Mikael Salasca (miksa599), Jonas Michelberger (jonmi043)

Lab report 2 - Task 2

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

The states of the vacuum cleaner domain in part 1 were determined by the agent location and the dirt locations. Therefore we had the possible internal state representations Unknown, Wall, Clear, Dirt, Home and our current position for every location. Moreover, there is an initial state and a goal state. The actions were none, move forward, turn right, turn left, suck dirt.

The branching factor defines how many child nodes a parent node has. In our vacuum cleaner domain in part 1 it is not uniform because of the walls. Therefore we use an average branching factor which is between 2 and 4 depending on the world. We have 4 moves possible if the agent is in the middle of the world, 3 if we are at an edge, 2 in the corners.

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 is instead of expanding the shallowest node expands the node with the lowest path costs. Thus, if all edges have the same costs they share a similar behavior except that BFS stops when it reaches the goal state while Uniform Cost Search continues searching at the level of the goal state to find a path with lower path costs. Therefore, Uniform Cost Search does more work by seeking at this level although it is not needed in this special case.

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

(h1+h2)/2 is admissible. Because if we have h1<=h* and h2<=h* we can deduce

(h1+h2)/2<=h*

b) 2h1

If we assume the extreme case of admissibility: h1=h*, then 2*h1 cannot fulfill h1<=h*

except h1 and h* are 0. There for it can not be admissible.

c) max (h1,h2)

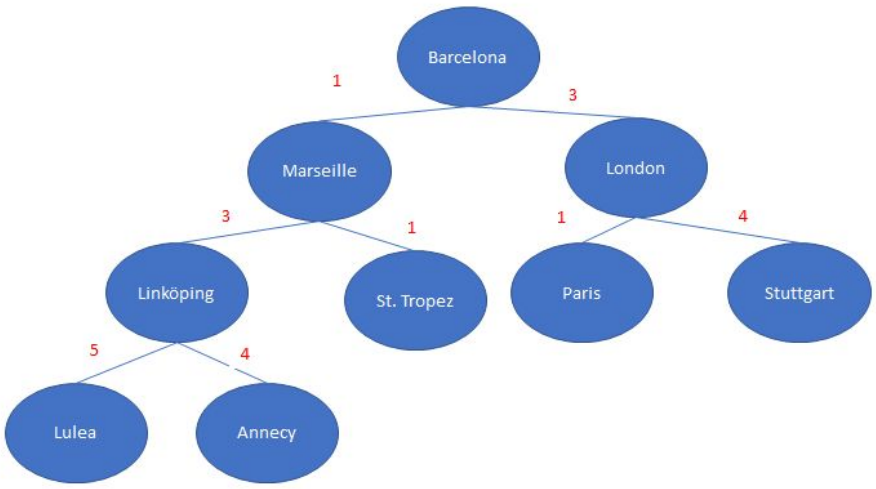
It is admissible since we choose the larger value. Because if we have h1<=h* and h2<=h* then we can deduce that max(h1,h2) <=h*.

4. If one uses A* to search for a path to one specific square in the vacuum domain, what could the heuristic function (h) be? What could the cost function (g) be? Is your choice of (h) an admissible heuristic? Explain why.

A heuristic function (h) could be the Manhattan distance which is the sum of the absolute pointwise distances. The cost function could be the path from the start node to the goal node. In this case the Manhattan distance is an admissible heuristic, because it does not overestimate the costs.

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.

Domain: Route planning to reach Stuttgart from Barcelona



Breadth-First Search:

Memory usage: FIFO Queue for the search frontier with the nodes to explore in the future. Uses a set for all explored nodes.

Barcelona - Marseille - London - Linköping - St. Tropez - Paris - Stuttgart

Depth-First Search:

Memory usage: LIFO queue / stack for nodes to explore in the future. Uses a set for all explored nodes.

Barcelona - Marseille - Linköping - Lulea - Annecy - St. Tropez - London - Paris - Stuttgart

Uniform-Cost Search:

Memory usage: Priority queue for nodes to explore in the future. Uses a set for all explored nodes.

Barcelona - Marseille - St. Tropez - London - Linköping - Paris - Stuttgart

6. Look at all the offline search algorithms presented in chapter 3 plus A* search. Are they complete? (i.e. Best-first search, Breadth-first search, Uniform-cost search, Depth-first search, Iterative deepening search, Bidirectional search, Greedy best-first search and A* search). Are they optimal? Explain why!

Best-First Search

Depends on evaluation function f. See A* and Greedy best-first below.

Breadth-First Search

Complete: Yes. If the goal node is at some finite depth level it will find it by exploring all shallower nodes until the goal node is found.

Optimal: Yes, if the path cost is a non-decreasing function of the depth because it always finds the shallowest goal node.

Depth-First Search

Complete: Graph Search version: Yes, because it expands every node (finite state spaces).

Tree Search version: No, it can loop.

Optimal: Both versions: not optimal, since we are exploring e.g. left part of the tree always first. Optimal goal could be on a shallow level on the right. Additionally, looping problem.

Uniform-Cost Search

Complete: Yes, if there is an optimal solution Uniform-Cost Search will find it by exploring all nodes with less path cost until it finds a solution.

Optimal: Yes. Everytime Uniform-Cost Search selects a node to explore, it chooses the optimal path with least cost.

Iterative deepening search

Complete: Yes, if the goal node is at some finite depth level. Then it can not follow infinite long paths or stay in cycles.

Optimal: Yes, if the path cost is a non-decreasing function of the depth. It is optimal if all path costs are equivalent, because in this case depth search finds the shortest path to a target.

Bidirectional search

Complete: Yes, if both directions use Breadth-First Search and the goal node is at some finite level.

Optimal: Yes, if both directions use Breadth-First Search and all step costs are identical.

Greedy best-first search

Complete: Incomplete even in finite state spaces. Can (like DFS) lead to infinite loops.

Optimal: Not optimal, the path found may not be the optimal one. Since it is only using the heuristic function it does neglect how to get to the reference points (compare to A*).

A* search

Complete: Complete if there are not infinitely many nodes to explore in the search space.

Optimal: A* is optimal, it always finds the optimal path between start node and goal node. Due to the requirement of a admissible heuristic (reason for the star*) it guarantees optimality.

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.

The most important requirement is the completeness for this problem. The implementation of a version of Breadth-First-Search would guarantee us a complete coverage of the vacuum cleaner when we have to choose an uninformed algorithm. In this case we would not look for a good time or space efficiency.