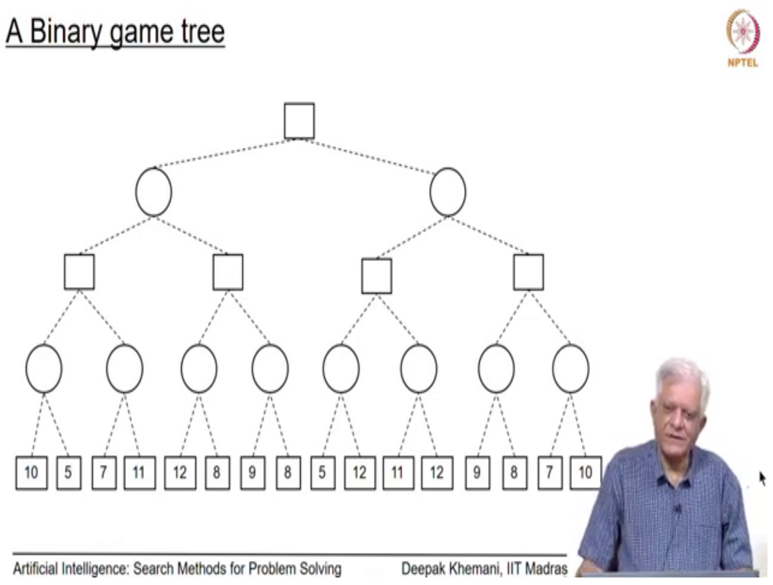
Artificial Intelligence: Search Methods for Problem Solving

Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Chapter – 08
A First Course in Artificial Intelligence
Lecture – 54
Game Playing
Algorithm Minimax and Alpha-Beta Pruning

(Refer Slide Time: 00:14)


A Binary game tree
10
5
7
11
12
8
9
8
5
12
11
12
9
8
7
10
Artificial Intelligence: Search Methods for Problem Solving
Deepak Khemani, IIT Madras

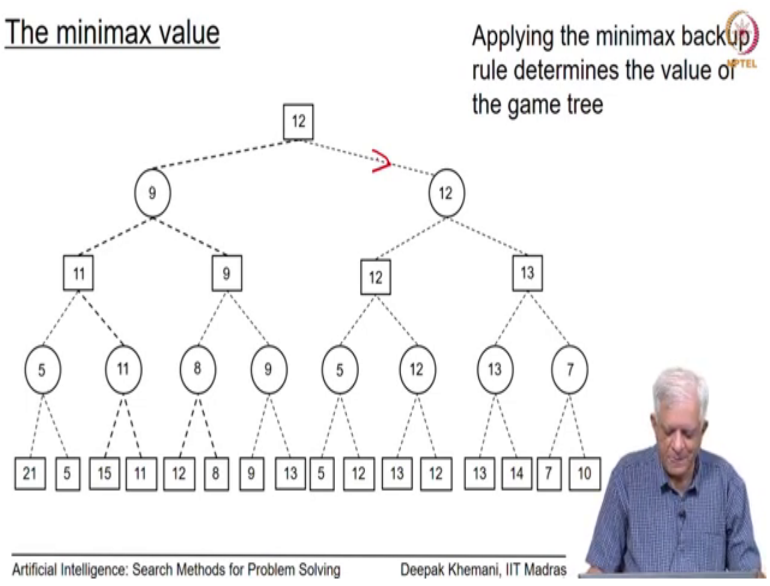
So welcome back, now we have down to the business of actually writing algorithms to play games. And we are assuming that our game playing programs will do a finite amount of look ahead and on the horizon, they would apply the evaluation function.

And then our task is to backup this is values given to us by the evaluation function; decide what is the value of the root node or the value of that game tree or the game sub tree, and based on that make a choice as to what, who Max should make essentially.

So, we have studied the minimax backup rule. So, if you remember the minimax backup rule it says that, at the min level you take the smallest value from its children; because it is a minimizing node and it looks for a smallest value.

At the max level, you take the largest value from its children; because it is a max node and it wants to drive the game towards plus 1 or whatever it has been scaled to plus large essentially. And we are also observed that, we can do this in a bottom of fashion.

(Refer Slide Time: 01:21)

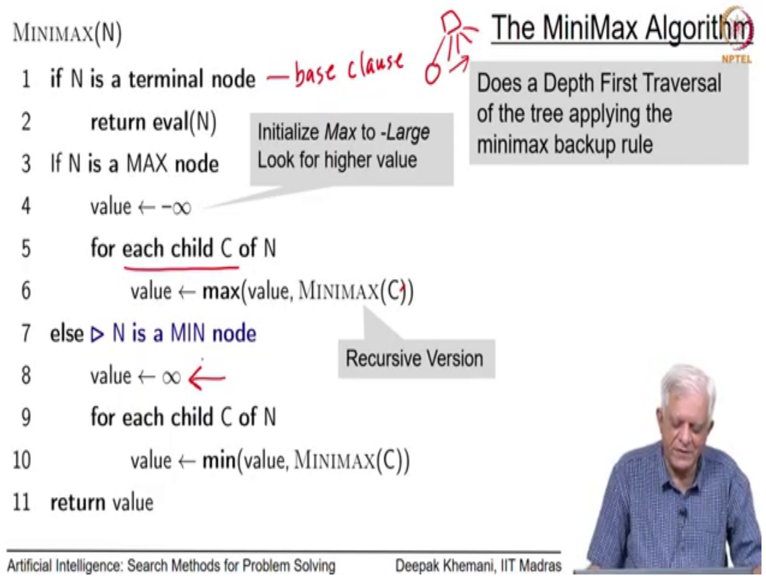

The minimax value
Applying the minimax backup rule determines the value of the game tree
12
9
12
11
9
12
13
5
11
8
9
5
12
13
7
21
5
15
11
12
8
9
13
5
12
13
12
13
14
7
10
Artificial Intelligence: Search Methods for Problem Solving
Deepak Khemani, IIT Madras

And so, this is how for this particular game tree, we back up the values essentially. So, you can see that, we are started from the bottom level; the first 5 is the smaller of the two values 21 and 5, 11 is smaller of 15 and 11 and so on.

We have finished a min row and now we are moving to the max row; 11 is maximum of 5 and 11, 9 is a maximum of 8 and 9, 12 is a maximum of 5 and 12 and so on. And in this fashion, we are moving up the tree and eventually we figure out that the minimax value of the game is 12 essentially.

And therefore, Max would make a move, would make a would make the move which from where this value comes and then wait for Min's move as we have just discussed essentially.

(Refer Slide Time: 02:06)



The MiniMax Algorithm

```
MINIMAX(N)
if N is a terminal node — base clause
    return eval(N)
If N is a MAX node
    value ← $-\infty$
    for each child C of N
        value ← max(value, MINIMAX(C))
else ▷ N is a MIN node
    value ← $\infty$
    for each child C of N
        value ← min(value, MINIMAX(C))
return value
```

Initialize *Max* to *-Large*
Look for higher value

Does a Depth First Traversal of the tree applying the minimax backup rule

Recursive Version

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

The minimax algorithm is the simplest algorithm that we can use for computing the minimax value. And here we have given a recursive version of the algorithm and we can also write iterative version. So, we know for example, that we can maintain an open list and keep adding to that. So, in the recursive version we do the following; we initialize Max to minus large and subsequently we will look for higher values.

Now, the base clause in recursion is that if it is a terminal node. And if it is a terminal node, we simply written the value given to us by the evaluation functionally. If it is not a terminal node, then we do recursive calls. So, remember that for Max, we will look at each of those children one by one.

So, recursively we will call the first child and this is what is shown here. We initialize the value of that max node to minus infinity or minus large, if you are implementing a real program.

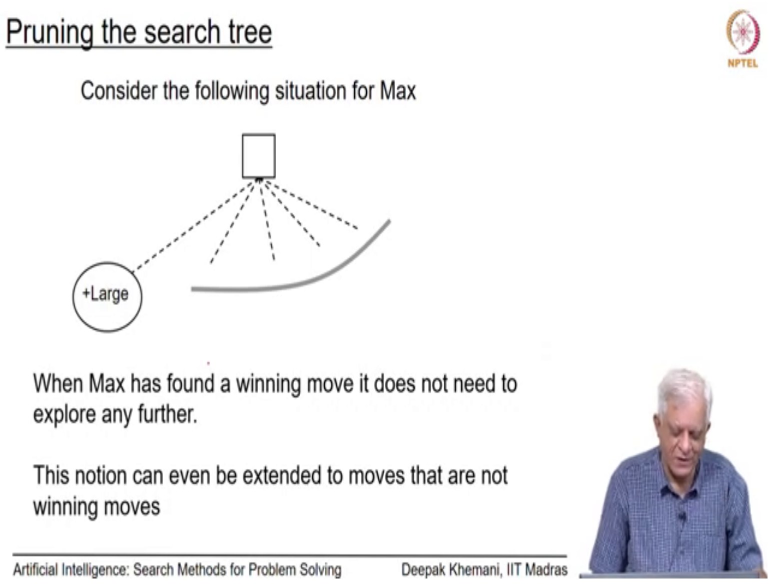
And then for each child C of N, we start with the leftmost child [vocalized-noise; we recursively call minimax the same algorithm. Compute the value of the child and Max chooses the maximum of the value that it had to start with and the value that is returned to it by the minimax algorithm recursive, the recursive call to the minimax algorithm.

Remember that we initialize max to minus infinity. So, the moment the first child gives a value, there becomes the value of the Max; and as it proceeds looking at other children, it will possibly update its values to higher values essentially.

If the recursive call was made at a Min node, we initialize it to plus large or infinity. And then we look for lower values from there onwards; because Min is always looking for a lower, the lowest possible value from all its children essentially. Now, the minimax algorithm, it does a Depth First Traversal of the tree applying the minimax backup rule essentially.

The minimax algorithm sweeps the entire tree, traverses the entire tree, in some sense blindly simply applying the minimax rule essentially. But we want to look at situations, where you may want to ignore some parts of the tree.

(Refer Slide Time: 04:51)



Pruning the search tree

Consider the following situation for Max

When Max has found a winning move it does not need to explore any further.

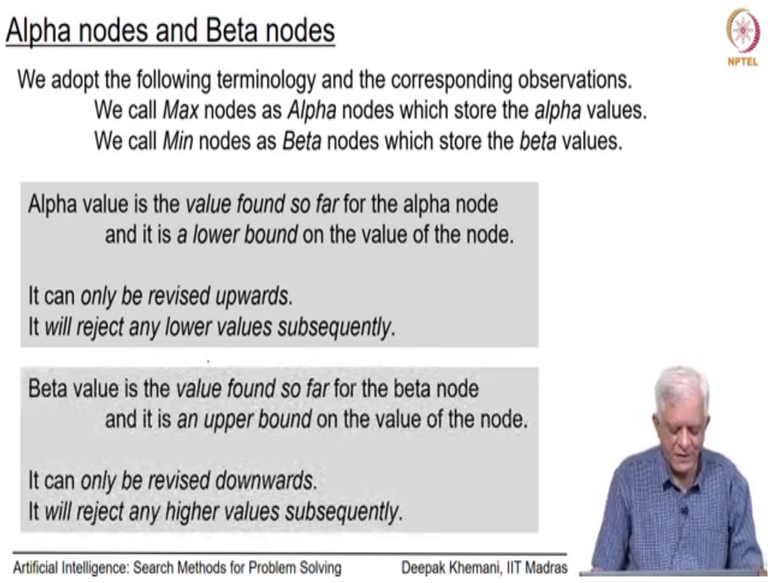
This notion can even be extended to moves that are not winning moves

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

So, this is the simplest kind of a situation that, if Max has found the value of plus large from one of the children and that is the best that Max can do, it cannot do any better than that.

So, there is no point in looking at the other children of Max. So, we could prune those children and save time by not making recursive calls to those children. So, in this case Max has found a winning move and it does not need to explore any further. But we will see that this process can be extended to moves that are not winning moves essentially. So, it does not have to be plus large for such pruning to happen, it can happen with other values and this is what we will study now.

(Refer Slide Time: 05:40)



Alpha nodes and Beta nodes

We adopt the following terminology and the corresponding observations.
We call *Max* nodes as *Alpha* nodes which store the *alpha* values.
We call *Min* nodes as *Beta* nodes which store the *beta* values.

Alpha value is the *value found so far* for the alpha node
and it is a *lower bound* on the value of the node.

It can *only be revised upwards.*
It *will reject any lower values subsequently.*

Beta value is the *value found so far* for the beta node
and it is *an upper bound* on the value of the node.

It can *only be revised downwards.*
It *will reject any higher values subsequently.*

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

Help us do this in a systematic way, we will adopt some nomenclature. So, we will adapt the following terminology and the corresponding observations with these nodes.

We will call Max nodes as Alpha nodes and they will store the value which we will call as alpha values. Remember that the algorithm minimax that we just very quickly refer to is doing a depth first traversal of the game trees sweeping from left to right and it has partial values as it goes along and keeps updating them as it does more search.

So, the partial, partially computed values will you will call as alpha values or even the fully computed values. So, Max nodes will be called as Alpha nodes and they will store alpha values; Min nodes will be called Beta nodes and they will store beta values. The alpha value is a value found so far for the Alpha node.

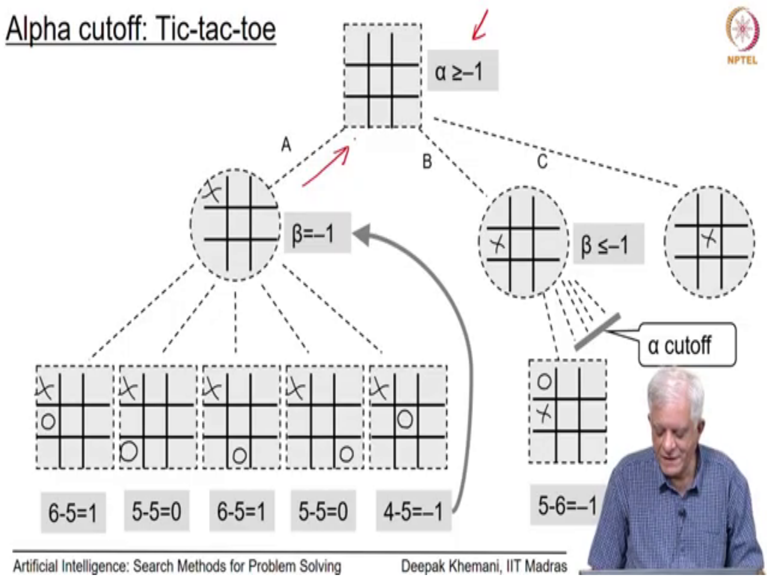
Remember that the algorithm is flipping from left to right. And it will be a lower bound on the value of the node. Why are lower bound? Because if it is got a value x from one child, left child let us say; then from any other child, it would only accept a value which is higher than x essentially.

So, any value it has found can only be increased for a Max node and that is why it is a alpha value is the lower bound for that. It can only be revised upwards and it will reject any lower values offered by any of its children subsequently. So, in that sense, we can think of the game tree as you know kind of a supply chain mechanism, where all the internal nodes are kind of traders in a sense that an Alpha node or a Max node is looking at all the suppliers and saying give me the and it will check, it will take the maximum value from the suppliers.

And the Beta node or the Min node is also doing the same except that it will choose the minimum values from its suppliers. So, once it has, once both of them have got one value, they can only look for better values; better values in the case of Max or higher values and in the case of Min they are lower values.

So, the situation for Min is similar, beta value is the value found so far for the beta node and it is an upper bound on the value of the node; because it will not going to go above that, it will only look for values which are smaller and it will reject any values which are higher subsequently.

(Refer Slide Time: 08:17)



Alpha cutoff: Tic-tac-toe
α ≥–1
A
B
C
β=–1
β ≤–1
α cutoff
6-5=1
5-5=0
6-5=1
5-5=0
4-5=–1
5-6=–1
Artificial Intelligence: Search Methods for Problem Solving
Deepak Khemani, IIT Madras

So, let us look at this simple game again Tic-tac-toe, same example that we saw a short while ago. And Max is considering making this move which is putting across on the corner square. And it is doing two ply look ahead, so it looks at each of the children one by one; remember this is depth first search which is happening.

So, first it looks at the first child and evaluates the first child; the value of the first child is 1 and you can do this as a small exercise that Max has 6 rows columns or diagonals available and Min has only 5. So, the evaluates 1. So, at this point you can say that, the Beta node which is A has got a value of 1.

So, Beta node A is only going to look for values which are smaller than 1. So, the search continues, it looks at the second node; it turns out that the second node is as you can see symmetric. So, it is not surprising that the value of that node is 0; the third node has a value 1;

the fourth node has a value 0, and the fifth node as we have seen earlier was the value minus 1.

And by the time all the children of this beta node are visited, the value of beta has become minus 1. What does this mean that? This is a value that beta will send to its parent alpha node and as you can see here; it means that alpha, alpha is going to be greater than or equal to minus 1.

Why is that? Beta, the first beta child which is A has given it a value of minus 1; the this child is fully solved and we know its value exactly and it is offering a value of minus 1 to the root node which is a Max node. So, Max has got minus 1 from this beta child. And so, it is only going to look for values which are higher than minus 1.

So, Max continues it is search in depth first fashion; it looks at the second possibility which is to place a cross on a side. And applies the and generates a first child of that node, which is let us say that Min is playing a naught in the corner and as you can see the value of that node is minus 1.

So, which means that, the value of this beta node B is going to be less than or equal to minus 1. Now, you compare these two values; the root node has a value it says I want to be greater than or equal to minus 1 and the node B, which has not yet been fully explored, only one child of B has been seen.

But we know at this point that, beta is going to be less than or equal to minus 1; because this value of minus 1 is an upper bound on this beta node and it is a lower bound on the alpha node. So, you can see that these two nodes, if you were to think of it are not going to do any business; because beta is going to look for a value smaller than minus 1 and alpha has already got minus 1 from the first child, N is looking for a higher value.

So, at this point there cutoff occurs and this cutoff is called as an alpha cutoff. Why is it called an alpha cutoff? An alpha cutoff is induced by an alpha value and it is induced below a beta node. And it is induced when the beta node is no longer going to be promising for the alpha node; in the sense that the beta value that it is going to return or offer to alpha is going to be less than the value that alpha has already gone from its left child.

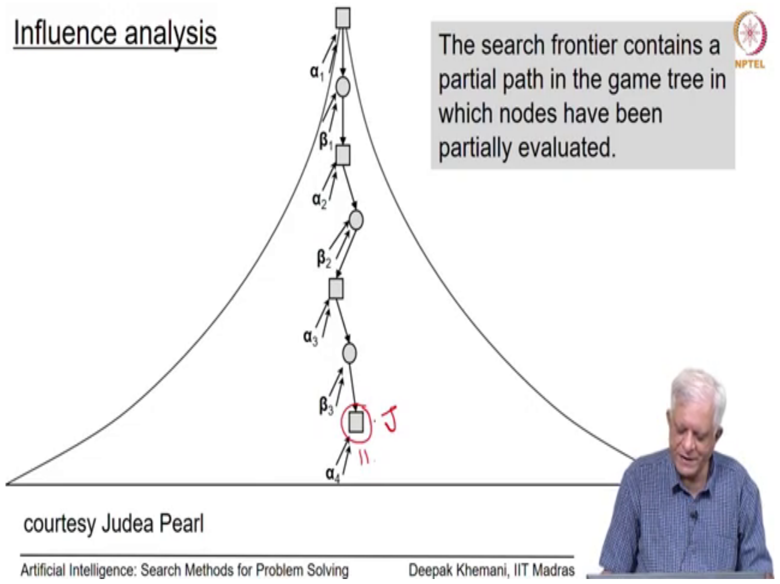
So, this these are the key features of cutoff that we can do. The first key feature is that, it must already have a value from one child and it evaluates the subsequent children in contrast to the value it has already gone. So, if the subsequent children are going to offer values which have worse, then we will simply tell them that do not need to; that you do not need to do any further processing of that particular node.

No point looking at this search tree below that. And you can simply say that, we are not going to do business; in other words we have done a cutoff at node. And having done that, it will now proceed to the third node which it is, which is the third move that Max can make, which is to place a cross in the center.

Now, in this case you can see that, even in the version in which there are symmetric, we have kind of remove symmetries. For the beta node 4 children were cut off, because you know each node has 5 successor; just like the first beta node had 5 successors, the second beta node also has 5 successors.

And 4 of them have been cut off. But imagine that instead of doing two ply search, we were doing eight ply search. Then the sub trees below all those 4 successors would have been cut off. So, pruning can sometimes be quite drastic, which is of course good for the program, the program will terminate faster.

(Refer Slide Time: 14:01)



Influence analysis

The search frontier contains a partial path in the game tree in which nodes have been partially evaluated.

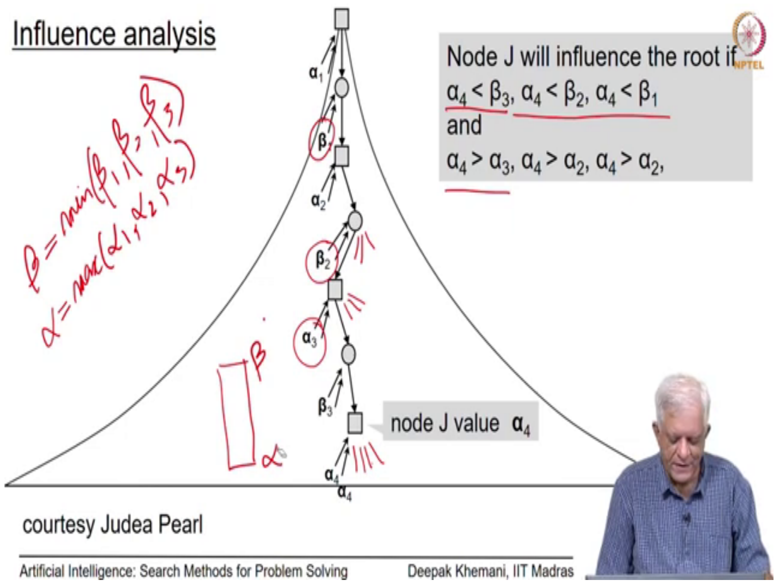
courtesy Judea Pearl

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

So, we can do an influence analysis and this is due to Judea Pearl from his book called Heuristics, where he describes this influence analysis of game play. And at any given point, you are exploring a node which we will call as J. And you are in the process of exploring that node; which means you have got some values from the left children which is called alpha 4 which the value that it has got so far is alpha 4.

And it is looking at other children below that and the question we want to ask is; till when should we continue exploring this node J? And by that we mean, looking at making recursive calls to the children of J and getting their values and seeing if you are getting a better value, ok.

(Refer Slide Time: 14:57)


Influence analysis

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

$\beta = \min(\beta_1, \beta_2, \beta_3)$
$\alpha = \max(\alpha_1, \alpha_2, \alpha_3)$

courtesy Judea Pearl

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

So, this is where my program was trying to get into. So, let say this is a node J and it has a value of alpha 4. Now, if you look at this tree; what does this tree have? This tree has a string of partially evaluated nodes; each of these nodes has been only partially evaluated.

So, for example, this beta 2 would have certain other children that would be evaluated; this alpha 3 would have certain children that have been evaluated, they are yet to be done essentially. So, each of them is partially evaluated and we are focusing currently on this node J and we are trying to see whether to look at the other children of J essentially.

Now, when will J influence a root node? You can see that because its immediate parent is beta 3; only if the value alpha 4 becomes less than beta 3, which is what I have written here, will it propagate it is value up upwards towards the root?

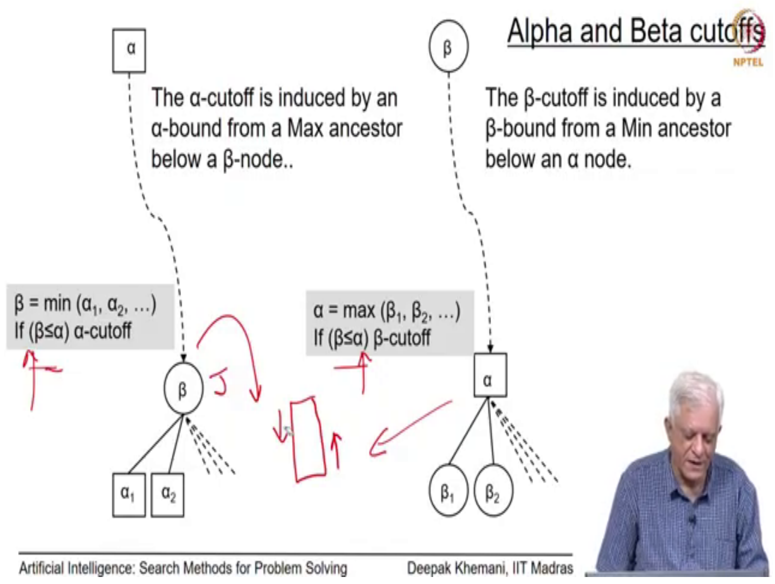
But not only it should be less than beta 3; it should be less than beta 2 as well, because that is also a beta node sitting there. It should be less than beta 1 as well, because that is another beta node sitting as an ancestor of this particular node. So, as depicted here, this value must be less than all the values of beta 3, beta 2, beta 1 and so on.

Likewise if it is to go pass it is Max ancestors, it must provide them a better value essentially. So, which means for example, for alpha 3; alpha 4 must be better than alpha 3 and better means higher than in this case, likewise for alpha 2 and likewise for alpha 1. So, essentially we can say that we can propagate a beta bound which is the smallest of all the beta values. And only as long as alpha 4 or the value of this node J is smaller than this beta, should you continue to investigate below node J.

Likewise, we can propagate a value alpha, which is Max of alpha 1 or all the ancestors essentially; and only if the value is higher than this alpha, we should propagate; we should continue to explore this node J. So, in some sense, this node J operates within a window, where the upper bound is beta and the lower bound is this alpha.

And only as long as the value that it is getting from its children is in this range, should you continue investigating the children; otherwise you should simply stop essentially and that then the cutoff would happen.

(Refer Slide Time: 18:13)



Alpha and Beta cutoffs

The α-cutoff is induced by an α-bound from a Max ancestor below a β-node..

The β-cutoff is induced by a β-bound from a Min ancestor below an α node.

$\beta = \min(\alpha_1, \alpha_2, \ldots)$
If $(\beta \leq \alpha)$ α-cutoff

$\alpha = \max(\beta_1, \beta_2, \ldots)$
If $(\beta \leq \alpha)$ β-cutoff

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

So, there are two kinds of cutoffs; the alpha cutoff that we saw earlier in the case of Tic-tac-toe game is induced by an alpha bound from a Max ancestor and the alpha cutoff happens below a beta node essentially.

So, if you situation is something like this, that you are exploring a beta node and this is a node J that you are exploring at this moment and you have looked at some children alpha 1, alpha 2 and you are looking at other children so on; the beta value that is going to be computed is going to be the smallest of the values of its children that it is exploring.

And the alpha, alpha is some ancestor Max ancestor which is on the path to the root; it could be the root or it could be some other node on the root. And for every search Max ancestor or alpha is a maximum of the values as we discussed in the previous this thing, alpha is max of these values.

So, only as long as this beta is going to be larger than alpha, only if as long as this beta promises a better value than alpha and better value for alpha means larger will search proceed. The moment beta becomes less than equal to alpha, we can do an alpha cutoff, below this beta node; which means we can stop investigating this beta node and do not make recursive calls to any further children.

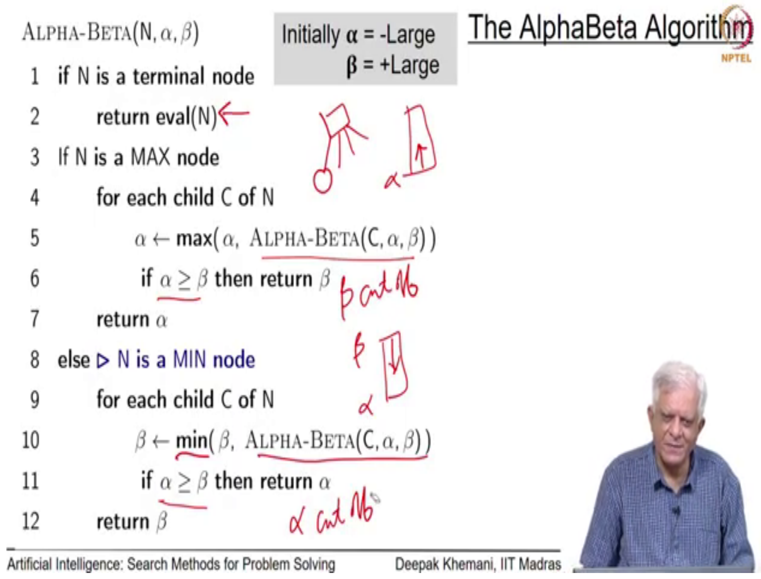
The beta cutoff is similar; the beta cutoff is induced by a beta bound from a Min ancestor and it happens below an alpha node essentially. In a very similar analysis, the alpha value is going to be the maximum of all the values of it is children beta 1; beta 2 and so on and there is a beta bound sitting on the top. So, alpha value must be less than that beta value for a to have influence on the root.

The moment that beta becomes, the moment that alpha becomes greater than beta or equal to beta or in other words when beta is less than equal to alpha, beta cutoff occurs. So, you will notice that the cutoff condition is the same for both alpha and beta that the difference is that, in alpha cutoff beta is a value being computed and in alpha; in beta cutoff, alpha is the value being computing.

In alpha cutoff, beta is a value being computed and in beta cutoff, alpha is a value being computed and the other one comes as a bound essentially. But remember that this is basically saying that the window is closed essentially.

So, when we are looking at an alpha node here for example, it is trying to increase the lower bound. And when you look at the beta node here, it is trying to decrease the lower bound. In either case, if this goes beyond the other bound; then we can stop searching essentially and that basically the idea behind alpha cutoffs and beta cutoffs.

(Refer Slide Time: 21:16)



The AlphaBeta Algorithm

Initially α = -Large
β = +Large

```
ALPHA-BETA(N, $\alpha$, $\beta$)
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($\alpha$, ALPHA-BETA(C, $\alpha$, $\beta$))
6          if $\alpha \geq \beta$ then return $\beta$
7      return $\alpha$
8  else ▷ N is a MIN node
9      for each child C of N
10         $\beta \leftarrow$ min($\beta$, ALPHA-BETA(C, $\alpha$, $\beta$))
11         if $\alpha \geq \beta$ then return $\alpha$
12     return $\beta$
```

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

So, here is a algorithm alpha beta algorithm which we will quickly see. So, in when we wrote minimax, we simply gave a board position N as an input to the algorithm. But when we write the alpha beta algorithm; we gave it two bounds, the alpha bound and the beta bound. So, the initialization of alpha is minus large and initialization of beta is plus large.

So, if you go back to the window analogy that I just spoke off, it amounts to saying that the windows completely open; beta is at the top most point, and alpha is the at the bottom most point, which is amounts to saying that the minimax value of the game could be anything within these two ranges.

The rest of the algorithm is similar to minimax, in the sense that we make, we have written the recursive version here; you can also write an iterative version in which you stored nodes into open and maintain open as a stack and do all that kind of stuff. So, as before if N is the terminal node, we return sorry; if N is a terminal node, we return the value N, that the value eval N which is returned by the evaluation function.

And if you remember in the minimax algorithm, we had initialized alpha and beta to minus infinity and plus infinity respectively; because alpha could only become better than minus infinity and beta could only become better than plus infinity, but in the alpha beta algorithm, we are propagating these bounds.

So, as the search proceeds, we have seen that this window becomes smaller and smaller as we go from left to right. And these are the bounds that are propagated to every recursive call that we are making. So, we do not need to initialize that minus infinity and plus infinity here; we can directly work with the bounds that are propagated. So, as before, if N is a Max node; then you start looking at a child see one by one first, second, third and so on.

And for each child, you make a recursive call to alpha beta again; and for each child, the value that you want to compute is the maximum of what it had got earlier. So, remember that, this is alpha value that you had got to start with when the call was made and you are simply trying to see if you can get a better value than this.

So, we at for each child we try to raise the value of alpha, if you can find a better value of alpha. If at any point alpha becomes greater than beta, which means this window is closing; then we simply return saying that we cannot give you a better value, just take the value beta that you send to us as a bound essentially.

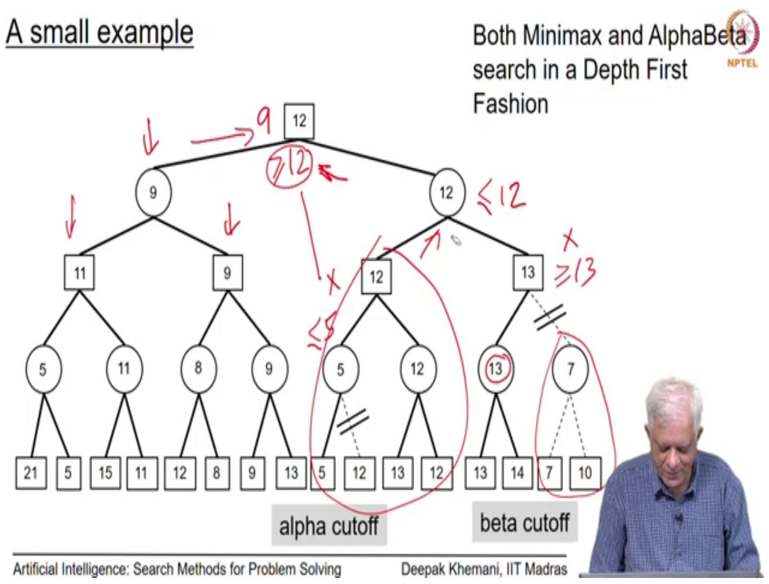
So, that is the that is a beta cutoff which is happening and it is happening below a Max node essentially. So, this is a beta cutoff. The situation for Min is similar that, Min has got also the same window beta and alpha and Min is looking for lower value of beta essentially.

So, when you make the recursive call to the child of Min, you choose the smaller of what that call returns and what you already have. And if at some point beta has gone below alpha, which means again alpha is greater than equal to beta; then you do an alpha cutoff.

So, this is a algorithm alpha beta, it is probably the most popular algorithm that is used in game playing. And simply because it is so easy to implement and it performs much better than minimax; minimax just does brute force search over the entire tree, whereas alpha beta, it is also called the alpha beta pruning algorithm, can prune large portions of the tree.

And as we will see, the amount of pruning it does depends upon the order in which moves are generated and we will come back to that in the next session. But let us just look at an example of alpha beta algorithm. So, it also does minimax like search, which means it searches in a depth first fashion left to right; but unlike minimax, it may not explore certain sub trees essentially and these happen when either the alpha cutoff appears or the beta cutoff appears.

(Refer Slide Time: 26:09)



A small example

Both Minimax and AlphaBeta search in a Depth First Fashion

So, let us look at the couple of small examples to understand this. So, here is a small example that we had seen earlier. And you can see that this is happening in depth first fashion; first we explore the leftmost sub tree, then this sub tree, then we get a value for this and then we do the right most part and so on.

Minimax would have explored the entire tree in the depth first fashion, whereas alpha beta in this small example has not looked at some nodes. So, what are these nodes? An alpha cutoff has appeared below the beta node 5, whose value is 5. And you can see that this has happened because this value 5 that.

So, this is going to be less than equal to 5 and this alpha has said, it is going to be greater than or equal to 12, and alpha is not going to be interested in this beta node. So, it simply says stop exploring below any further and this cutoff occurs below the beta node.

Student: (Refer Time: 27:13).

Yeah.

Student: 12 was not there (Refer time: 27:14).

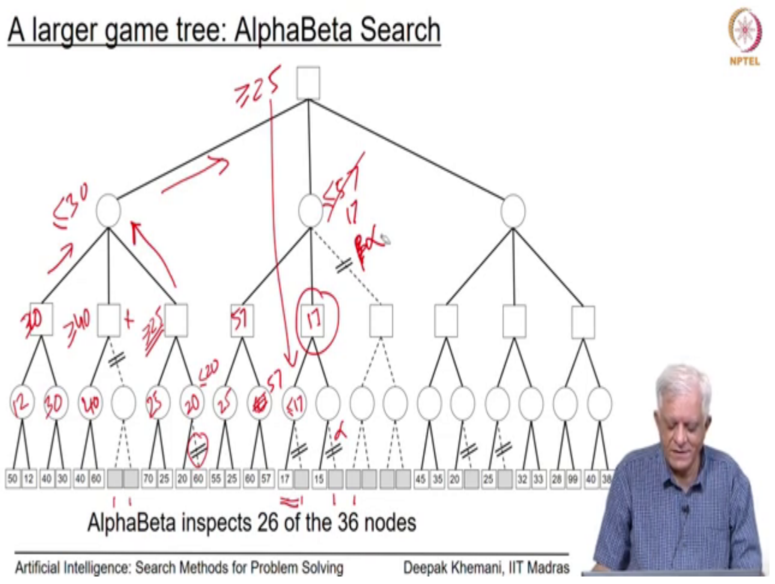
Thank you, it should not be 12; it should be 9, because 9 is a value that is coming from here. This 12 value is actually only coming from this side essentially, but that has not yet been done. So, when this part of the sub trees explored, we know that this value is 12.

And when beta starts looking below this, this thing. So, remember that this is a beta node; so this is going to be less than equal to 12. And this beta node has evaluated to 13 and so, this alpha says I am going to be greater than or equal to 13. And again the beta node says that, there is no need to proceed any further.

I am looking for a value less than 12 and you are giving something which is 13 or more. So, no point in, no point in progressing further. Observe that the sub tree below that has actually a value which is 7, but this 7 is not going to influence this beta node; because Max is sitting in between and between 7 and 13, Max will choose 13.

And therefore, between 13 and 12, Min has already chosen 12 and says that I am not interested in a value greater than 13. Let us look at a slightly bigger example in which there are more cutoffs.

(Refer Slide Time: 28:44)



A larger game tree: AlphaBeta Search

AlphaBeta inspects 26 of the 36 nodes

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

So, here is the larger tree, it is got 36 nodes and you can see and this is the final tree that I have drawn and I would encourage you to do this exercise yourself. And in particular, observe that in the nodes that have not been visited by alpha beta which are the shaded nodes here.

If you were to put in any value, it would not change the minimax value of the entire game tree. So, just to do a quick, quick study of what, what is the cutoff happening here; you can see that from the leftmost side, we get 12 here, because it is a Min node and we get 30 here, because it is a Min node.

And between 30 and 12, Max would choose 30 and therefore, this would be less than equal to 30; it is getting one value from here, which is less than equal to 30. Now, when you go to the second sub child of this Min node, you are getting a value of 40 from here and this says that I am going to be greater than or equal to 40 and therefore, Min says this you can cut that off essentially.

Then Min goes to the third child and it is getting a value of 70 and 25 here. So, this will be 25 and this says I am going to be greater than or equal to 25. So, this says this I am going to be greater than equal 25, so which means Min is still interested.

Because, for example, if it was 26 or 27 or 28; Min would still prefer it to 30. So, the search continues and when we find the first value of 20 for Min child; this is saying I am going to be less than equal to 20, but Max is its parent is getting a value of 25, so it says do this cutoff.

So, this is an alpha cutoff which has happened. In a similar fashion by the time this has gone up, this value of 25 would have gone here and it would have gone here. And so, this would be greater than equal to 25 and cutoffs can be very deep in the sense.

Then when Max, when the algorithm is exploring this node 17 here and it can see that this is going to be less than equal to 17 or the beta value is actually an upper bound; but so I have just written less than equal to 17. Then the fact that its Max ancestor is 25 influences a cutoff at the very deep level.

So, even without knowing the value of its parent, a cutoff can be influenced by any ancestor and this is what we had studied earlier. So, if you go through this whole exercise, you will see that there are some more cutoffs which are happening here.

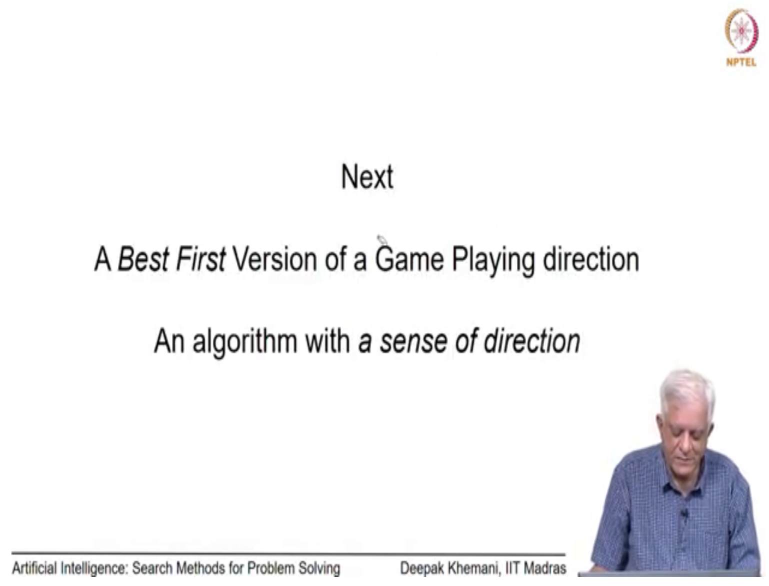
So, this is an alpha cutoff and this is a beta cutoff; because beta is getting some value. So, what is that value let see. This is 25 from here and this is 60 from here sorry, this is 57 from here. So, this would choose 57. So, this is going to be less than equal to 57.

This one is going to be less than equal to, this is less than or equal to 17 and the other one is going to be less than equal to 15. So, we are not even interested here. And at most this can be 17 essentially. And because this is 17; so the, this 57 has gone down to 17 and again the root is interested in 25.

And therefore, this will be sorry, this will be an alpha cutoff we know that, ok. So, this is basically our first look at the alphabet algorithm, it does pruning on the search tree. We will come back to this in the next session; we will maybe take another example.

Then we will look for an algorithm which is better than alpha beta in the sense that, alpha beta has the same property that it is like the minimax algorithm, searching the tree in a predetermined fashion from left to right.

(Refer Slide Time: 33:24)



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

We would be interested in looking at an algorithm which is the best first version, which has a sense of direction, which does not necessarily explore the tree from left to right.

We will also see that with a couple of examples of alpha beta that, given certain order of nodes and by order of nodes we mean the order in which the tree is generated; alpha beta may do more or less cutoff. So, it would be dependent, the alpha beta pruning would be dependent on the order in which the children are generated.

But the best first version that we will see would be independent of that. So, for any order the best first search would do the same pruning and that is the next algorithm that we will see. So, see you in the next session.