Linköping University

TDDC17 - Artificial Intelligence

Lab 2

Search

Author:
Janne Väisänen
janva415@student.liu.se

Assistent:
Olov Andersson
olov.a.andersson@liu.se

September 18, 2014

Linköping University
INSTITUTE OF TECHNOLOGY

Q&A

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

A: The actions are

- Go East
- Go North
- Go West
- Go South
- Suck dirt
- No op (stop)

States consists of

- The hoover position (x and y).

The branching factor is four since we at most have four successor node at any given node.

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

A: The difference between breadth-first and uniformed-cost search is basically that uniformed-cost search order is it frontier using priority-queue by path cost. Maybe one could be fooled to believe that if step cost is same for all step then the two algorithms behave identically. Not so, if we take closer look at the implementations of the algorithms we find that the goal-test is preformed at different points in code for the two. For breadth-first the goal-test will occur at the point of node generation whilst uniformed-cost search will perform goal-test when node is expanded. This makes sense since in the case of uniformed-cost search the first goal-node it comes across might be suboptimal. Hence it needs to ensure that there are no other nodes with cheaper paths. So in this case it has to add all the nodes at that depth to it's frontier(which is a priority-queue). There is an second difference explained in the book as well. If uniformed-cost search comes across two paths to same node it compare path-costs for the two nodes and only keep the node with cheapest path-cost.

Q: 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)

A: Heuristics are admissible if they never overestimates the cost to reach the goal. At least a) and c) are admissible. max(h1,h2) in c) "picks" either h1 or h2 according the description of the assignmet both are admissible so clearly c) must be admissible. Now let's consider a). Let h3 be a admissable heuristic such that it estimates cost to reach goal as actuall cost of reaching the goal from current node,at any point in time (this would be the worst case scenario for such a function). then

$$h3 = (h3 + h3)/2 = (2 * h3)/2 = h3 \leq 2 * max(h1, h2) \leq (h1 + h2)/2 \tag{1}$$

What i'm trying express above is that (h1+h2)/2 never can be greater than the worst case, that is heuristics estimating cost to goal as actuall cost to goal.As goes for b) we can't be sure whether it is admissible or not. It might be that h1 estimates the cost to be less then half the cost of actual one. As side note A* requires consistency to be optimal(maybe handy to know, come exam-time:).

Q: 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?

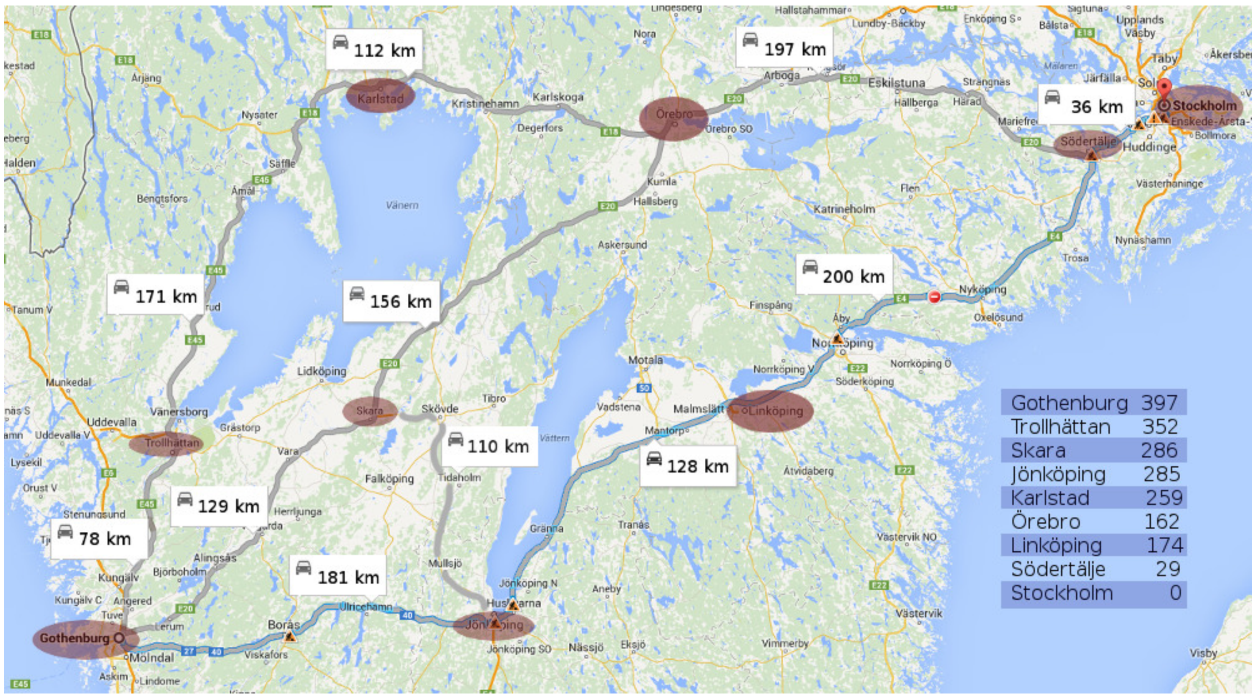
A: The evaluation function for A* looks as follows

$$f(n) = h(n) + g(n) \tag{2}$$

Where $h(n)$ is the heuristic function, a cost estimate to reach goal from state at node n, and $g(n)$ is the cost to reach n from start node. The formulas says n for node but we actually only depend on states of n. $h(n)$ could be the manhattan path from state at node n and $g(n)$ then shall be accumellated acuall cost from start node to current.The relaxation of the problem here would be that we ignore the fact that in some cases we need to pass around obsticles to get to destinited goal. I believe this is concistent as well. ($h1(n) \leq h(n\prime) + c(n, a, n\prime)$ $n\prime$ is successor node , c cost-function getting from n to $n\prime$ given any valid action a).

Q: 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.

A: I'm gone borrow the problem with finding a route from the course-book. So the available action would be Go(target-city) where target-city is a city that has a road from current city to target city and is the first city from current city to target city on that road. So for instance if we are currently in Gothenburg action go is applicable for Trollhättan, Skara and Jönköping. .Agent states are defined by In(city) which just say agent is currently located in city. The initial state is In(Gothenburg) and the goal state is In(Stockholm). The result of doing Go(target-city) when we are in state In(current-city) is simply that agent is moved to current-city (this would be the transition model). Path costs are given the total sum of step-costs on given route and each step-cost is the distance by car between two adjacent cities. The map in figure 1 describes possible routes between cities. Only the cities mark red are considered in this assignment (zoom in for a better view about 150% worked for me).



Gothenburg	397
Trollhättan	352
Skara	286
Jönköping	285
Karlstad	259
Örebro	162
Linköping	174
Södertälje	29
Stockholm	0

Figure 1: map and table used for A*

The first algorithm will be A*. I'm using the map above. The lower right corner table consists of Straight line distances from each city to Stockholm in km:s. A* uses a priority queue to order it's fringe.The priority is calculated using the simple formula.

$$f(n) = g(n) + h(n) \tag{3}$$

Given a node n $f(n)$ is an estimate of total cost of path to reach goal. So the lower $f(n)$ the higher priority the node gets. $g(n)$ is actual cost of getting from start node n. h(n) is heuristic function that estimates cost of cheapest path to goal. h(n) shall never overestimate such costs. g(n) is here calculated by adding distances between visited cities on the map. So for instance g(n) of reaching Karlstad by Trollhättan would cost 78 + 171= 249. h(n) will be the staight line distances from table in lower right corner. So f(n) in this case would be 249 + 259= 508. So here's the search-tree. We also keep track of those states that have already been explored.

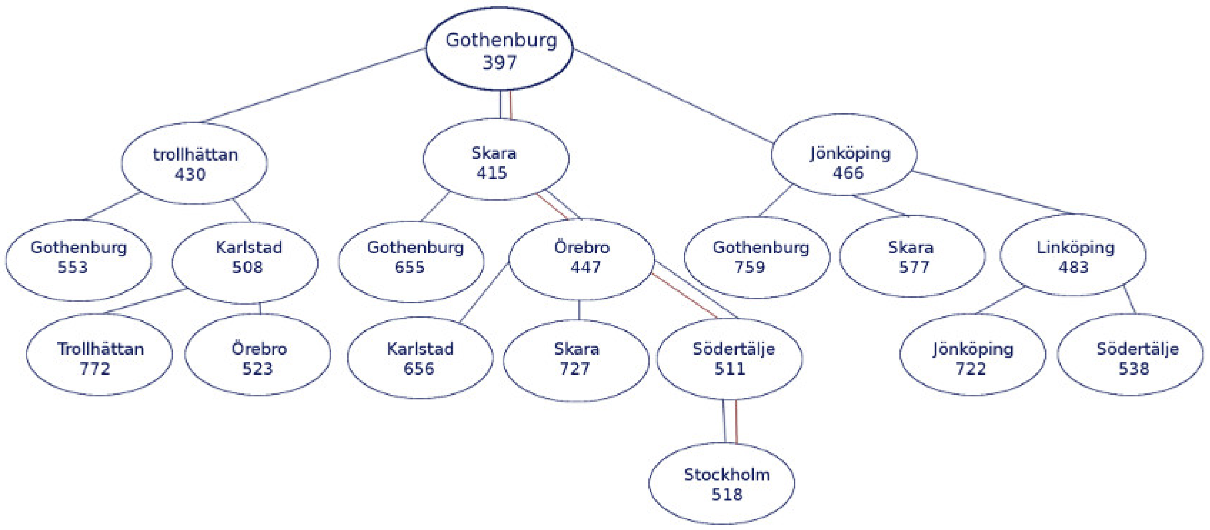


Figure 2: A* search tree

The red line just indicates the chosen solution which is the optimal one. The numbers indicate f(n) in given node, that is the estimated cost to goal-node from start-node going through given node.

The second algorithm i've chosen is **depth-first-search**. I'm attempting to illustrate the search-tree in image bellow. The blue straight lines are those leading to nodes which are explored and visited. Each node keeps track of which action lead to this node. A corresponding state of the agent is also known at each node. The red lines illustrates that in this case Trollhättan and Skara are kept in the datastructure representing the fringe, In this case a LIFO. We also keep track of where we have already visited in a datastructure which we will call explored (contains the states).

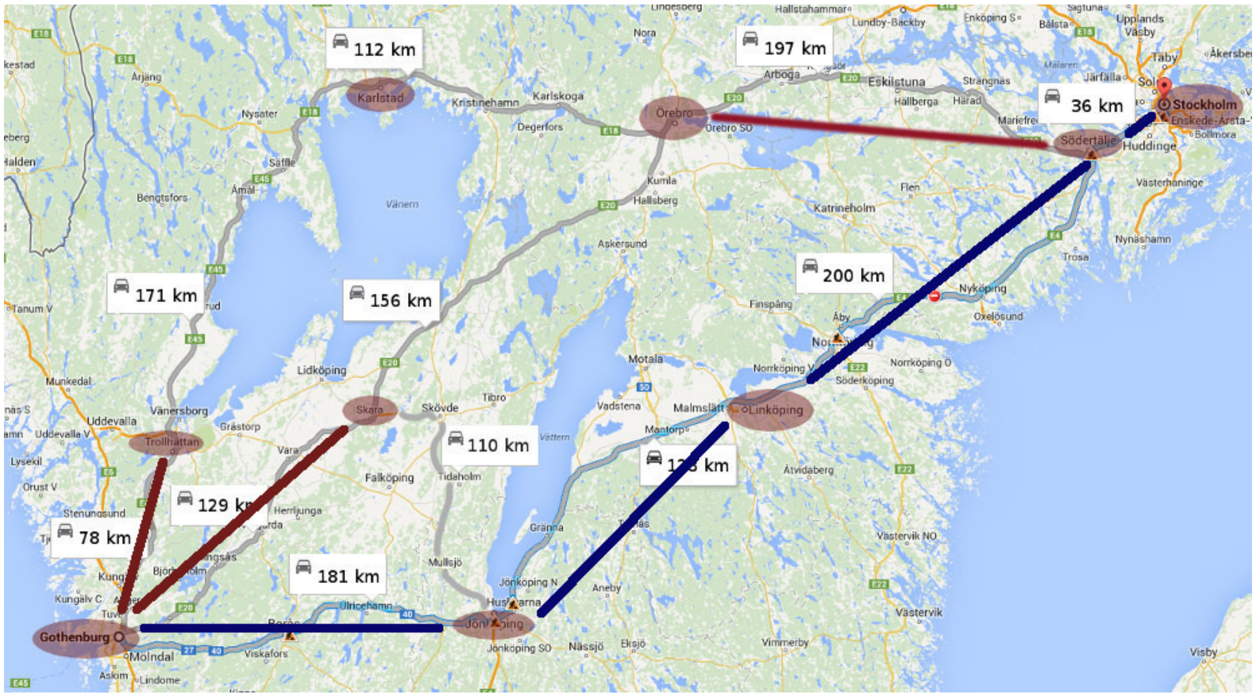


Figure 3: Illustration of depth-first-search

Image shows a possible tree for depth-first-search. The interpretation i make of the tree is the following. We have reached the goal In(Stockholm), nodes corresponding to Trollhättan and Skara are on the fringe and that states in(Gothenburg), in(Jönköping), in(Linköping), in(Södertälje) have been explored and put on the explored list.

The last algorithm is **breadth-first-search** it works in a similar fashion as depth-first-search except it uses a FIFO-queue as fringe hence the order nodes are explored is different. Namely all nodes at a certain depth are explored before deeper nodes are considered. The image illustrates this by using different colors for each depth of the tree (note the fringe keeps nodes crossing these boundaries).

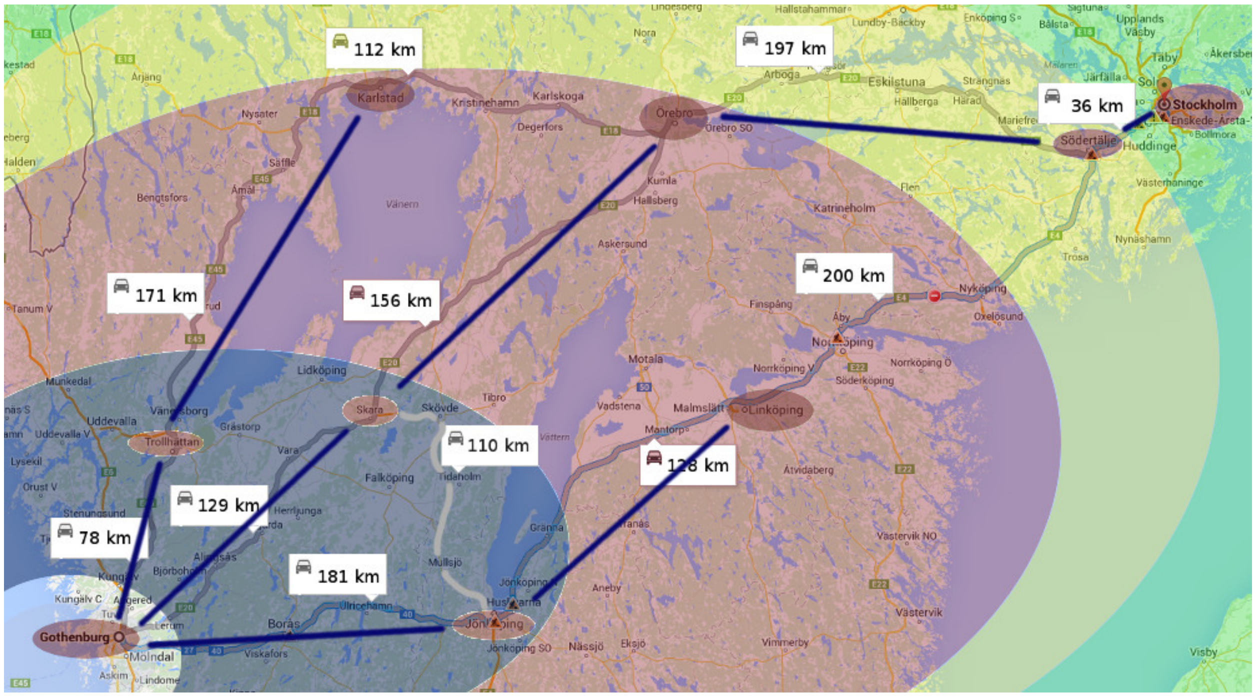


Figure 4: illustration of breadth-first-search

Blue lines in the picture depicts a search-tree with its root at Gothenburg. Again as with depth-first-search visited nodes are kept on explored data-structure. Whenever we visit a node we create child-nodes with states corresponding to such that can be reached by applying an action from current state and put them in the back of the fringe (a FIFO queue). The search progresses by dequeuing the fringe from the front generating new child nodes as long as goal hasn't been reached.

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

A: **Breadth-first** -is complete if the branching factor is finite. If there exist a goal node it will be found. Because if the shallowest goal node is at some finite depth it will found after expanding all other nodes at shallower depths. That's so, because we require branching factor to be finite.

It is optimal if path cost is a non-decreasing function of the depth of the node. Because breadth-first expand shallowest nodes first, the first goal-node breadth-first comes across will be the one at the shallowest depth. Since we require that the that path-cost should be non-decreasing all goal nodes at a deeper depth have greater path-costs.

Uniformed-cost -Is complete if step-cost $\varepsilon > 0$ and branching factor b $b < \infty$. Then if there is a goal node it has a finite path-cost. Since expanding a node implies increasing path-cost it also implies we will reach a goal-node in finite number of node-expansions. If there $\varepsilon = 0$ was allowed it could get stuck in infinite loop since expanding such a path would not mean increasing path-cost. Then if there exist a loop whit path-cost 0 it would always stay on such a path.

-It's optimal when $stepcost \geq \varepsilon$ for some $\varepsilon > 0$. Uniform-cost-search has a function $g(n)$ which calculates path cost to node n. Let $n\prime$ be the a parent of n, $p(n\prime)$ path cost from initial node to $n\prime$ and $s(n\prime, n)$ the step cost from $n\prime$ to n. Then $g(n) = p(n\prime) + s(n\prime, n)$.The frontier is order in such away that cheapest nodes comes (lowest $g(n)$) comes first. Ordering is accomplished using a priority queue. So at any given time, the node which has the lowest path cost so far is in the front of the queue.Hence when a node n is selected the path to that node is optimal. (This holds because of the graph separation property). Since $s(n\prime, n)$ for any $n\prime, n$ it follows that first goal node has to be optimal.

So, whenever a node n is selected for expansion the path to n is optimal. This follows from the graph-separation property(frontier separates the state-space graph into an explored and an unexplored regions every path from initial-state to unexplored state has to pass through a state in the frontier). Whenever we add nodes to the path the cost will increase since step cost is non-negative. Hence the frontier consists of the optimal an optimal path always.

if the step-cost 0 is allowed it could get stuck in an infinite loop.

Depth-first The easiest answer is that it's neither complete or optimal. Not complete since algorithm always strives to expand deepest node in a given direction first. If such a branch turns out to be infinite(unbound) and lacking goal nodes then algorithm will never terminate.

As for optimality, since the algorithm always strive towards deepest node in certain branch before considering other branches it might come across a suboptimal node with goal state.For instance in the graph bellow if C and G are goal state consisting nodes and the algorithm always prefer right branch (looking from A downwards)before left branch, then c would be the chosen goal.

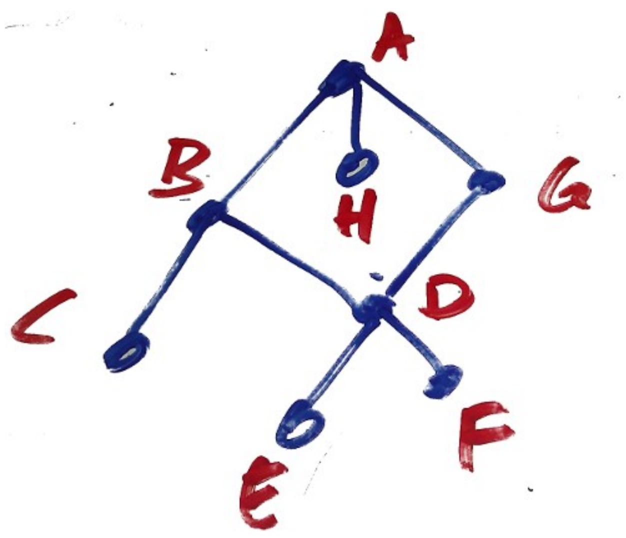


Figure 5: search graph

For graph-search version there are ways to avoid loop paths in finite state-space. In such case it will eventually visit all nodes. In infinite state space both tree and graph search will fail because if it's ”unlucky” it will choose an infinitely deep branch with no goal.

Depth-limited search Is neither optimal or complete Depth limited works the same way as a depth-first search, down to a certain cut-off depth. If there exist goal node bellow the cut-off it will never be found. So for instance if A is initial node at depth zero,the goal node is E and the cut off is two the depth-limited would fail. The upside is that by limiting the depth we can limit the search-space and avoid infinitely depth non-goal-paths and loops. otherwise it pretty much has the same properties as depth-first search.

iterative deepening depth-first Is complete and optimal under same restrictions as breadth-first. basically depth-limited runs depth-first search over and over with increasing cut-off depth.This means all nodes n at certain depth d are considered before any nodes $n\prime$ at $d+1$. So for the same reason as breadth-first is optimal, this is optimal and complete.That is goal first goal-node found is a goal node at the shallowest depth.

Bidirectional search is complete and optimal and complete if uses breadth-first in both directions and under the same assumption as for breadth-first.

A* search is optimal, complete and optimally efficient if the heuristic function is admissible for tree search and consistent for graph-search version.

A* Tree search A* uses the function $f(n) = g(n) + h(n)$ where $g(n)$ is actual path-cost to reach node n and $h(n)$ is an estimated cost from n to goal. The book describes it as "cost of cheapest solution through n". admissible heuristics, simply put means never overestimating the cost of reaching the goal. So if $h(n)$ is admissible then $f(n)$ also gives a path-cost which is lower then the actual cost to reach the goal from initial state. Now if the cost of the optimal solution is called C^* and we have a suboptimal goal $g2$ on the frontier we know that $h(g2) = 0$ as for all goal nodes. We get

$$f(g2) = g(g2) + h(g2) = g(g2) > C^* \tag{4}$$

That is since cost of h(g2) = 0 (reaching goal from goal node) then actual full path cost is simply g(g2). We stated that g2 is suboptimal hence this cost must be greater then C^* because that's the cost of the cheapest path.

Consider the frontier node n that is on optimal solution path. If $h(n)$ is admissible, that is it doesn't overestimate the path-cost from n to goal nodes. We get

$$f(n) = g(n) + h(n) = g(n) \leq C^* \tag{5}$$

furthermore

$$f(n) \leq C^* < f(g2) \tag{6}$$

What this is saying is that g2 never gets expanded. So from $(1),(2)and(3)$ we can conclude A* is optimal for tree search. The first goal node that gets expanded is the cheapest one.

A* Graph search As stated above is optimal if $h(n)$ is consistent. Consistency of heuristic function can be expressed as follows.

$$h(n) \leq c(n, a, n\prime) + h(n\prime) \tag{7}$$

Where $c(n, a, n\prime)$ expresses the cost from any node n to any other node $n\prime$ which we can reach by taking some action a. $h(n\prime)$ is estimated cost from $n\prime$ to goal node. Finally $h(n)$ is as before the estimated cost from node n to some goal node. So simply put this is saying for every step we take towards goal node the estimated cost should at least not decrease (should be as expensive or more expensive). Maybe one way of viewing this is that cost estimates get more accurate as we move towards goal. From this we can conclude that if $h(n)$ is consistent then cost function f(n):s estimates when moving towards goal will increase. Formally this can be expressed as.

$$f(n\prime) = g(n\prime) + h(n\prime) = g(n\prime) + c(n, a, n\prime) + h(n\prime) \geq g(n) + h(n) = f(n) \tag{8}$$

So if $h(n)$ is consistent then cost of $f(n)$ along path to goal are. We take advantage of this fact in next part of the proof. Now we want to convince our selves that if A* select a node n to be expanded then the optimal path to that node has been found. If this path was not optimal then there would exist a node $n\prime$ such that it would have been on frontier(follows from graph separation see course book p78). Further more because we require consistency (which implies non-decreasing f-costs along any path) the f-cost to $n\prime$ would have been cheaper and hence been selected for expansion first but this was not the case. Hence the first goal-node to be selected for expansion has be optimal.

complete If C* is cost of optimal path to goal node n. Then A* is complete give the the condition that number of nodes such that $f(n) \leq C*$. This condition holds if all step cost are ε and branching factor is finite [1 p97]. TODO fyll på inte färdig

Q: Assume that you had to go back and do lab 1 once more, but this time with obstacles. Remember that the agent did not have perfect knowledge of the environment but had to explore it incrementally. Could you still use the search algorithms you have learned to guide the agent's execution? What would you search for? Give an example

A: Hmm.. Since we have no prior knowledge of he world we can only rely on events in the past. One way could solve it would be to do an exhaustive search of the whole world using some complete search algorithm. This could have been achieved for instance using some variation of depth first search (DFS is not complete in it simplest form). This should be possible even thou we don't know the size of the world we could probably be shore it's not infinite.