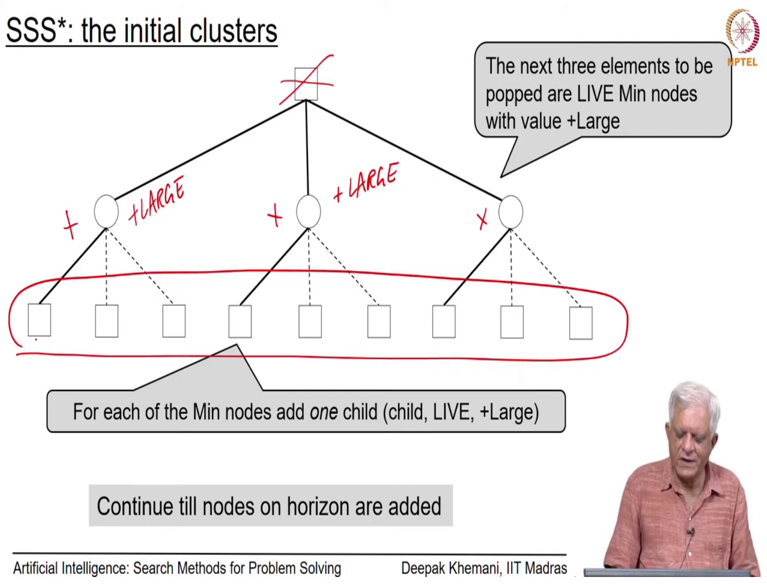
Artificial Intelligence: Search Methods for Problem Solving

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

Chapter – 08
A First Course in Artificial Intelligence
Lecture – 57
Game Playing SSS*: A Detailed Example

(Refer Slide Time: 00:14)



SSS*: the initial clusters

The next three elements to be popped are LIVE Min nodes with value +Large

For each of the Min nodes add *one* child (child, LIVE, +Large)

Continue till nodes on horizon are added

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

So, remember in the forward phase we construct the initial clusters and the way we do that is to start off by adding the root node to the priority queue with a status of being LIVE and a heuristic value of being plus large to OPEN. Then remember that it goes through a loop, you pop the top element of the root and since root is a max node we will add all children of root and for each child we will add the same LIVE status and the value plus large.

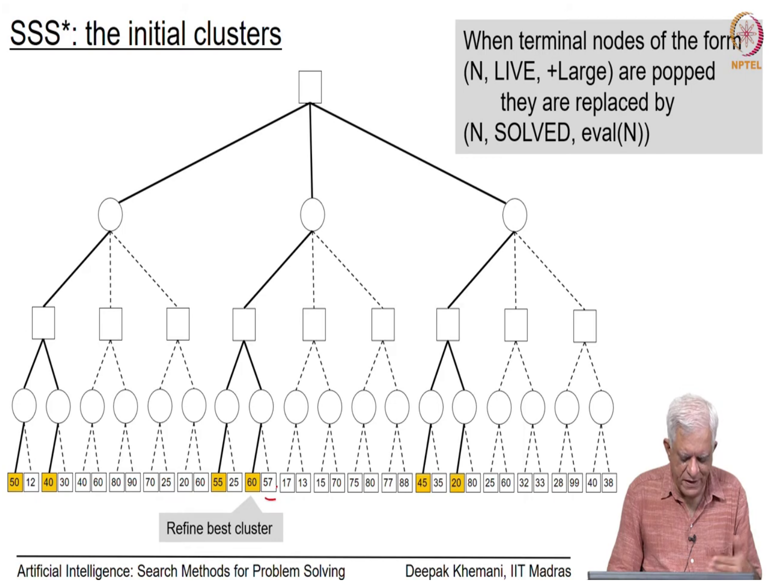
At the next level because we are looking at min nodes we will pop this three. So, by the time we have finished the first stage this has gone out of our priority queue. So, only these three nodes remain, each of them has a value of plus large. So, it does not matter which one we pick, but most implementations will treat a priority queue with equal values as first in first out.

So, the first one will get extracted first and for each of these childs we will each of these children we will add one child with a value of plus large essentially. So, remember that that is how we construct the initial clusters.

So, by the time we are done with this forward process we will be left only with the leaf nodes in the priority queue. All the internal nodes would have vanished in some sense I mean they exist in the game tree, we know who is the parent or who and who is the child of who. But, the priority queue will contain only the leaf nodes essentially. So, if you are doing two ply search, then this would have gone and this would have gone and this would have gone and only this nine nodes would have remained.

And, they would have had a value of plus large to start with. Then they would get popped one by one and replaced with the actual evaluation functions, but in the example that we are seeing we will go down a little bit further, we will look at a four ply tree.

(Refer Slide Time: 02:34)


SSS*: the initial clusters

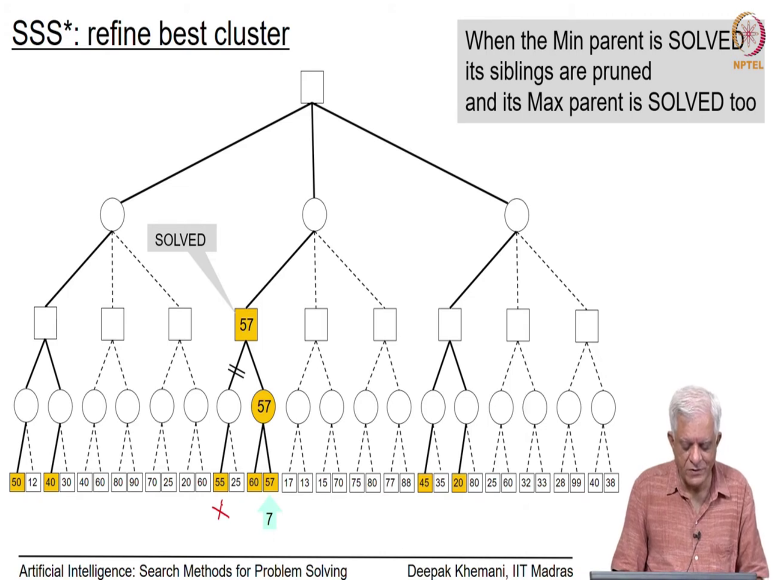
When terminal nodes of the form (N, LIVE, +Large) are popped they are replaced by (N, SOLVED, eval(N))

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

So, we have seen this that this process gives us a set of clusters and when we reach the leaf nodes, then we replace them with their evaluation function values and you can see in this particular example that the values are the ones which are colored orange. So, there are six clusters here and the values are 50, 40 reading from left to right 55, 60, 45, 20.

And, then in the backward phase where we are popping a solved node because all these six nodes will be labeled as solved and these are the only six nodes which will exist in our priority queue. At the head of the queue would be this node with value 60, we will pop that and since it is a solved node and since it is a max node and it is not the last child of it is parent. We will add it is sibling which is this node number with a value 57 to the priority queue and that would continue to be the best node.

(Refer Slide Time: 03:52)

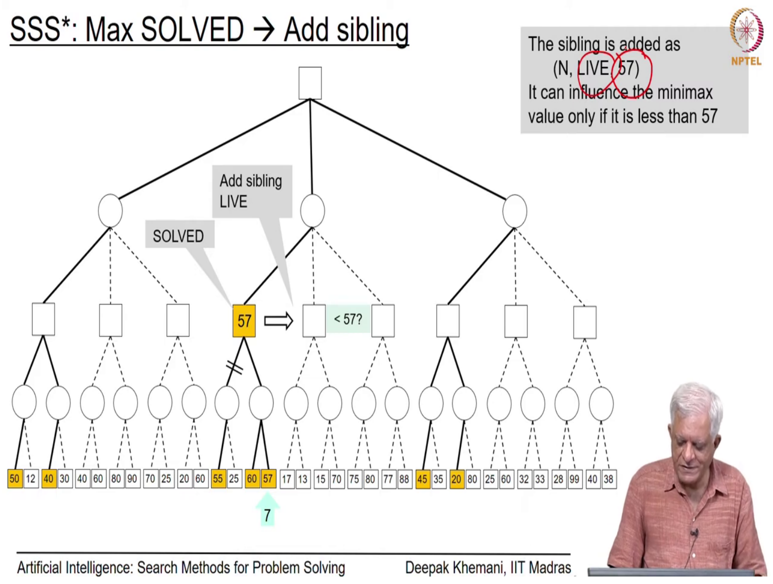

SSS*: refine best cluster
When the Min parent is SOLVED its siblings are pruned and its Max parent is SOLVED too
NPTEL
SOLVED
57
57
50 12 40 30 40 60 80 90 70 25 20 60 55 25 60 57 17 13 15 70 75 80 77 88 45 35 20 80 25 60 32 33 28 99 40 38
7
Artificial Intelligence: Search Methods for Problem Solving
Deepak Khemani, IIT Madras

So, this is what happens next we refine the best cluster and the value of the cluster becomes 57 and its min parent because now this node which is the 7th node that we have expanded it is the last child of its min parent. Its min parent is labeled as SOLVED. And, we have said that every time a min node is labeled SOLVED we can immediately label its max parent as solved.

In this particular example we also prune one strategy away from our priority queue because it can never compete with the strategy which is known to be 57. This particular strategy can at most be 55. So, in fact, that will be thrown out of the priority queue and at this moment we would have five nodes in the priority queue the values would be 50, 40, 57, 45 and 20.

Now, this 57 is the best cluster. It will get popped from the priority queue and if you remember the algorithm when you pop a labeled or SOLVED max node, if it is not the last child you will add its sibling to the priority queue. So, that is what happens next.

(Refer Slide Time: 05:11)



SSS*: Max SOLVED → Add sibling

The sibling is added as (N, LIVE, 57)
It can influence the minimax value only if it is less than 57

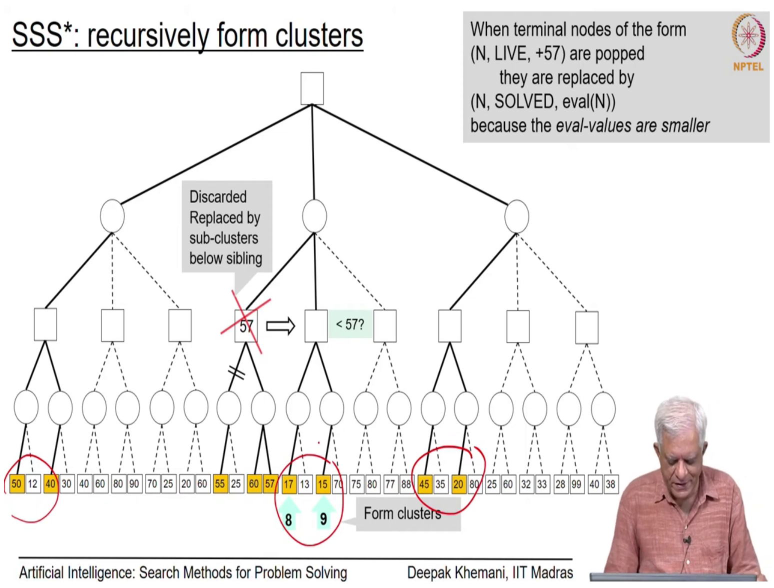
Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

So, max is solved we will add its sibling to the priority queue and we will add it with an upper bound of 57. So, again remember why do we do this. Because the parent of 50 parent of this cluster or the parent of this node which is the value of 57 is the min value the parent will be interested only as long as this sibling remains less than 57.

The moment this sibling becomes higher than 57 this parent is going to lose interest in that, but we do need to know whether this cluster which contains which whose upper bound is 57 is going to go down in value or not. And, that is why we add its sibling, but with a bound of 57 and then we recursively in solve the cluster.

But, in our iterative version of the algorithm what happens is that we remove the cluster with value 57 and we add this new node which we will add as a LIVE node and with a upper bound of 57 essentially. See whenever it is a LIVE node the value is an upper bound; whenever it is a SOLVED node its value is exact essentially. So, we will put this as a LIVE node with upper bound 57.

(Refer Slide Time: 06:39)


SSS*: recursively form clusters

When terminal nodes of the form (N, LIVE, +57) are popped they are replaced by (N, SOLVED, eval(N)) because the *eval-values are smaller*

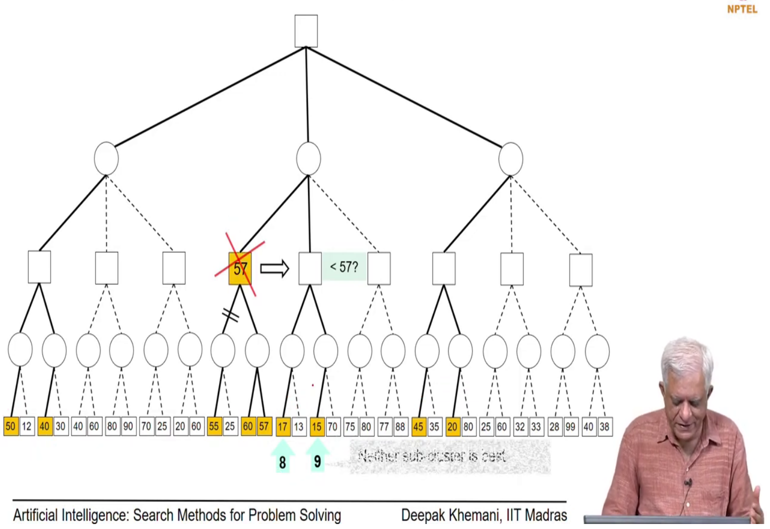
And, therefore, and then we will for form the clusters in practice when these clusters are formed ah. We will put an upper bound of 57 at every level and when we go down to the horizon level at that point we would replace the value by the actual value which in this case is 17 and 15. These are the 8th and the 9th nodes that we have inspected.

Why will they be replaced by the values of 17 and 15? Because if you remember the algorithm say that at the horizon choose the minimum of the bound that it gets and the value that you get from the evaluation function. The bound that we are propagating here is 57 and the value that we are getting is from 17 and 15. So, they will get a value of 17 and 15. Later on we will see in the same tree an example where the opposite happens.

When the value is greater than the bound in that case actually the bound is used, but in this case now we have thrown away this cluster with a value 57 and in instead replace it with two clusters which are the 8th and the 9th nodes that we have seen which have the values 17 and 15. So, again at this stage we have six nodes in our priority queue. So, we had the four earlier clusters that we saw here two here, two here and these are the two new ones that are added.

(Refer Slide Time: 08:15)

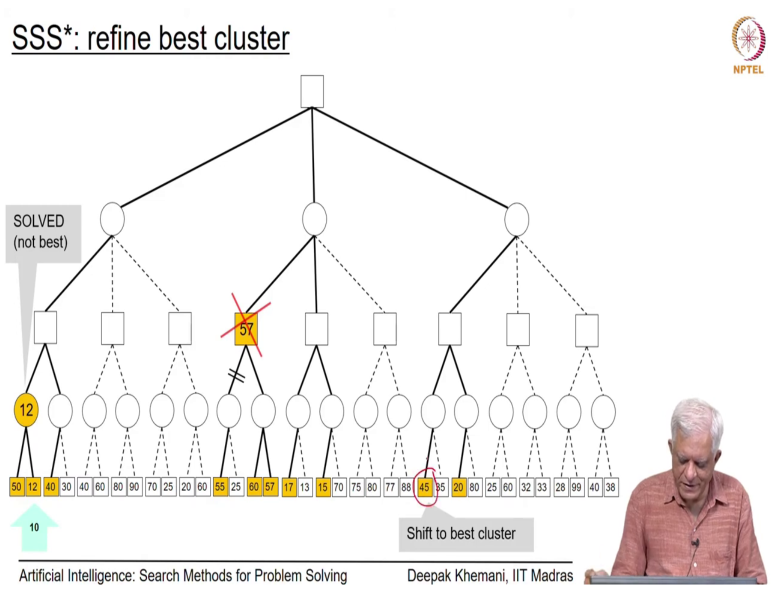
SSS*: shift if clusters not promising



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

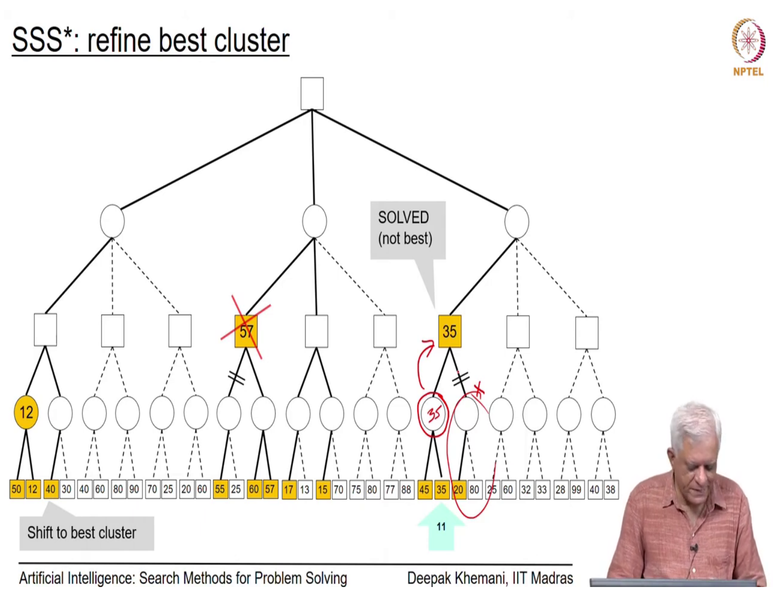
These new clusters that we have added are not the best. So, neither of them is the best. So, our algorithm will pick the best cluster from the priority queue which has a value of 50 as you can see here and it will refine the cluster. Remember, what does refinement mean here that because it is a Max SOLVED node, add its sibling and compute its value.

(Refer Slide Time: 08:38)



In this case it is a horizon node. So, it is value is immediately determined to be 12. And, the value of this particular cluster has now gone down to 12 essentially. It started with 50, but now it has become 12 because we have refined it further we have looked at a further a sibling of that in that node, but this value of 12 is now no longer the best. The best value is here at 45 and we will shift our attention to this node.

(Refer Slide Time: 09:15)



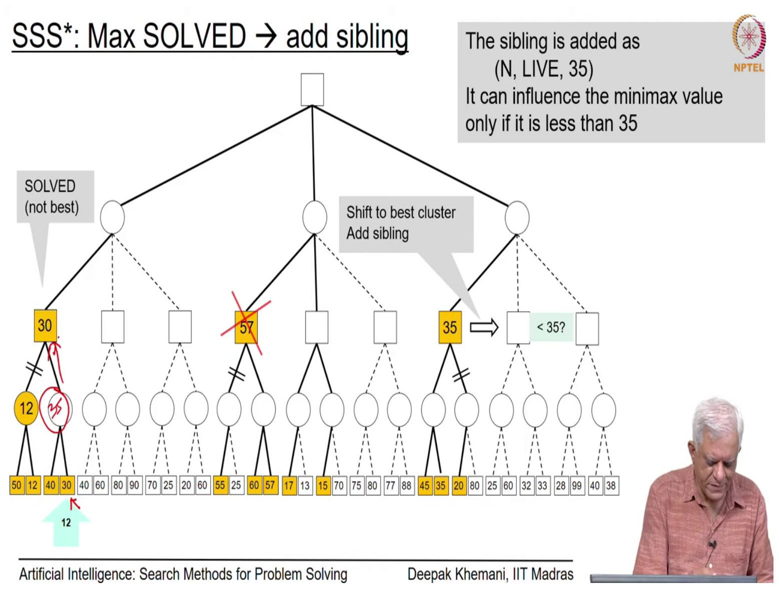
SSS*: refine best cluster

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

So, once we refine this cluster with a which had a value of 45, the moment we inspect this node 35 we can see that its min parent is 35 and it is also solved and because it is a min node which is solved immediately we label it is man max parent as solved with the same value 35.

And, we also delete any siblings of this cluster which were existing and this is a the cluster that we are deleting from this thing. So, we have solved this; we have got a value of 35, but if you observe this 35 is not the best.

(Refer Slide Time: 09:53)


SSS*: Max SOLVED → add sibling

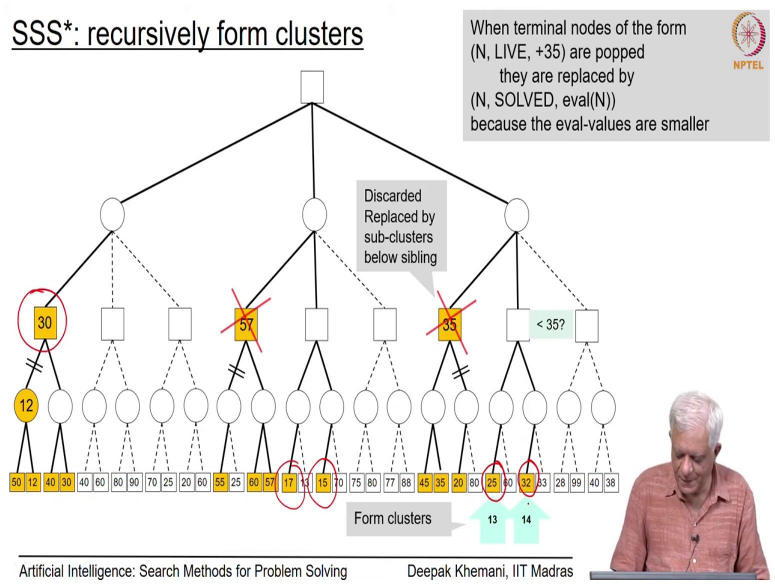
The sibling is added as
(N, LIVE, 35)
It can influence the minimax value only if it is less than 35

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

And, so, we shift our attention to its other sibling which is with which is a value 40 and then we solve that essentially. Once we solve that it is value get refined to 35, but the similar process that we have seen so far that this cluster becomes solved the min parent because it is the last because this one is the last max child, its min parent becomes SOLVED with a value of 35.

And, whenever a min parent min node is SOLVED its parent immediately gets sorry, not 35 30. This is the value that we are getting. It immediately its max parent gets SOLVED and its sibling gets pruned from the priority queue, but now this 30 is not the best and we shift our attention back to this cluster with value 35 and we add its sibling with a upper bound of 35. So, we have seen this process and we repeat this again here.

(Refer Slide Time: 11:05)



SSS*: recursively form clusters

When terminal nodes of the form (N, LIVE, +35) are popped they are replaced by (N, SOLVED, eval(N)) because the eval-values are smaller

Discarded Replaced by sub-clusters below sibling

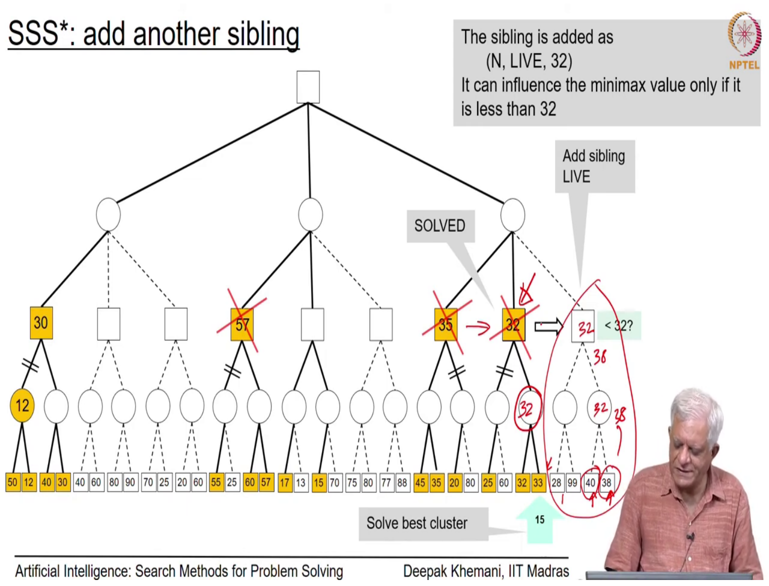
Artificial Intelligence: Search Methods for Problem Solving

Deepak Khemani, IIT Madras

We generate its children refine form the clusters and these are the 13th and the 14th nodes that we are inspecting their values are less than the bound which is 35. So, we keep those values 25 even 32 and then we refine the best cluster.

What is the best cluster at this point? You see that one cluster that is still alive is the value with 30. There are two clusters with poor value, but they are not in contention yet and there are two clusters here one with 25 and 32. So, 32 is the best and so, we refine that.

(Refer Slide Time: 11:48)



SSS*: add another sibling

The sibling is added as
(N, LIVE, 32)
It can influence the minimax value only if it is less than 32

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

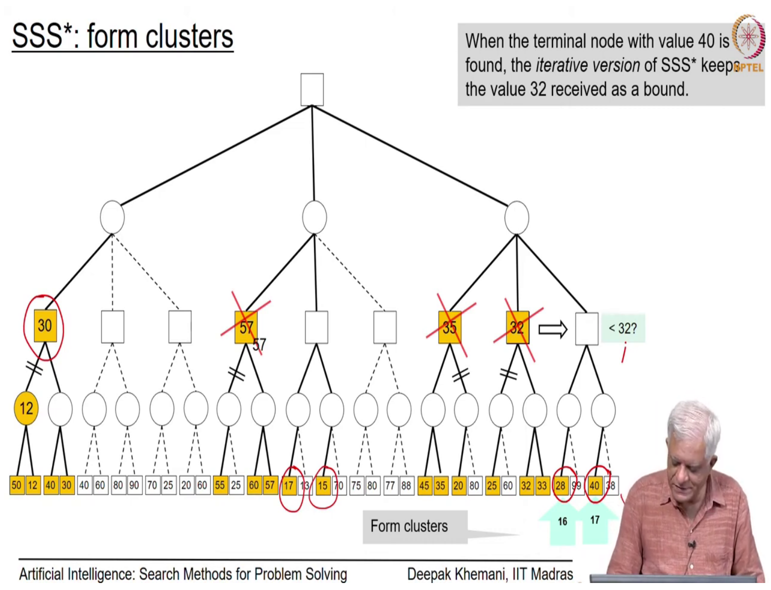
We looked at refinement means as you know we look at it is sibling this in the algorithm is implemented automatically. It says that if a max node is SOLVED in this case it was 32, add its sibling and solve it. The sibling is of value 33, it is on the horizon. It is immediately determined and immediately we know that the value of this its parent because that was the last max child, its min parent becomes SOLVED.

And, the moment a min node get SOLVED its parent becomes solved and it is value become 32 and because this 32 is not the last child of its min parent we will again add its sibling with a upper bound of 32. So, observe that when we went from 35 to 32 the value of the cluster went down to 32. Now, we want to go and explore whether it can further go down from 32 to in the next sibling.

Now, in this example that we are looking at, when we refine this cluster here which is this recursive sub call to the sibling you will observe that the value of one node on the horizon is 40 which is greater than 32 the bound that it is getting. So, the iterative version of the algorithm will actually replace it with the value of 30 and when it looks at 38 this also replace not 30 sorry, 32 when we inspect 38 it will also replace it with the value of 32 and therefore, this will become 32 and therefore, this will become 32.

But, in practice if you do not look at this iterative version if you look at the recursive version of the algorithm that is what we will illustrate is that you will solve it with their actual values of 40 and 38, and then you will find that that this becomes 38 and this becomes 38, but because it is larger than the bound that it got it gets pruned off essentially, but the iterative version would have replaced this 40 and 38 immediately with 32 and therefore, the mini max value it would have returned is 32 essentially.

(Refer Slide Time: 14:12)



SSS*: form clusters

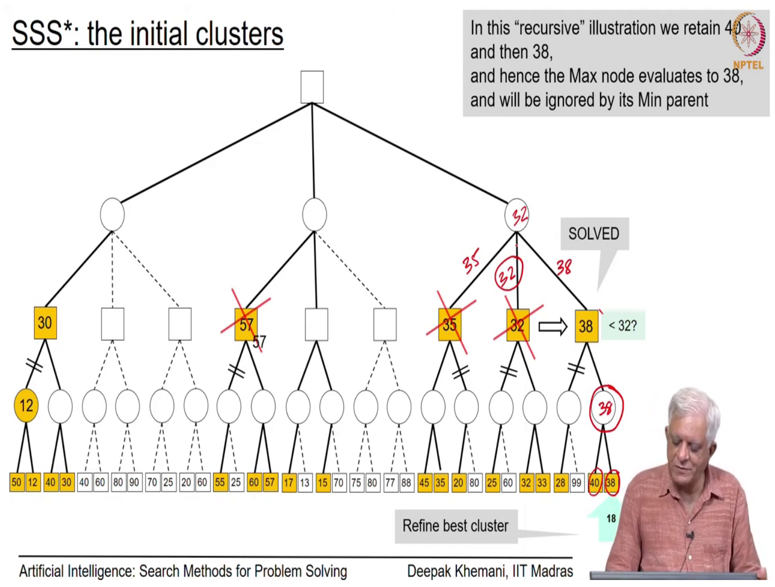
When the terminal node with value 40 is found, the *iterative version* of SSS* keeps the value 32 received as a bound.

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

So, let us just go through this process we form the clusters. This is the 16th and the 17th nodes that we have seen which is 28 and 40. So, again if you look at what is the status we have this clusters sitting here with a value 30, we have this cluster sitting there with a value 17, another cluster with a value 15. And, now two more clusters one with a value 28 and one with the value 40. The value 40 is the best cluster, so, we refine that.

What happens when we refine that? We add its neighbor because this is this is already solved. We know its value 40 and because it is a max node we add its sibling and solve it and because the sibling is on the horizon it immediately gets a value of 38. Actually it gets a value of the minimum of 38 and 32 which is the bound that it is getting. So, in the iterative version of the algorithm this will also gets stored with a value of 32.

(Refer Slide Time: 15:17)



SSS*: the initial clusters

In this "recursive" illustration we retain 40 and then 38, and hence the Max node evaluates to 38, and will be ignored by its Min parent

Artificial Intelligence: Search Methods for Problem Solving

Deepak Khemani, IIT Madras

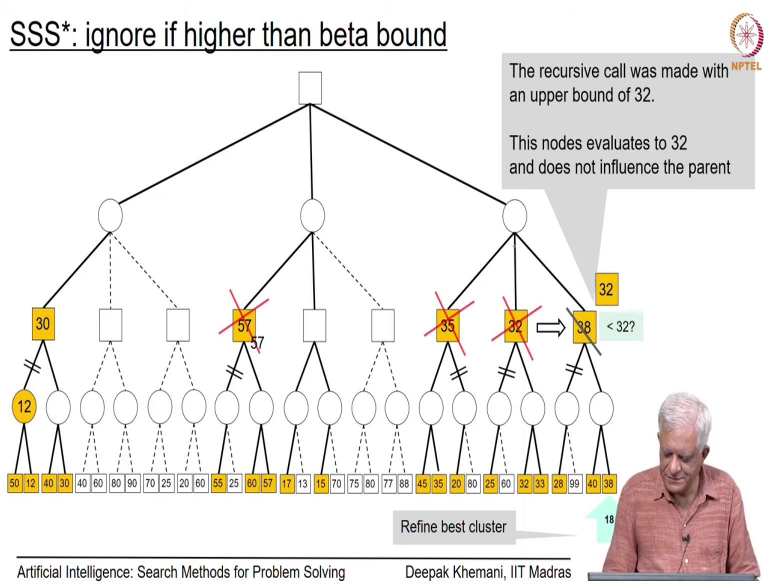
But, in the recursive more abstract algorithm we it will get a value of 38 and then it will gets it is the last child. So, its parent would get SOLVED which is the min node and once a min node gets SOLVED its parent gets SOLVED with a value of 38.

So, this is what would happen if you were to take a recursive view. In practice, keep in mind that this will get a value of 32. In practice we are interested in the mini max value and not so much as to from where the value is coming from; the where the value is coming from is also important because max has to make a move, but we can sort of take care of that little bit separately.

We find that the value of this strategy is 38 or 32 as the case may be if we had replaced this 40 and 38 with 32, then this would have been 38, 32 and this is the last child which is it has no more siblings left for its min parent. And, in that sense we have solved the min parent and the min parent will get a value of 32.

Observe that it will not get a value of 38 because it is a min node and of the other options that were available to is 35 was available from here 32 is available from here 38 was available from here it is going to choose 32 essentially. So, min would actually make the middle move in this case, but the mini max value of the min node would be 32.

(Refer Slide Time: 16:51)



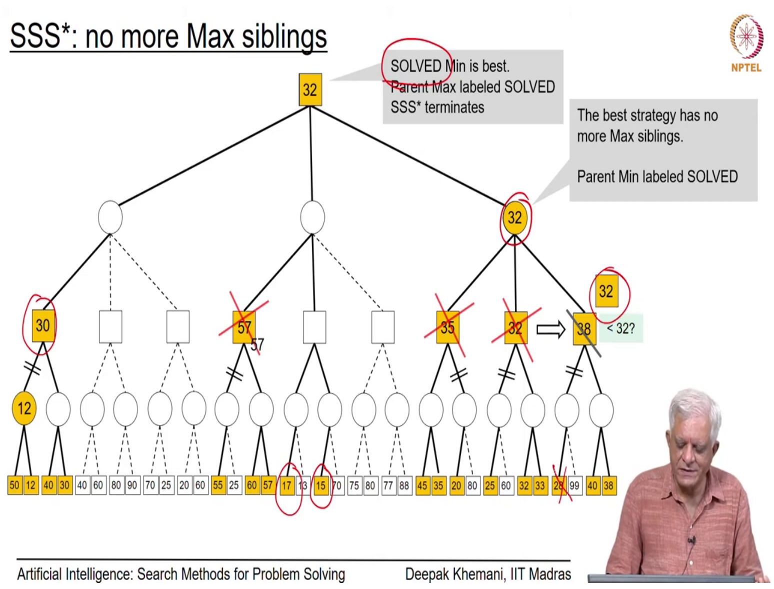
SSS*: ignore if higher than beta bound

The recursive call was made with an upper bound of 32.

This nodes evaluates to 32 and does not influence the parent

And, because this is a min node which has become solved its max parent also will become solved essentially. So, as I said the recursive call was made with an upper bound of 32. This node evaluates to 32 in the algorithm that we have written and does not influence the parent. It gets evaluated to 38 essentially.

(Refer Slide Time: 17:18)



SSS*: no more Max siblings

SOLVED Min is best.
Parent Max labeled SOLVED
SSS* terminates

The best strategy has no more Max siblings.

Parent Min labeled SOLVED

Artificial Intelligence: Search Methods for Problem Solving

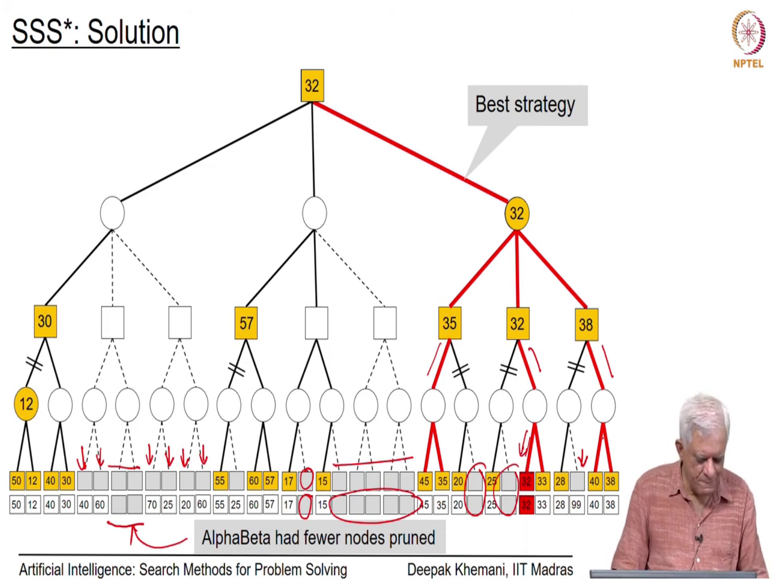
Deepak Khemani, IIT Madras

So, what happens is that instead we consider a value of 32, the best strategy has no more max siblings and so, this was the best strategy the value with 32, it has no more max siblings left. So, it is parent becomes labeled with the value 32 and because it is a min node which is SOLVED its parent gets labeled with the value 32.

So, the parent now is fully SOLVED, the root is completely SOLVED and remember that the termination criteria was when the root is solved and it is at the head of the priority queue.

When will it be at the head of the priority queue? When it is a best strategy. Clearly 32 is better than 30 which is still hanging around in the priority queue it is better than 17 which is hanging around is better than 15 which is hanging around and this one has obviously, been pruned away at some point. So, it is a best strategy it will be at the head of the priority queue and the moment we pop the root node, we can return the mini max value as 32.

(Refer Slide Time: 18:20)


SSS*: Solution
Best strategy
AlphaBeta had fewer nodes pruned
Artificial Intelligence: Search Methods for Problem Solving
Deepak Khemani, IIT Madras
NPTEL

So, that is what we get. The best strategy is shown here in red and remember that a strategy is freezes the choices of Max. So, max has said on the top level that it will choose the right most choice, then it has to of course, account for all the choices of min and for each of the choices of min it has decided that this is the choice that it is going to make.

And, then of course, min gets to have the last say in this game tree because the leaves are max nodes and one expects that min will drive the game towards this value 32 and this is the game tree that is explored.

So, in this diagram I have shaded the nodes which were not visited by the SSS star algorithm. So, at the top level you can see that we have this SSS star tree that is generated and just below that we have copied what are the leaves for the same tree that alpha beta would have visited.

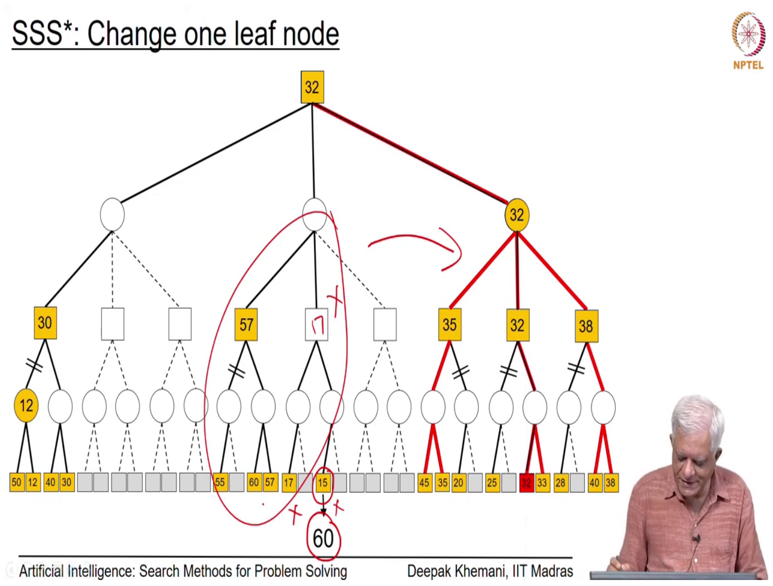
If we had done alpha beta searching from left to right then we can see that alpha beta would have pruned these two nodes, but these two nodes are also pruned by SSS star; then alpha beta would have pruned this node and this node is also pruned by SSS star. Likewise alpha beta will have pruned these five nodes they are also proven by SSS star and likewise for the other nodes that alpha beta has pruned.

So, every node that alpha beta prunes SSS star also prunes. So, in that sense SSS star is distinctly better because apart from that there are certain nodes which alpha beta saw, but alpha, but SSS star has pruned. So, you can see that SSS star has not been influenced by the ordering of the nodes, it has only looked at the nodes in a particular order and it has pruned more nodes than alpha beta in general.

So, in that sense alpha SSS star is a superior algorithm to alpha beta. Imagine that instead of this four ply tree that we have drawn here if you are doing eight-ply search, then every one of these internal nodes which is pruned would have pruned off a huge sub tree.

And, you can then imagine the kind of savings that pruning does for you in general and alpha beta is much better than Mini max and SSS star does more pruning than alpha beta. But, SSS star is a little bit more involved in implementation because you have to maintain the priority queue and that kind of stuff whereas; alpha beta can be implemented in a much more simple fashion.

(Refer Slide Time: 21:17)

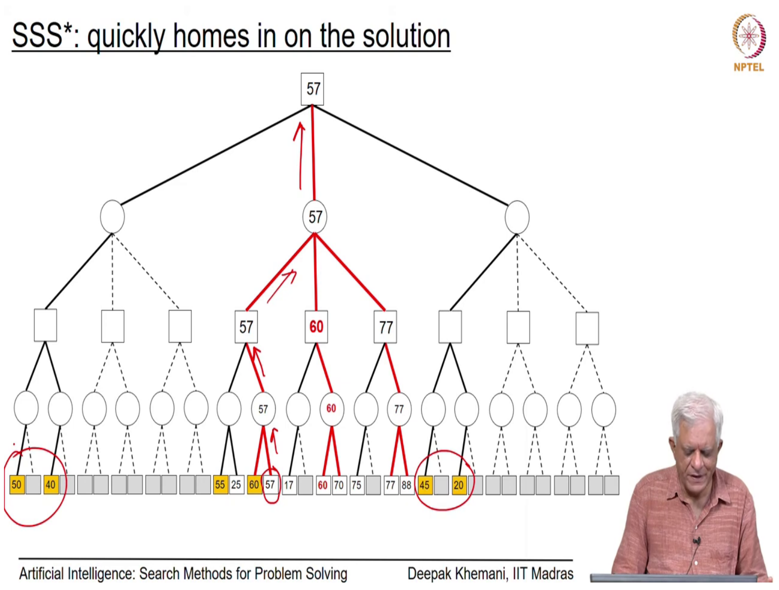

SSS*: Change one leaf node
32
32
30
57
17
35
32
38
12
50
12
40
30
55
60
57
17
15
45
35
20
25
33
28
40
38
60
NPTEL
Artificial Intelligence: Search Methods for Problem Solving
Deepak Khemani, IIT Madras

Let us just see that how changing one value in the tree that we examined changes the way SSS star performed. Now, you will see that SSS star had initially focused on the on the middle sub tree.

But, at some point because it ran into this cluster which is a value 15 and then this whole thing reduced to a value of 17 here and, then it lost interest and these two clusters kind of languished away at the back of the priority queue and it shifted its attention to the right side of the tree eventually after exploring the little bit more on the left and then found the solution on the right side of the tree.

So, let us see that if we change this value of 15 to a value of 60 what would SSS star have explored essentially? So, if we run through this algorithm and I will encourage you to do that just change all the just keep the same tree and I change this value of 15 to 60.

(Refer Slide Time: 22:35)



SSS*: quickly homes in on the solution

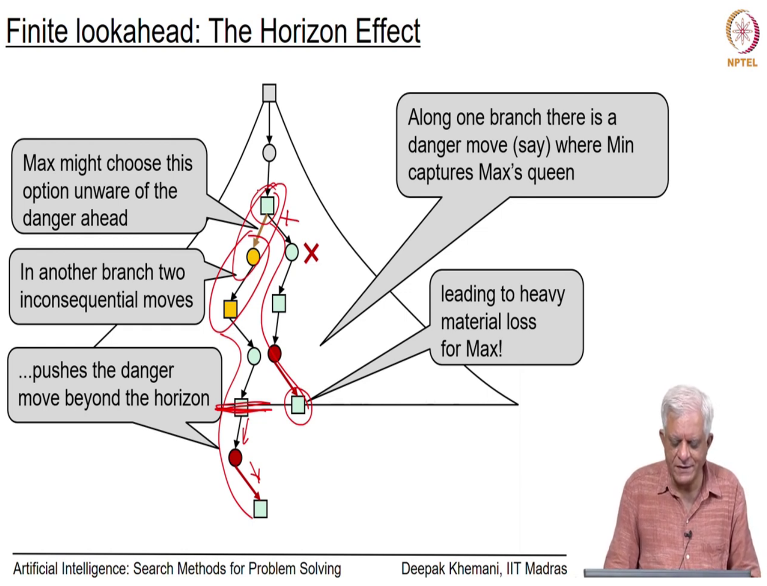
Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Madras

And, then you will see that SSS star very rapidly focuses towards the best strategy. The best strategy in our case comes from this node 57 which is what its min parent selects and then its max parent selects it and then this.

So, the game that would be played would be towards this node 57, but you would notice the important thing is that SSS star has not even explored this part the right part of this the sub tree or the left part of the sub tree because in this particular case it found that the center move was the best and it did not have to change its mind anyway.

So, that is just to illustrate that under that if the game tree is has certain properties, then SSS star would just simply go towards the go towards the solution. So, which is another way of saying that if your evaluation function is perfect, then a breadth first algorithm would not have to look here and there too much and it would reach the goal soon.

(Refer Slide Time: 23:57)



Finite lookahead: The Horizon Effect

Max might choose this option unware of the danger ahead

In another branch two inconsequential moves

...pushes the danger move beyond the horizon

Along one branch there is a danger move (say) where Min captures Max's queen

leading to heavy material loss for Max!

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

So, finally, I want to end with one property of limited look ahead or finite look ahead which is called the horizon effect. And, this has been studied because people have observed that sometimes this limited look ahead creates the kind of problems where some critical event if it goes past the horizon, then the algorithm is not able to make the best choice.

So, let us assume that there is a path in the game tree which is as follows as depicted here and the and we are searching this limited which is 6-ply search and there is a node let us say along the branch a node where min captures the max queen this node depicted in red.

And, in fact, alpha beta would see this because the value of this leaf node would plummet it would lead to heavy material loss because queen has a very high value in the game of chess. And, consequently it would not have made this choice essentially and it would have made perhaps a different choice. So, at this level this choice would not have been made essentially.

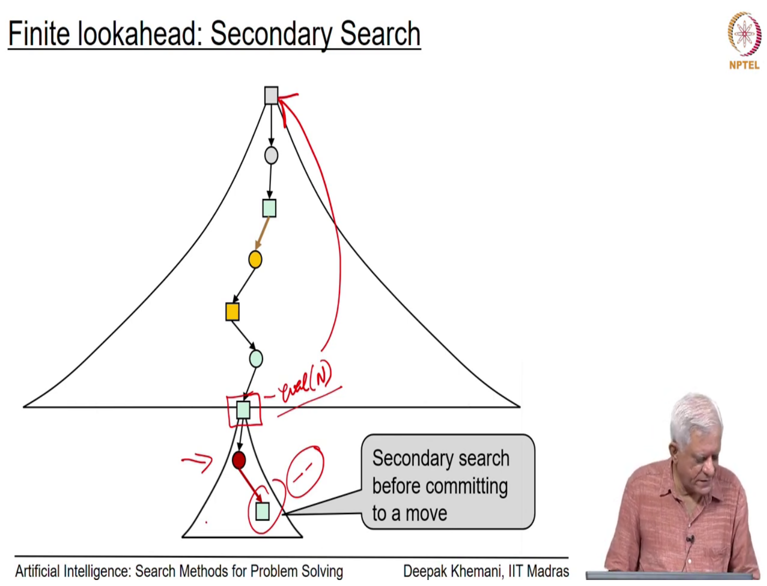
Now, what people have observed that if there are some inconsequential moves that you are making in another branch below that it is just like you know some you push one corner pawn and opponent puts one corner pawn, which has nothing to do with what is happening in the game and then you repeat this set of moves that you had done here you repeat them here.

This danger move which is in the red circle for min, it gets pushed beyond the horizon because the search is only going to be at this level. It is going to terminate at this level, max might not realize the danger which is lurking beyond that move and that if max were to make this move then min would capture it is queen and it cannot see it because it is beyond the horizon.

And, this has gone beyond the horizon because of this couple of inconsequential moves that you made somewhere on this board and remember, that in algorithm we will try out all possible moves.

So, in some branch these two inconsequential moves will be there and in that branch this danger move would have got pushed beyond the horizon. And, it is possible that max might actually end up making this move on this thing. So, what people have done to kind of to some extent, take care of this problem.

(Refer Slide Time: 26:57)


Finite lookahead: Secondary Search

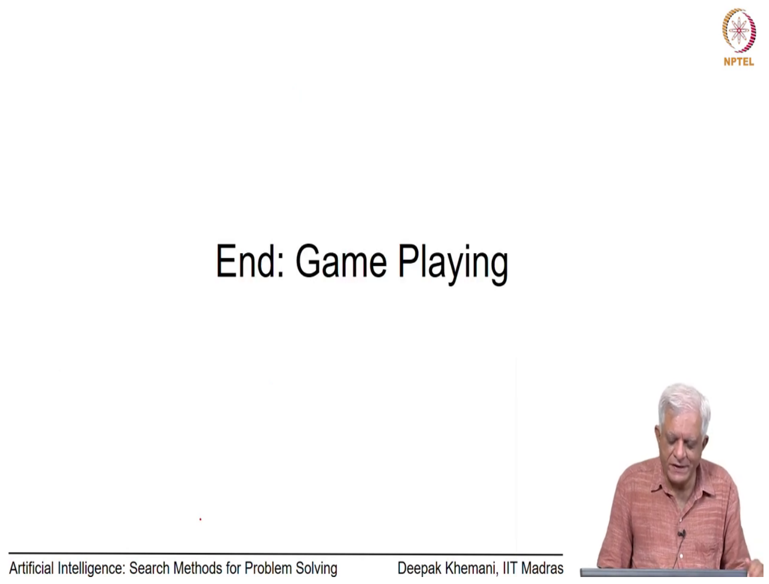
Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

Is to do a bit of secondary search. So, when you are searching when you have decided that you want to select a Mini max value from a certain node which is this one and you selected this node because you could not see this danger move which is lurking just beyond the horizon. You can do a secondary search just to make sure that the eval value that you get from here does not get disrupted too much.

Because you have chosen this path based on this eval value and this eval value has become the Mini max value of the game and that is why this path gets chosen what you cannot see is that just beyond the horizon this eval value is going to go rapidly down. So, here it is going to get a lot of negatives here and we cannot see that.

So, you can try and kind of cater to this by doing a secondary search a limited look ahead search from that node that is the mini max gives us the Mini max value just to see that the evaluation function does not fluctuate too much. And, in practice this has been found to work quite well. So, just before making a move most algorithms tend to do a little bit of secondary search just to make sure that that move is essentially.

(Refer Slide Time: 28:26)



End: Game Playing

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

So, with this I think we will end our study of game playing. Game playing has traditionally found a lot of excitement among students and whenever I teach this course on AI I find that students like this part the best. And, especially if you introduce a assignment in which they have to implement games for a particular game and what I often do is to hold a competition between the programs of different students.

And, the program which wins the tournament gets the maximum marks and so on and so forth and it has turned out to be quite an excitement. So, I would encourage you to take up a small game one of the games that we use is called Othello or Reversi.

But, you can take up any game you can take up checkers or you can take up chess or if you are very adventurous you can take up go and try to implement a game playing program for playing that game essentially. So, maybe you can create an online site where other people can come and play your game.

So, we will move away from games after this and to a new topic in the next session.