LABORATORY ASSIGNMENT 2

SEARCH

Part 2 – Theory:

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

A state is the specific situation of the "world" (our program's world) at a specific time, which means that in our domain the state would be our position in the maze (and the knowledge of the obstacles and dirt positions). The actions are our operators to modify the world, which in our domain would translate to the different movement options our agent have, we have four movement options (north, west, south and east) so 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?

The UCS algorithm is designed to select the node with the lowest cost so, in a context where all the actions have the same cost, it lacks of sense. In that kind of environment the BFS seems to work better (in spite of the big expansion of his conceptual tree).

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

1. **(h1+h2)/2 :** Is an admissible heuristic, as neither h1 or h2 can overestimate the cost the average of them is a valid solution.
2. **2h1 :** This wouldn't be admissible, because it would be an overestimation.
3. **max(h1,h2) :** As long as h1 and h2 are admissible heuristics, choosing the highest one shouldn't be a problem.

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?

For the heuristic we can consider the distance between our agent's position and the square position (a straight line distance). The cost could be the number of steps that the agent would have to take in order to get to the cell. It might not be the best heuristic, but is an admissible one because is realistic and it doesn't overestimate the cost.

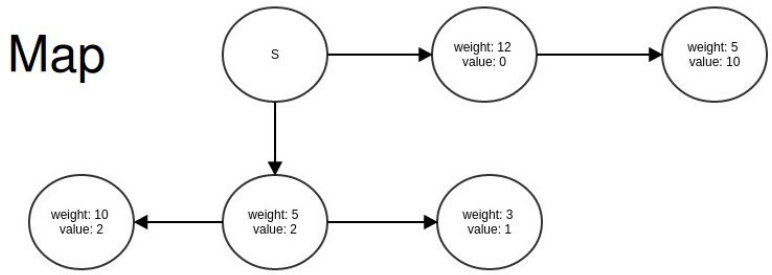
Oier Saizar
Juan Ramón del Caño

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.

Our problem will be the next: You're an adventurer and have a backpack that can only hold, let's say 15kg, and you have to select a path where there will be objects with a certain weight and value. You want to get the best amount of values, but you have a weight limit, so you can decide if you want to pick up the object or no. And also our explorer can't come back (for simplicity).

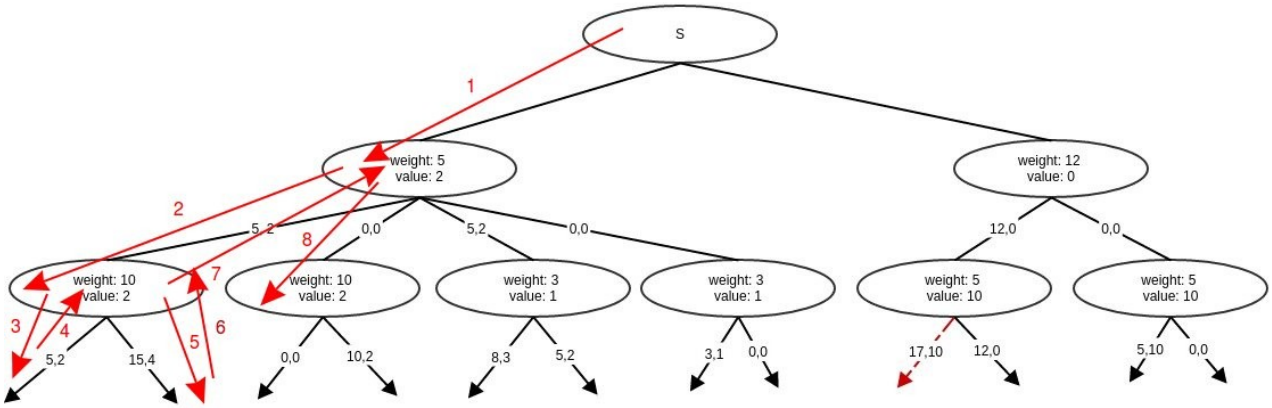
Our algorithm should get the best path and the best decisions about picking up the objects.

We're going to write a small problem, because the branching factor can be tricky as we have the decision of picking up the object or not and it grows pretty quickly.



Depth First Algorithm:

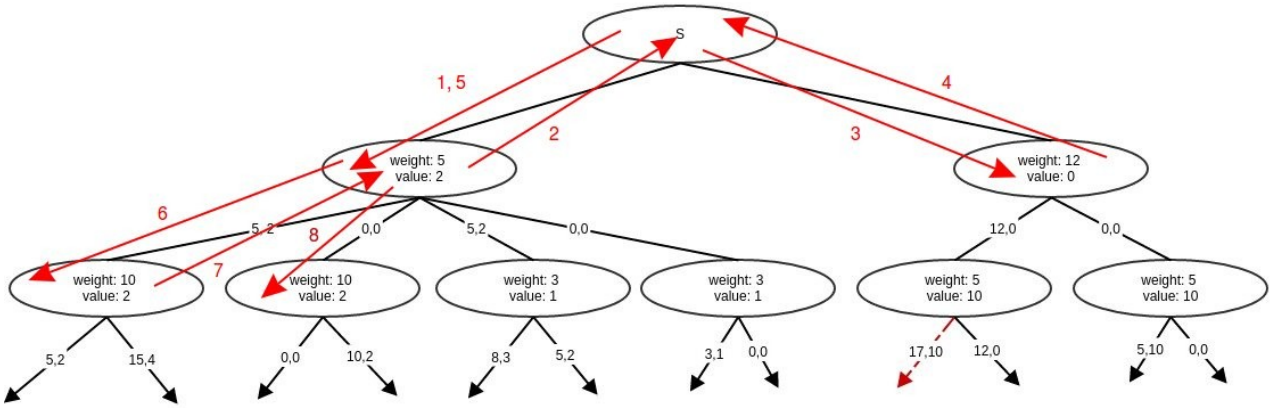
In this case we would get a good result with this algorithm, as it only has to check the different paths and store the best one. The memory usage would also be optimal, because it only has to maintain in it the actual branch of the search.



Oier Saizar
Juan Ramón del Caño

Breadth First Algorithm:

For this problem, using a BFS would probably be a bad decision. Of course it would find the best solution, but as we have to explore the entire tree it doesn't matter if the algorithm founds it sooner or later, and as the BFS has to maintain the entire tree in memory, it would certainly be a waste of it.

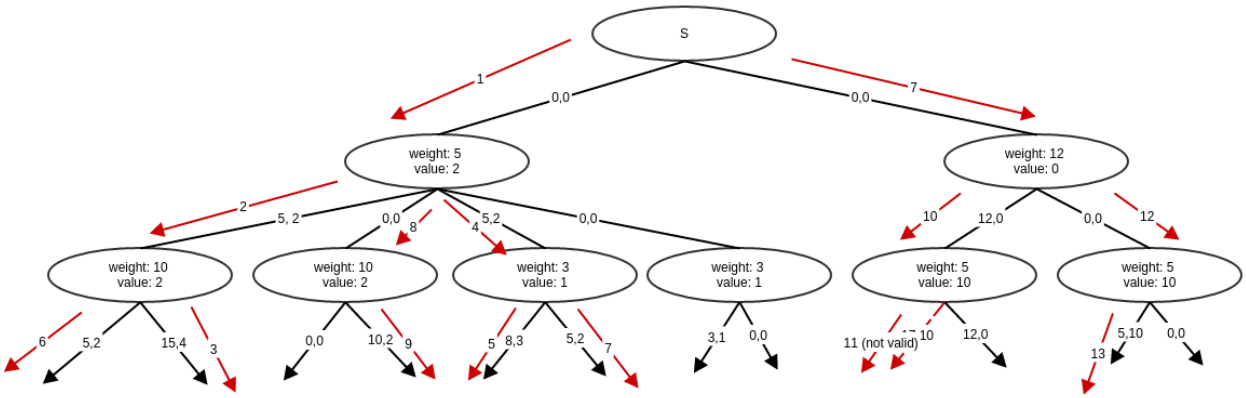


A* Algorithm

To use this algorithm a good heuristic is needed. As the most important part of our problem is the value of the items, our heuristic must seek for the greatest item values.

$$H = value$$

We also have to take into account that the agent can't carry more than 15kg on his bag, to accomplish this, we will check if a step is valid or not.



As we can see in the diagram, A* tries to find the path that most suites the heuristic. If the most valued steps have been explored, the algorithm will try to find better paths exploring the nodes that there were left behind previously.

Oier Saizar
Juan Ramón del Caño

The memory usage of the A* algorithm it's quite similar to the one on Breadth First Search. The algorithm has to maintain all the tree on the memory to be able to jump from one branch to another while it finds the branch that most suits the heuristic. We can say that A* is a guided version of BFS.

Oier Saizar
Juan Ramón del Caño

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

First of all we consider an algorithm complete if it always finds the solution (if there's one) and optimal if the solution found is the shortest (or the one with the smallest cost).

- **Breadth First:** It is complete, as long as the solution is on a finite depth. It is also optimal assuming that our optimal solution is the one located in the smaller depth of the tree.
- **Uniform Cost Search:** The completeness is guaranteed (IF there is not an infinite sequence of zero-cost actions, because then the algorithm could get stuck). UCS is optimal for two reasons: when the algorithm selects a node as a goal, it gets the shortest path (if not, then it gets a nearer node which is in the way to the goal node) and also, as there is no nodes with negative cost, the algorithm will not add additional nodes to the solution.
- **Depth First:** Well, it's kind of tricky. The algorithm can be complete, but we must avoid selecting repeated nodes or we could get stuck in an infinite loop. It's not considered an optimal algorithm because it will give us the first solution found exploring the left subtree.
- **Depth Limited:** It's pretty obvious that this can be an incomplete algorithm, if the solution is deeper than the limit we'll not going to fin it. It will also be nonoptimal if we choose Q>d.
- **Iterative Deepening:** Just as the breadth first, is complete (as long as the branching factor is not really big, we don't have limited memory, etc.). The same with the optimality (yes, it is).
- **Bidirectional:** As it is an implementation of the previous searches from two directions it is complete, but we cannot assure that the first solution it's going to be the optimal, for this requires a little more research to check if there's any shortcut somewhere.

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.

As the agent must explore the entire board, a DFS backtracking looking for the unknown cells would be a good idea, because it would be using the minimum amount of memory. Once the agent has finalized the backtracking it should be in the starting position and have a perfect knowledge of the environment, then it can use a BFS for planning the route to the home position in the minimum amount of steps.

Oier Saizar
Juan Ramón del Caño