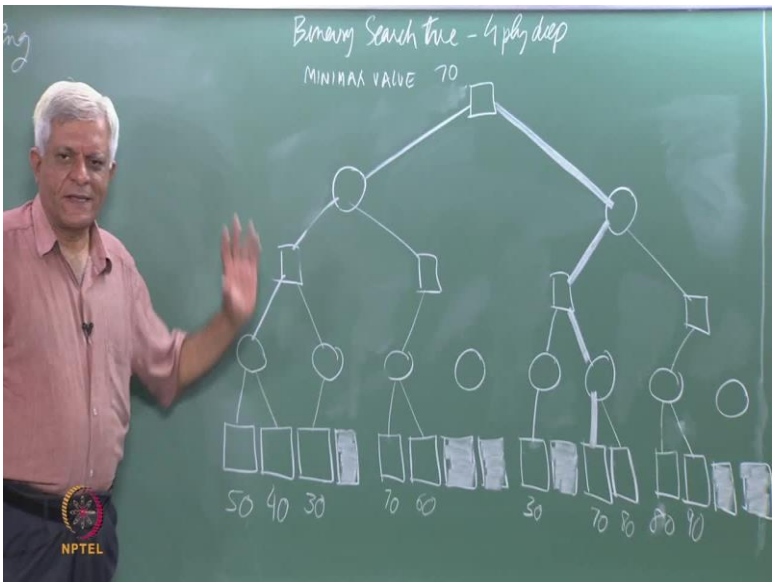
Artificial Intelligence
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture No. - 29
Game Playing SSS

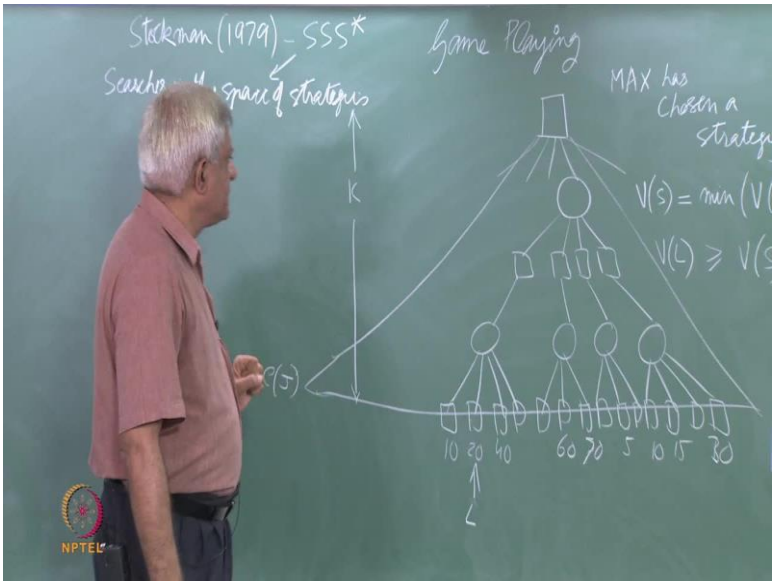
So, we are looking at game playing algorithm.

(Refer Slide Time: 00:16)



And we saw in a last class the alpha beta algorithm, and the tree that was explored by alpha beta we saw here. It is a binary game tree 4 ply deep and those shaded nodes are the ones which alpha beta did not explore essentially. Now, alpha beta algorithm searches from left to right and we want to now look for a algorithm which is not uninformed, which is not blind in this manner, but has a sense of direction. So, just as we moved from death first search to best first search by introducing heuristic function, we want to look at an algorithm which will go towards what it thinks is a good solution essentially. And indeed such an algorithm was given by stockman in 1979 and it is called SSS star.

(Refer Slide Time: 01:06)



Now, to understand this algorithm first we must understand what is the space in which this best first search will happen, and look at what max does. So, max is playing a game of the many choices that it has available, it makes one choice. So, of course, on the surface level it is selecting a choice, but it is making a choice on the basis of having searched a key up to some depth, which is let us say k ply search we are doing.

And it is, it is on a basis of this look ahead that max has made this move. And on the basis of having applied a evaluation function on the horizon essentially. So, in effect max has force looked ahead at or foreseen what would be min's response? So, in other words, max is comfortable with all of min's responses, and for each of those min's responses max has thought ahead that I will make one particular move and then, max is taken into account, all of min's responses. So, let us say, this is also a 4 ply search tree. Max has looked at this entire set of possibilities, and on the basis of this has made the move that max has made essentially.

So, you can say that, this sub tree that we are seeing is what max has concluded at the result of this search. He says if max will make this move, whichever move min makes max has an answer to that and then, whichever makes min makes max has taken that into consideration while evaluating this thing. Now, you will remember that, such a sub tree is called a strategy. So, max has chosen a strategy of course, alpha beta algorithm does not search in the space of strategies, it is does not even consider strategies as such, but if

we are going to look at a different algorithm which is SSS star. SSS star searches in the space of strategies.

So, let us say, this strategy that max has chosen this strategy and let us call this S, and we have discussed this earlier, what is the value of this strategy? And value in terms of the leaf nodes that max has taken into account. So, let us say, this is 10, 20, 40, and let us say all these are bigger values 60, 70. Let us say this is 5, 10, 15, 30. Some values, I am not filling all the values, but let us say we have some values 10, 20, 40, 60, 70. There is a 5 here 10, 15, 30. What is going to be the mini max value when max uses this key?

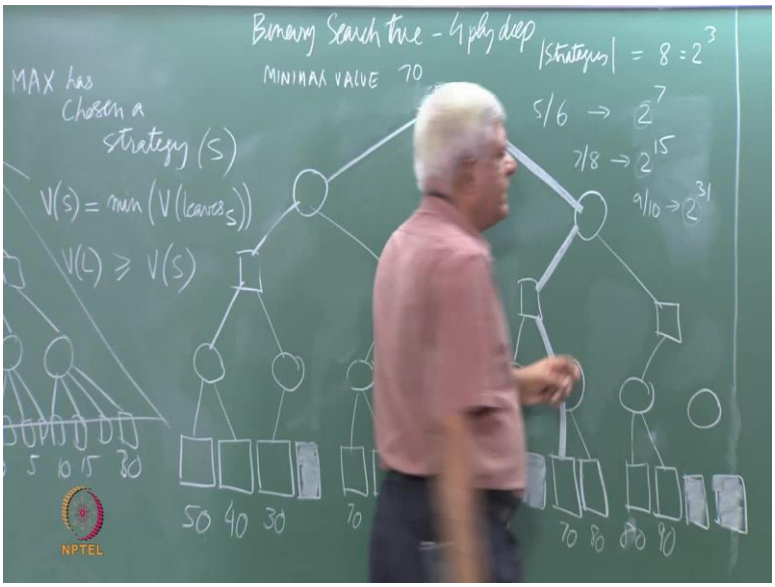
It is going to be the minimum of all these values. Because once max has frozen max's strategy, it is only up to min to choose and min will lose some analysis and say that, okay if I make this move and if then, if I make this move then, I will get a value of 5, which is what min is going to drive it at. And max has no more choices left because max's strategy has frozen essentially. So, the value of a strategy is the minimum of the value of leaves in this strategy S. So, I just use this notation to say that these are the leaves of the strategy S, and the value of the strategy is the minimum of the value of the leaf of this of the strategy.

Now, as a corollary, if were to choose a random leaf in a strategy. So, if I chose some leaf L value of L. So, let us say this is L, some leaf, it does not matter. How does it influence us, what relation does it have with value of S? It is greater than or equal to the value of the strategy, which means the value of a leaf is an upper bound on a strategy in which contains that leaf. Now, let us look at this binary search tree again, if I were to select this leaf 50, which has this value 50, how many strategies will this be a part of?

You can see that, this of course, is min's choice and here max has made a choice, and that choice has 50 as this leaf. Max has to encounter so, so this strategy would be this choice for max, both these choices for min, this particular choice for max, and both these for min essentially. Now, since max has to consider both these choices for min, this as well as this. When max makes this move, max is either selecting a strategy in which max is making this move here, and this move here. I mean max is considering this move, max is considering a strategy in which max is making that move and this move, and the strategy may have this move precisely.

Alternatively, this leaf will also be part of a strategy where max makes that move, this move here, but the other move at this place here, that also will contain this particular strategy. So, you need to visualize this a little bit, that in this particular game tree every leaf belongs to 2 strategies, and that is the one strategy is when max makes this choice here, and the other strategy is when max makes the other choice here. Because max has to take into account this choice of min here. So, there are 16 leaves, and how many choices, how many strategies does max have? 2 here, choices here, 2 choices here, and 2 choices here so, max has 8 strategies available to that and as an.

(Refer Slide Time: 09:19)

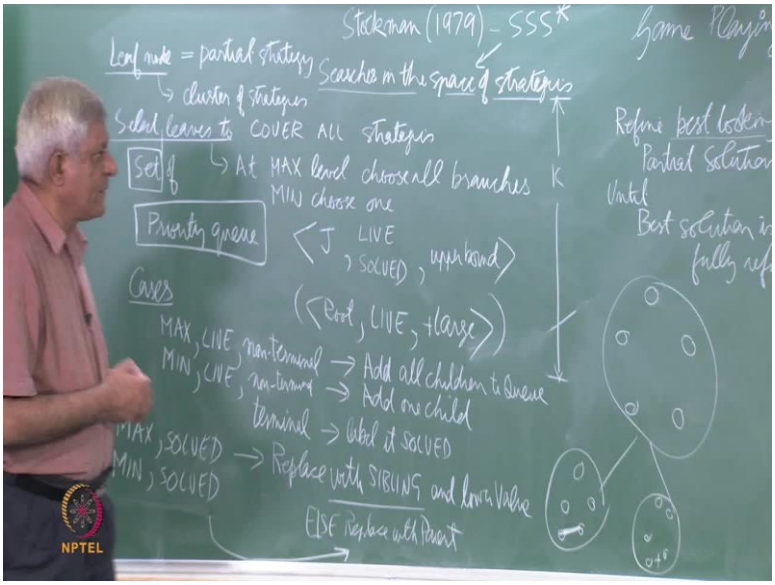


So, I will just write it here number of strategies equal to 8 for a 4 ply binary search tree, and as an exercise you can see that for 5 or 6 ply it is the same it is. So, 8 is equal to 2 raised to 3, for 5 or 6 it is 2 raised to 7, for 7 or 8 it is 2 raised to 15, for 9 or 10 it is 2 raised to 31 and you can extrapolate from here. And I will just give this as figures, you can work this out yourselves that as you go deeper, the number of strategies increases considerably. And as an exercise you can so, this 2 is a branching factor that is playing the role here. If he replaces 2 with B what would the figure be like this?

It is an interesting exercise, but you can do that later. For this particular tree, there are 8 strategies and if I choose any arbitrary node, I will get 2 of those 8 strategies, choose any node and you will get 2 strategies. So, what does SSS star do? It is searches in the space of strategies, and it must be exhaustive in the sense that it must not miss out on the best

move. So, it must consider all strategies to begin with and then, choose one from that. That is of course, brute force, but we do not do brute force here especially.

(Refer Slide Time: 10:57)



Now, a leaf node, you can say is represents a partial strategy. In other words, it is a strategy which is not completely refined, if I have seen only one leaf in a strategy, I have seen only part of the strategy. So, for example, if I have seen this leaf 20, its value is 20, I have, I know that this part of this particular strategy, it may be part of other strategies as well. But whichever strategy it is a part of, it has this property that it is an upper bound of the value of this strategy.

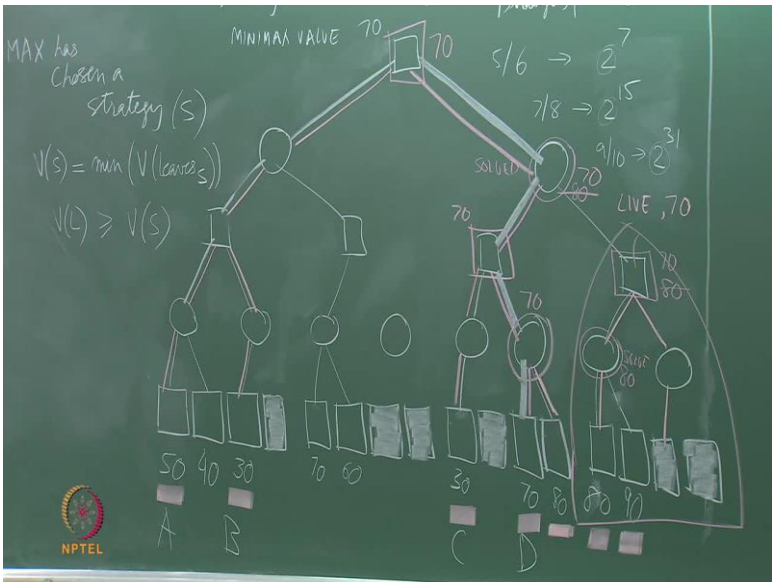
A leaf node also represents a cluster of strategies, which is what we were discussing in the moment ago that, if I were to choose this node 50, the left most node then, it represents a cluster of 2 strategies. And the 2 strategies are the one where max makes the left choice at the top level, and if min were to make this choice then, max has decided the left choice, but if min were to make this choice, max could either choose this or it could choose that. So, those are the 2 strategies.

This particular node 50 represents, and this 50 would be an upper bound on both those strategies. So, what we need to do is to select leads to cover all strategies, that is the first point of SSS star algorithm, that somehow you select the leaves in such a manner that, you have a representative from all possible strategies, which means you have covered all strategies. How do you do that?

It is very simple, you extract a sub tree from the game tree in the following fashion that at max level choose all branches. Why should we do that? Because we want to cover all strategies. We want to not miss out on any strategies so, for example, at the root level, we must consider all possible moves that max makes, and likewise at deeper levels essentially.

So, at max level choose all branches, at min level choose 1. If I do this, so, I should write here set of, I will construct a set of leaves which will cover all strategies essentially. So, let us do that for this game tree that we had considered for the alpha beta algorithm. So, we want to now explore how SSS star will do it, but first it starts off by selecting a set of leaves where each leaf represents a cluster of 2 strategies in this case, but you must select the leaves so, that all strategies are covered. So, that we do not miss out on any strategy.

(Refer Slide Time: 14:48)

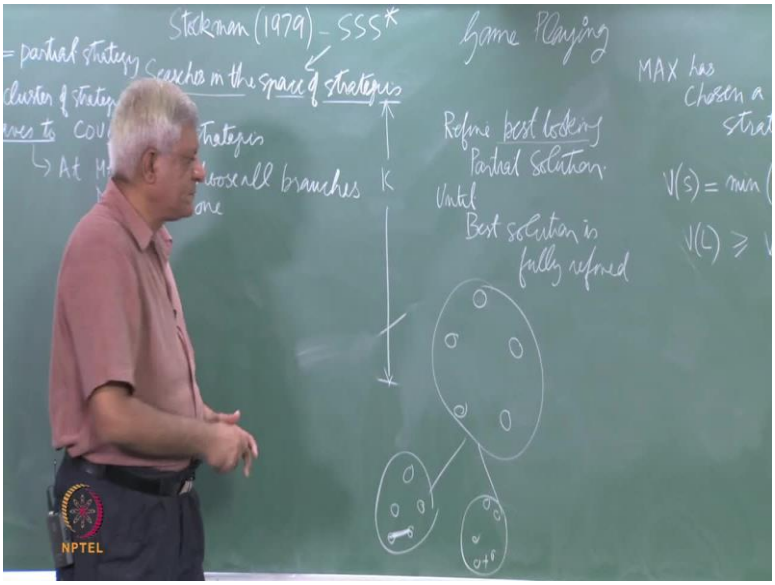


And the way to that is to select all choices for max, one choice and let us say, arbitrarily we are choosing the left most choice for min then all choices for max and one choice for min. Likewise here, one choice for min all choices for max, one choice for min. SSS star, this algorithm starts of by selecting this set of nodes, leaves.

So, let me just underline these leaves here, we are looking at this one, this one, this one, and this one. And I will ask you to think about this and convince yourself that of the 8 strategies that we talked about, these 4 leaves cover 2 each, and they are all disjoint and they cover all the 8 strategies essentially. Now, max is basically tasked so, remember the

best first algorithm that we had talked about and then, we also looked at the travelling salesman problem which is the high level algorithm is that define.

(Refer Slide Time: 16:28)



Best looking partial strategy, a partial solution in that case, in our case the solution is the strategy till best solution is fully refined. This is the high level algorithm for best first search A star fits into this, branch and bound fits into this, best first fits into this. At all point, we are maintaining some sort a priority queue in which we are able to pick the best value strategy.

So, in A star the best is defined in terms of the f value, which is h value plus g value. If you remember the travelling salesman problem, we said that at the top most root level, the set of cities represent all possible to us. And then, we partition this set into 2 sets one which contain one edge, and the other which did not contain this edge.

So, this was a partial tour, both these are partial tours, they are not fully specified and we had figured out a way to evaluate what is the lower bound cost on those tours, and we always picked up best looking tour and refined it further essentially. We want to do something similar here in SSS star is that, we first cover all the possible solutions and that is what we have done by choosing the set of leaves to form the clusters which cover all strategies.

So, in this case we have 4 clusters, and in each cluster we have one representative leaf 50, 30, 30, and 70, and that leaf represents an upper bound on the value of the strategy that this cluster belongs to, this leaf belongs to, each leaf belongs to 2 strategies. So, we will follow this, we find the best looking partial solution until the best solution is fully refined when it is fully refined, when the strategy is completely defined. So, this algorithm SSS star, it uses a priority queue of clusters, but clusters are not represented explicitly, they are represented by the representative node essentially. And each element in the priority queue is represented by the name of the node by a value which is either live or solved and a bound.

So, initially when we start the algorithm, we start it with the root and we say it is live by live we mean which is not solved essentially. So, you can now also make a comparison or with the AO star algorithm that we had seen. The AO star algorithm keeps refining a solution till the root gets solved, root gets level solved, which means that the solution is completely refined.

And at some point I had also made this observation that, solving a game tree is similar to solving an of problem because at the max level we have to make one choice, so it is an odd node. At min level you have to consider all choices, and it is like a min node except that you do not sum up the pass cause of the parsing solutions as we did in A star, but we take the minimum of the cost of the parsing solutions here essentially.

So, it is very much like the A star, AO star algorithm and this is the terminology we use in the AO star also that is we had nodes levels solved or not level solved. In this case we are explicitly calling them live. The algorithm will proceed till the root gets level solved so, that is a similarity with AO star you should observe. AO star was the best first algorithm if you remember and this is also going to be the best first algorithm.

So, we start with this and this bounds. Initially we insert this into the priority queue, this triple where the root node and level it live and plus large. And always the highest value will be the head of the root, the priority queue a max queue then, the algorithm picks the head element which is the value with the largest bound or heuristic value if you want to call it, and does the following.

So, this is, these are the cases when you pick the head node from the root that, it is max and it is live then, you add all the children to the queue with the same bound and call

them live essentially. So, I should in fact, say non terminal. So, some where there is a test so, remember the terminal nodes are those on the horizon, and at the horizon we allowed to apply. In fact we have to apply the evaluation function and get a value for that node.

If it is max, it is a max node and if it is a live node and if it is non terminal then, add all children to the queue. So, you will see that this is the step, at max level choose all branches, at max level we are adding all the children. So, initially we would have added this root then, we will add both these children to this thing and then, we will call them live essentially. If it is a min and live and non terminal then, add one child.

So, you can see these two steps correspond to the initial formation of the clusters essentially. And since this is a priority queue with this plus large values, all this will be taken care of first and this whole tree would be constructed here. This tree that you see in this pinkish color here at max level we will add both the children, we will remove max and add both the children. At min level we will remove min and add one child.

In this case we arbitrarily choose a left side at max level we add both this children at min level we add one child. And all this will vanish and so, only 4 entries will remain in my parity queue with the values initially large, but the moment we take a terminal node. If it is terminal, you label it solved. So, these 4 nodes will be terminal, we will label them solved and we will put the values that we have, which is 50, 30, 30, and 70. So, at the end of this so, if you remember the AO star algorithm had these 2 phases, the forward phase and the backup phases essentially.

So, in some sense this is like the forward phase, whenever we are looking at the live node it is like forward phase because we are adding the children. Either all children, if it is a max node or one child if it is a min node, but we are moving forward or moving down in the tree. But once it is solved nodes, we go into the backward tree essentially. So, now, we have these 4 nodes and all 4 will get label solved essentially. Because they have this plus large value, they will always be ahead of the queue.

Even if this one will gets label solved, it will go at the rear of the queue because those will have plus large. Then, eventually all 4 will get plus these values, and they get sorted 70, 50, 30, and 30 in the priority queue. We always pick so, it is a priority queue, I do not need to repeat that, we always take the node at the head of the queue. So, once we have

this 4 solved nodes in my, in the priority queue, the next node that will get picked is this one 70, one is 70.

So, these represent 4 clusters let us call them A, B, C, D. Each of these clusters represents 2 strategies, and each of these values represents an upper bound on those 2 strategies. And the best looking cluster is d, and d comes to ahead of the priority queue and so, SSS star picks that algorithm, it is a max node. So, if it is a max and solved, there are two cases, one is that, it has a sibling left in the tree all the other cases when it does not have a sibling left from the tree. So, both these nodes 30 and 70 and in fact, all 4 have a sibling still left in the tree.

So, they have this case essentially. So, in case they have a sibling, you replace with sibling and lower value, whichever is the lower of the 2 values is you give that value to that cluster, why? Because remember that, the value of a strategy is always the minimum of the value of the leaves essentially. And if you are looking at one more leaf in the strategy, and if it has a lower value, we must keep the lower value. If it has a higher value, we must keep the lower value from the first one, it does not matter where the lower value comes from.

So, keep the sibling as a representative, replace with sibling and with a lower value, if there is no sibling. So, I will write else here, and this else means that there is no sibling, replace with parent. So, we will come to this so, let us follow these steps. So, this 70 would be the head of the queue, and it will get removed from the queue, and it is sibling would be added. So, which amounts to say that we are exploring this node, and we are looking at this node then, the value of this cluster would still be 70 because that is the lower of this two values 70 and 80.

So, this would still be at the ahead of the queue because it has its value 70, it has no more siblings left so, this will get added to the queue and this will be get label solved. So, let me say this represents label solved as a, as we had done in the AO star algorithm. If it has no sibling, we place it with the parent with that value. So, still this cluster D is represented by this node here and its value is 70.

At this point, observe that because this is a max node here, this is never going to play a role any further because it has an upper bound of 30, and max is getting 70 from here, max knows it is getting 70 from here. This is like an alpha cut off will take place this

place, either you do it explicitly or it will languish at the end of the queue never coming to the fore essentially does not matter.

That is, that step of alpha cut off is captured by this step here. If it is min and solved now, just think carefully about this, if it is a min node and it is solved and implicitly we are not saying this here, it is at the head of the queue, it is at the head of the queue. So, this is solved and it is at the head of the queue. It has a sibling, but the sibling is never going to play a role, why?

Because it is a min node, it is, it is got an upper, it is, it is value got a, got a value of 70, it is sibling has an upper bound of 30, and this sibling is never going to influence this max node, and it is going to not consider this at all. So, this is implemented by saying that, if it is a min node and if it is a solved node then, just replace it with its parent. So, now, this and the, parent gets label solved.

So, I should say that, parent and solved in both cases whether it is a max node or a min node. For a max node if there are no siblings then, the parent gets label solved and gets the value. If min node, it does not matter whether there are siblings or not, but its parent will get label solved, and it will get a value of 70 in this case. Now, we go back to this step that node is solved and that is a max node, its sibling is not yet solved essentially it is not a terminal node we are somewhere up in the tree.

So, we replace this with the sibling, the same process if max is solved replace with sibling and lower value. In this case the value is 70. So, we basically this node becomes live with a value of 70. It amounts to say that I want to evaluate this with a bound of 70 essentially. Now, since it is a live node, it is like a recursive call essentially henceforth. It is a live node, we follow this, add all children for max, one child for min. So, it is like a recursive call which means we add all children for max and one child for min.

So, we first look at this and then so, all these added to the queue, this comes to the head of the queue because it has a value of 80, it is a terminal node. The moment we see a terminal node we missed it. This comes to the head of the queue then, it gets a played biased child with a value of 80 then, there is no more siblings left, this is still at the head of the queue. So, this gets added to the queue with the value of 80 and solved. The moment this gets label solved with a value of 80 its parent gets label solved with a value of 80, and when the, when a max node.

So, remember that we are always considering the nodes at the head of the queue and the highest value node we are looking at that all times. This is at the head of the queue sorry, this is not correct with the value of 80, but with value of 70 essentially the lower of the 2 values. Now, there are no more siblings left for this max node at the head is the max node with the label solved so, this is already gone away actually, it is been thrown away, only one copy is been kept it is. So, no more siblings are left so, this min node gets label solved with the value of 80.

Now, since it is a min node and it is solved, it does not care about it, and it is at the head of the queue, it does not care because this means whatever the bounds that are coming from elsewhere will be lower than this, it is in the hydraulic queue and max is going to. So, max is getting a value of 80 from here it is getting a value of utmost 50 or 30 from here depending on what happens in these other nodes which we have not seen, but utmost the value of 50.

So, they are behind in the queue, the moment a min solved node comes into the head of the queue as we have said here, we just replace the parent with the solved node. In this case we replace this with a value of 70. So, what have we done? We have looked at this. This SSS star algorithm has looked at 1, 2, 3, 4, 5, 6, 7, nodes essentially before it terminated. Notice, it terminates with the same value of 70 which the alpha beta found and which is of mini max would have found essentially.

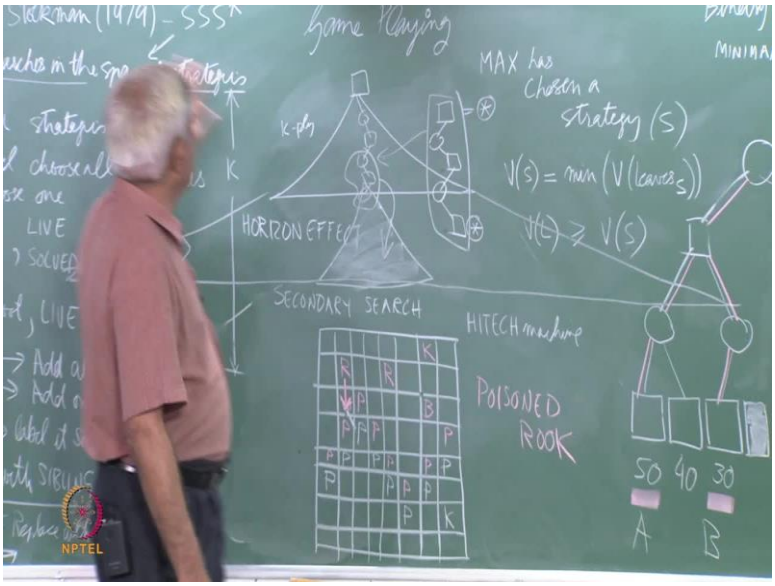
So, all the three algorithms mini max algorithm, alpha beta algorithm, SSS star algorithm, we find the same mini max value obviously, otherwise they would be different. So, they find the same move with the same mini max value. Mini max would have seen all these 16 nodes, alpha beta sees only these 10 unshaded nodes that we have drawn here, and SSS star sees only these 7 underlined pink nodes that we have seen here.

What is more important? Observe that it is attention is always been focused towards this side of the tree where the best moves lie for both the players, and it has basically solved this part and ignored this part altogether essentially. It is not seen this node for example, which was seen by alpha beta of this node, which was seen by alpha beta and so on essentially. So, in fact, it was shown by stockman that, alpha beta that, SSS star, if any node is seen by SSS star, alpha beta will also see that essentially. Where? Yes because this was 70, this also should be 70 and solved.

So, SSS star is the best first variation of alpha beta you can see. The only difference between what we said earlier when we talked about best first search, we used a notion of a heuristic function which was domain dependent. In this case we do not use the heuristic function in that sense, but we use an estimate of the solution. A heuristic function also gives an estimate of a solution, but here we have a different mechanism to arrive at an estimate is that is by sampling.

By sampling all strategies to get upper bounds on all possible strategies, and that in the process we found these clusters. And then we always refined the best looking cluster. So, that is where the best first nature of the algorithm comes into play. So, I want to end with a couple of things, one is that if you are talking about a real world game playing program.

(Refer Slide Time: 40:10)



Then, you do a certain amount of search, you do a cape line look ahead, but there is a danger of lurking here. Of course, the farther you can see the better, but because the tree is growing exponentially, you cannot do too much search and most algorithms do not go beyond 10 ply also essentially.

Now, this something called the horizon effect, and this is as follows that, this supposing, there is one particular line of play which is of interest, which is being evaluated by certain mechanism as which happens naturally in this. So, let us say this sequence of nodes has a role to play in determining that either this better or something else is better,

it does not matter. It is an important sequence of events moves. So, each is a move. So, max move, min move and so on.

Now, supposing in this you were to insert a set of arbitrary moves which are pointless let us say, max makes some move, max makes a move, min makes a move, max makes a move, makes the move. And let us for arguments sake say that, this state is equal to this state. It is possible, if you are looking at a game like chess for example, max might make a knight move, min might make a knight move, max might take the knight move back, and make min take its knight move back.

So, both have made a move and undone the move and so, they are in the same state here. But what happens to the search? In search supposing, we insert this whole sequence here then, this part gets pushed out of the horizon because of this arbitrary move that you are doing, this cart gets pushed out of the horizon which means that, something that is important which was happening in these moves is no longer noticed by this algorithm because it is search is only till the horizon.

So, this effect is called the horizon effect, and what people have done is that, we often do a secondary search, before making a move they do a little bit of secondary search to verify that the move is indeed not a there are no catastrophic lurking behind that move or something like that. So, very often people do this. Obviously, the secondary search will only do this much amount of search, which is at last less than actually searching this whole thing up to this depth, that would have been meant much more worth essentially.

And finally, I want to end with an example, which shows that there can be a limitation to search. So, this is a well known example which was fed to this hi-tech program, this hi-tech was a multi processor, 64 processor chess playing machine developed by Berlino in Seymour, and this position was given to hi tech and it illustrates that search is not always capable of doing something, unless it is full search of course, which are other forms of reasoning can do. So, the position is as follows.

So, let us say, these are pawns P, P, P, P, P. So, white has this position. So, it is a very contrite position somebody invented it just to show that search can have its drawbacks. So, these are pawns, 8 pawns white house, one in each column, and white has only the king left here essentially. Let us say, the opponent also has 8 pawns which are kind of

head to head with these 8 pawns. In some sense, creating the kind of a dead lock, but the opponent has other pieces also.

So, opponent has for example, a bishop here and a rook here, and a rook here, and a king here. So, the opponent is strong in terms of material advantage. This white player has 8 pawns and the king, this red player or black player whatever you want to call has 8 pawns and a king, but also has 2 rooks and a bishop essentially. So, there is a significant material advantage essentially. What happens if this red moves this rook here? That makes this move and now its white's turn to move and this position was given to hi-tech essentially, this is called the poisoned rook.

So, if you just look up poisoned rook on the web, you will get this position and some story behind this essentially. So, what is the story? Red moves the rook here, in some sense offering it to this pawn, right? White can capture this rook like this, if you know the chess rules, and white, the game playing algorithm that we have been talking about will basically do this. They will do some k ply search, apply the evaluation function. What is the evaluation function? We had seen it is a combination of material advantage and positional advantage.

And then, choose a best move based on this evaluation function. What will this poor white program do? It will see that, it is getting to capture the rook. As a result of which, it makes this move of capturing the rook, but this rook as a title says, it is a poisoned rook because what happens? Once this pawn moves away from here, this impregnable fortress that right had. So, observe that these rooks cannot attack any of the pawns otherwise because rooks can move only in this direction, and the only bishop that black has it is the one which will attack its own color.

So, if you know chess you will see that you will be for example, all black squares and these will be all white squares, and it has a black bishop, it can never attack, it can never break, left to itself black can never break into this fortress that white has constructed. But the moment white moves one pawn from this chain, it opens its territory to black and then, black can actually as you might say, invade from here and win the game essentially.

So, even players, most even players will look at this position and say yes. The only thing that the white can do is to move the king around and then, black cannot do anything, but

this chess program which was not able to reason at a what you might say as a higher level or a meta level, or see further it can only do this search of a tree thought that it was going to get some material advantage.

So, it captured the rook and actually lost the game essentially. So, which is the, thus the lesson here is that to build intelligence systems, you need not just one form of reasoning, but many forms of reasoning working together. The other form of reasoning is that kind of reasoning that we are talking about here know, analyzing the structure in some way or making some difference about this thing which this program is not able to do.

So, we will stop here with games, there are other games plays algorithm, game playing algorithms that we will not consider. For example, Berlino had a algorithm called B star, which we will not consider, which is also a kind of a heuristic search algorithm which had a sense of direction, but we will limit ourselves to the SSS star, which is a much simpler algorithm to talk about essentially. So, we will end with games here and move on from this place essentially.