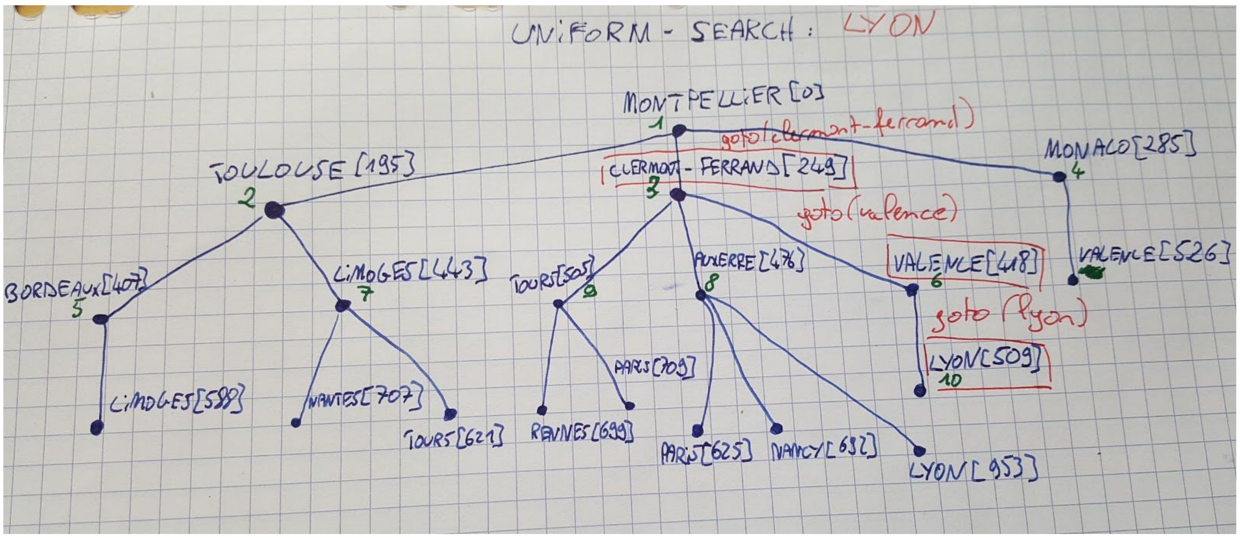


	Frontier (LIFO queue)	Explored
INIT	[montpellier]	
POP	montpellier	[montpellier]
Add Children	[monaco,Clermont-ferrand,toulouse]	
POP	monaco	[montpellier,monaco]
Add Children	[valence,Clermont-ferrand,toulouse]	
POP	valence	[montpellier,monaco, valence]
Add Children	[lyon,Clermont-ferrand,toulouse]	
Goal Node	lyon	

Solution :: [goto(monaco), goto(valence), goto(lyon)]

Uniform Cost Search

In the uniform search algorithm, the cost path from root to a node N is calculated. Then, at each steps, we'll choose the node with the lowest cost. We repeat this method again and again until to reach the goal node. In our case, we visite toulouse first, then clermont-ferrand, etc. The goal node (lyon) will be the tenth visited node. Once the goal node reached, we return the path.



		Explored
INIT	[montpellier]	
POP	montpellier	[montpellier]
Add Children	[monaco[285],Clermont-ferrand[249],toulouse[195]]	
POP	toulouse	[montpellier,toulouse]
Add Children	[bordeaux[407],limoges[443],Clermont-ferrand[249],monaco[285]]	
POP	Clermont-ferrand	[montpellier,toulouse,clermont-ferrand]
Add Children	[tours[505],auxerre[476],valence[418],bordeaux[407], limoges[443],monaco[285]]	
POP	monaco	[montpellier,toulouse,clermont-ferrand,monaco]
Add Children	[tours[505],auxerre[476],valence[418], Valence[526],bordeaux[407],limoges[443]]	
POP	bordeaux	[montpellier,toulouse,clermont-ferrand,monaco, bordeaux]
Add Children	[tours[505],auxerre[476],valence[418],valence[526], Limoges[443],limoges[588]]	
POP	valence	[montpellier,toulouse,clermont-ferrand,monaco, bordeaux,valence]
Add Children	[tours[505],auxerre[476],valence[526],Limoges[443],limoges[588],lyon[588]]	
POP	limoges	[montpellier,toulouse,clermont-ferrand,monaco, bordeaux,valence,limoges]

Add Children	[tours[505],auxerre[476],valence[526],limoges[588],lyon[588],nantes[707],tours[621]]	
POP	auxerre	[montpellier,toulouse,clermont-ferrand,monaco, bordeaux,valence,limoges,auxerre]
Add Children	[tours[505],valence[526],limoges[588],lyon[588],nantes[707],Tours[621],paris[625],nancy[692],lyon[953]]	
POP	tours	[montpellier,toulouse,clermont-ferrand,monaco, bordeaux,valence,limoges,auxerre,tours]
Add Children	[valence[526],limoges[588],lyon[588],nantes[707],Tours[621],paris[625],nancy[692],lyon[953], paris[709],rennes[699]]	
Goal node	lyon	

Solution :: [goto(clermont-ferrand), goto(valence), goto(lyon)]

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

BFS is complete and optimal if the cost of each action is the same. It is complete because if the shallowest goal node is at depth d, it will eventually find it after expanding all nodes that are located at smaller depth. It is optimal in the case when each action has equal cost because it always stops at the shallowest goal found. In case when costs are not the same, shallowest node is not necessarily optimal.

Uniform Cost Search is complete and optimal. It is optimal because at every step the path with the least cost is chosen and paths never get shorter as nodes are added which ensures that the search expands nodes in order of their optimal path costs. It is complete only if the cost of every step is nonnegative.

DFS is complete for graph search version because it avoids repeated states and redundant paths and will eventually expand every node. DFS is not complete for the tree search version because it may loop infinitely on one branch. The depth-first search is not optimal for both the graph search and tree search versions. When all of the edges have been explored, the search will backtrack until it reaches an unexplored nodes.

Depth-limited search is not complete and it is not optimal. It is not complete because if the limit is less than the depth where the goal is located the algorithm will terminate without finding a solution. It is not optimal if the limit is bigger than depth because it acts similar to DFS.

Iterative-deepening DFS is complete when the branching factor is finite and it is optimal when the path cost is a non-decreasing function of the depth of the node. It is complete since the algorithm combines the space efficiency of DFS and completeness of BFS. It is optimal because it operates similar to BFS but by using a lot less memory.

A* search is complete and optimal. It is complete because it is guaranteed to terminate on finite graphs with nonnegative edge weights since it eventually reaches a contour equal to the path of the cost to the goal state. It is optimal for a given admissible heuristic and admissible heuristic never overestimates the cost to the goal state, which means that value given by heuristic will be lower than the suboptimal solutions we might see during the search.

7. Assume that you had to go back and do Lab 1/Task 2 once more (if you did not use the 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 would choose BFS for the second task of Lab 1 because BFS is an optimal search algorithm for the world in Lab 1 since the cost of each action was the same.

We would search for unknown positions in the world. For example, if there is an unknown position our algorithm would give us the path to that position and in case there is no more unknown positions our search algorithm would terminate. Then we can just search the home and follow the path suggested by BFS to go home.