University of Waterloo
CS 486
r2knowle: 2023-10-1

Assignment # 1

Q1a) We will expand the nodes in the following order:

$$s- > h- > k- > c- > a- > b- > d- > m- > e- > n- > g$$

Thus arriving from s to g.

Q1bi) This heuristic is admissible, to prove that this is the case we will use contradiction. Assume that the heuristic is not admissible this means that for some node n:

$$h(n) > h^*(n)$$

where $h^*(n)$ is the true cost of node n. This means that the true cost of n is actually shorter with a barrier, and since a barrier only every reduces the path size this is impossible. Therefore the heuristic is admissible.

Q1bi) Using depth first search with this heuristic gives us the nodes expanded in the following order:

$$s- > h- > k- > c- > a- > b- > d- > m- > g$$

Which gets us from s to g.

Q1bii) Using A^* gives us the nodes expanded in the following order:

$$s- > h- > k- > f- > p- > q- > a- > r- > t- > g$$

Which gets us from s to g.

Q2a) Yes, in the event that all the paths have an equal cost. To prove this lets use contradiction, assume that breath width search does not expand values in the same state as uniform-cost search does. That would imply that the node n's ancestors would be different in uniform search then breath first search. However if the ancestors are different then that would imply that breath first search is on an unoptimised path which is impossible as breath first search minimizes the number of nodes visited and since the path cost is constant this should minimize cost. Therefore because we have a contradiction breath width search is a special case of uniform-cost search.

Q2b) No, as there is no heuristic which guarantees us to always expand the child node (branch) consider the example in 1, we cant define a heuristic such that we will only expand the child node.

Q2c) Yes in the event that the heuristic function is 0, A* will preform exactly the same as uniform cost. This is because the cost of each node in A* is its uniform search cost plus the heuristic and therefore if the heuristic is zero they will be the same.

Q3a) We can represent the TSP by the following problem representation:

States = any combination of citites, where no city is repeated twice.

Representation = we can represent each state by the letter of the city in the order they are visited.

Inital State = the inital state is always city A.

Goal State = The goal state is when all the cities are visited, cost minimized.

Sucessor Function = The sucessor function is to move to a new state, where a new city letter is added to the end of our current state i.e we visit a new city.

Cost Function = The cost between any state is the minimum eucliean distance between all the cities they have in common and the one city they have different. The total cost of any state is the total eucledian distance between all the neighbouring cities in the representation.

Q3b) We can have 3 of the following heuristics:

h1(n): For this heuristic the cost for any city is equal to the minimum distance from this city to any other city that we have. This is admissible as the true cost of any node will always be greater then or equal to the minimum distance, if the true cost is less than it would imply that there is a new minimum path and is thus a contradiction.

h2(n): For this heuristic the cost for any city is equal to 0. This is always admissible as the actual cost of adding a city must be greater then 0. This heuristic will turn our A* algorithm into a uniform search algorithm.

h3(n): the cost for this city is equal to the distance from the current city. This ensures that we will always pick the city that is cheapest from our current node to the next node. This is admissible as there is no path that is cheaper between our current node and the next node as if there was we would use this node instead.

We should expect that h2 will get dominated by both h3 and h1, as h2 will not get rid of any states. For h1 we expect that it should dominate h3 as the number of states we need to expand should be less as there are far less global minimums (i.e minimum distances in the entire graph) then local minimums (i.e distance between the current node and the next node).