TDDC17 Artificial Intelligence

LAB 2 – Part 2

Davgr686, Ottde625
Linköping Universitet

18/09/25

In part 1 of this lab our goal was to create a custom breadth first search, as well as a depth first search. We were to implement the actual search of the lab, the rest was already implemented. In code, the two search algorithms are very similar. Both pushes "children"-nodes to a queue of nodes. The difference is that in the BFS version, the child nodes are pushed to the end of the queue. In DFS, they are pushed to the beginning of the queue.

The search consists of a while-loop, which stops once the "frontier" is empty, meaning that there are no more nodes to expand (which also means there are no more accessible "dirt"-nodes). While there are nodes in the frontier, we remove the first node in the queue and check whether or not that node is a goal node. If it is, the path to move from the current position to that goal node is returned. Otherwise, we expand the node (parent) that was removed, potentially adding four more nodes (children) to the frontier. If the children-nodes are valid, they are added to the frontier. If not, they are disregarded. A node is considered valid if the node has not been previously explored and is not a wall. As previously mentioned, this loop continues while there are nodes to expand in the frontier. If the while-loop breaks without returning a path to a goal node, an empty path is returned, indicating that the vacuum is done vacuuming.

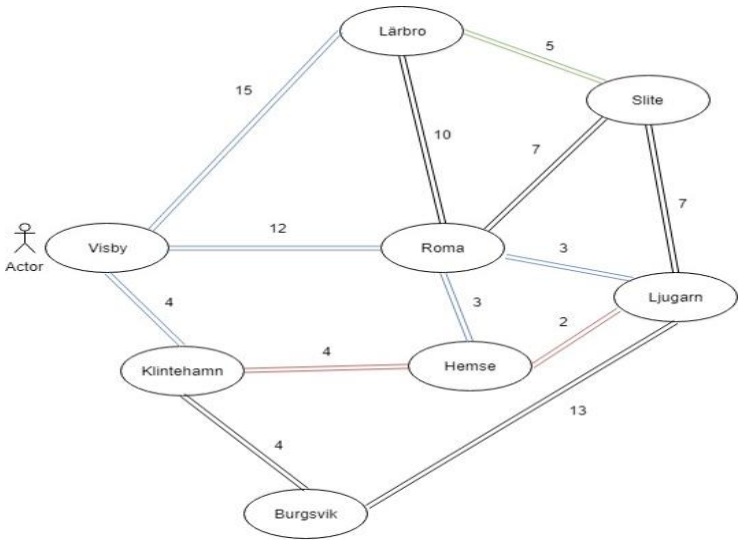
1. The states of the vacuum in part 1 was the current position of the vacuum, i.e. the coordinates. The actions of the vacuum are to move to a neighboring node of the current node, as well as vacuuming dirt. The branching factor is 4.
2. The difference between Breadth First Search and Uniform Cost Search is that BFS uses a FIFO queue to store the nodes that should be expanded, while UCS uses a priority-based queue that takes the lowest cost into account. When the cost of each action is equal, the priority queue in UCS will act like a normal FIFO queue. Therefore, there will be no difference between the two algorithms when the cost is the same for every action.
3. A and C. A because the average of h1 and h2 can never be larger than both h1 and h2. C since the maximum of h1 and h2 means that largest of the two will be used. Since both are admissible, the max of the two is also admissible.
4. A possible heuristic could be the straight-line distance between the current node and the goal node.
5. See bottom of document.
6. **Breadth First Search** – BFS is complete if the goal node is at a finite depth in the tree (if the depth is finite BFS will find every node available and therefore eventually find the goal node). BFS is also optimal if the cost of each action is the same. If different paths have different costs, BFS is not necessarily optimal.
   Uniform Cost Search – UCS is almost the same as BFS, with the difference that it chooses the node with the lowest cost from the root node and expands on that. It is complete, for the same reasons as BFS. It is also optimal since the algorithm expands on the node with the lowest cost. Therefore, eventually the path with the lowest cost will be found with UCS.
   Depth First Search – If the state space is finite, the graph search version of DFS is complete. Eventually, the algorithm will expand every node and find the goal node. The tree search version is not complete since it may loop infinitely between two nodes in the tree (in one

branch). DFS is not optimal because of the fact that it may expand one branch which eventually leads to the goal node but is not the most efficient path.
Depth Limited Search – DLS is the same as DFS but has a maximum depth. The algorithm will stop expanding on a branch once the maximum is reached. This algorithm is complete if the limit is larger than the depth of the tree (this means that the goal node will be present in the search space of the algorithm). It is not optimal for the same reasons as DFS.
Iterative Deepening Search – This algorithm combines BFS and DFS. It uses DFS up to a certain limit. Once everything up to that limit is explored it increases the depth-limit and continues using DFS up to the new depth-limit. The algorithm is complete if the goal node is at a finite depth and optimal if the path cost is the same for every node.
Bidirectional Search – Bidirectional search uses two searches "at the same time". It searches from the goal node towards the current node and from the current node towards the goal node. The two searches then meet in the middle. Bidirectional search is complete if BFS is used for both of the two searches. It is also optimal if BFS is used and the cost of each action is the same.

7. In lab 1 we implemented Breadth First Search. This was because we did not have an end goal. We only wanted to find the "shallowest" node that was unexplored. If we wanted to improve the algorithm and performance more, we could take the amounts of turns we would have to do to get to the node into account. The less turns the agent would have to do, the better. This could also be our heuristic if we wanted to implement A* search.

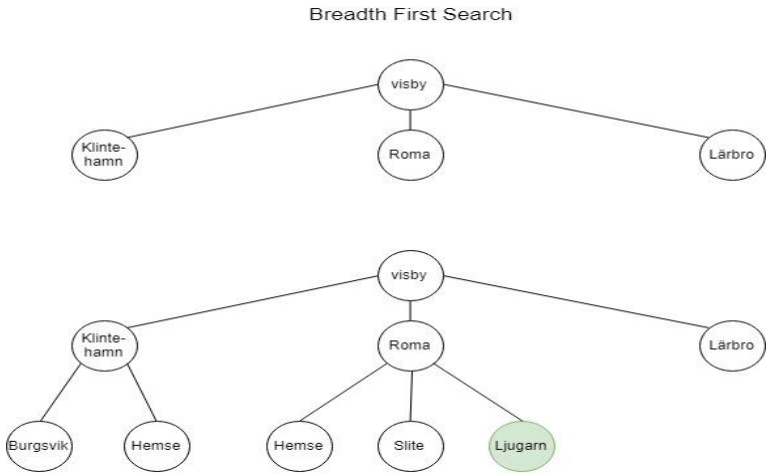
Examining three search algorithms

We have a road map of Gotland where we want to find the shortest path from Visby to Ljugarn. The roads distance is represented with the numbers next to the roads. We chose to examine Breadth First Search, Uniform-cost Search and A* Search.



Breadth First Search

Breadth first search will return the path with the least amount of towns passed. BFS will return the "shallowest" nod with the goal state and does not take the distance from one town to another into account. The cost of the actions in our world are not the same for all, this means that BFS could chose the path with the least amount of nodes in them but with the longest distance and will not be optimal.



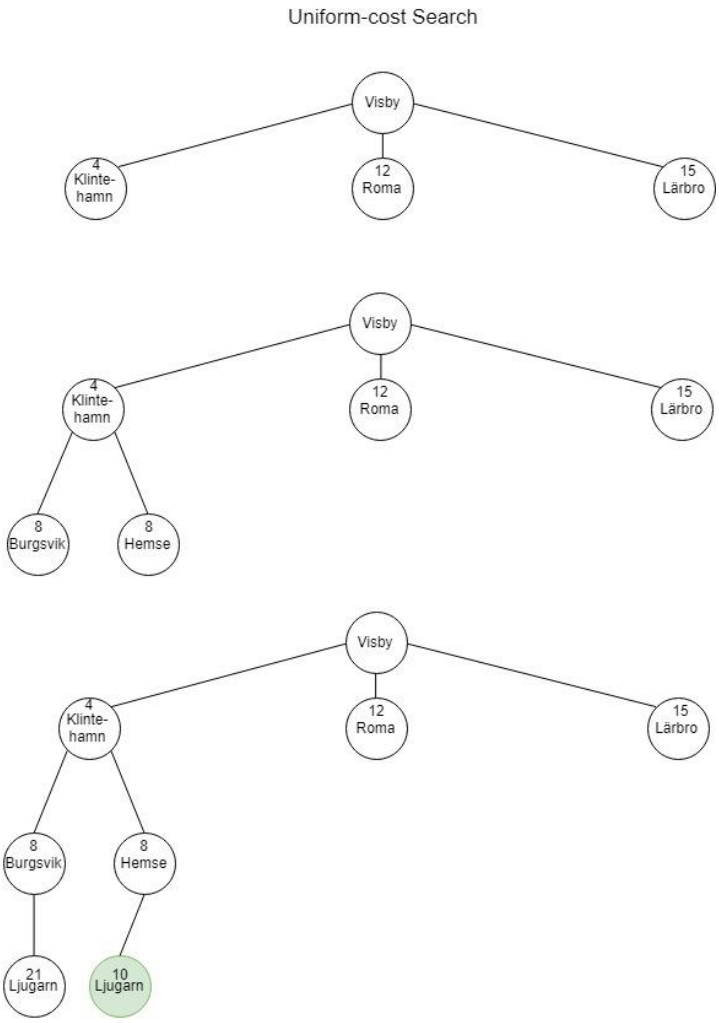
Memory for Breadth First Search

	Frontier	Explored
INIT	Visby	
POP	Visby	Visby
Add Children	Klintehamn, Roma, Lärbro	
POP	Klintehamn	Visby, Klintehamn
Add Children	Roma, Lärbro, Burgsvik, Hemse	
POP	Roma	Visby, Klintehamn, Roma
Add Children	Lärbro, Burgsvik, Hemse, Slite, Ljugarn	
Goal Node	Ljugarn	

The path will be: Visby -> Roma -> Ljugarn with a total cost of 15.

Uniform-cost Search

Uniform-cost Search takes the distance into account while choosing what path to expand. It uses a priority based queue that is based on the lowest path cost from the root node. Because the cost of the actions are not the same for all, Uniform-cost Search is a better fit for this kind of problem comparing to BFS.



Memory for Uniform-cost Search

	Frontier	Explored
INIT	Visby	
POP	Visby	Visby
Add Children	Klintehamn, Roma, Lärbro	
POP	Klintehamn	Visby, Klintehamn
Add Children	Burgsvik, Hemse, Roma, Lärbro	
POP	Burgsvik	Visby, Klintehamn, Burgsvik
Add Children	Hemse, Roma, Lärbro, Ljugarn	
POP	Hemse	Visby, Klintehamn, Burgsvik, Hemse
Add Children	Ljugarn, Roma, Lärbro, Ljugarn	
Goal Node	Ljugarn	

The path will be: Visby -> Klintehamn -> Ljugarn with a total path cost of 10. Compared to BFS, UCS is more effective in this example.

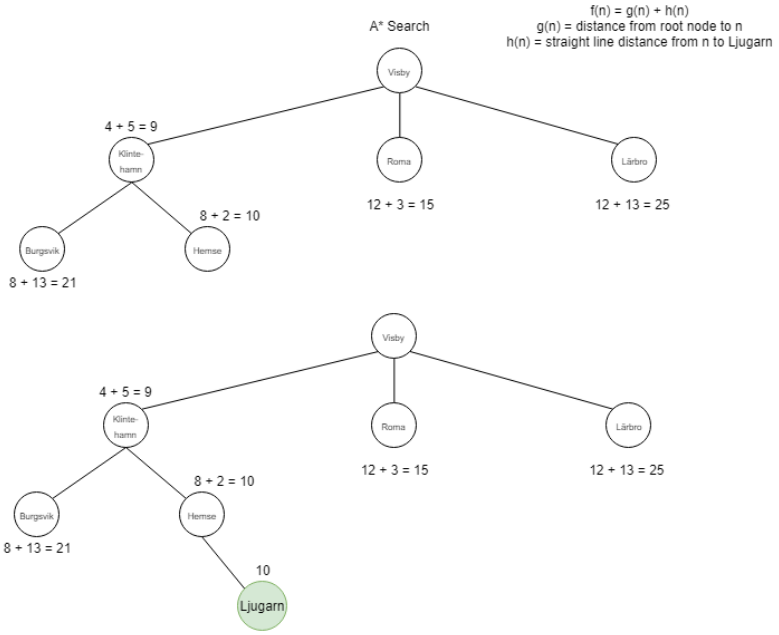
A* Search

A* Search takes the distance and an estimated distance to the goal into account while choosing what path to expand. It uses a priority-based queue that is ordered on f(n).

f(n) = g(n) + h(n) where *g(n) = the distance from root node to n, h(n) is the straight line distance from node n to goal node.*

The search will continue until a goal node has a lower f(n) than any other node in the priority queue.

We chose to use the straight line distance from node n to goal node because it is admissible and will not overestimate the cost.



Memory for A* Search

	Frontier	Explored
INIT	Visby	
POP	Visby	Visby
Add Children	Klintehamn, Roma, Lärbro	
POP	Klintehamn	Visby, Klintehamn
Add Children	Hemse, Burgsvik, Roma, Lärbro	
POP	Hemse	Visby, Klintehamn, Hemse
Add Children	Ljugarn, Burgsvik, Roma, Lärbro	
Goal Node	Ljugarn	
Goal Node		

The path will be: Visby -> Klintehamn -> Ljugarn with a total path cost of 10. A* chose the same path as Uniform-cost Search but it found the path in fewer iterations. It did not have to expand Burgsvik and therefore it is more efficient.