

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

Chapter – 08
A First Course in Artificial Intelligence
Lecture – 55
Game Playing
A Cluster of Strategies

(Refer Slide Time: 00:14)

Influence analysis

$\beta = \min(\beta_3, \beta_2, \beta_1, \dots)$
 $\alpha = \max(\alpha_3, \alpha_2, \alpha_1, \dots)$
 continue as long as $\alpha < \beta$

Node J will influence the root if
 $\alpha_4 < \beta_3, \alpha_4 < \beta_2, \alpha_4 < \beta_1$
 and
 $\alpha_4 > \alpha_3, \alpha_4 > \alpha_2, \alpha_4 > \alpha_1,$

courtesy Judea Pearl

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

So, welcome back. The last time we were looking at this algorithm called alpha beta, pruning for game playing. It is one of the most popular algorithms which is used for many board game applications. Let us do a little bit of a recap of that and also study some of the properties of that alpha beta algorithm.

So, if you remember, we had said that if we are evaluating a particular node which is J and remember that the depth first algorithm is sweeping this thing and going towards the right from left to right.

So, everything that you see on the left hand side of the paths that we have seen that is alpha 4 beta 3 alpha 3 beta 2 alpha 2 beta 1 and so on are partial values that the alpha nodes have got and the beta nodes have got.

Remember, that alpha nodes are max nodes and they are lower bounds on their the values that they contain are the lower bounds, because max nodes are trying to maximize the value and likewise beta nodes a upper bounds on the values of that on that node essentially.

So, the question we were asking is should we continue exploring node J or not and remember that by when we say that we mean shall we continue the recursive call to J or not. In this case we have looked made a couple of calls to J and we have got some value alpha 4. When will alpha 4 be meaningful in the game, when will it influence the root essentially.

So, we had observed that that node J will influence the root. Whenever the alpha value which is alpha 4 is smaller than beta 3, smaller than beta 2, and smaller than beta 1, because only then will it get selected and likewise it should be greater than alpha 3, greater than alpha 2, and greater than alpha 1, because only then this particular value would be selected.

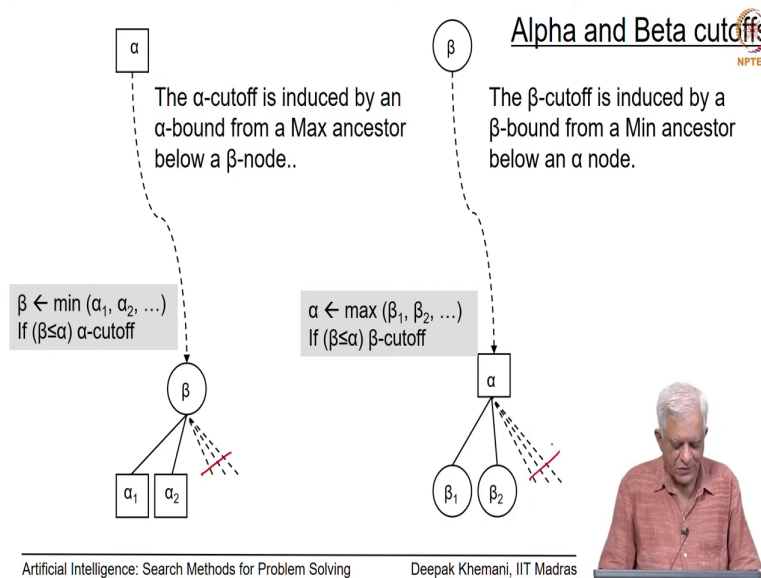
So, if this is to be meaningful it must satisfy all these properties. So, we can put this property succinctly as follows that we look at the beta bound which this node J gets and this beta bound is the minimum of the value of its beta ancestors which is beta 1, beta 2, beta 3 likewise we have an alpha bound which is the maximum of the value of this alpha ancestors which is again alpha 1, alpha 2, alpha 3 and so on.

So, we should first of all continue searching below J only as long as the alpha bound it gets is greater than, smaller than the beta bound that it gets essentially. In other words, we can think of a window where the lower end is alpha and the upper end is beta and every recursive call

gets these two bounds; alpha and beta and it is meaningful to search only as long as the window is open, which means that beta is greater than alpha.

So, now, when we are exploring a beta node that beta node will try to reduce the bound a little bit, because it is looking for lower values and whenever we are looking at alpha nodes, alpha nodes will try to increase the bounds. So, that is of course, welcome for us because it takes us to better values of the game, but we must stop to exploring if this window closes, which means alpha has become greater than or equal to beta greater than or equal to beta.

(Refer Slide Time: 03:57)



And so, this was formalized as follows that the alpha cutoff is induced by an alpha bound from a max ancestor below a beta node. So, whenever we are developing this beta node, if that beta value which is the minimum of alpha 1, alpha 2, and so on is becoming smaller than alpha bound that it get then we introduce a beta cutoff, when the beta cutoff happens here.

Likewise if you are exploring an alpha node a beta cutoff happens below an alpha node and it happens, because of the influence of a beta ancestor and if the alpha value which is looking for higher and higher values, if it becomes larger than beta then a cut off works and this is called the beta cutoff essentially.


(Refer Slide Time: 04:49)


ALPHA-BETA(N, α, β)

- 1 if N is a terminal node
- 2 return eval(N)
- 3 If N is a MAX node
- 4 for each child C of N
- 5 $\alpha \leftarrow \max(\alpha, \text{ALPHA-BETA}(C, \alpha, \beta))$
- 6 if $\alpha \geq \beta$ then return β
- 7 return α
- 8 else $\triangleright N$ is a MIN node
- 9 for each child C of N
- 10 $\beta \leftarrow \min(\beta, \text{ALPHA-BETA}(C, \alpha, \beta))$
- 11 if $\alpha \geq \beta$ then return α
- 12 return β

Initially $\alpha = -\text{Large}$
 $\beta = +\text{Large}$

The AlphaBeta Algorithm





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

So, we had looked at the game tree and the alpha beta algorithm. We initialize alpha and beta to be minus large and plus large which in some sense means that the window is completely open and as the game progresses alpha nodes will find higher values and beta nodes will find lower values and the value will become narrow and narrow essentially.

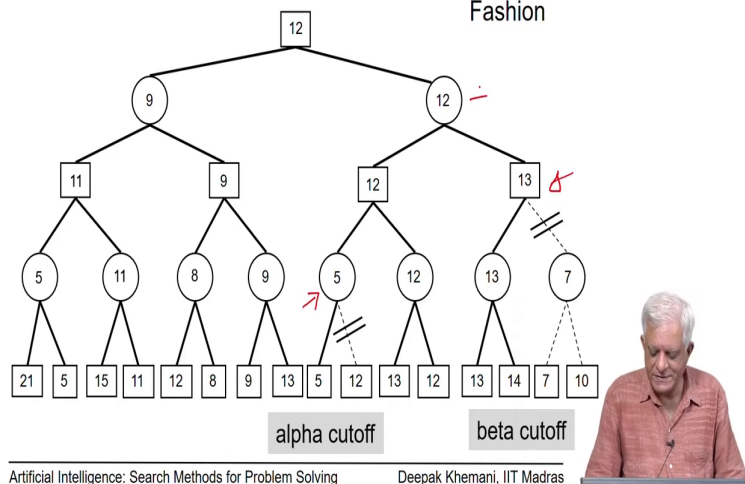
And we had seen through this process that in this recursive version of this algorithm that for max nodes for each child we take the maximum of alpha which is the bound that it is got from above and the recursive call value for the child which is alpha C alpha beta of C alpha

beta and keep choosing the maximum as long as alpha remains smaller than beta. The moment alpha becomes greater than beta we do a cut and return the value of beta and similarly for a min nodes.

(Refer Slide Time: 05:48)

A small example

Both Minimax and AlphaBeta search in a Depth First Fashion



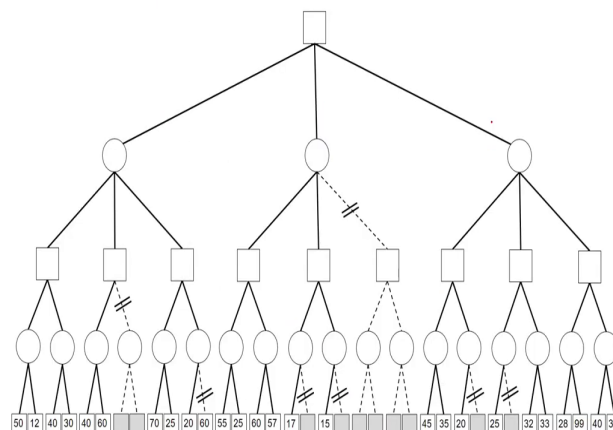
So, we had seen this in the last class. We also had seen a couple of small examples. So, let us quickly kind of run through them and we had seen that in this small game tree they were to cut offs; there was one alpha cut off which was below the beta node with a value 5 here and this was influenced by the root value of 12, because the root is assured of a value of 12, it will not be interested in a value which is less than 5 which is what this beta node is threatening to do.

Likewise there was a beta cutoff below this alpha node with a value 13 and the reason for this was similar that this node was looking or likely to become either 13 or more, because 13 it

has already got from the left child and its parent beta node which is the value 12 is looking for a value smaller than 12 so; obviously, there cannot be no agreement between them and there is a beta cutoff then we had seen a larger tree which had 36 leaf nodes and we saw that in this particular example there were 26 nodes that alpha beta has inspected out of 36.

(Refer Slide Time: 06:48)

A larger game tree: AlphaBeta Search



AlphaBeta inspects 26 of the 36 nodes

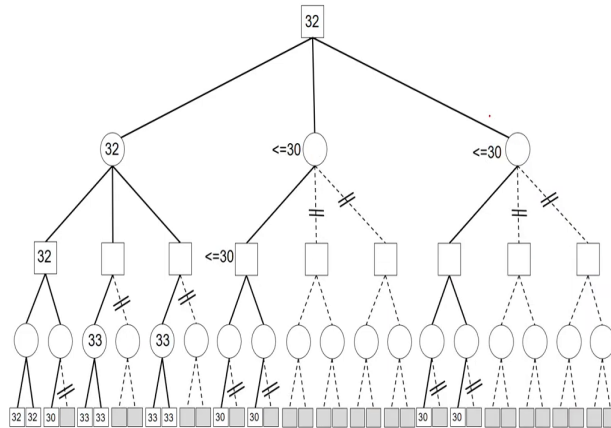


So, 10 nodes had not been seen, but one thing that we want to kind of emphasize is that the number of cut offs which take place is also dependent upon the order in which you inspect the nodes. Now, remember that this is a game tree which means for example, at the top level max has three choices. So, let us call them abc the fact that we draw a first and be second and C third has no significance.

We could have drawn the game tree in any order. So, we could have actually permitted the children or the sub tree of any sub tree by rotating it across the vertical axis and we would

(Refer Slide Time: 08:38)

An extreme example of pruning



AlphaBeta inspects 11 of the 36 nodes



Now, a question that you might ask is what is the maximum number of cut offs which can take place in any given tree and to look at this question here is a example of a contrived game tree in which the values are chosen such that at every possible juncture there is a cut off and you can see that in this contrived game tree where the values are just 30 and 32 and 33 chosen, so that the cut offs are maximized. You can expect to visit as little as 11 of the 36 nodes. So, clearly it depends upon how those nodes are placed essentially.

(Refer Slide Time: 09:12)

AlphaBeta: Reordering move generation

AlphaBeta pruning *depends on the order of move generation.*

Having called k-ply search, *while waiting for opponent's move*, one can reorder the moves for the *next* call to k-ply.

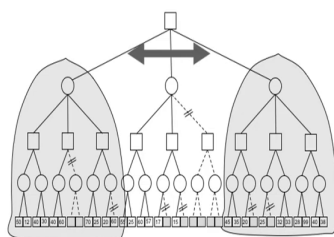
The MiniMax value *does not change.*

The same value is found, *but faster.*

GAME-PLAY(MAX)



- 1 while game not over
- 2 call k-ply search
- 3 make move
- 4 get MIN's move



Now, if you remember our high level game algorithm which calls alpha beta, in this case we have called it k ply search, but this k ply search could have been minimax or it could have been alpha beta or it could have been another algorithm that we will study, but once we call this k ply search we make a move and then we wait for MIN's move or the operands move to come.

People have asked that in tournament conditions can we use the time that we get while the opponent is thinking to do something meaningful and one of the thing that has been proposed is as follows that, because the alpha beta pruning depends on the order of move generation and if you look at a few examples you will realize that if the best moves are found early then there is more cut off and if the best moves are found late then there are less cut offs.

So, if you have identified what are the best moves and you would have done this during your k ply search then you could do the following. That having called k ply search while waiting for opponents move one can reorder the moves for the next call to k ply, because again you are going to call k ply, again remember that this game playing cycle goes into loops.

You call k ply, you make a move, you wait for the opponent move and during this waiting time you can do this little bit of extra work of reordering the moves which so that next time the game tree would be, your algorithm will perform faster. The minimax value that you will get, because of this reordering does not change the minimax value. The value is still the same, but the difference is that it would be found faster.

So, what do we mean by reordering? For example, these two sub trees that I have highlighted you could have exchanged them looked at the right subtree first instead of the left or in other words you could have drawn them such that you know they are rotated about the y axis and this is something that is done by game playing algorithms.

(Refer Slide Time: 11:19)



Next

A Best First Version of a Game Playing direction

An algorithm with a sense of direction



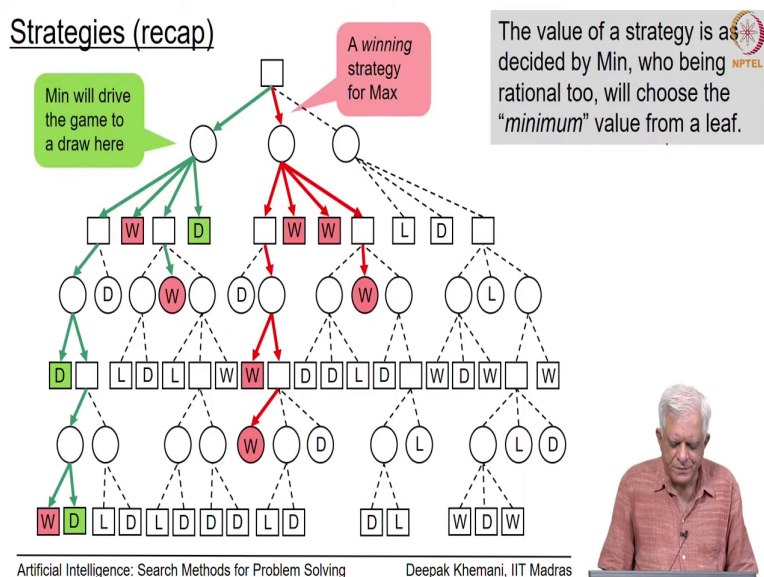
Artificial Intelligence: Search Methods for Problem Solving

Deepak Khemani, IIT Madras

Then we had said that our interest is in looking at an algorithm which is best first version of game playing. So, if you remember that when we were looking at basic search, the idea of best first search or heuristic search was that of all the options available you should choose that which seems to be the most promising in some way and in some sense that an algorithm has a sense of direction.

Now, our alpha beta algorithm does not have a sense of direction in the sense it always searches from left to right in the way that we have defined the algorithm. We did see that if you reorder the tree maybe its performance can change, but instead of reordering the tree can we look at an algorithm which will go towards the best moves initially.

(Refer Slide Time: 12:14)



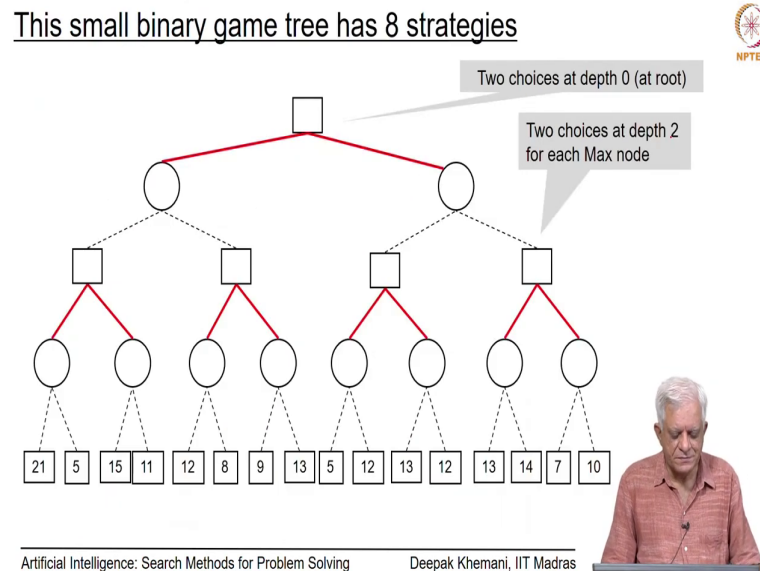
So, let us go to that algorithm. To do that, we have to go back to this idea of strategy that we had discussed earlier. So, remember that a strategy for max and we are writing the program for max is a subtree of the game tree where we choose one child for a max node and all children for a min node and we continue doing that till we reach leaf nodes.

So, in this diagram we had studied earlier, we have shown two strategies; one is let us say a red strategy in which all the leaves are labeled with W nodes. So, it is a winning strategy, because min has no choice, but to choose one of these winning leaf nodes essentially.

Whereas, the strategy on the left which is a green strategy has a mix of win and draw nodes and clearly, because max has chosen, max has choices min is the one which is left to choose, min will drive the game towards a draw in this particular strategy if max were to say that this

is my strategy. So, clearly the task of max is to find a good strategy and this is the approach that we will take in the algorithm that we are interested in.

(Refer Slide Time: 13:28)



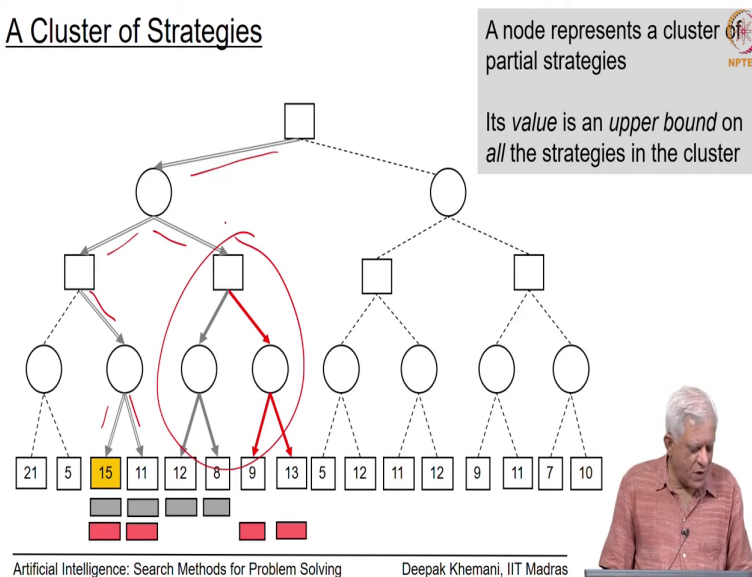
Now, one thing that we want to do is to cover all possible strategies, because we want our algorithm to be complete in the sense it must give us the minimax value and it must tell us what is the best move to be made.

If we consider that small binary tree that we had worked with, it has 8 possible strategies and this can be computed as follows that at the max level in this case there are two choices. So, at the root level there are two choices, then at the third level, because the strategy for min will have both of these things again you have two choices.

So, you have 2 into 2 on the left hand side and 2 into 2 on the right hand side. So, in this total is 4 plus 4 which is 8 and so, I will leave this as a small exercise for you to think about that if you have a k ply look ahead search tree with branching factor b, how many distinct strategies exist in that game tree. As we have seen in this small example which is a 4 ply with branching factor to search tree there are 8 strategies and the task of max is to pick the best of these 8 strategies.

(Refer Slide Time: 14:53)

A Cluster of Strategies



So, how do we do that let us explore that next. The first thing to observe is that any node, any leaf node that we select it represents a cluster of partial strategies. So, this node with a value 15 that we have seen you can see that its part of two strategies one we can call as a gray strategy and the other as a red strategy. The double arrows that you see here, they are common

to both the red and the gray strategies and the two strategies differ on this part of the this part of the tree essentially.

So, if we choose one node any arbitrary node in amongst the leaves in a game tree that node represents what we call as a cluster of strategies or its these are partial strategies, because they have not been completely specified. Only one node in that has been specified.

Now, if we remember that the value of a given strategy is the minimum of the value of all the leaves in that strategy. You can also observe that this value which is 15, which is the only node that we have so far looked at is an upper bound on all the strategies in this cluster.

In this particular cluster there are two strategies. So, both these strategies cannot have a value greater than 15, because of the fact that the value of a strategy is a minimum of the value of its leaves and if we know one leaf then you know that it cannot become higher than that.

So, that leaf becomes an upper bound. So, what do we want to do? We want to construct all possible strategies, cover all possible strategies. So, essentially we want to identify enough leaf nodes so that we will cover all strategies.

Now, we have seen in this example that one leaf node covers two strategies. So, and since we know that there are 8 strategies in this tree, we can imagine that we will need 4 leaf nodes to do that. So, how do we do the initial clusters?

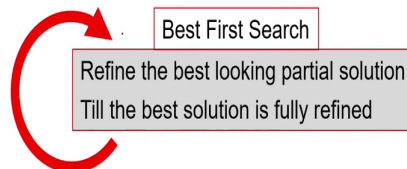
(Refer Slide Time: 17:03)

Algorithm SSS*



Devised by George Stockman in 1979

A Best First approach to searching the game tree



A solution in a game tree is a *strategy*

A *partial solution* is a *partial strategy*
and stands for a *cluster of strategies*



This is an algorithm called SSS star. It was devised by George Stockman in 1979, relatively much later in the game playing world where alpha beta was discovered many-many years ago.

It is a best first approach to searching the game tree and it works as follows which is our idea of heuristic search in general is that refine the best looking partial solution till the best solution is fully refined.

This is the theme that we have been following in many algorithms that we have studied essentially. In the in the case of games, a solution in a game tree is the strategy and a partial solution is a partial strategy, which means that if you have only specified some leaves in the strategy then it is a partial strategy and every partial strategy stands for a cluster of strategies, because the other leaves can be filled in some different ways. So, ours theme of SSS star is

refine the best looking partial solution till the best solution is fully refined and the space in which we search must cover all possible solutions.

(Refer Slide Time: 18:20)

Algorithm SSS* searches in the Solution Space



Algorithm SSS* first defines clusters to cover *all* strategies
- necessary for completeness

The procedure for constructing the initial clusters is as follows

Start at the root
Repeat
 At the Max level choose *all* children
 At the Min level choose *one** child
Until the horizon is reached

* Without loss of generality we always choose the leftmost child



Now, algorithm SSS star searches in the solution space, in the space of strategies and it does the following. You first define clusters to cover all strategies. So, remember we said there are 8 strategies in our small tree and we need to cover all those 8 clusters and we had kind of guessed, then we need 4 leaf nodes to cover those 8 clusters.

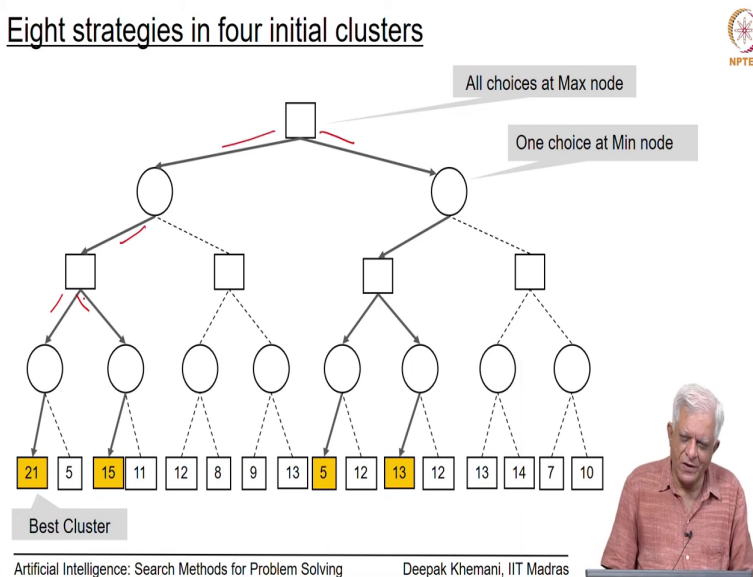
Those nodes can be found as follows and there is a procedure for constructing the initial clusters which says that start at the root and repeat the following that at the max level choose all children, because you want to cover all possible strategies and at the min level choose one child, because you just want to have a partial strategy and you do this till the horizon is reached essentially.

This one child that we choose for min can be any child, but without any loss of generality we will always choose a leftmost child. Now, this algorithm is for constructing the initial clusters.

It is not specific to the tiny game tree that we had seen. For any game that you are implementing, this is a procedure that you will need to follow to construct the initial clusters that at the max level choose all children and at the min level choose one child and we have said that we will choose always the left child without any without creating any difficulty.

(Refer Slide Time: 19:44)

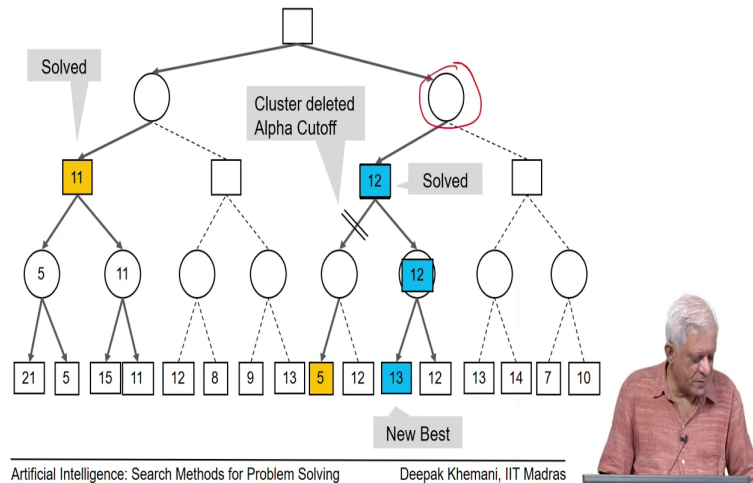
Eight strategies in four initial clusters



So, in our tiny game tree we follow this procedure and you can see that we would have constructed this, identified this 4 leaf nodes shown in orange here ah. So, you can see that at the top level we chose both the branches for max, at the min level we chose only one branch,

(Refer Slide Time: 22:03)

Refine the best partial strategy



So, we do the same thing. With this node we refine it; our previous cluster has now got merged into one cluster. So, when we started with 4 clusters, now we have only 3 left and when we refine this 13 node its sibling is 12 and it gets solved and in a similar fashion its neighboring cluster which had a value of 5 also gets deleted.

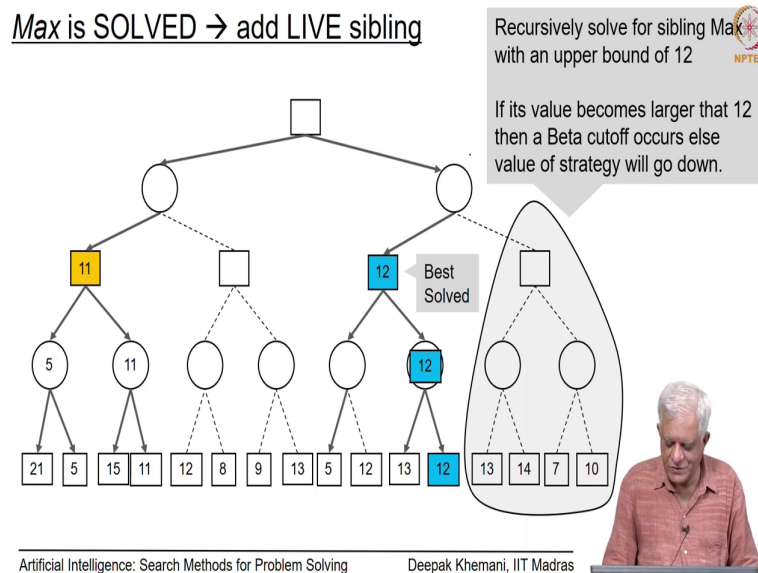
So, at this point we are left only with two clusters; one is the value 11 and one is the value 12 essentially. What is what do these values mean 11 and 12? It means that the cluster with value 11 can at most be, the minimax value can at most be 11 and on the right hand side cluster it can be at most 12.

So; obviously, we prefer the right hand side cluster and now we refine, we have to refine that cluster, but refinement now follows the slightly different path we will formalize this algorithm a little bit later, but the basic idea is that if you look at the min parent of this max

cluster, in the cluster that we have constructed this is the only child which exists, but they have they, but it has other children.

In this case there is only one other child and that other child could have a lower value or a higher value, we need to investigate that. If the other child had the lower value than the value of this strategy will go down, if the other child has a higher value then the value of the strategy will remain 12.

(Refer Slide Time: 23:33)

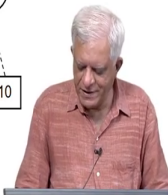
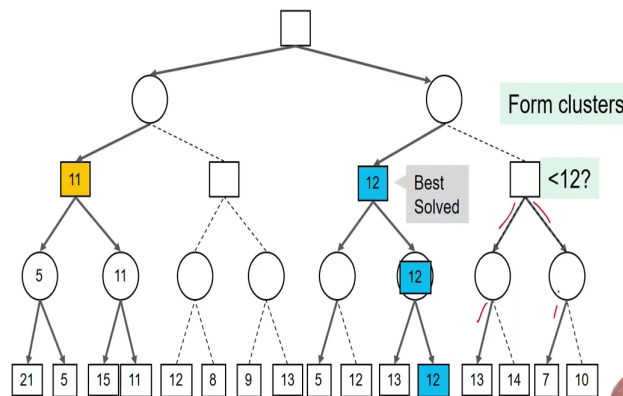


So, let us explore that. So, since this is the best solved node and it is a max node. What we do next is to add the sibling of this max node as the next problem to be solved. So, in other words we make a recursive call to SSS star algorithm and the subtree that we have shown here, but this is solved with an upper bound of 12. Why this upper bound of 12?

Because if this is going to be greater than 12 then we are not interested in that, because its present is the min node, if it is going to be less than 12 then we need to find that out, because we may want to shift to the other strategy which has a value of 11 essentially. So, we do this recursively which means we again consider the subtree and make clusters and then refine the best cluster and so on.

(Refer Slide Time: 24:31)

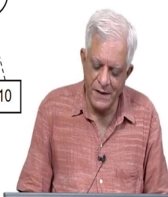
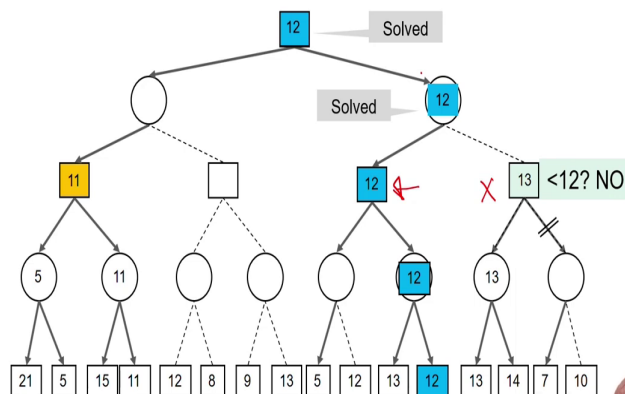
Recursively solve sibling Max node



So, this is what we are doing. Now, we are doing this sub cluster and we want to check whether its value is going to be less than 12. So, we do the same thing as before, we choose to all children for max and one child for min and then we have got the clusters. In this case two clusters; one has a value of 13 and the other has the value of 7 and we will refine the best cluster essentially and the best cluster has a value of 13 and its neighbor has a value of 14.

(Refer Slide Time: 25:36)

When SOLVED *Max* is last child of parent



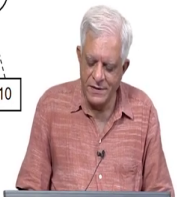
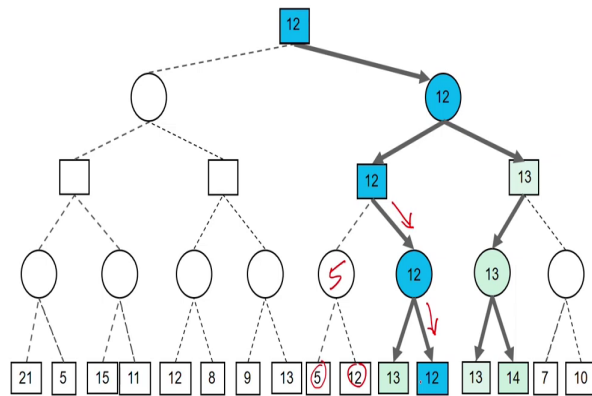
So, you can see that the value of this subtree is 13, because of the 4 children that is the minimax value which comes from this node, which is a value of 13. So, the question we were asking is that is this value going to be less than 12? The answer to that was no and therefore, we just ignore it, we discard it and we do not have any more siblings of this max cluster that we had solved earlier.

We explored one sibling, but it turned out that it was going to give us a higher value which its min parent was not going to accept. So, there are no more siblings and therefore, the min parent gets solved. Once a min parent get solved, it is solved and it is also the best which means that the other strategy which is of value 11 could not be become more than 12. So, we know that the root will get a value of 12 from this cluster and we have solved the game tree.

So, we have seen that in this SSS star algorithm the algorithm searches the tree always looking for the best cluster to refine and it terminates when the best cluster is completely refined. So, once we have a value for the root we know that it is completely refined or it is a full solution and the algorithm get terminate and this is the strategy that it returns, this is a strategy that it returns as the this thing.

(Refer Slide Time: 26:51)

Strategy returned by SSS*



The value is 12, the dark green nodes show the paths that is expected to be followed in the game ah, because min has two options; 12 or 13 and we have already seen that it is not going to be interested in 13. It will choose 12 and max has already decided that of the options that it had.

Remember that this cluster would have given it a value of 5, but now max has chosen max has choices and has already ignored that and so, max has already decided that it is going to

make this move at 12, which is what the algorithm return to us and we expect that from this min node min will choose the smaller of the two value is 12 and 13.

So, that is where we expect the game to proceed essentially. So, this was the very kind of a gentle introduction to this algorithm SSS star what we will do next in the next session is to look at the algorithm more formally and in a little bit more detail. So, we will put it down as a piece of pseudo code and look at an example which is on a larger game tree. So, we will do that in the next session.