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 – 35
Finding Optimal Paths: Branch and Bound

So, let us continue a study of search algorithms for Finding Optimal solutions. And at the moment we are looking at this algorithm called branch and bound which is blind search algorithm, but unlike depth first search and breadth first search and did that we saw earlier this one caters to problems where there are edge costs involved in the in the graph and guarantees an optimal solution.

(Refer Slide Time: 00:49)

State Space (with no edge costs): Blind Search

Depth First Search
Deepest candidates are best
Dives headlong into the search space optimistically

Breadth First Search
Shallowest candidates are best
Stays as close to Start as possible
Wades into the search space and finds a solution with the *shortest number of hops*

Depth First Iterative Deepening
Depth First masquerading as Breadth First

Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Madras

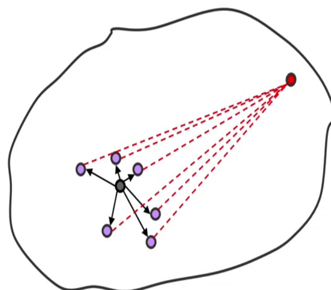
So, we are moving on to now state space search. And so far we saw algorithms for blind search or uninformed search which had no edge costs involved. The first algorithm we saw was depth first search, where the deepest candidates are the best. And it dives headlong into the search space optimistically hoping that it will reach the goal state.

The conservative algorithm was breadth first search where the shallowest candidates are considered to be the best. And it stays as close to the start as possible. And it gradually wades into the search space layer by layer which is why it guarantees that you find the solution with the shortest number of hops. And we saw a depth first iterative deepening which was basically depth first trying to behave like depth first search in its behaviour though it retain the advantages of having lower memory cost.

(Refer Slide Time: 01:48)

State Space: Heuristic functions (recap)

The heuristic function estimates the distance to the goal.

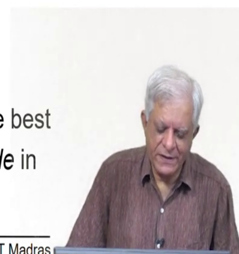


This estimate, h(n), can be used to decide which node to pick from OPEN

Best First Search

Candidates that appear to be *closest to the goal* are best

Chooses the candidate with the *lowest h-value node* in the hope of finding a solution *sooner*.



Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Madras

We also saw informed search algorithms, and we looked at the notion of heuristic functions. And this is a recap of what we had done earlier. A heuristic function is a function which estimates the distance to the goal. And this estimate is used to guide the search process. So, the best first search algorithm candidates that appear to be closest to the goal are considered to be the best, and it chooses the node with the lowest h-value in the hope of finding a solution sooner. So, while breadth first search was aimed at finding a shortest solution when edge costs are equal, best first search is aimed at finding the solution as quickly as possible.

(Refer Slide Time: 02:36)

Branch and Bound

- Organize a search space that does not preclude any solution.
  - Necessary for finding the optimal solution
- Search space could be the state space, in which a partial sequence of moves is extended.
- Search space could also be the solution space in which an abstract solution is refined.
  - As seen with the TSP example
- Continue looking for a solution (extend/refine) until
  - A complete solution (with known cost) is available, and
  - no other possible solution with a smaller cost exists.

The basic idea behind B&B is to prune those parts of a search space which *cannot* contain a better solution.

Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

And then we started looking at branch and bound. So, what since we want to guarantee the optimal solution, we want to work in a search space in which will not preclude any solution. So, that if wherever the optimal solution is we are going to find it essentially. It could be the state space in which a partial sequence of moves is extended as we will see today. It could

also be the solution space in which an abstract solution is refined as we saw with the TSP example in the last time.

So, the basic idea is that you continue looking for a solution extend it or refine it, and there have been people who have said that they have basically both the same thing. You can think of extending a solution also as a kind of refinement until a complete solution with known cost is available.

And it is the cheapest which means that no other possible solution with a smaller costs exist. So, the basic idea behind branch and bound is prune those part of the search space which cannot contain a better solution. So, in some sense, we are still trying to work towards the solution.

(Refer Slide Time: 03:53)

A Tiny Search Problem with Edge Costs

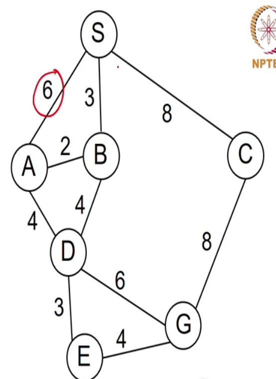
The *MoveGen* function

S → (A,B,C)
A → (S,B,D)
B → (S,A,D)
C → (S,G)
D → (A,B,G,E)
E → (D,G)
G → (C,D,E)

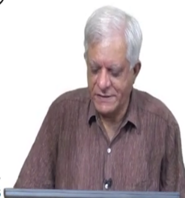
The State Space

Note: The placement of nodes in the graph may not reflect the edge costs accurately.

Cost of a path is the sum of the cost of edges in the path.



What is the *shortest path* from S to G?



Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Madras

So, let us look at the state space version of this and we look at a tiny graph again as we did earlier, except that this time in our graph we have weights for the edges. So, for example, the edge between S and A has a weight of 6, the cost of a path is the sum of the weights on the edges in that path. Now, I have drawn this graph roughly speaking to reflect the distances, but it may not be entirely accurate. So, go by the edge labels rather than by the actual placement.

So, I try to draw the graph, so that the distances also look visually what we have indicated them to be by edge basis essentially, so that is why the edge between S and C is longer than the edge between S and B. Edge between S and B is 3 units, and between S and b C is 8 units. The problem that we want to address is what is the status shortest path from S to goal. So, when we say shortest path in this context, we mean the path which has the least cost some of the edge weights, not the number of hops.

(Refer Slide Time: 05:13)

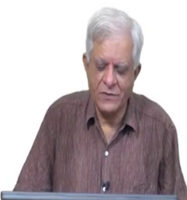
Branch & Bound

Each candidate is tagged with an *estimated cost of the complete solution*.

Estimated cost of (full) solution = actual cost of partial solution

Refine the best looking partial solution
Till the best solution is fully refined

Branch & Bound extends the cheapest partial path

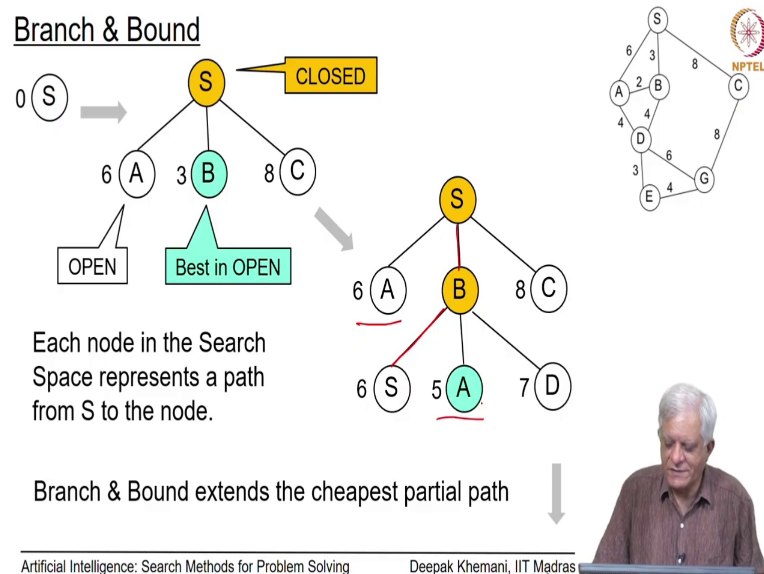


Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Madras

So, let us see how this work. Each candidate is tagged with an estimated cost of a complete solution. So, at all points, we are have in some sense the complete solution is in mind. In the case of branch and bound, the estimated cost of the full solution is very simply taken to be the actual cost of the partial solution. So, it is a gross approximation, but it is the useful approximation because it helps find the optimal path.

The algorithm that we have been following is still the same that refine the best looking partial solution till the best solution is fully refined. And the version of this algorithm that branch and bound implements extends the cheapest path partial path bound so far.

(Refer Slide Time: 06:04)



So, let us look at how branch and bound works with this small graph that we have drawn. Remember that the algorithm says extend the cheapest path essentially. And in our representation, the search space will contain partial paths and each node in the search space

will be a partial path starting from S to that node essentially, S is a start node. Initially, of course, we have only the start node and its edge cross and it is cost is 0 because that is why you are at the beginning. And the basic idea is to extend the cheapest partial path.

So, when we expand S we get its 3 neighbours A, B and C, and the cost of A is 6, the cost of B is 3, and the cost of C is 8. So, when we say that cost of a node essentially we mean the cost of a paths from the start state to that node. In this case, there is only one edge. So, the edge cost becomes a cost of the node, but as the graph grows as the search space grows the costs of every partial path will get incremented essentially. So, keep in mind that each node in the search space represents a partial path from S to that node.

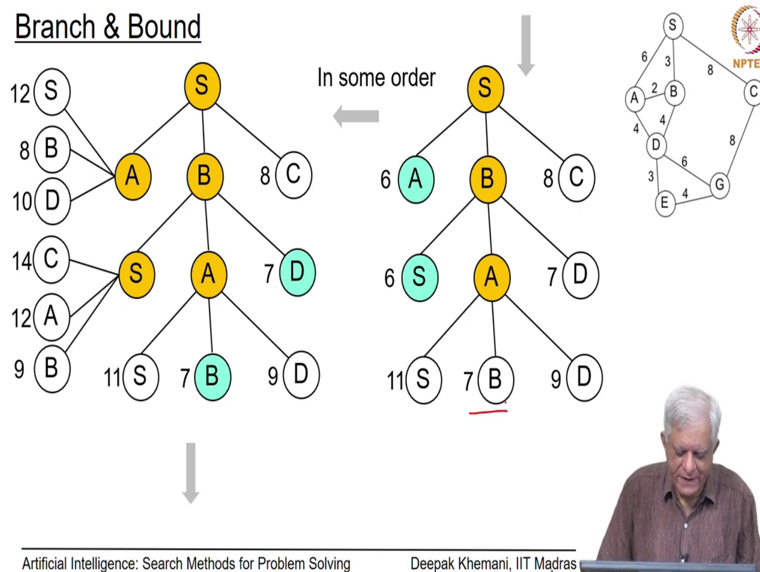
So, we have three partial paths here from S to A, S to B and S to C. And we have indicated in cyan or blue or green whatever you want to call it, the best node in the amongst the open nodes. And every node that we have finished expanding, we will put in the closed as before. So, we maintain both open and close as before. And as before we always pick up the best node from the open which means that we implement a priority queue for doing that and proceed till we have found the paths to the goal node.

So, since B is the best node we have now expanded B n we have got the 3 children of B which is S, A and D. Now, observe here that the labels against this three new nodes that we have added well label of S is 6.

It means it is a path starting from S going to B and then going back to S and that because the edge weight is 3, it is 3 plus 3 which is 6, whereas, if you go from S to B to A, then the new cost is 5. So, you can see that we have found two paths from the source or start to a one is a direct edge which is a cost 6, and one is a path through B which is the cost of 5.

Now, branch and bound the way that we are discussing it now implements every such partial path as a distinct node in the search spaces essentially, hence always peaks the cheapest partial path and extends it.

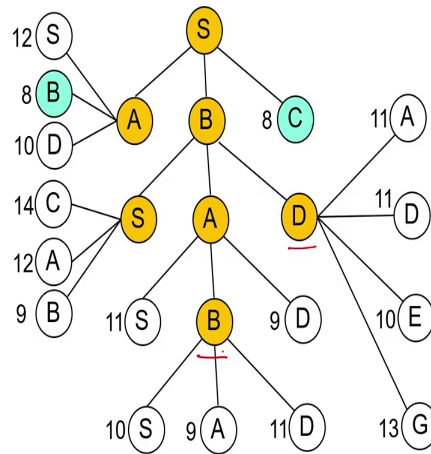
(Refer Slide Time: 09:13)

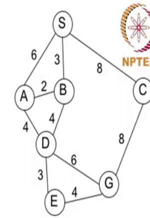


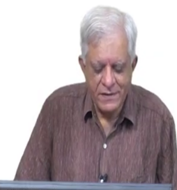
So, here we have then we extend it the cheapest node A which are the cost of 5, and we found new paths to S, B and D. S, B and D are the neighbours of A. So, again just to remind you the new node B for example, which has the label which has the weight of 7 means that you have gone from S to B and from B to A and from A to B back resulting in this total cost of 7 essentially.

So, at this point, there are two nodes which appear to be cheapest which is A and S, both have a edge cost of 6. And branch and bound will in some order expand both of them through generate this new tree where you see the children of A and S on the left hand side. And at this point, B and D, the green or cyan colour nodes have become the best. And the algorithm will next proceed to expand them again in some order.

(Refer Slide Time: 10:20)

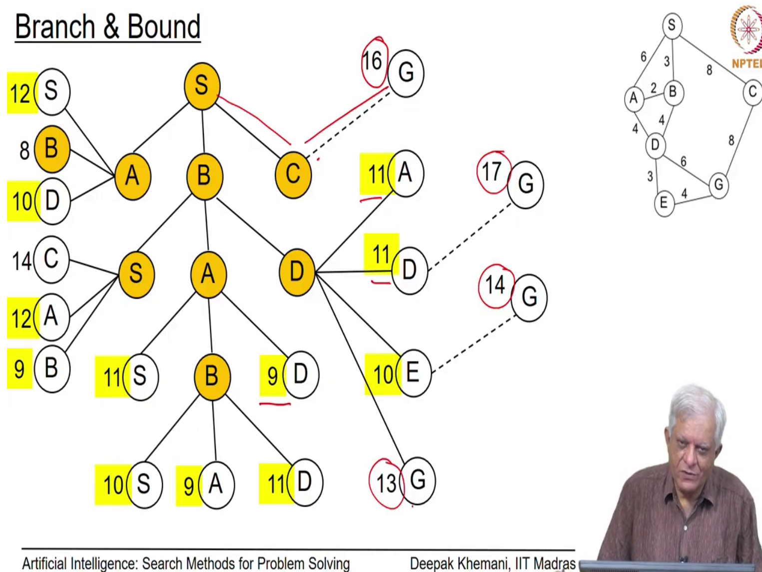






And what we get is this new graph in which remember that this D and this B was the new node that we expanded. And all the parts going from these nodes are more expensive than the earlier parts which existed. So, we have this node B with a cost of 8, and node C which is with a cost of also 8 which are the best nodes. So, branch and bound will continue in this fashion always picking the best nodes to expand.

(Refer Slide Time: 10:52)



And eventually it will go through these nodes which I have highlighted in yellow. So, for example, this node 11, A with cost 11, D with cost 11, D with cost 9 and so on and so forth. You can see that it has found many different paths to the goal one is from S to C and C to D which has a cost 16.

It has also found another path with a cost 17, another cost with the path with the cost 14 and so on. And only after it has finish expanding the yellow coloured nodes, finding longer and longer paths will it eventually focus on the deepest path which is from S to B to D to G which is of course 13.

So, at the moment as depicted in this graph, the nodes which we with which have been highlighted with yellow in the cost with edges edge the partial costs that have been highlighted with yellow are cheaper. Branch and bound actually first explore them before

picking up this node G. And it will when it picks a node G, it will be the lowest cost node. And we would know that a shorter path cannot exist.

(Refer Slide Time: 12:14)

Dijkstra's algorithm

- Dijkstra's algorithm begins by assigning infinite cost estimates to all nodes except the Start node.
- It assigns colour white to all the nodes initially.
- It picks the *cheapest* white node and colours it black.
- Relaxation: Inspect all neighbours of the new black node and check if a cheaper path has been found to them.
- If yes, then update cost of that node, and mark the parent node.

Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Madras

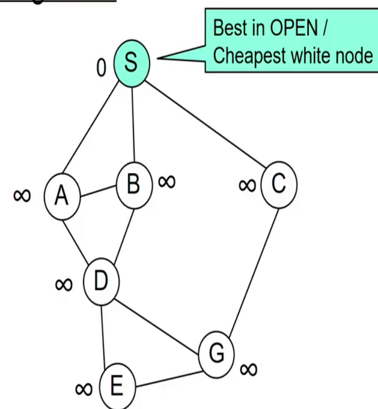
Now, you would have recognized this that this is very similar to the Dijkstra's algorithm for singles so shortest path to all nodes which is a well known algorithm studied in Data Structures course. And just to do a quick recap Dijkstra's algorithm begins by assigning infinite cost estimates to all nodes except for the start node. And it assigns the colour white to all nodes initially. And it picks the cheapest white node and colours it black.

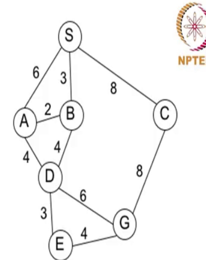
The node that it has just coloured we relax do a process of relaxation which means it we inspect all the neighbours of a new black node and check if a cheaper path has been found to them. If a cheaper path is found, we keep track of the cheapest path. What Dijkstra's algorithm does differently from branch and bound is that it only keeps one copy of the node

unlike branch and bound which keeps many copies. And it keeps a copy to the parent node instead of storing the entire partial path.

(Refer Slide Time: 13:23)

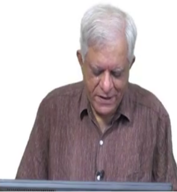
Dijkstra's Algorithm





Colour cheapest node and relax the connected edges

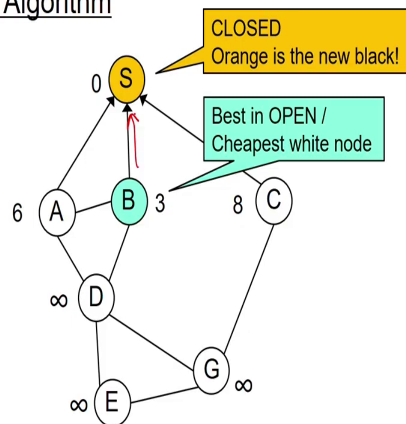
Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

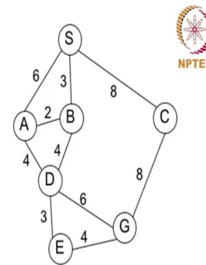


So, let us see how this algorithm will work. Initially as before the start node is the cheapest and it has a cost of 0, all other nodes as you can see have been labelled with the costs infinity or some very large number if you are implementing the algorithm. And the algorithm says pick the cheapest node colour it black, and relax the edges which means that we will look at the neighbours of S which are A, B and C.

(Refer Slide Time: 14:01)

Dijkstra's Algorithm





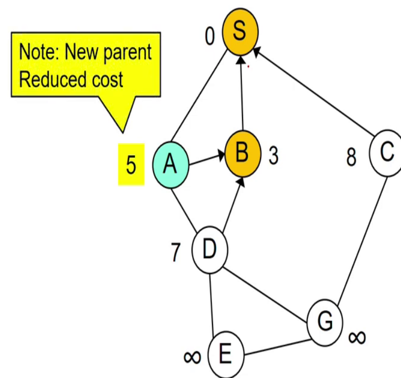
Colour cheapest node and relax the connected edges

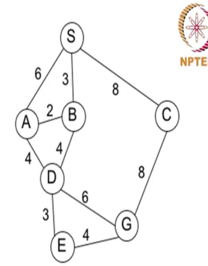
So, this is what happens. Observe that I have used the colour orange. So, maybe orange is the new black. So, think of orange as black in as far as Dijkstra's algorithm is concerned. And at an any point you I have indicated like in the branch and bound algorithm, the cheapest node in open in cyan essentially.

So, you can see that they are three, three white nodes or 3 nodes in open A, B, and C, with the costs with respect to be 6, 3 and 8. And B is the cheapest. And from each node there is a parent pointer to where it came from. So, from B we have this parent pointer to S, and likewise for the other nodes.

(Refer Slide Time: 14:49)

Dijkstra's Algorithm





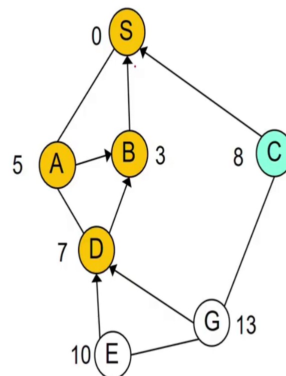
Colour cheapest node and relax the connected edges

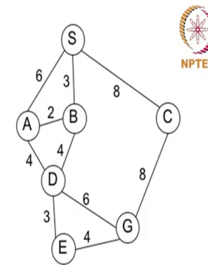
Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

So, now it colours B as black within our cases orange and relaxes the edges for its neighbours. You can see that at this point, it has found a better path to the node A. And the pointer from A now points to B which means that the best paths to A is from the node B, not from the node S even though it is only one edge essentially that is because we are counting edge costs.

(Refer Slide Time: 15:23)

Dijkstra's Algorithm





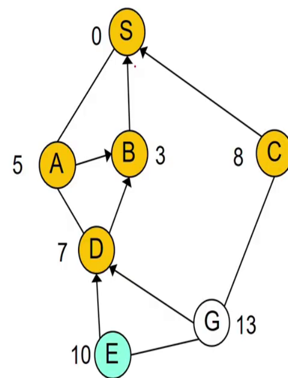
B&B has found a path to G but it is not sure if it is best.

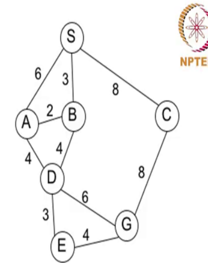
Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

The next node to be relax would be D at which point you will see that a path to the goal node has already been found. But we are still not sure whether that is a cheapest path, because there are nodes which are open. For example, C as a cost of 8 and if you imagine that that see that the edge between C and G was let us say two units, then the path from S to C to G would have been 10, and that is why at this point Dijkstra's algorithm and also branch and bound we will not pick the node G, but it will extend the cheapest partial path which in this case is the node C.

(Refer Slide Time: 16:01)

Dijkstra's Algorithm





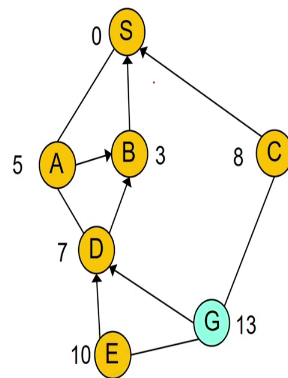
If cost(C,G) were to be less than 5,
then C would become the parent of G

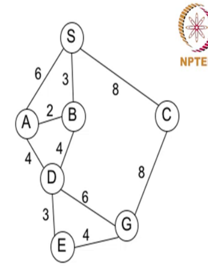
Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

And after that E is still remaining with a cost of 10. So, what is the paths to E? The paths to E is parent of E is D as we can see, and the parent of D is B, and the parent of B is S. So, the paths to E is S-B-D-E. And that is a cost edge that is a cost of 10. Again if E to G was only two units, then this would have been a cheaper path to the goal G. But as it is not. So, G still considers D to be its parent because from there the path the cost of reaching G is 13 nodes.

(Refer Slide Time: 16:40)

Dijkstra's Algorithm



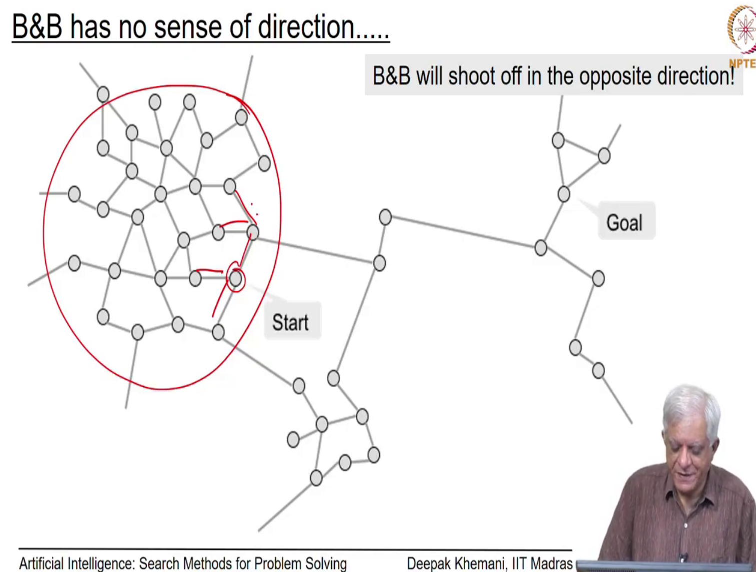


When B&B does pick G it has found the cheapest path to G

Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

And finally, when the algorithm does pick the node G it has found the cheapest path to the goal node. Now, Dijkstra's algorithm is not designed to find the path to specific goal node. It is a algorithm which gives the shortest paths to all nodes from the start node. And therefore, it does not have nearly this sense of direction that we are interested in and we will move on to that shortly essentially.

(Refer Slide Time: 17:11)



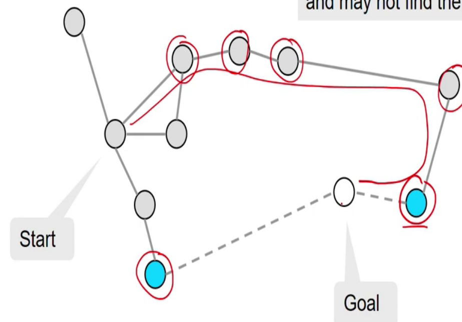
So, this is what I meant by saying that transient bound has no sense of direction which is also which also applies to Dijkstra's algorithm. So, it will keep exploring this part of the space because these nodes are closer to the start node and it always extends the shortest path. So, it will extend the path to this first, then maybe the paths to this and so on and so forth. And it will spend a lot of time exploring this neighbourhood.

Just imagine this is a village and all the small town, and then there is a village next door, and then in a another village a little bit far away is your goal node, our algorithm branch and bound will keep focusing on where it is essentially because those nodes are the closest nodes. And it wants to be sure that when it finds a path to the goal, it will have found the shortest path.

(Refer Slide Time: 18:12)

Best First only looks ahead

Best First only looks at the distance to goal, and may not find the shortest path



Algorithm A* combines the best features of both!

Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

We have seen that best first search looks ahead only. And it is possible that the best first search algorithm may not be may not find the shortest path. So, in this case, if you see if this is the as the goal node is shown and two candidates in open which are in contention which has shown in cyan you can see that this node is closer to open to the goal.

And therefore, the paths that it finds would have been the path which goes like this. And at all points while assuming that this node is the one with the shortest path, but they have other nodes which appear to be closer to the goal.

So, for example, this one, and then this one, and then this one, and then this one, and then this one and that is how the algorithm will end up finding a path which is not necessarily shortest.

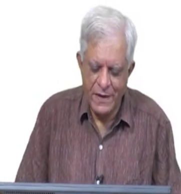
So, essentially we want to look for an algorithm which has this sense of direction that best first has, but also guarantees that you find the shortest path.

And the algorithm that we are going to look at is called a star and it combines the best features of both the algorithms that we have seen branch and bound and best first and in fact, also the Dijkstra's algorithm. So, that is what we will do in the next session.

(Refer Slide Time: 19:46)

State Space Search

Algorithm A*



Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Madras