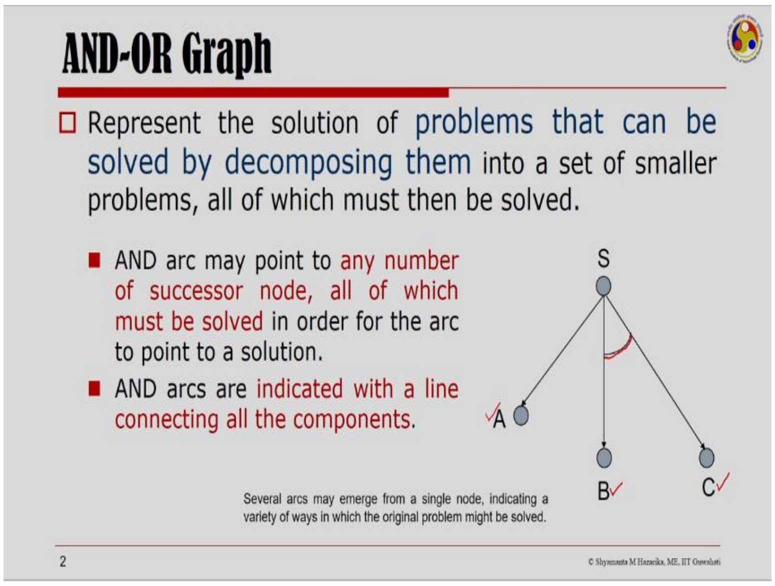
Fundamentals of Artificial Intelligence
Prof. Shyamanta M Hazarika
Department of Mechanical Engineering
Indian Institute of Technology - Guwahati

Module - 3
Lecture - 7
Searching AND-OR Graphs

Welcome to fundamentals of artificial intelligence. Today we will look at searching AND-OR graphs. AND-OR graphs are structures that are used for representation of solution of problems that can be decomposed into smaller problems. Solution of these smaller problems has to be accomplished in order to complete the solution of the original problem. We have looked at AND-OR graphs while we were discussing production systems, particularly decomposable production systems, which allowed a problem to be broken up into smaller problems.

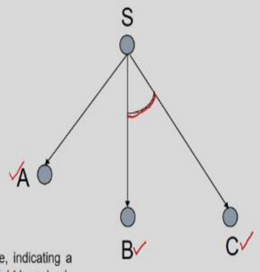
And solution to these smaller problems lead to the solution of the original problem. Let us look at what we mean by an AND-OR graph in more detail now.

(Refer Slide Time: 01:33)



AND-OR Graph

- Represent the solution of problems that can be solved by decomposing them into a set of smaller problems, all of which must then be solved.
  - AND arc may point to any number of successor node, all of which must be solved in order for the arc to point to a solution.
  - AND arcs are indicated with a line connecting all the components.



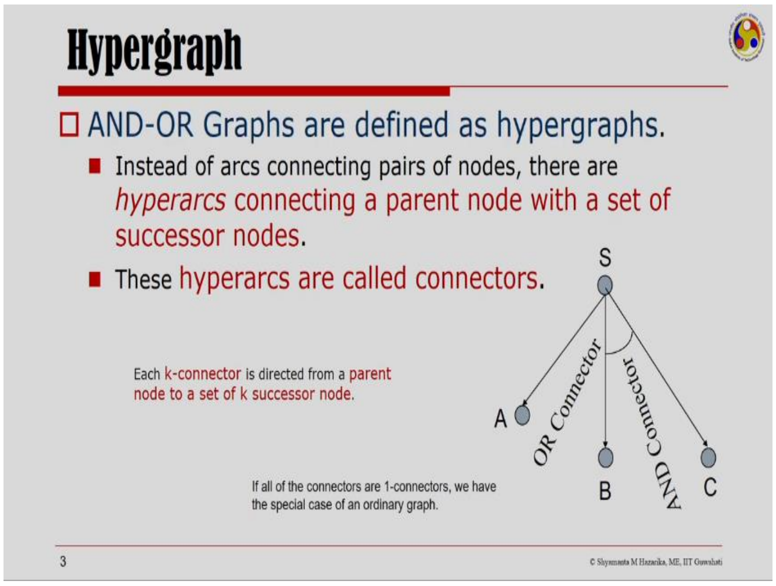
Several arcs may emerge from a single node, indicating a variety of ways in which the original problem might be solved.

2 © Shyamanta M Hazarika, ME, IIT Guwahati

Let us say we have a problem S that could be solved by either going to A or by B and C together. It is interesting to note there that several arcs may emerge from a single node, indicating a variety of ways in which the original problem might be solved. In this example here, S could be solved if I have solved A. Or, it could be solved by solving B and C, both of them.

When it is solved by solving B and C both, we tend to name it as an AND arc. And AND arc point to a number of successor nodes, all of which must be solved in order for the arc to point to a solution. Here, in the example that we were looking at, B and C both needs to be solved to solve S. AND arcs are indicated with a line connecting all the components. So, for B and C which are AND arcs, we will try to use this arc here to indicate it that it is an AND arc.

(Refer Slide Time: 03:01)



Hypergraph

- AND-OR Graphs are defined as hypergraphs.
  - Instead of arcs connecting pairs of nodes, there are *hyperarcs* connecting a parent node with a set of successor nodes.
  - These hyperarcs are called connectors.

Each k-connector is directed from a parent node to a set of k successor node.

If all of the connectors are 1-connectors, we have the special case of an ordinary graph.

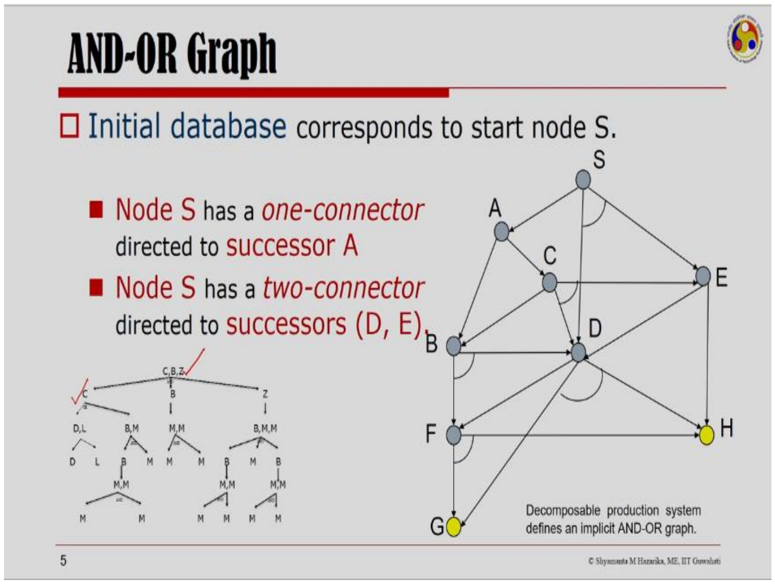
3

© Shyamanta M Hazarika, ME, IIT Guwahati

Before we move on to understanding AND-OR graphs, let us review what we mean by hypergraphs. So, hypergraphs are where the arcs connecting pairs of nodes, are referred to as hyperarcs. They connect a parent node with a set of successor nodes. These hyperarcs are called connectors. So, coming back to the example that we were discussing; here, B and C together has to be solved to solve S.

So, the link between these 2 is referred to as a two-connector. So, each k-connector actually is a directed arc from a parent node to a set of successor nodes. Here, between B C, which is to be both solved for S to be solved, we have an AND connector. If all of the connectors are actually one-connectors, then we have the special case of an ordinary graph. So, hypergraphs are used to define AND-OR graphs.

(Refer Slide Time: 04:13)



AND-OR Graph

- Initial database corresponds to start node S.
  - Node S has a *one-connector* directed to successor A
  - Node S has a *two-connector* directed to successors (D, E)

Decomposable production system defines an implicit AND-OR graph.

5

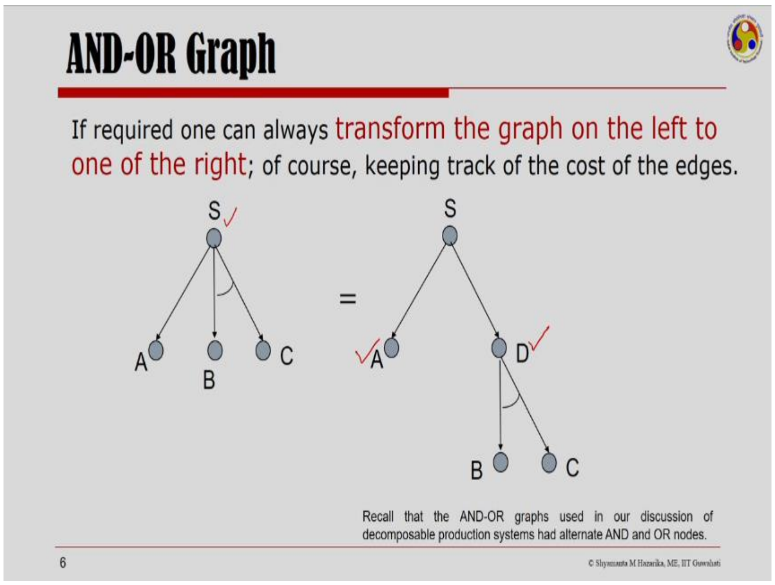
© Shyamanta M Hazarika, ME, IIT Guwahati

Here is an example of an AND-OR graph. For every k-connector with k > 1, we could see that we are denoting with a curved line. So, here is a two-connector. Here is again a two-connector. Here is a two-connector. Node S here has a one-connector that goes to successor A and has a two-connector that goes to D and E. The decomposable production systems that I was referring to actually defines an implicit AND-OR graph and the initial database corresponds to the start node S.

Kindly recall what we did in our production system's lecture. We looked at a decomposable production system for rewriting rules. So, we had an initial database C B Z which could be broken up into 3 databases, C, B and Z. Now, each of these could be applied rewriting rules. Like, C could be rewritten as D L or C could be rewritten as B M. So, here we have an OR node.

But, when we had splitted the main problem into 3 component problems C, B and Z, we have an AND node. So, decomposable production systems defines an implicit AND-OR graph. One difference that you have, must have noticed by this time is, that when we are talking of decomposable production systems, the AND and OR nodes are at alternate levels. Whereas, in the graphs that we have introduced today, both the AND arc and the OR arc are at the same node. But, there is not much of a difference. We can move from one to the other.

(Refer Slide Time: 06:28)



AND-OR Graph

If required one can always transform the graph on the left to one of the right; of course, keeping track of the cost of the edges.

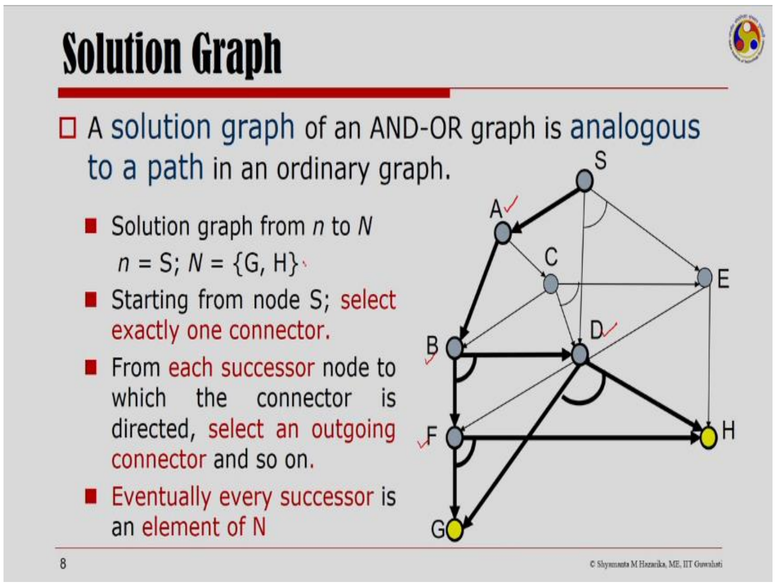
Recall that the AND-OR graphs used in our discussion of decomposable production systems had alternate AND and OR nodes.

6

© Shyamanta M Hazarika, ME, IIT Guwahati

So, let us look at this with example. Here is our graph S with an A and a two-connector leading to B and C. We can replace this with the graph on the right, where we maintain the fact that there are 2 possibilities of solution for S. One going to A and the other going to B and C together. So, for that, we introduce an auxiliary node D. And D will be solved if we could solve B and C. So, we can transform the graph on the left to one on the right. But of course, we need to keep track on the cost of the edges.

(Refer Slide Time: 07:19)



Solution Graph

- A solution graph of an AND-OR graph is analogous to a path in an ordinary graph.
  - Solution graph from *n* to *N*
    n = S; *N* = {G, H}
  - Starting from node S; select exactly one connector.
  - From each successor node to which the connector is directed, select an outgoing connector and so on.
  - Eventually every successor is an element of N

8

© Shyamanta M Hazarika, ME, IIT Guwahati

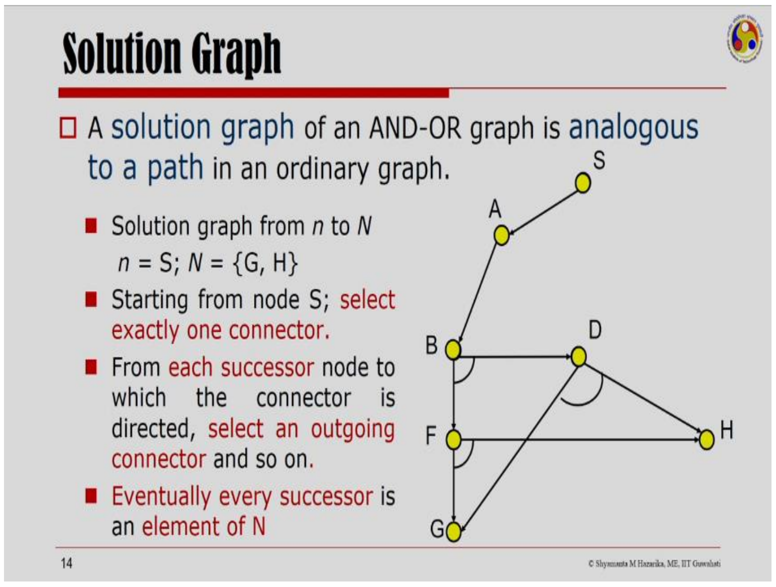
Having introduced the concept of an AND-OR graph, let us now look at, what do we mean by solution of an AND-OR graph. Solution of an AND-OR graph is analogous to a path in an ordinary graph. So, let us look at this example AND-OR graph. Here, if we are looking for a

solution from node S to the set of solution nodes which are G and H, we want to explore starting at S and select a connector to take us to the successor nodes.
Thereafter, from each successor node, we again select an outgoing connector, so on and so forth. Eventually, every successor needs to be an element of the set N, which is a set of goal nodes. So, let us take S first. And then, we will take the one-connector. We come to the node A. The node A, we need to realize, has 2 paths; one going to B and one going to C. But then, these are OR and therefore we need to pick up one of them.

So, we proceed to B. Now, it is interesting. If you focus your attention to the node B, you would realize that node B has a two-connector leading to nodes F and node D. What this mean is, in order to solve B, I need to solve F as well as solve D. So next, I follow that two-connector, arrive at F, arrive at D and look for solution to F and D. The next node to be expanded is D. And now again, we see that D is a two-connector leading us to G and H.

So, we solve D by going to G and H. And now, we solve F which is a two-connector again leading us to F and H one more time. So, after we have traversed from S through A to B via F D, we have arrived at G H, which is the set that we want to have arrived at from N.

(Refer Slide Time: 10:09)



Solution Graph

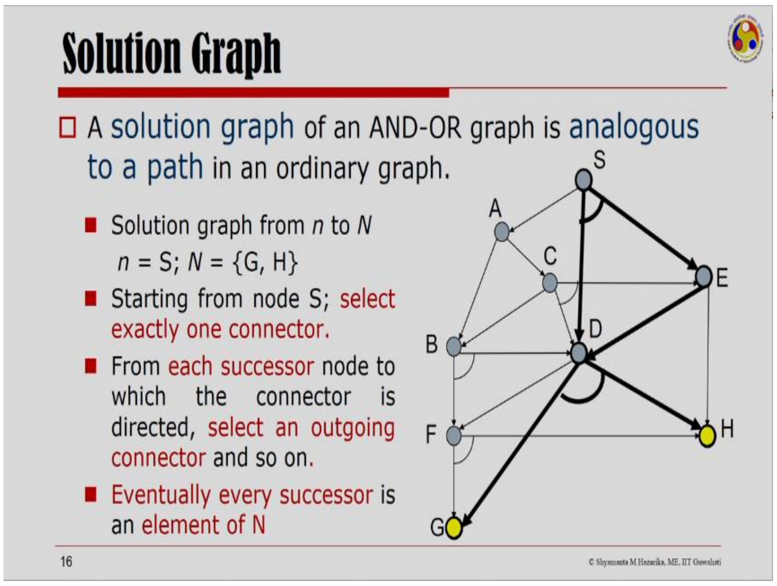
- A solution graph of an AND-OR graph is analogous to a path in an ordinary graph.
  - Solution graph from *n* to *N*
    n = S; *N* = {G, H}
  - Starting from node S; select exactly one connector.
  - From each successor node to which the connector is directed, select an outgoing connector and so on.
  - Eventually every successor is an element of N

14 © Shyamanta M Hazarika, ME, IIT Guwahati

So, this is the solution graph. Now, if we focus on each of the nodes of the solution graph, it is interesting to note that once G and H are solved, I know that D would be solved. Because D is an AND node between G and H. Once I have G and H solved, I can also see that F that has a two-connector leading to G and H; I can have that as solved. And then, once F and D are solved, I have B marked as a solved.

Thereafter, A could be marked as solved and thereafter S could be marked as solved. Because for solving S all I need is solving A. This is the solution subgraph for the AND-OR graph that we started with.

(Refer Slide Time: 11:12)



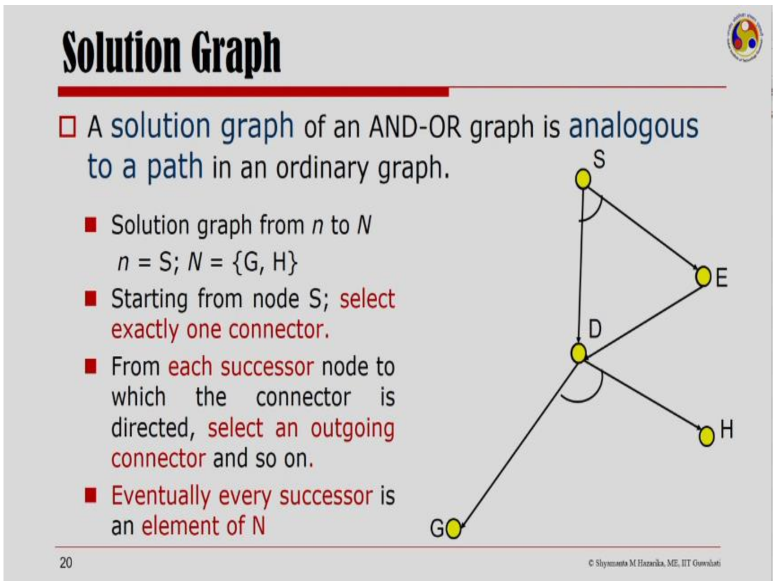
Solution Graph

- A solution graph of an AND-OR graph is analogous to a path in an ordinary graph.
  - Solution graph from n to N
    n = S; N = {G, H}
  - Starting from node S; select exactly one connector.
  - From each successor node to which the connector is directed, select an outgoing connector and so on.
  - Eventually every successor is an element of N

Let us now see if there are more than one solution to this graph. Recall that we had started by following the one-connector from S leading to A. Instead of that, if we follow the two-connector, that is, we move this way leading from S. Then, we would have to solve first by moving to E and then, moving to D, because they are on an AND arc. Once we have solved D and E, we can look at E which has a one-connector to D.

So, we can mark that. Therefore, we have got the next node on E which is D. At D, we realize we have a two-connector leading to G and H. So, we will mark that and we have arrived at the solution set. So, this is an alternate solution subgraph for the AND-OR graph that we are illustrating here. We can now think of going back and looking at if we have solved each of the nodes on our traversal.

(Refer Slide Time: 12:33)



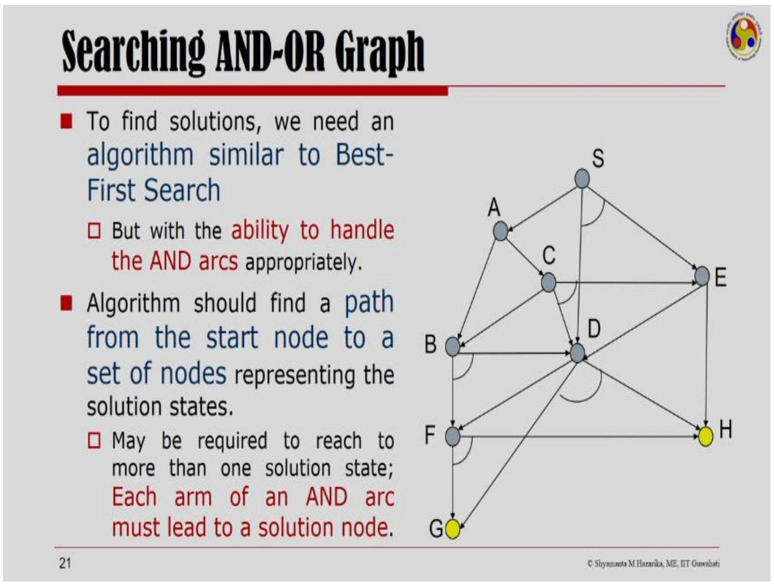
Solution Graph

- A solution graph of an AND-OR graph is analogous to a path in an ordinary graph.
  - Solution graph from *n* to *N*
    n = S; *N* = {G, H}
  - Starting from node S; select exactly one connector.
  - From each successor node to which the connector is directed, select an outgoing connector and so on.
  - Eventually every successor is an element of N

So, as before, you would see that we have solved G and H. So, that means we have solved D which is an AND of G and H. And now, if we see S; before we can mark S to be solved, we need to realize that we need to mark E. We have one-connector between D and E and D is solved. Therefore, we can mark E as solved. Once we mark E as solved, we can now mark S as solved.

So, this is the solution subgraph of the AND-OR graph. So, as we can see from this example, for a given AND-OR graph as simple as the one we have shown here, we have highlighted 2 solution graphs. The idea is, how do we get to the most optimal solution graph. For that, we need to search through the AND-OR graph and we need an algorithm, something like the best-first search.

(Refer Slide Time: 13:43)



Searching AND-OR Graph

- To find solutions, we need an algorithm similar to Best-First Search
  - But with the ability to handle the AND arcs appropriately.
- Algorithm should find a path from the start node to a set of nodes representing the solution states.
  - May be required to reach to more than one solution state; Each arm of an AND arc must lead to a solution node.

21

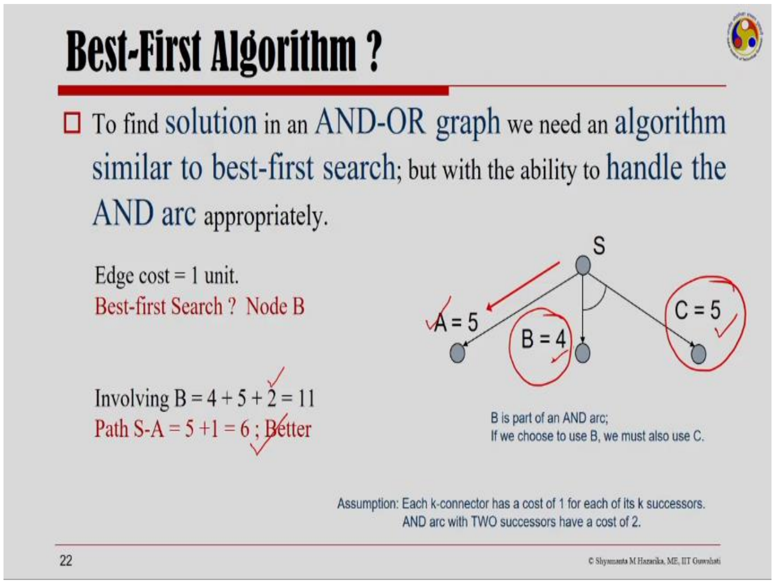
© Shyamanta M Hazarika, ME, IIT Guwahati

But then, as we will see in the next few slides, the best first search may not be tuned to do a searching on the AND-OR graph. This is because of its inability to handle the AND arcs appropriately. So, we are looking for some algorithm which does something like the breadth-first search, but should be able to handle the AND arcs appropriately. This is what we will try to understand in this part of the lecture.

And another thing is the algorithm that should find a path from the start node to the set of node representing the solution states, may be required to reach more than one solution state. Now we should realize that this is precisely because each arm of an AND arc must lead to a solution. And this is something we must take care of when we are writing this algorithm. Then, after we realize that we have expanded a node and we have reached the solution nodes, we should go back and see if the parent can be marked solved.

This is only possible if all the successor nodes of a k-connector arising from a given node has been solved. Then only I can mark that particular node n as solved. And that we need to ensure. So, we will look at this.

(Refer Slide Time: 15:29)



Best-First Algorithm ?

- To find solution in an AND-OR graph we need an algorithm similar to best-first search; but with the ability to handle the AND arc appropriately.

Edge cost = 1 unit.
Best-first Search ? Node B

Involving B = 4 + 5 + 2 = 11
Path S-A = 5 +1 = 6 ; Better

B is part of an AND arc;
If we choose to use B, we must also use C.

Assumption: Each k-connector has a cost of 1 for each of its k successors.
AND arc with TWO successors have a cost of 2.

22

© Shyamanta M Hazarika, ME, IIT Guwahati

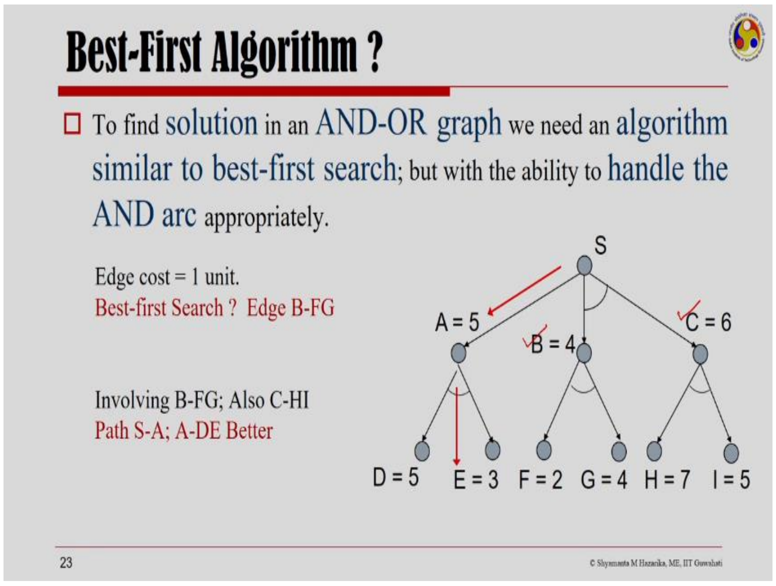
But before that, we will look at what is that makes the breadth-first algorithm inadequate to deal with searching AND-OR graphs. So, here is an example to start with. I have a node S and I have an arc coming out of this to A. And an alternate solution involves B and C together. So, this portion of the graph here, shows that I have two-connector. In order to solve S, I would need to solve both B and C.

Now, what we have here, with the nodes marked is a heuristic value. This is something like the heuristic value that we have looked at when we were doing problem solving as search and trying to look at informed search. Only difference here is that, these heuristic values contain the promise of the node and gives us a measure of how good it is to take it to the solution. We do not store here the cost of coming to this present node.

So, if you assume that each edge costs 1 unit. And I was just doing breadth-first search, then my algorithm would be choosing node B for expansion. This is because node B has the minimum cost to the goal. But then, is it the best path? If you look at the cost of the path, you will see that B is part of an AND arc. So, if we choose B, we must also use C. So, involving B would mean that I would be expanding 4 units at D, 5 units at C.

And because this is a two-connector, I would be spending 2 units there. So, I would be having an estimate of 11 on this path. Whereas, the other path from S to A; all I spend is 5 units that takes me to the solution from A. And 1 unit because of the edge. So, I have 5 + 1 and which is 6. And this is definitely better. So, actually I must traverse this path. So, this is something which is not possible from just looking at the heuristic value of that particular node.

(Refer Slide Time: 18:40)



Best-First Algorithm ?

- To find solution in an AND-OR graph we need an algorithm similar to best-first search; but with the ability to handle the AND arc appropriately.

Edge cost = 1 unit.
Best-first Search ? Edge B-FG

Involving B-FG; Also C-HI
Path S-A; A-DE Better

S
A = 5
B = 4
C = 6
D = 5
E = 3
F = 2
G = 4
H = 7
I = 5

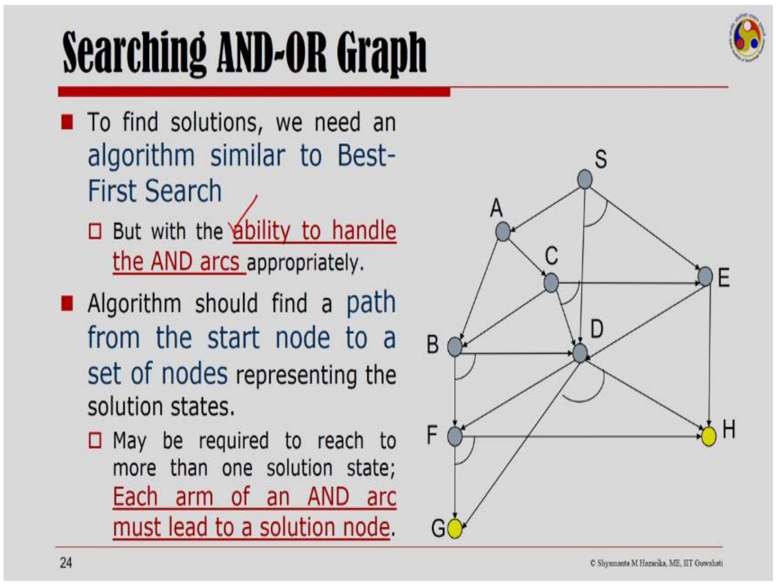
23

© Shyamanta M Hazarika, ME, IIT Guwahati

Let us try to understand this with one more depth of search. Let us say, now I expand node A and arrive at B and E. Node B is expanded to, let us say F G, C is expanded to H and I. Now, if you look at the edges, then possibly one involving B F G here, has the lowest cost. Because, this would be 2 + 4 + 2 which is 8. This would be 5 + 3, 8 + 2, 10. Whereas this one here would be 7 + 5, 12 + 2, 14.

So, possibly this is the best. But the question is, is it true? If you look at it more closely, you will realize that, even if this path has a cost which is lowest, but then involving this path in this solution would also mean that I have to involve C. And therefore, this cost would be added. Under this present scenario, the best would be to go along this path and expand D and E further.

(Refer Slide Time: 20:14)



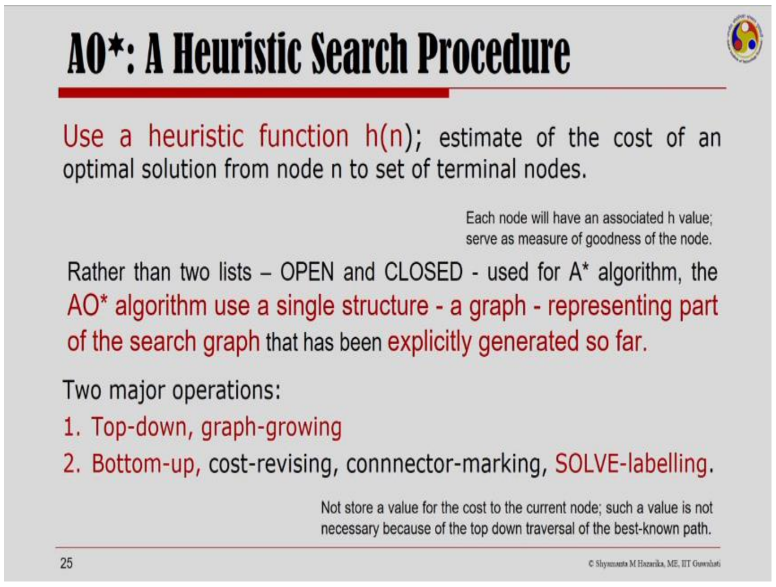
Searching AND-OR Graph

- To find solutions, we need an algorithm similar to Best-First Search
  - But with the ability to handle the AND arcs appropriately.
- Algorithm should find a path from the start node to a set of nodes representing the solution states.
  - May be required to reach to more than one solution state; Each arm of an AND arc must lead to a solution node.

24 © Shyamanta M Hazarika, ME, IIT Guwahati

So, we had looked at these inadequacies of the best-first algorithm. Its inability to handle the AND arcs; and therefore, would look at a more specialized version of searching AND-OR graphs.

(Refer Slide Time: 20:35)



AO*: A Heuristic Search Procedure

Use a heuristic function h(n); estimate of the cost of an optimal solution from node n to set of terminal nodes.

Each node will have an associated h value; serve as measure of goodness of the node.

Rather than two lists – OPEN and CLOSED - used for A* algorithm, the AO* algorithm use a single structure - a graph - representing part of the search graph that has been explicitly generated so far.

Two major operations:

1. Top-down, graph-growing
2. Bottom-up, cost-revising, connnector-marking, SOLVE-labelling.

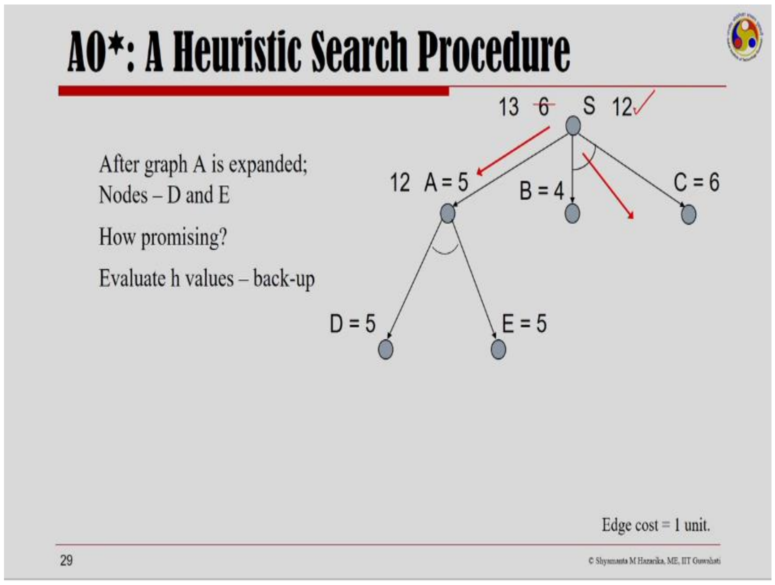
Not store a value for the cost to the current node; such a value is not necessary because of the top down traversal of the best-known path.

25 © Shyamanta M Hazarika, ME, IIT Guwahati

We have a heuristic search procedure that is called the AO * algorithm that we will look at here. So, the AO * uses a heuristic function. It estimates the cost of an optimal solution from a node n to a set of terminal nodes. And each node is somehow connected to its successor. It remembers where it came from. Each node will have an associated h value that serves as a measure of goodness of the node.

Rather than the 2 lists that we have used while we were doing A *, the open and the closed list; the AO * algorithm uses a single structure, a graph that represents part of the search graph that has been explicitly generated so far. 2 major operations are involved in this algorithm. One is the top-down graph-growing, where we get to the partial solutions. And then there is another which is the bottom-up cost-revising connector-marking and basically, if you remember from my explanation of the solution graphs that we looking at, the solve-labelling. We label which of the nodes are solved and get to the solution.

(Refer Slide Time: 22:14)



AO*: A Heuristic Search Procedure

After graph A is expanded;
Nodes – D and E

How promising?

Evaluate h values – back-up

Edge cost = 1 unit.

29

© Shyamanta M Hazarika, ME, IIT Guwahati

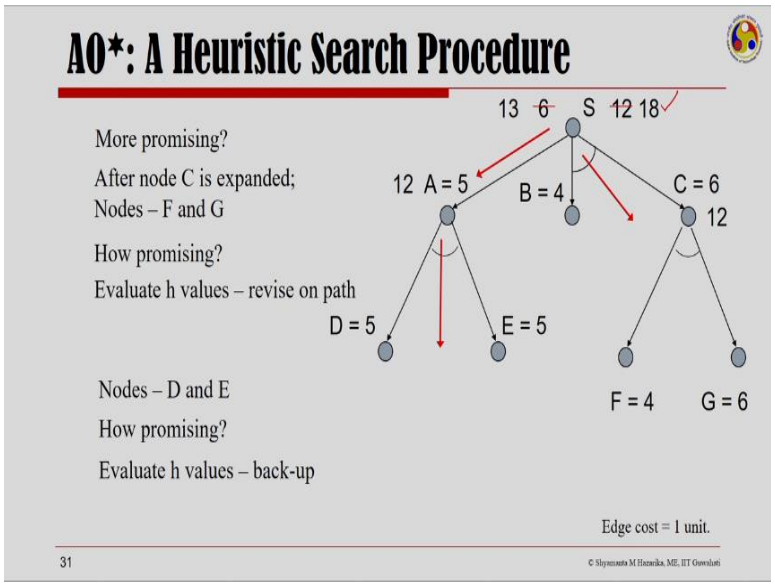
So, let us look at an example here. We have an S node which has a one-connector to A and a two-connector to B C. Let us assume that the edge costs are one unit each. The graph is initialized to S. Now, after the graph is initialized to S; the question is, which hyperedge do I follow? So, we have alternatives of moving from S to A or moving along the path S to B C. We start computing the cost.

So, if I start at A, I add one edge cost this. So, I will have 6; 5 + 1, 6. If I take the two-connector option, then I have 4 + 6, 10 + 2, because of the two-connector. That gives me a value of 12. Given these 2 values, I will definitely follow the path S to A. Thereafter, I expand the node A. I get nodes D and E. Question is, how promising are they? So, I evaluate their h values which is 5 and 5 here. And I revise the value for A now.

So, I have 5 + 5, 10. And this being a two-connector, I have 2. So, I have a 12 here. The moment I have a 12 here for A. Now, this side would be 12 + 1, 13 instead of 6. And now, a little close examination will make you realize that this path no longer is promising for us.

This has an estimate of 12 and it becomes more promising. So, I will rather follow that than going into expanding D or E further.

(Refer Slide Time: 24:35)



AO*: A Heuristic Search Procedure

More promising?

After node C is expanded;
Nodes – F and G

How promising?

Evaluate h values – revise on path

Nodes – D and E

How promising?

Evaluate h values – back-up

Edge cost = 1 unit.

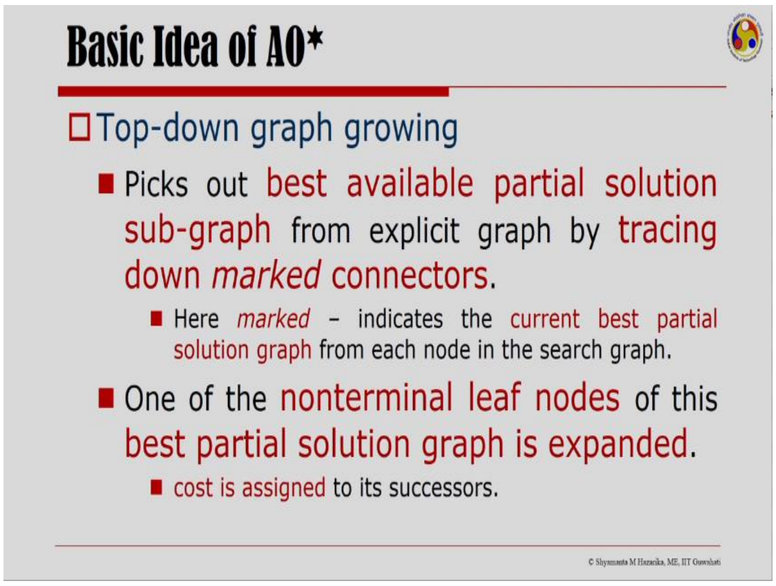
31

© Shyamanta M Hazarika, ME, IIT Guwahati

So, let us say we then expand node C and get to 2 nodes F and G. The node F has a value of 4 and G has a value of 6. Once again, we will try to get to the estimate of 6 revised vis-a-vis the new values that I have for F and G. So, that would be 4 + 6, 10 + 2 because of the connector here. That would make it 12. So, I would have 12 here. And then, I would have a revised value going up.

12 + 4, 16 + 2 because of the two-connector. And that would give a value of 18. That is, when I have expanded the nodes of C, I have now an estimate of 18 for the solution that starts with the two-connector from S. This makes me jump to the other side that I have already come up to A and the search now comes this side. So, I look at evaluating node D and E and see how promising they are. And then backup their h values. So, this is the basic idea of the first portion of the algorithm, where we looked for a top-down partially graph exploration.

(Refer Slide Time: 26:12)



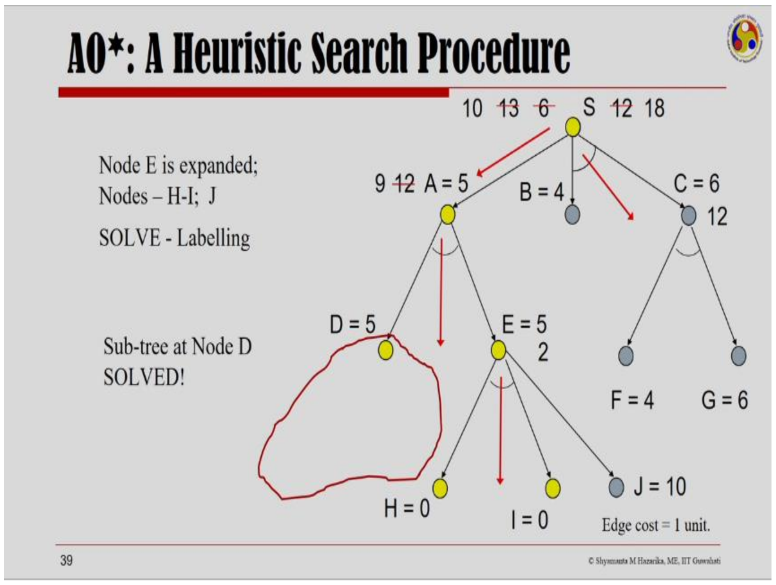
Basic Idea of AO*

- Top-down graph growing
  - Picks out best available partial solution sub-graph from explicit graph by tracing down *marked* connectors.
    - Here *marked* – indicates the current best partial solution graph from each node in the search graph.
  - One of the nonterminal leaf nodes of this best partial solution graph is expanded.
    - cost is assigned to its successors.

© Shyamanta M Hazarika, ME, IIT Guwahati

We have the top-down graph growing. So, we pick up the best available partial solution subgraph by tracing down-marked connectors. And here, I refer to marked as those connectors which indicate the current best partial solution from each node. One of the nonterminal leaf nodes of this best partial solution is expanded and the cost is assigned to its successors.

(Refer Slide Time: 26:47)



AO*: A Heuristic Search Procedure

Node E is expanded;
Nodes – H-I; J

SOLVE - Labelling

Sub-tree at Node D
SOLVED!

39

© Shyamanta M Hazarika, ME, IIT Guwahati

Now, let us look at the other portion of the algorithm, which is the bottom-up cost-revising solve-labelling portion of the algorithm. So, we continue where we left from. That is, we are right now looking at this particular path. And we have node D and E. Let us say, at this point we expand node E. Now, node E has two-connector leading to H and I and a one-connector leading to J.

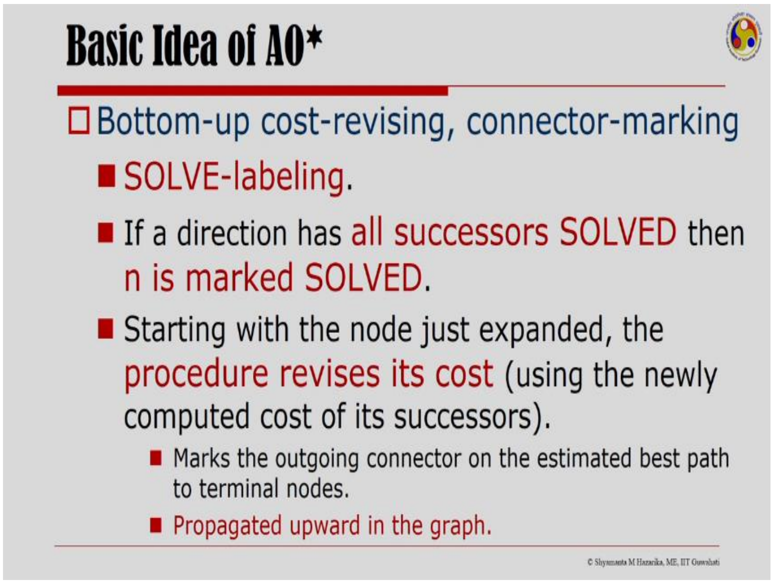
So, if you look at which way to follow at this point of time, you will realize that the cost of the two-connector is 0 + 0 + 2 which is 2. And the cost of the one-connector going to J is 10 + 1 which is 11. So, this side it is 2 and this side it is 11. So, we will rather have this path. So, that is what is done and in the algorithm. So, it follows this path. And once it follows that path, we realize that H and I are part of the solution set.

So, we realize that. And before we mark them father, we revise the cost of the different nodes from which we have followed up to H and I. So, this H I cost which is 2 makes the estimate of E being revised to 2. If the estimate of E is revised to 2, then 5 + 2, 7 + 2, 9. Estimate of A would be revised to 9. Once estimate of A is revised to 9, 9 + 1, the estimate at the top would be revised to 10.

Once this revision is over, now we will look for solve-labelling. So, we have H which is part of the solution set. We have I which is part of the solution set. Now here, I should point this out to you that I have marked the solved nodes in yellow color, so that I can see which node I am talking of as I am propagating it bottom up. So, H and I are solved. That would make node E as a solved node.

Once I have solved E, now for the sake of this discussion and illustration here, let us assume that the subtree rooted at node D has been solved. And somehow, I could mark D as solved. So, once I have marked D as solved, I can now mark A as solved. And once A is solved; because, from S, I have a one-connector coming to A. Therefore, I could mark S as solved. This is when I have the solution for the AND-OR graph that I started with.

(Refer Slide Time: 30:16)



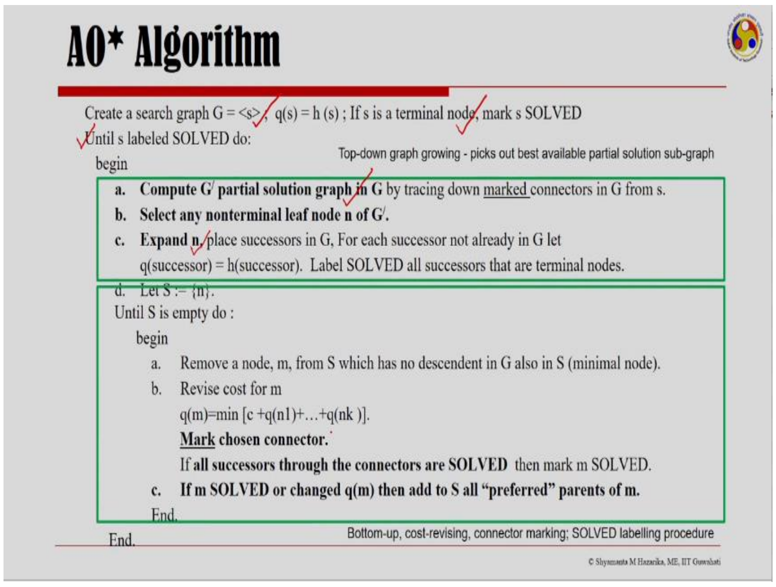
Basic Idea of AO*

- Bottom-up cost-revising, connector-marking
  - SOLVE-labeling.
  - If a direction has all successors SOLVED then n is marked SOLVED.
  - Starting with the node just expanded, the procedure revises its cost (using the newly computed cost of its successors).
    - Marks the outgoing connector on the estimated best path to terminal nodes.
    - Propagated upward in the graph.

© Shyamanta M Hazarika, ME, IIT Guwahati

So, this is the second portion of the algorithm. The basic idea of bottom-up cost-revising connector-marking solve-labelling exercise. So, in this exercise, if a direction has all successors solved, then N is marked solved. Starting with the node that we have just now expanded, the procedure revises its cost using the newly computed cost of its successors and marks the outgoing connectors on the estimated best path to terminal nodes. And this is propagated upward in the graph.

(Refer Slide Time: 30:59)



AO* Algorithm

```
Create a search graph G = <s>, q(s) = h (s) ; If s is a terminal node, mark s SOLVED
Until s labeled SOLVED do:
  begin                    Top-down graph growing - picks out best available partial solution sub-graph
    a.  Compute G/ partial solution graph in G by tracing down marked connectors in G from s.
    b.  Select any nonterminal leaf node n of G/.
    c.  Expand n, place successors in G, For each successor not already in G let
        q(successor) = h(successor).  Label SOLVED all successors that are terminal nodes.
    d.  Let S := {n}.
    Until S is empty do :
      begin
        a.  Remove a node, m, from S which has no descendent in G also in S (minimal node).
        b.  Revise cost for m
            q(m)=min [c +q(n1)+…+q(nk )].
            Mark chosen connector.
            If all successors through the connectors are SOLVED  then mark m SOLVED.
        c.  If m SOLVED or changed q(m) then add to S all "preferred" parents of m.
      End.
  End.                     Bottom-up, cost-revising, connector marking; SOLVED labelling procedure
```

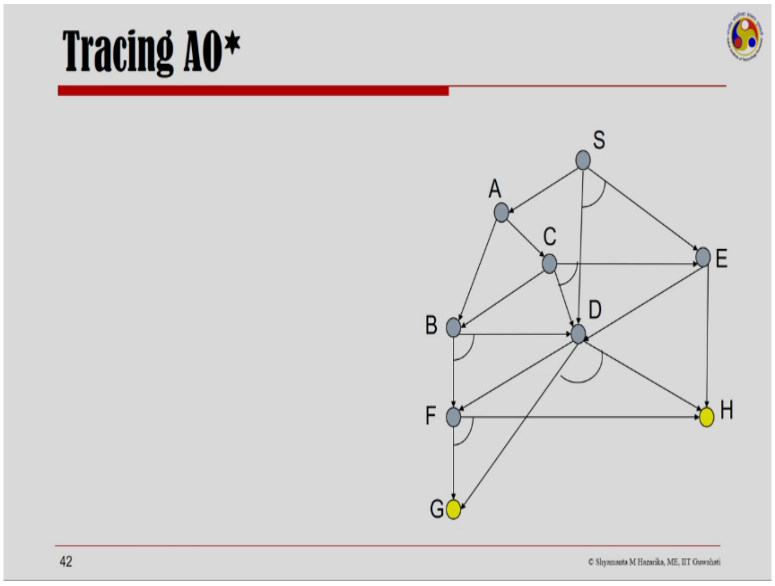
© Shyamanta M Hazarika, ME, IIT Guwahati

Now, having looked at these 2 ideas, I am in a position to formally put the AO * algorithm. One thing that I would love to point you out here is the following. We first initialized the graph to the start node. And if that is part of the terminal node, that means it is part of the

solution set. Then there is nothing to be done. I mark S a solved. If it is not, then I am looking for top-down partial solution graph.
And then, I am looking for a bottom-up solve-labelling exercise. And for this, I keep on looking at this, I keep on looping until I can mark S as solved. So, in the first portion here, I pick up the best available partial solution graph by tracing down marked connectors. How do we mark them is coming from the next portion of the algorithm? Initially to start with, I have no marked connectors and just follow one with the lowest cost.

Later, I have the marked connectors and follow the path of the marked connectors. Now, this is where one needs to understand that, unlike A * algorithm where we also keep the estimate of the cost of coming to the current node, apart from the estimate of the cost of going from the node to the solution; the AO * algorithm only talks of keeping the promise of each node. This is because, every time I am coming through the best possible way via the marked connectors.

So then, we select the nonterminal leaf node of that particular partial solution graph and expand N. The next portion of the algorithm is that I start with looking at N and I mark the chosen connectors. The chosen connectors are those which have the minimum cost currently. So, once all successors through the connectors are solved, then I mark that particular n as solved. And once we reach up to S, as we have seen in our explanation of the AND-OR graphs, we will terminate the search.
(Refer Slide Time: 33:37)



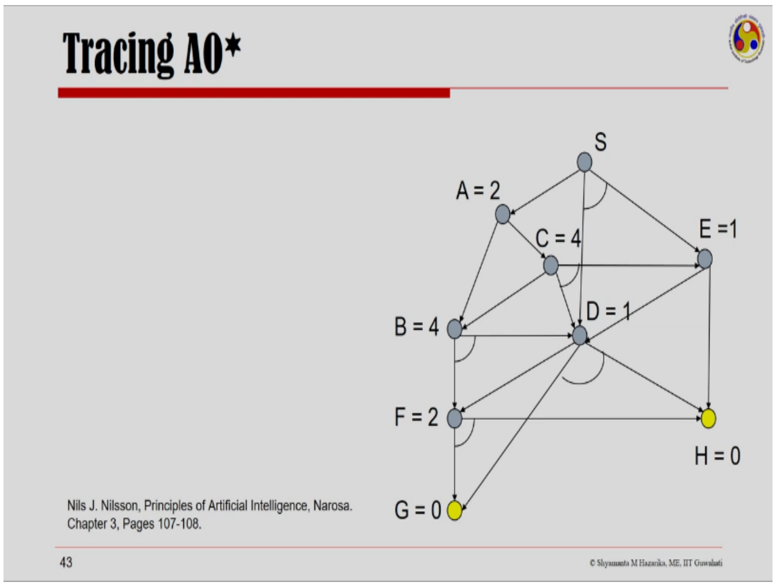
Tracing AO*

42

© Shyamanta M Hazarika, ME, IIT Guwahati

So, let us try to illustrate and trace the AO * algorithm in our particular example. So, on the right is the AND-OR graph that we have been looking at.

(Refer Slide Time: 33:56)



Tracing AO*

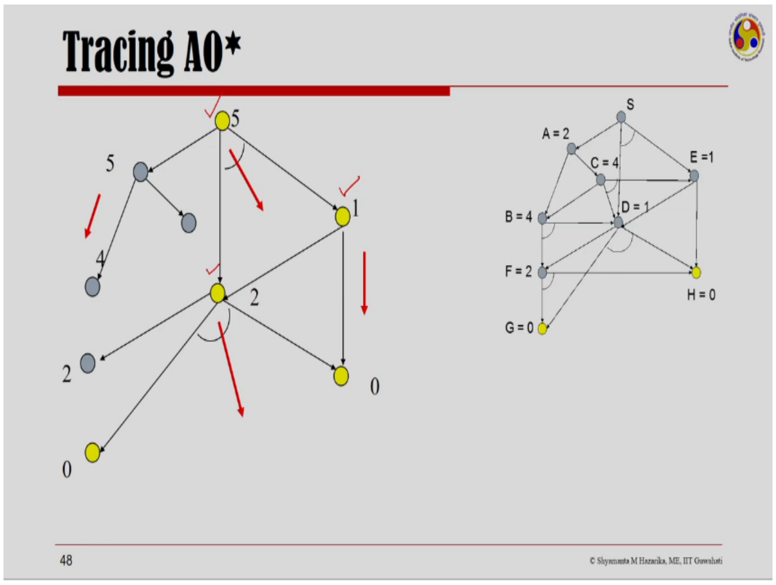
Nils J. Nilsson, Principles of Artificial Intelligence, Narosa. Chapter 3, Pages 107-108.

43

© Shyamanta M Hazarika, ME, IIT Guwahati

Let us assume a couple of heuristic values to each of the nodes. So, we initialize and start with S.

(Refer Slide Time: 34:04)



Tracing AO*

48

© Shyamanta M Hazarika, ME, IIT Guwahati

And we see that the price to the A via the one-connector is 3, whereas the price to D and E would be 1 + 1 + 2, 4. So, instead of 4 or taking that path, we take the path from S to A. Next, we expand A to get B and C. And once again compute the prices. So here, we have 4 + 1. This is now revised to 5 and this side now cost becomes 5 + 1, 6. Whereas, if you remember, we had computed the price via the two-connector to be 4.

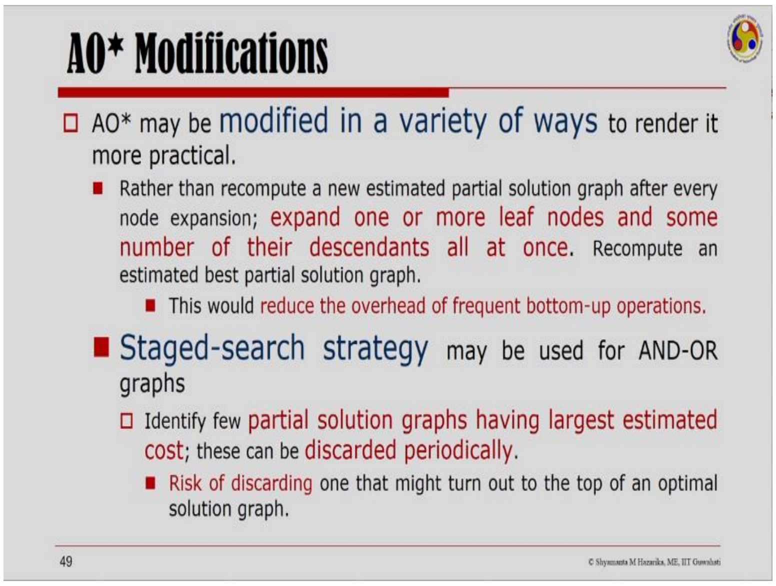
Therefore, the search proceeds via the two-connector starting at S. The next node that we expand is the node D. So, at D we have 2 options to do. One that takes us to node F and another that is a two-connector taking us to G and H. The price that one has to pay for moving to F is 2 + 1, 3. That is the price of this arc. And the price for the two-connector is 0 + 0 + 1 each for the connectors; so, 2 in total.

So, the preferred path is what is marked here. This was the forward or the top-down partial graph drawing. Now, once we know that the nodes at this end are part of the solution node; that means, I can mark D as solved, because I have G and H which are part of a two-connector and both of them are solved. So, D is solved for me. Once you have marked D solved, I can next look for expansion of node E.

So, node E I expand. And this path that is marked has a cost of 1 which I follow. Once H is marked solved, I can mark E as solved. And once E is solved and D is solved; so, I can mark my S solved. This gives me the solution to the AND-OR graph and I arrive at a solution of the given problem. So, we have looked at searching through AND-OR graphs and we have realized that AND-OR graphs are represented through hypergraphs.

AND-OR graph searching is through an algorithm called the AO * which uses some philosophy of the best-first search. But somehow brings with it the ability to deal with the AND arcs. And the AO * algorithm can be best looked at as 2 intertwined processes top-down graph-growing process that arrives at partial solution graphs and a bottom-up cost-revising connector-marking solve-labelling exercise that lets us move from the solution set to the parents, one after another marking them as solved, until we reach the start node S and mark it as solved and arrive at the solution of the AND-OR graph.

(Refer Slide Time: 38:19)



AO* Modifications

- AO* may be modified in a variety of ways to render it more practical.
  - Rather than recompute a new estimated partial solution graph after every node expansion; expand one or more leaf nodes and some number of their descendants all at once. Recompute an estimated best partial solution graph.
    - This would reduce the overhead of frequent bottom-up operations.
  - Staged-search strategy may be used for AND-OR graphs
    - Identify few partial solution graphs having largest estimated cost; these can be discarded periodically.
      - Risk of discarding one that might turn out to the top of an optimal solution graph.

49

© Shyamanta M Hazarika, ME, IIT Guwahati

The AO * algorithm that is used for searching AND-OR graphs can be modified in a variety of ways to render it more practical. We can recompute a new estimated partial solution graph all at once. That is, rather than recompute a new estimated partial solution after every node expansion, we can expand one or more leaf nodes and some number of their descendants all at once.

Recompute and estimated best partial solution graph. Now, what would this lead to it, some reduction in the overhead of frequent bottom-up operations. The other modification could be what is called the stage search strategy. We should be able to identify few partial solution graphs having largest estimated costs. So, this we could keep on discarding periodically. But then, doing this as the risk; the risk of discarding one that might turn out to be the top of an optimal solution graph.

Having said that the general AO * algorithm with its 2 operations the top-down and the bottom-up intertwined gives us the best optimal solution to an AND-OR graph. In our next class, we will look at game trees and algorithms to do searching in game playing. Thank you very much.