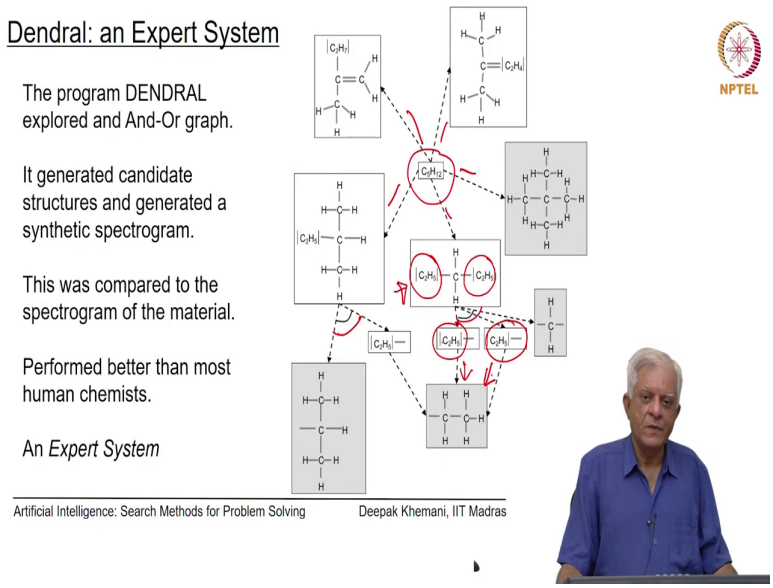
**Artificial Intelligence: Search Methods for Problem Solving**
**Prof. Deepak Khemani**
**Department of Computer Science and Engineering**
**Indian Institute of Technology, Madras**

**Chapter – 06**
**A First Course in Artificial Intelligence**
**Lecture – 70**
**Algorithm AO***

(Refer Slide Time: 00:14)

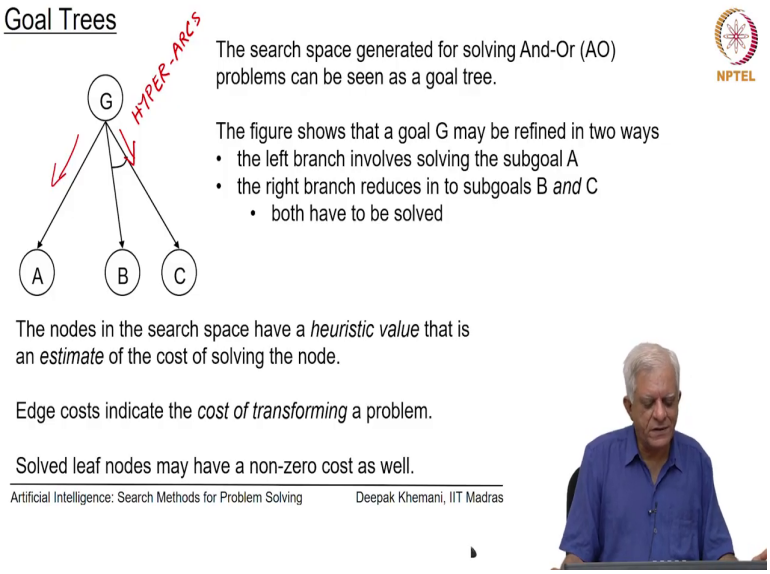



Welcome back. So, we are looking at this approach to Problem Solving which is based on decomposing the problem into smaller problems. And we saw that this process of decomposition can be captured in an abstract manner in the form of And-Or graphs or And-Or keys.

And we saw a couple of motivating examples one from symbolic integration and one from chemistry in which a program called Dendral; that is the last thing we discussed was assistant to chemist to try and elicit the structural formula sitting behind a given compound.

You know the molecular formula, for example, here you know that it is C 5 H 12, but you do not know how those atoms are arranged. And this is where general explore the different possibilities and help the chemist essentially.

(Refer Slide Time: 01:13)




So, let us now move on to how we actually solve these things and the approach to solving problems on android cases; also sometimes referred to as goal trees. We will come back to that. This is a little bit similar to what we did in backward state space planning and we will

come back to that again when we look at logic and theorem proving then backward reasoning or goal directed reasoning is a very powerful mechanism indeed.

So, the search space generated by the algorithm that we are interested in which will be called AO algorithm in fact, AO star algorithm. It can be seen as a goal tree. So, here, you can see in this figure that there is a goal G that you want to solve; and you can refine it or simplify it or decompose it in two different ways.

In the left-hand side, you can see that you reduce the problem G to a problem A and we saw examples of this when we were looking at symbolic integration that you can do that. On the right-hands branch, you see that you reduce the problem G into two sub goals; B and C and but they are joined by And edge and the semantics of that And edges that to solve G, you have to solve both B and C essentially.

So, at this point in this graph, you have basically two choices; should you reduce the problem G to the problem A or should we reduce the problem G or the goal G to goals sub goals B and C essentially, both would have to be solved essentially.

Now, to help, guide, search, we would include the notion of heuristic functions here as well like we did in the case of A star and best first search and the idea of a heuristic function is that it is an estimate of how much effort is needed in solving the problem using that particular choice.

So, here there are two choice points remember; one is that you can reduce it to A and the other is that you can reduce it to B and C together. So, we would like to associate values to help us, guide, search otherwise you would end up doing brute force search and we will do that using heuristic functions as before.

We will also edge costs to edges. This cost is the cost of transforming the problem, it is not the cost of solving the final solution, whatever the costs associated with it maybe, but it is a cost of transforming the problem that you do not want to endlessly keep transferring problems

if you can solve it quickly, the better for you because remember that one of the goals is to find solutions faster.

The SOLVED nodes or the leaf nodes may have non-zero cost as well. So, it really depends on what is the kind of problem that you are trying to solve. So, if you are doing symbolic integration for example, then if you want to do integral z square dz, then we will not associate the costs with that because we are just doing a lookup on the table. But on the other hand, if you are let us say planning to build a house for yourself, then the house as you can imagine can be broken up into sub problems.

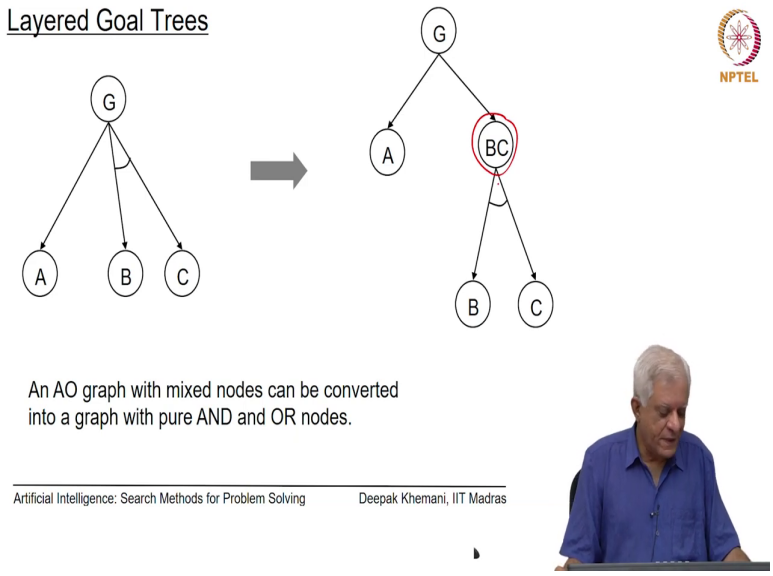
So, for example, you first construct the basement or the foundation, then you construct the ground floor; when constructing the ground floor, you construct the walls, you construct the windows, install the doors and all that kind of stuff. Every primitive action that whatever level we want to reason with will have certain associated cost. So, we will allow for SPLVED nodes to have a certain cost associated with that.

But on the other hand like we said in the cases of problems like symbolic integration, we may say that core cost is 0 so, the entire effort that we will measure in symbolic integration is how quickly you reduce it to solve node essentially which would be measured in terms of the number of edges that you have traversed and that is where we will introduce cost to edges as well.

So, they are going to be two kinds of cost one is the cost of transforming a problem and the other may or may not be is the cost of implementing the primitive solutions. Now, in this small graph, we have seen that it is a mixed kind of a graph that the goal node has two choices, we can think of them as two hyper arcs and you have to choose between one of them.

But it is if one is more comfortable, one can always break down the problem into layer problem in which a node is either an OR node or an AND node in this we cannot call G either OR or AND. We can call it OR if you think of BC as a hyper arc.

(Refer Slide Time: 06:26)

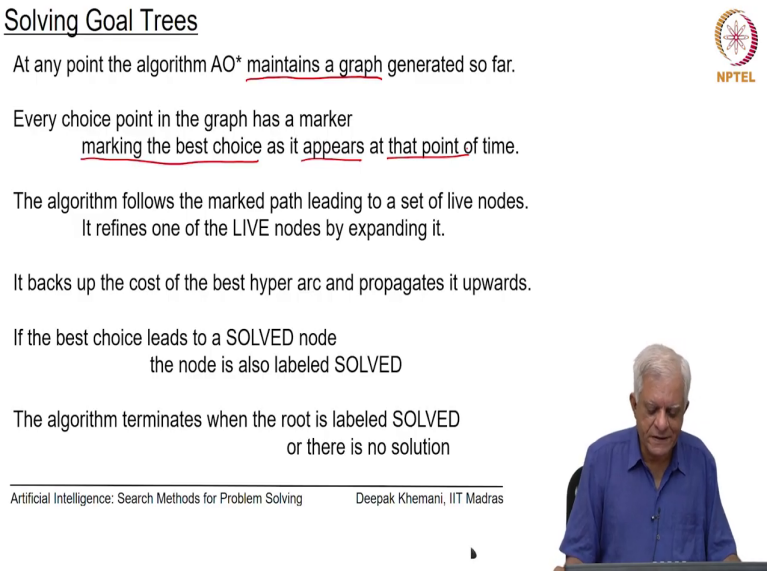



But we can also always reduce it to something which is neater or cleaner in that sense well instead of BC, we introduce a new node which we call BC which itself is broken down into this thing. So, if we do this, then you could have layered goal trees and again I would draw your attention to the game trees that we search.

Game trees are kind of layered trees. And with different kinds of activity going on at each level. So, in game trees at the max level, you have to choose one move and at the min level, you have to cater to all the moves. So, the min level who is a little bit like an AND node there; because you have to explore all possible ways of proceeding from there.

So, there are similarities as I said and but you can always break down a problem which is of mixed nature into a layer problem where alternate layers are AND nodes and OR nodes essentially.

(Refer Slide Time: 07:30)




Now, how do we solve goal trees. We said that we will be using a heuristic function to guide, search and let us look at this process. Now, this algorithm that we will call as AO star algorithm, it maintains a graph which is different from the earlier approaches where we maintain some kind of priority queue which contain the open and the closed things. The AO star maintains a graph explicitly. So, this is one thing where it is different from the other process.

Every choice point in this graph remember that we have this hyper edges or hyper arcs coming out of nodes and every choice point in this graph has a marker which tells you that as

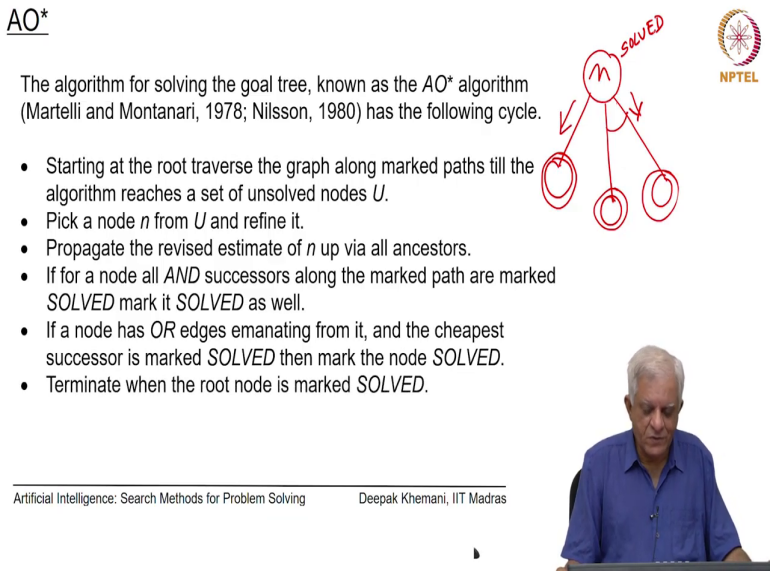
far as the analysis that you have done so far what is the best choice, marking the best choice as it appears at that point of time essentially and this is going to be used to decide as to which approach the algorithm will refine further. The markers will tell you where are the unsolved or live nodes that you want to work on.

So, the algorithm will follow the marked path leading up to a set of live nodes and it refines one of the live nodes by expanding it that is the basic idea behind the algorithm. Then, it backs up the cost of the best hyper arc and propagates it upwards essentially. They have been refined it you may have revised the actual cost a little bit like what we did in recursive best first search algorithm, you back propagate the values up essentially, but in this case of course, you keep the entire graph.

In the special case, when the best choice leads to a SOLVED nodes or a set of SOLVED nodes if it is an AND edge, then the node is also labeled as SOLVED. So, in this way, the labels SOLVED will propagate upwards into the graph again this is similar to what was happening in the SSS star algorithm where we propagated SOLVED nodes from leaves towards the root essentially.

The algorithm will terminate when the root is labeledSOLVEDor if there is no solution and again, this termination criteria is similar to what we did in s SSS star. So, as we look at the algorithm, you can compare and contrast it with SSS star.

(Refer Slide Time: 10:07)




So, let us see a small example for this algorithm called AO star and for the record it was invented by Martelli and Montanari 1978, also mentioned by Nilsson who was at Stanford, one of the pioneers in AI and one of the first textbooks that I studied and many people studied were written by an Nilsson.

So, the algorithm is as follows that starting at the root, you traverse the graph along the marked path. We said that these markers would be present till the algorithm which is a set of unsolved nodes or live nodes which we will call as U to for unsolved.

Pick some node n from U and refine it which means that you know decompose it further to or transform it to another problem. Propagate the revised estimate of n up to all the ancestors.

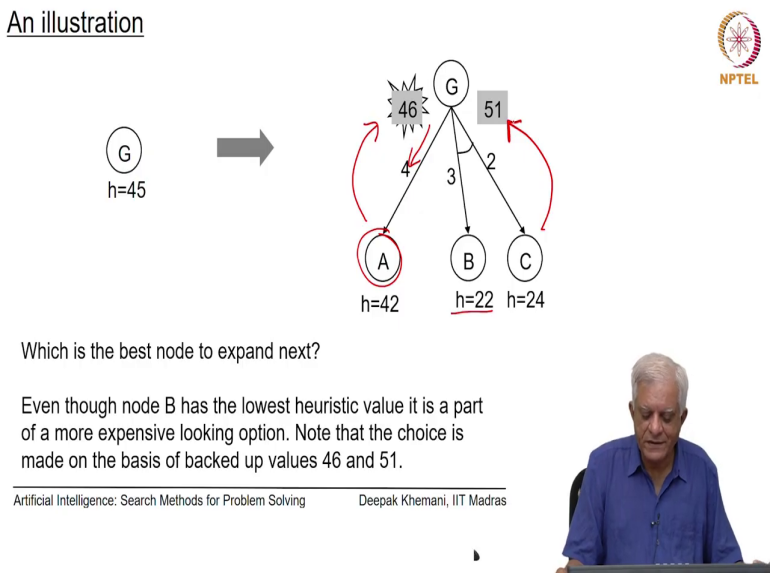
So, once you have refined your estimate of how expensive it is, you propagate the cost upwards.

For a node if all its AND successors are marked along the marked path are marked SOLVED mark it SOLVED as well and propagate it up. If it has OR edges emanating from it, then the cheapest successor should be marked SOLVED between that the best choice is leading to a SOLVED node, then also you can mark it SOLVED.

So, a node n can be marked SOLVED if the best OR choice is marked SOLVED, we will use double circles to mark it as SOLVED, then you mark that is SOLVED. Also, if which is an AND choice like this, then both its successor must be mark SOLVED.

In either case, if all the children along the hyper edge so, there are two hyper edges remember here one, here one here. If the best hyper edge is mark SOLVED, then you can mark this node n is SOLVED which means we have a label very similar to what we had in the SSS star algorithm and as we said terminate when the root node is marked as solved essentially.

(Refer Slide Time: 12:34)




So, let us illustrate this with some synthetic example that we will construct as we go along. So, let us say that you are looking at the problem, we call it G because it is a goal that you want to achieve and somehow we have estimated that the cost of solving that is going to be 45 essentially.

So in fact, this is the only node that is available to you. So, you start solving this, refine this. And let us say that you come up with two ways of solving, it one is as we said earlier that either you reduce it to A or you reduce it to B and C to be solved together. Let us assume that the heuristic values of A are 42 as shown here and B is 22 and C is 24 essentially and we are given some edge costs to that as well.

Now, the question that you want to ask is which is a next node that you should expand essentially? We have three choices A, B or C. Now, even though B appears to be the cheapest

node because it is the cost of 22, it may not be the best choice because we are looking at it from the perspective of the root node which is the goal node and from that perspective to if you were to refine the solution which is which contains B.

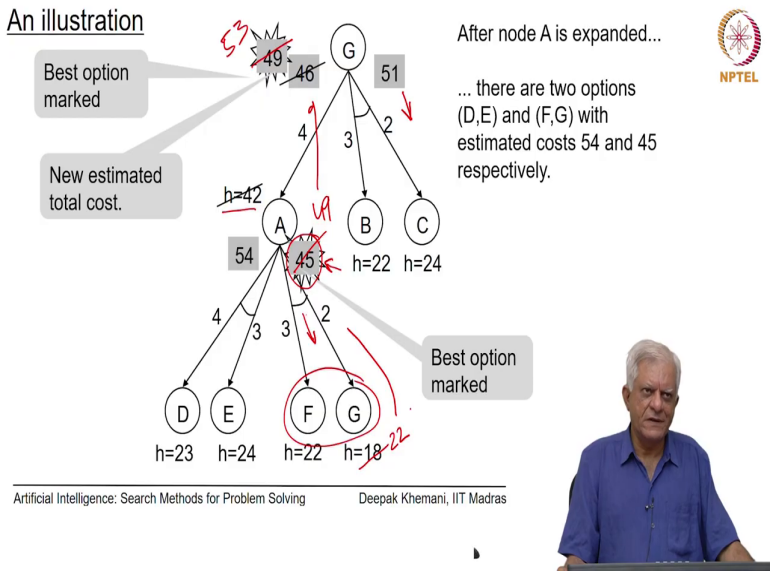
Then you would have also to refine the solution which contains C and therefore, you would have to add the cost of B and C which is 46 plus the two edge edges which is 3 and 5 that would be 51.

Whereas, if you refine the node A, it has a cost estimated cost of 42 plus edge cost of 40 of 4 and therefore, the total cost is 46. So, you can see that this is the best choice that you should, this should be marked as the best choice. in this diagram, I have marked it using this explodes explosion kind of a figure which says that this is the choice you should make essentially.

So, there are two phases as we saw in this. In the forward phase, you refine, you follow the markers and refine the best node in this case the best node is A or from the unsolved nodes. In the backward phase, you no we are not refined A as it, we have only; only created these three choices and as look at the heuristic values.

Then, in the backward phase, you have back propagated these values so, this goes to this and these choices goes to 51 and in the next cycle, now, we will follow the marker and the marker will tell us go to A and refine A.

(Refer Slide Time: 15:22)




So, let us look at that. This is where we started with. We had these two choices A and B, C let us call it BC. A was estimated to cost 46 at the root level problem would have cost about 46 and if you have chosen BC, you would have cost 51. So, we had marked A as the best choice and when we refine the node A, it presents us with some similar; some similar situation.

So, in this example, we have four nodes D, E, they form AND arc and F, G they also form an AND arc. So, there are two hyper edges coming out of this and the heuristic values as shown here D is 23, E is 24, F is 22, G is 18 and the edge cost are 4, 3, 3, 2 and the backed up cost which are the best options which is there says that F, G is the best better of the two choices and its marked by this thing which is a marker which says that next time you come in the next forward cycle from A, you must refine it in towards F and G.

So, at all stages, you mark best option. Sometimes one may mark the best option using an arrow, but it is something very similar. It is just a question of saying which is the best choice, but that is not enough. We know where to go from A, but the cost of A has changed, it was 42 to begin with and now the best choice has become 45.

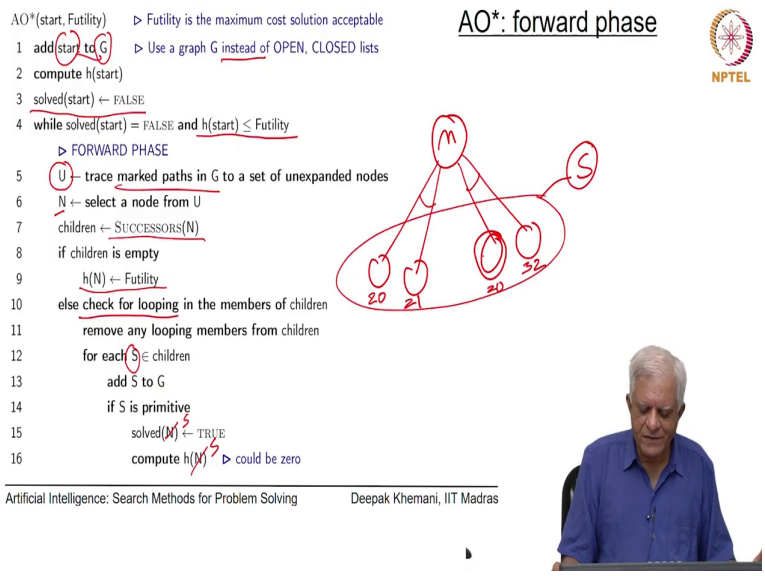
The cost has gone up, we must propagate this changed cost to the root node and when we do that, we find that the cost of root has now gone up to 49. It is still better than 52 so, it still remains the best choice.

But this is where thus graph would look like after two expansions. In the first expansion, we expanded G and we identified A as a best choice. Then, in the second expansion, we expanded A and we identified F and G as the best choice and backed up the values appropriately all the way up to the root.

Now, it is possible for example, if this instead of 18, let us say for the sake of argument this was let us say 22 so, this is plus 4 right so, then, we would have had to propagate this 22 here and this would have become actually 49. And then, we propagate this, this would have become 53 and then, the marker would have shifted to this one thing.

So, if G, the cost of G was 22, then by the time you backed up to the root, algorithm in the next cycle would have shifted away from solving node A and shift it towards solving node B and C. We will see a little bit more detail example later, but first we will look at the algorithm in a little bit more detail.

(Refer Slide Time: 18:39)




So, if you were to implement this algorithm, this is how it would look like. This is again thanks to my colleague Baskaran, who is carefully drafted this algorithm and it goes as follows. This slide shows the forward phase of the algorithm in which you have to follow the markers and then pick a node to expand. So, let us look at this.

Initially at the very beginning of course, we simply add start to the graph G. G here stand for the graph and the only node we have is a start node in the graph essentially. So, as we said, we are using a graph instead of open and close like we did in the earlier algorithms. Compute h value of start that we assume for every node that we generate, and we say that it is live, and we say it by saying that solved status is FALSE.

We could have; we could have used two labels live and solved, but this particular variation which has been published in other sources as well uses this as a Boolean variable called

solved and it is either TRUE or it is FALSE. If this case because it is not solved, it is FALSE which is equivalent to saying that it is live essentially.

So, as long as the root is not solved, we proceed and this is exactly like what we were doing in triple S star, but we have put in another condition that the estimated cost of solving the node should be less than some acceptable cost which we call is futility and futility sub cost beyond which we have say that we do not, we are not interested in a solution it could be anything, it could be a very high value or it could be tending to infinity if there is no solution even then you would label it as futility.

But basically beyond a certain cost, you are not willing to work on the problem any further essentially. So, in the forward phase, as we said we generate a set U of unsolved nodes or live nodes by tracing the marked paths in the graph. Remember that we have said that there are going to be marked paths in the graph and the set of unexpanded nodes or unsolved nodes, we call as U. From this U, we select some node N for expansion.

Now, you can give some thought to that if there is are many nodes in this unsolved node which one should you select. Now, there are two arguments to this, one is that anyway you have to solve all of them because these are part of some kind of AND node. So, it does not matter in which order you solve them, but there is another argument which people often use is to say that failure first.

That if some node is looking more expensive from the set U, then address that node first; because you will find out faster whether that solution or that branch of the solution is going to become more expensive. So, look at the difficult part of a AND arc first so that if necessary, you will backtrack early and shift your attention to another part of the tree. But of course, if the solution is going to lie along refining that part partial solution, then you will have to solve all of them so, it order would not matter.

So, you have pick this node N from the set this thing and as before we generate SUCCESSORS of N by whatever mechanism we have, if it has no SUCCESSOR, then we simply say that it cannot be solved and we label all we assign futility value to N. Futility is a

very large number beyond which we are not willing to go. So, if it is not, if it has children, then we check for saw looping essentially.

Now, remember that for example, when we did the symbolic integration problem, you could you know for example transform cot x to tan x and you could also have some tan x to cot x or you could transform tan x into something else and so on and so forth so, you could go back and forth in. We do not want to do that and for that purpose, we were introducing a notion of avoiding loops.

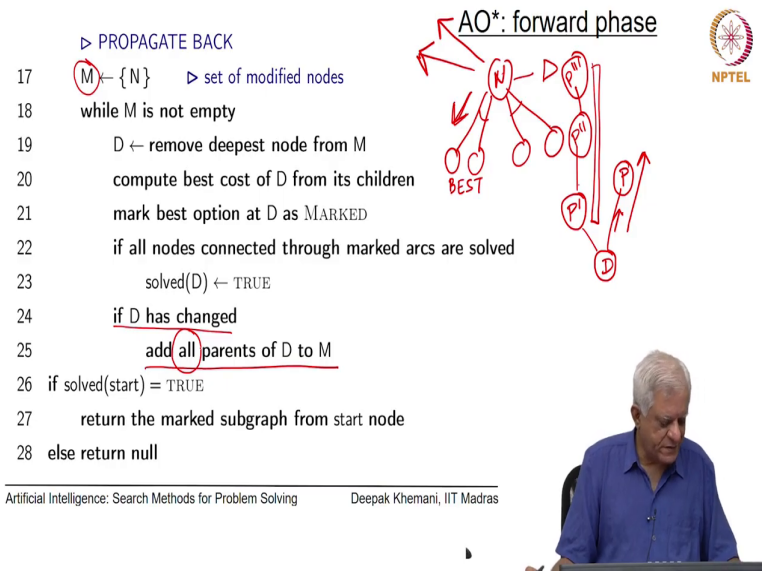
Of course like it happened in the case of DFID, if we keep looping and we keep adding edge costs to the solution, then eventually your attention will turn away, but you might as well catch looping earlier if it is possible. So, if there are any looping, then remove looping members from the children. So, now, we are left with a set of children S that you will explore.

So, what is the situation that for example, you had n and let us say you have two sets of children and this entire set of children is called S in this diagram here. So, you add this nodes S to the graph and or each of them we are calling S here, if that node is a solved nodes, then we say that solved of that node is TRUE.

So, if that node S is a solve node, we label it as solved otherwise we compute its h value. So, for example, the h value of this might be 20 and this maybe 21 and maybe this is solved and maybe this is 32 whatever and it may have a certain cost associated with that as well.

So, for example, this may be 20, we said that solved nodes can may or may not have costs associated with that. But essentially, this is what you do and having done this so, you have generated the children of the node that you picked n, each of these child we have called S and for each of these nodes S, we have computed the heuristic values and if it is labeled as solved, then you want to label it then you label it as solve as we see in the third child here essentially.

(Refer Slide Time: 25:28)




Now, comes a backward phase or the propagation phase and that works as follows; that this node N that we just expanded. So, let me draw this again. N is a node we expanded and it has two children or two hyper arcs below it and these are kind of AND edges. I do not know whether I do this yeah.

So, this node N, we add to a set called M. Now, this M is a set which are we are going to revise essentially. So, initially, we just initialize M with the set N and as long as M is not empty of course, M will not be empty because we have just added N to that.

We remove the deepest node from M in this case there is only 1 node. When we call it D so, now, this node N, we are calling D in this algorithm. Compute the best cost of D from its children. So, D has this 4 children and whichever is the best cost, we compute that and back it up to N essentially. So, if the so, this is the best, then we put a marker saying that this is the

best choice for the node N. And we have seen that if the best choices were marked solved, then we would also mark D as solved essentially.

Now, if D as change which means what that either its heuristic value has changed or is status of being solved or live as changed, then if D has changed, add all parents of D to M essentially. So, this is where we start propagating the change upwards towards the root essentially.

There is an argument which you should think about which says as to why should you (Refer Time: 27:38) propagated to all parents and the idea is basically like this that if for example, if this was the D which is changed and D was generated as a child of let us call it P and of course, we need to propagate the change to P, but supposing there was another P prime which was also a parent of P and P double prime which is the parent of P double prime and let us say P triple prime which is the parent of P triple prime.

So, if you were to only propagate it from the parent that we came, then these ancestors would not reflect the change in the value of D. So, D had a heuristic value to start with when it was generated, but after D was expanded and its children were generated, its value changed and it is this change that we are trying to propagate.

So, if you do not propagate it to other branches, then it is possible that the algorithm may have a mistaken notion of what it is like and it may unnecessarily either explode that branch which is this P prime, P double prime, P triple prime or worse it might ignore that branch you know if you are not realize that it is becoming better essentially.

So, for that reason, this all is a necessary condition for algorithm to be correct and as we said, if we have labeled the start node as solved, then we can terminate and return the marked sub graph as the solution exactly like we did in the case of triple S star.

So, this we saw in two parts. Here is this whole algorithm just for the sake of completeness I have put it up on one slide here and this is the algorithm AO star which is also very popular algorithm.

In the next session, we will look at an example of AO star and we will also talk about the fact that the algorithm is admissible only when the heuristic function unless it makes the actual cost. We will not go through the details argument as we did in the case of A star, but the rest of the argument would be similar. So, we will do that in the next session and see you then.