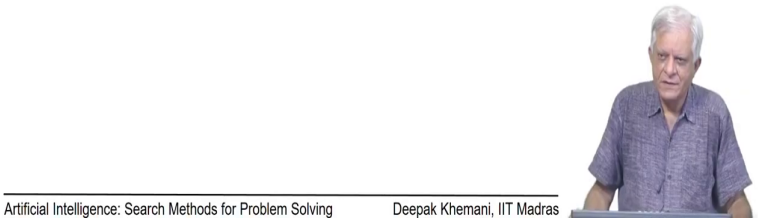
Artificial Intelligence: Search Methods for Problem Solving
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

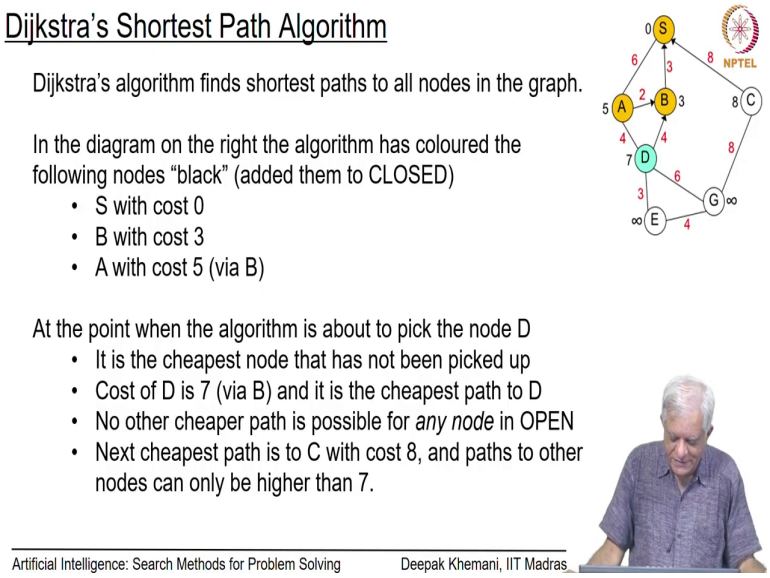
Chapter – 05
A First Course in Artificial Intelligence
Lecture – 38
Finding Optimal Paths: Is A* Admissible?

(Refer Slide Time: 00:14)



So we have been looking at this algorithm A star, which I said is a major algorithm in the area of search and we have seen an example of how it works. Now, let us prove that it is an algorithm which is Admissible, which means that if there is a path to the goal then it will always find an optimal paths to the goal.

(Refer Slide Time: 00:47)



So, let us just do a quick recap of what Dijkstra's algorithm does remember that Dijkstra's algorithm only looks at what we have called as G values and it always picks the node from OPEN which has the lowest G value. So, in the graph which is from the example that we had studied earlier, you can see that it is about to pick the node D and it has already seen nodes S A and B not in that order, but in the order S B and A.

Now, we had said that Dijkstra's algorithm finds the shortest path we have not given a formal proof of it. But let us go through a slightly in less formal argument about this. So, if you look at this situation here at this stage of the algorithm has colored the following nodes black, when you say black that is because the algorithm uses the word black in our diagrams we have colored them orange. In other words they have been added to the closed list essentially.

First it added node S with cost 0, it always starts from the start state then it added node B with cost 3 because that was the cheapest node on open at that point, then it added A with cost 5. Now, you would remember that A has a direct edge of cost 6, but Dijkstra's algorithm found a path which is shorter than that which is of cost 5. So, this is a property of Dijkstra's algorithm that whenever it picks a node it always picks a node by a finding an optimal path to that node.

So, at the point when this algorithm is about to pick D you can see that the following is true that it is the cheapest node that has not been picked up. So, of the nodes that we have not yet inspected there are 4 nodes which are D C E and G, E and G still have an associate cost of infinity which means we have not even found a paths to them, so there is no question of picking them up.

In the terminology that we have been using in our algorithms E and G have not yet been generated essentially both D and C have been generated, but D we have found a cost path of cost 7 and D we have found a path of cost 8 which is just an edge from S to C.

So, between D and C you can see that D is the cheapest node that has not yet been picked up; I mean you can even count nodes E and G but their costs are infinity. Also the cost to the node D is the cheapest path to D. Now, it is figured out that of all the different possible paths to D, what are those paths those paths are the one that it we have identified by the back pointers that is S to B and B to D.

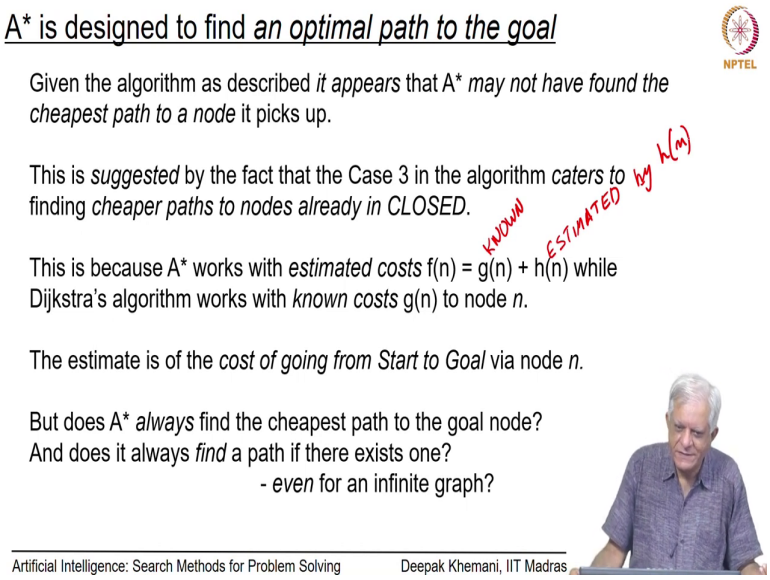
But there are other paths possible for example you can go from S to A and then A to D that would cost 10 units or you can go from S to B and B to A and A to D that would cost 9 units and there are various other combinations possible you can even go back to S and then come back as branch and bound might have explode.

But at the point when Dijkstra's algorithm picks up this node D it is the cheapest and there is no other cheaper node available at all, because the algorithm says that pick the cheapest node

and color it black. So, there is no other node at all and this is the cheapest node and the paths that we have found to it is also the best path essentially.

The next cheapest node would be of cost 8 which is C and all other paths of to all other nodes would have been more expensive. So, at this point in Dijkstra's algorithm the cheapest partial path is that you can see is to node D with value C. And when it picks it up there are no other possible cheaper paths available to D, because if there were any cheaper paths they would have shown up somewhere during search essentially.

(Refer Slide Time: 05:11)



Now, A star unlike Dijkstra's algorithm A star is designed to find an optimal path to the goal node. Dijkstra's algorithm simply says that find shortest paths to all nodes in the graph A star is more to do with a I related search and which means it is solving a particular problem of

reaching a particular goal state or a goal description which may have 2 or 3 or 4 goal states associated with them, it is not designed to find shortest paths to all nodes.

So, the question we want to ask is this our eventual question is that does it find the shortest path to the goal. But an intermediate question we can or an observation we can make is that as we have described the algorithm it appears that A star may not have found the cheapest path to a node that it picks up from the OPEN.

So if you remember the algorithm, algorithm says compute the F values of all the nodes in open or all the nodes that you are adding to OPEN, create a priority queue based on those F values pick the node with the lowest F value; remember that F value is the sum of G value plus H value and check whether it is a goal or not. If it is not the goal then add it to CLOSED and generate its neighbors and see if we have found cheaper paths to those neighbors.

Now, if you remember there were 3 kinds of cases 3 kinds of neighbors that we dealt with, case 1 was when the neighbors were new nodes that had not been generated earlier we should correspond to a cost of infinity G value of infinity to them you simply update the G values by adding the H cost from the node that you have just expanded and add them to OPEN.

The second case was that you have found a new path to a node which is already on OPEN, but like Dijkstra's algorithm you may find a better path to a node that is already on open. So, if you go back to the Dijkstra's algorithm you can see that first we had found a direct path from S to A which is of cost 6, but then at the moment when we picked up A we had found a shorter path which is of cost 5.

So, in a similar manner algorithm A star also can find shorter paths to paths on open that was the case 2 that we studied. And essentially what we need to do is to update the parent pointer for that new that that node that neighbor and move on essentially, because we already updated the cost.

The case 3 which is of interest to us at this moment was the case when you found the shorter paths to a node enclosed. Now, this is something which was not possible in Dijkstra's

algorithm and in the and we just now kind of argued in a sense that whenever it picks a node it always has found the shortest path to that node.

That is not the case in A star algorithm, because we have seen that the algorithm incorporates the case 3 and that case 3 caters to the fact that you might have found the cheaper paths to nodes already in closed. So, this is where A star is different from Dijkstra's algorithm.

Why does this happen? This is because A star works with estimated costs of f of n remember f of n is the estimated cost from start node to gold node and a path which passes through the node n and therefore f n was estimated as the sum of g of n plus h of n.

Where g of n was known and h of n was estimated, it was estimated by h of n the heuristic function. So, it was an estimated cost so distance to the goal essentially. The Dijkstra's algorithm always works always worked with known cost g of n and therefore it knew that when it had picked reached the particular node that the estimate the known costs was always the best cost that was available.

But since a estimated cost f of n for A star depends so much upon both the values g of n and h of n. And h of n is only an estimate you cannot be sure that f of n at the spot that you have picked that node n at that point g of n is the lowest value for that node n essentially that is.

Because the algorithm picks nodes not based on value of g of n that is what branch and bound or Dijkstra's would have done. But based on values f of n which incorporates the heuristic distance h of n which is not necessarily perfect way of estimating the cost; it is just the function that we have used to try and speed up the algorithm

So, remember that when Dijkstra's algorithm picks node n it looks at the cost from source node to that node n. Whereas, when A star picks a node n it is picking an estimate which is the cost from start to goal and the algorithm and the path is via the node n essentially.

And we have seen that is the case and that is why algorithm has the case three, because you may find newer paths to nodes on closed and you may then have to propagate those cost that was part of the algorithm that we have seen essentially.

Later on as we move along we will see that under certain conditions A star can behave like Dijkstra's algorithm. Which means that if it has picked a path to a node n and it is about to pick node n from the open list it would have found it would have found an optimal paths to that node n.

But that will come under conditions that we will state later at the moment those conditions are not true and that is why the algorithm A star as stated generally incorporates the case 3, where we have to update costs of nodes which are already in closed and maybe even propagate them.

The question that we are really interested in because A star is designed to find the paths to a goal not necessarily to all nodes in the graph, that does A star always find a cheapest part to the goal node that is a question we are going to answer next. And the other related question is that does it always find a path if there exists one. What if the graph is infinite? If the graph is infinite, but there is a path to the goal node does it guarantee that it will find the paths to the goal node.

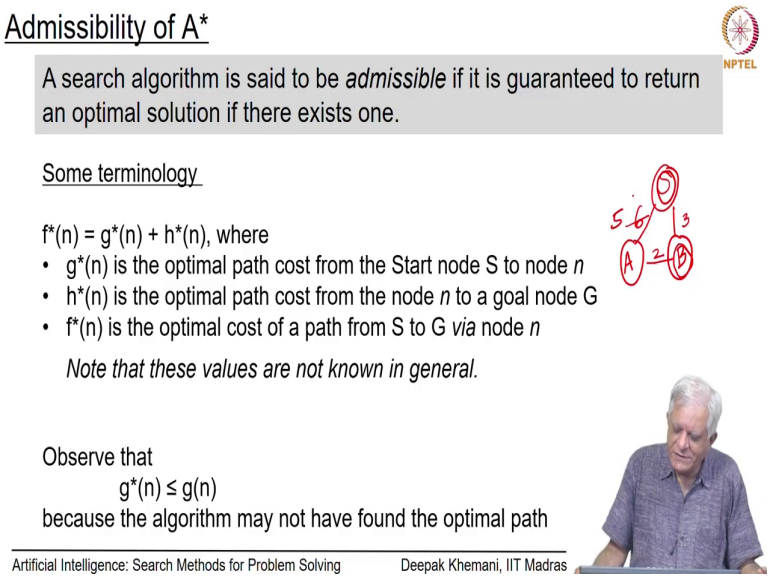
So, if you remember the discussion we had when we started with search which is with blind algorithms like depth first search and breadth. First search we had observed that depth first search can get caught in an infinite loop, if it goes down an infinite path if the search space is infinite.

Whereas, breadth first search which waits through the search page level by level, would have found a paths to the goal if there existed one even if the rest of the graph was infinite.

So, the question we asking here is that is A star does it display a similar behavior, that if there is a paths to the goal even for an infinite graph, does it find that path to the goal? And the

answer clearly is yes as we will show shortly not only it finds a paths to the goal it always finds the optimal paths to the goal, but there are some as they say conditions attached.

(Refer Slide Time: 13:23)



So, we say that a search algorithm is said to be admissible, if it is guaranteed to return an optimal solution if there exists one. So, only condition we are stating here about the graph apart from some more conditions that we will say it under which it will be admissible. But by admissible we mean that if there is a paths to the goal then does the algorithm return the optimal paths to the goal.

So, we need some terminology for doing this analysis and for doing the formal proof and we use the terms f star of n, g star of n and h star of n. And as before f star of n is a sum of g star of n plus h star of n. Now, when you put star on these functions then we are kind of implicitly saying that they are optimal in nature.

So the star whenever we say A star we mean that the algorithm is will find the optimal paths, later on we will study an algorithm called A star and that would mean that it would find an optimal solution.

In this case we are using this functions f star of n g star of n and h star of n, g star of n is a optimal path cost from start node to the node n it is a property of that node n and it is a optimal paths that comes from start to n. This is the value that Dijkstra's algorithm discovers at the point when it picks that node n. h star of n is a optimal path covers from node n to goal node G.

Now, this is clearly not known to us essentially, you know because we are just estimating the heuristic distance by some mechanism. For example, difference between coordinate values and things like that or number of blocks in place as we saw in the case of 8 puzzle or rubics cube and so on essentially.

So, these are just some estimates which are used to estimate some heuristics which are used to estimates distances to the goal and they are by no means accurate. So, we do not know what is the actual cost to the goal. If we knew the value of h star n, then you would notice that the algorithm does not really need to search too much it will kind of directly home into the path which goes takes you to the goal.

But we do not have such perfect heuristic functions and that is why we are try we have forced to impose certain other criteria on heuristic functions to make them sure that to make it sure that the algorithm becomes admissible and we will see what these conditions are shortly.

So, corresponding to g star of n and h star of n is the value of f star of n which is the optimal cost of a path from start to goal. Remember that f star the estimates that it uses the A star algorithm that the estimates that it uses which is f of n are always from start to goal and when you say f of n we basically mean a path from start to goal, but which passes by a n. And so in that manner f star is the optimal costs of a path from start to the goal which goes via n essentially.

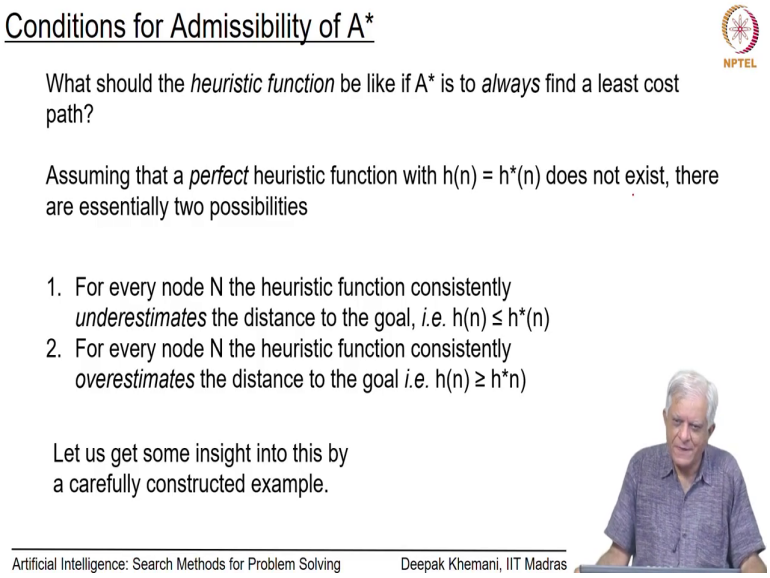
And as I have been saying here note that these values are not known in general, we rarely know any of these values sometimes we will see under certain conditions we will get to know g of g star of n, but not but definitely not h star of n and therefore definitely not f star of n.

You can also observe that in general g star of n will always be less than g of n and the reason for that is that because the algorithm at that point may not have found the optimal path to that node A to the to the node n sorry. When I was saying a I meant with the example that we were considering with Dijkstra's algorithm that initially if you remember we had S and then we had h cost of 6 to A and then we had an S cost of 3 to B.

And we so at this point when algorithm is looking at node A it estimates g of n g of a to be six, but later on when it expands node B also after expanding node A it found a new paths to A and this value went down from 6 to 5.

So, in general at the point where node n is added to open its g value may be greater than or equal to G star of n, because it may not have found the optimal path here essentially. So, this is a property we will use when we are arguing about the proof essentially.

(Refer Slide Time: 18:23)



Now, what are the conditions for admissibility of A star, when you say conditions we basically mean on the problem statement and in particular on the heuristic function.

So, what should be the heuristic function be like, if A star is to always find the least cost path and when we say this we mean that it always finds a least cost path to the goal node we are not yet concerned with intermediate nodes.

So, assuming that the perfect heuristic function does not exist; if you have the perfect heuristic function which the people in theory of computing would call an oracle which always knew what is the right direction to go to then, of course we would not have any problems to solve and we would not be discussing all this algorithm.

But we do not have a perfect heuristic function where h of n equal to h star of n, which means that there are only 2 possibilities that either the possibility one is that for every node n the heuristic function consistently underestimates a distance to the goal node; which means h of n is always less than or equal to h star of n.

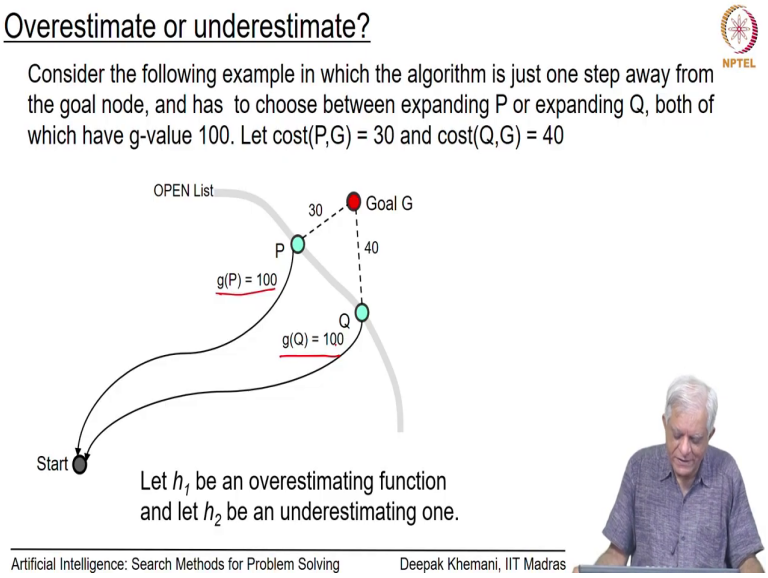
So, equality has been incorporated here we can hope to be equal to h star of n. But the property that we want to investigate is that is it better to underestimate a heuristic use of heuristic function which underestimates the distance to the goal or should you use a heuristic function which consistently over estimates the distance to the goal. So, we say that the heuristic function consistently over estimates the distance to the goal which is case 2 here, if h of n is always larger than or equal to h star of n.

So, the question that I would like to pose to you and perhaps you can pause the video a little bit ah, think about this come up with an answer and then resume the video. So, that you can verify what your intuition says and whether it matches what is really in the case essentially ok. So, at this point you should take a small break think about this question and come back with that answer.

What we will do now that you are back is to get some insight into this by looking at a carefully constructed example. We want to simply first get a feel of whether it is better to underestimate the distance to the goal or whether it is better to overestimates the distance to the goal. So, I hope you have given some thought to this question and you have some idea of what it should be.

Now, let us look at this very small example designed to get some insight into this question and get an answer to this question though a formal proof will follow later.

(Refer Slide Time: 21:21)



So, should we underestimate the distance to the goal or should we overestimate the distance to the goal that is a question that we are asking at this moment

So, this is a small example in which for the sake of simplicity we assume that there are 2 nodes on open P and Q that we have to choose between. And again for the sake of simplicity we have assumed that their G values are identical, because we want to see what is the influence of whether you the heuristic function over estimates or whether the heuristic function under estimates of the distance to the goal.

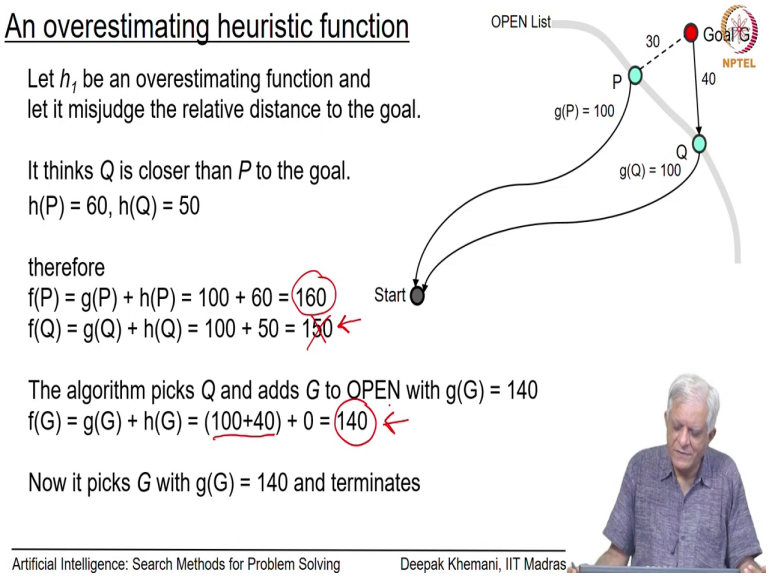
To investigate this answer we will also assume that both these nodes P and Q are just one step away from the goal node, which means there is an edge connecting P to the goal node and

there is an edge connecting Q to the goal node and as depicted here the cost of these 2 edges are 30 and 40 respectively.

So, you can see that the shortest path to goal is by a P, because the total costs of that path would be 100 which is the cost of reaching up to P and plus30 which is the cost of going from P to G. The other path which goes via Q has a cost of 140 100 plus 40 as shown here.

So, let us try out 2 different heuristic functions one heuristic function we will underestimate the distance to the goal and the other heuristic function will overestimate to the distance to the goal. So, let h 1 be an overestimating function and let h 2 be an underestimating function. Let us see how search would proceed from here essentially

(Refer Slide Time: 23:11)



So, let us first look at the over estimating heuristic function. What do we have? We have a function heuristic function h 1. So, h 1 of n for any node n and it is an over estimate function and just to drive home our argument let it misjudge the relative distance to the goal.

Now, we can see from the graph that P is closer to the goal it is an actual cost of 30 and Q is a little bit farther it has an actual cost of 40, but we have now overestimating function. So, this function thinks it is more than 30 and 40 as the case may be. But it also misjudges which one is closer.

So, this heuristic function thinks that Q is closer than P to the goal, it assumes that h of P is 60, whereas actually it is 30 the actual distance h star of P is 30. But h of P is 60 likewise h star of Q is 40 which is the actual cost from Q to G, but h of Q is 50.

So, both 50 and 60 are over estimates, but on top of overestimating the distances it also thinks that Q is closer to the goal so obviously it will explore the path which comes via the node Q. So, it will compute the f values and you can see that f value of P is the G value of P plus the h value of P which is g value is 100 h value is 60. As we have just mentioned h likewise the f value of Q is 100 plus 50 which is 150.

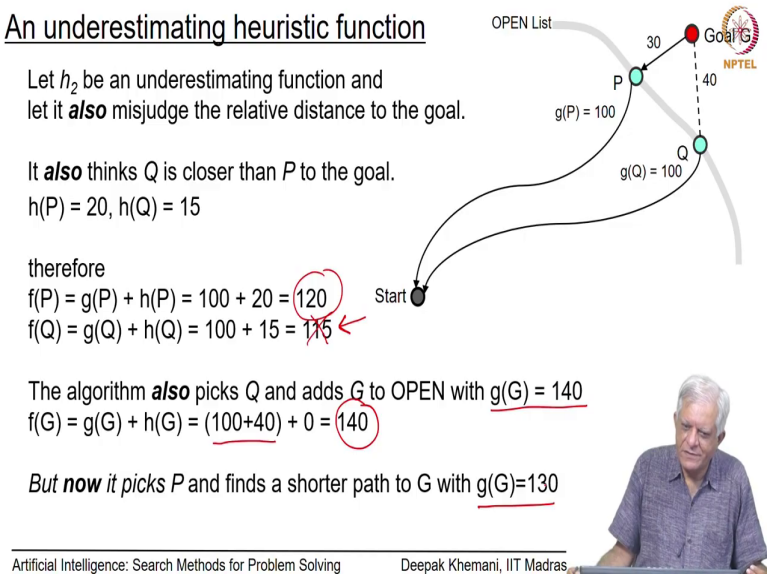
So, clearly between these 2 nodes this one is the better node and A star should pick this node and expand it put it into close generate it is neighbors. So, let us assume for simplicity that the only neighbor is g, so it will add the goal node G to the open list. That is what it does the algorithm picks Q adds G to the open with a g value of 140; remember the G value is the actual value of the path it is found to the goal node.

What is this path that what is the path that we have found? We have path of found going from start through some nodes up to Q and then from Q to G. So, the f value of the goal node which is what we are interested in as far as the A star algorithm is concerned is again the sum of g values and h values.

The g value as we have just observed is 100 plus 40, because that is the actual value of the path bound to the goal, h value is 0 because you are already at the goal and we have said that by definition the estimated distance is 0.

So, the f value of G is 140, so remember that 140 has always already gone from open. So, we are left with 2 nodes 160 and 140 and between them 140 is cheaper. So now, algorithm picks the node G and as stated in the algorithm when it picks the node G it terminates it has found a paths to the goal and it reports that path back. So, clearly we can see here that an overestimating function could terminate could make A star terminate with a path to the goal which is non optimal.

(Refer Slide Time: 26:42)



What about an underestimating function? So, let us investigate that now. So, we assume that there is another function we will call it h 2, but that underestimates the distance to the goal

and for the sake of argument we will assume that this also misjudges the relative distance to the goal.

So, this function h 2 also thinks that Q is closer to the goal as compared to P; it thinks that h of Q is 15 the heuristic value of Q or the estimated distance from Q to the goal is 15. Whereas in fact it is 40 and it assumes that the heuristic value of P is 20 and it thinks that P is just 20 units away from the goal.

Now, you can see that both these value 15 and 20 are underestimates and also like in the previous case they are mischarging this function is mischarging which node is actually closer to the goal. It thinks that Q is closer to the goal, whereas actually P is closer because the cost of going from P to G is actually 30 and the cost of going from Q to G is actually 40.

But the algorithm thinks that Q is closer and therefore it now computes the f values for both these nodes and the f values in this case will the heuristic values of 20 and 15 respectively. So, the f values are 120 and 115.

So, like the version which use h 1 this version also thinks that Q is closer to the goal. So, it will pick this node Q from OPEN it will remove it from OPEN and again it will find a path to the goal. In fact, the same paths to the goal that the previous version found and it will add that node Q to node G to OPEN.

Now, it adds the node G with the same cost of 140, why is that? That is because 140 is the actual cost of the path found by the search algorithm to the goal node G. And the actual path is still the same 100 plus 40 and therefore it puts this node on to open. And now we can see that between the 2 options that it has whether to pick P or whether to pick Q, it will pick P because P is less than h of f of P is less than f of g.

And now it will find this shorter paths to the goal node G, just like Dijkstra's algorithm would have updated the G value of G to 130 this also does the same and it terminates now

with a path which is the optimal path. So, what this small thought experiment has given us the insight that it pays to underestimate the distance to the goal.

So, I sometimes try to use this analogy to say that supposing you were out there to buy some product. So of course nowadays the most sought after products are mobile phones and let us say that you have chosen a particular model P Q R made by some company and you are want to buy that particular phone. And either you are searching on the net at different locations or you are in some physical location in a mall let us say which has got 2 or 3 mobile stores and you want to go from one to the other.

The question that I would pose is that supposing the first store offers you the phone for a price of let us say for arguments sake 5000 rupees essentially. Now, you have a you have to make a decision whether to buy it from there which is like saying that I have terminated my search or whether to look at other stores and find out if you can get it better.

Now, when would you go to the other store you would only go to the other store, if you thought that the other store might give it to you cheaper. In other words you are underestimating the actual cost which will be quoted or the actual price that will be coated by the other stores.

And if you underestimate you will go and search ask the other store essentially the point here is not that you the other store is going to sell it to you cheaper. The point here is that you will end up exploring all possibilities which would lead to an cheapest phone or an optimal cost solution. And to ensure that you do not miss out on any possible optimal solution it makes eminent sense to always underestimate the distance to the goal.

So, having now got this inside next what we will do is we will do a formal proof of this fact essentially. So, we will do that in the next session and go through a little bit of logic or proof methodology. So, see you soon.