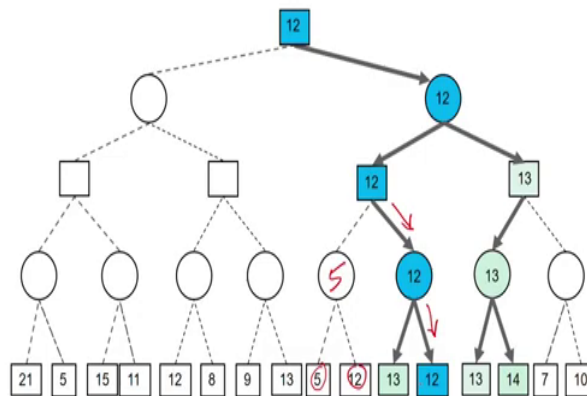


Artificial Intelligence: Search Methods for Problem Solving
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Chapter – 08
A First Course in Artificial Intelligence
Lecture – 56
Game Playing SSS*: A Best First Algorithm

(Refer Slide Time: 00:15)

Strategy returned by SSS*



So, in the last session, we had informal Introduction to the SSS star Algorithm. Now, let us go ahead and formalise the algorithm. We saw that SSS star algorithm returns strategy and just to recap a strategy is a subtree of the game tree in which you have one choice for max; max has frozen max has choices and you have to consider all choices for min because you do not know what min will play and we keep doing that till the end.

So, the strategy that SSS star or the algorithm that SSS star pursues is to look for the best strategy and it starts off with a cluster of partial partial strategies and keeps it finding the best strategy.


(Refer Slide Time: 01:09)

```

SSS*(root)
1 OPEN ← empty priority queue
2 add (root, LIVE, ∞) to OPEN
3 loop
4 (N, status, h) ← pop top element from OPEN
5 if N = root and status is SOLVED
6   return h
7 if status is LIVE
8   if N is a terminal node
9     add (N, SOLVED, min(h, eval(N))) to OPEN
10  else if N is a MAX node
11    for each child C of N
12      add (C, LIVE, h) to OPEN
13  else if N is a MIN node
14    add (first child of N, LIVE, h) to OPEN
15 if status is SOLVED
16   P ← parent(N)
17   if N is a MAX node and N is the last child
18     add (P, SOLVED, h) to OPEN
19   else if N is a MAX node
20     add (next child of P, LIVE, h) to OPEN
21   else if N is a MIN node
22     add (P, SOLVED, h) to OPEN
23   remove all successors of P from OPEN

```


SSS*: Iterative version



Start by adding root to the priority queue and continue till root appears SOLVED at the head

Forward phase: add LIVE nodes till the horizon. Evaluate on the horizon

Backward phase: when SOLVED apply the backup rule



Artificial Intelligence: Search Methods for Problem Solving

Deepak Khemani, IIT Madras

So, here is a version of SSS star which is iterative in nature. So, this gives the complete algorithm and in the following slides, we will look at each of the three possibilities in a little bit more detail. This iterative version of the algorithm it maintains a priority queue just as best first search algorithms that we studied earlier did.

And, the nodes in the priority queue are the nodes in the game tree and apart from the name of the node it contains two possible values that every node is going to be either LIVE or

SOLVED; by LIVE you mean that you do not know it is min max value and, by SOLVED you mean that it you know it is min max value.

And, it will have a value associated with it which is that if it is a partial it is a LIVE node then the value would be an upper bound on the particular cluster; if it is a SOLVED node, then the value would be exact essentially.

So, as you can expect we start by at the root node we add the root node to the priority queue and we will continue this algorithm till the point when the root appears again at the head of the priority queue and it is appears as a SOLVED node. So, as long as root is not SOLVED, we will continue the following.

We will construct the strategies which is what happens when we encounter that LIVE node, remember that the way that we construct clusters is that we add children for MAX nodes we add all the children and for MIN nodes we add one child. We will see this again in a little bit detail in the following slides, but the process of constructing the clusters is done whenever we encounter a LIVE node ok. So, we add more LIVE nodes till we reach the horizon and then we evaluate those nodes at the horizon. We will see this in more detail.

The other option is when we look when we retrieve a pop node retrieve meaning when you see when we are popping nodes from the priority queue the node set would be popped is the ones with the highest h values and if they are LIVE; that means, we need to add more clusters if they are SOLVED; that means, we have SOLVED and we have to go in some sense in a backward phase and apply the min max backup rule essentially.


So, whenever we encounter SOLVED nodes we move a little bit higher into the game tree and we will keep doing that till eventually we reach the root. So, let us look at these three components which are lines 1 to 6 in this is the initial initialization phase; line 7 to 14 is the forward phase in which you are constructing clusters and lines 15 to 23 are the backward phase in which you are backing up values essentially.

(Refer Slide Time: 04:31)

```
SSS*(root)
1 OPEN ← empty priority queue
2 add (root, LIVE, ∞) to OPEN
3 loop
4 (N, status, h) ← pop top element from OPEN
5 if N = root and status is SOLVED
6   return h
...
```

SSS*: Initialize priority queue

Terminate when the root pops out of the priority queue with *status* = SOLVED.



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

So, this is the initialization phase and you can see that that is what we are doing here. We start off by adding this triple which is the root node and it is LIVE in the sense that we do not know its value and a value infinity. In practice, instead of infinity if you have programming you would use some plus large or something like that.

And, it says that the sky is the limit in some sense for the value of the root node that the min max value can be as high as plus large. And, of course, what we do throughout in the algorithm is that we go through a loop in which we pop the top element from this priority queue. And, the top element will have a name, it will have a status which is LIVE or SOLVED and it has it will have some h values.

We will keep doing that and because we are writing algorithm which is iterative now, which is kind of going back from the recursive version to the iterative version is that if the root

comes to the top of the; top of the stack top of the priority queue, we can terminate because root has got the best possible value all other clusters which are partially defined have values which are lower.

Why lower? Because they are not at the top of the priority queue; root is at the top of the priority queue and it has a value which is known and it is a SOLVED node. So, we can terminate and return the value h. So, till the point that the root does not appear SOLVED at the head of the priority queue, we do the following.

(Refer Slide Time: 06:21)

SSS*(root)


- 1 OPEN ← empty priority queue
- 2 add (root, LIVE, ∞) to OPEN
- 3 loop
- 4 (N, status, h) ← pop top element from OPEN
- ...
- 7 if status is LIVE
- 8 if N is a terminal node
- 9 add (N, SOLVED, min(h, eval(N))) to OPEN
- 10 else if N is a MAX node
- 11 for each child C of N
- 12 add (C, LIVE, h) to OPEN
- 13 else if N is a MIN node
- 14 add (first child of N, LIVE, h) to OPEN
- ...

SSS: popping a LIVE node

NPTEL

For terminal nodes invoke *eval(N)* and choose *minimum* of bound and *eval(N)*

else insert all children of Max and one child for Min



Artificial Intelligence: Search Methods for Problem Solving

Deepak Khemani, IIT Madras

This is a forward stage. In the forward stage, what we see is that the node that we have popped its status is LIVE essentially. What do we do when we have LIVE nodes? So, you might recall that we have to keep adding it is keep constructing the clusters below that. So, there are these are the cases that we had studied.

We had said that if N is a MAX node then for each child C of N add C, LIVE to h with the same heuristic value whatever MAX node has we add the children and say that this is the upper bound that you have to work with.

On the other hand, if N is the MIN node, then we just add the first child with the same three values LIVE and h essentially. We keep doing that till we reach the horizon and the horizon is determined by a predicate which tells you that N is terminal and when it is terminal at that point we replace its value.

So, first of all we labelled it SOLVED which is not the case when we are generating strategies in the earlier two cases, the label is still LIVE they have to be SOLVED. A cluster gets SOLVED, a leaf get SOLVED only at the horizon because at the horizon you apply the evaluation function and so, you apply the evaluation function to this node N and you keep the minimum of what it got as a bound from the call that was made to it earlier when it was added.

So, for example, remember that we are adding it with a value h in both these cases and now, we are popping it. We are when we are reach the horizon we are saying it is SOLVED and also we are determining its value and the value is going to be the smaller of the two values.

One of them is the value it gets from the top and the other is the value it gets from the evaluation function. Now, remember that we started off by saying that the value that we will start with is plus large or infinity.

So, the effect of the forward phase is that you go down to the horizon and effectively compute the value of the node as returned by the evaluation function. This is happens in the first phase.

So, you keep going down and keep at the terminal nodes we evoke eval N and we choose the minimum of the boundary it has got and the value that is returned to us by the eval node. In

the other two cases we insert all children for a MAX and one child for min. So, we have seen this for MIN right. So, this takes care of the forward phase of the algorithm.

The forward phase of the algorithm happens whenever we are looking at the partial strategy we want to refine it further. So, we generate clusters and which means we add children which are also LIVE nodes till we reach the horizon and the horizon is when we apply the evaluation function.

So, once you can imagine that all when the cluster formation is happening in our small example that we saw there were four clusters. So, we constructed this tree which (Refer Time: 09:56) the tree downwards selecting all children for MAX and one child for MIN, and at the lowermost level at the horizon level we had values for those four nodes and those values are the values returned by the evaluation function.

So, once we have all the leaf nodes which have been evaluated and that will happen automatically because this is a priority queue and all the nodes have been inserted initially with the status LIVE and with value h and this value h was infinity to start with.

So, all of them will have value infinity and one by one they will get picked up and replaced by their actual evaluation functions. And, they will get replaced in sorted order in some sense because it is a priority queue. The highest eval value will be at the head of the queue.

So, once we have popped all LIVE nodes and replace them by SOLVED nodes at the horizon then we start popping solved nodes from the horizon which is equivalent to what we said earlier was that you refine the best strategy why the best? Because it is a priority queue. And, the cluster with the highest eval value will be popped first essentially.

(Refer Slide Time: 11:16)


SSS*(root)

```
1 OPEN ← empty priority queue
2 add (root, LIVE, ∞) to OPEN
3 loop
4 (N, status, h) ← pop top element from OPEN
...
15 if status is SOLVED
16   P ← parent(N)
17   if N is a MAX node and N is the last child
18     add (P, SOLVED, h) to OPEN
19   else if N is a MAX node
20     add (next child of P, LIVE, h) to OPEN
21   else if N is a MIN node
22     add (P, SOLVED, h) to OPEN
23     remove all successors of P from OPEN
```

SSS*: popping a SOLVED node

If Max node is not the last child then add sibling (looking for a lower value)

Else for both Max and Min add the parent as SOLVED



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

That happens in the third phase of the algorithm that we have written from lines 15 to 23. So, if when you pop this element and if the status is SOLVED then you identify who the parent of this is. So, if this is N then you have to keep track of the parent P because eventually you have to back up the values to the parent P. So, we identify P as a parent of N; if N is a max node N is a last child then we add P, SOLVED, h to the.

So, we saw in this small example and we will see in a slightly larger example again that once you have looked at all the children of a MIN node and you have evaluated the last node. Remember that we will always be keeping the lower of the values, once the last node has been solved it is parent can be labelled as SOLVED essentially.

But, if it is not the last node if it is an earlier node for example, here and that corresponds to this case here that if it is just a MAX node so, we have already gone past the case where it is

not the last child. So, it is just a child. So, what we do is we add the next child this thing. So, if we have solved this one here then we add the next child here and then make this add that to the priority queue.

If N were to be a MIN node for example, this P were to be SOLVED then because of the fact that it is SOLVED completely and even if this it is parent had many other children. But this value is at the head of the priority queue and your popped it from the head of the priority queue which means it is going to be better than the other children. The other children are only partially refined, this node is completely refined.

So, at this point you can go ahead and say that the root that is parent is also SOLVED. So, that is the third case that we are looking at if you have popped a SOLVED MIN node. So, remember it is a SOLVED node and it is a MIN node, then we can just add its parent and say the label is SOLVED and whatever the value of h is.

Not only do we do that because we know that its other children are not going to be considered any further, we can remove them from the priority queue; we know you no longer need to keep them. So, this is the algorithm and let us try to look at this again from the perspective of the iterative version that we have written now and also look at a slightly larger example ok.

So, this is what we are just saying that if MAX node is not the last child, then add its sibling else for both MAX and MIN add the parent as SOLVED essentially. Its only the differences when you are looking at MAX nodes and because their parent is MIN you need to explore whether you are going to have a sibling which is a lower value.

Whereas, if you are looking at a MIN node, then because it is coming out of a priority queue it is the one with the highest value we do not have to look at this look at its siblings and we can simply say that the parent is SOLVED. So, this is in some sense a key to this iterative version of this algorithm.

(Refer Slide Time: 14:53)

```
SSS*(root)
1 OPEN ← empty priority queue
2 add (root, MAX, LIVE, ∞) to OPEN
3 loop
4   (N, player, status, h) ← pop top element from OPEN
5   if N = root and status is SOLVED
6     return h
7   if status is LIVE
8     if N is a terminal node
9       add (N, player, SOLVED, min(h, eval(N))) to OPEN
10    else if player is MAX
11      for each child C of N
12        add (C, MIN, LIVE, h) to OPEN
13    else if player is MIN
14      add (first child of N, MAX, LIVE, h) to OPEN
15  if status is SOLVED
16    P ← parent(N)
17    if player is MAX and N is the last child
18      add (P, MIN, SOLVED, h) to OPEN
19    else if player is MAX
20      add (next child of P, MAX, LIVE, h) to OPEN
21    else if player is MIN
22      add (P, MAX, SOLVED, h) to OPEN
23  remove all successors of P from OPEN
```

SSS*: A programming note

Add a parameter "player" that can be *Max* or *Min*
– easier to check the case



So, let us look at an example and see how it works, but before we do that if we are implementing this then perhaps instead of having a triple in the priority queue you can add a fourth parameter which you can call as player.

So, you have the name of the node as before you have the status as before you have the h value as before, but this fourth parameter called player can keep track of who you are talking about is it a MAX node or is it a MIN node.

So, instead of saying that if N is MAX or if N is MIN which of course, when you are writing the algorithm at a higher level it makes sense, if you are implementing it this might make it easier for you to for you to check whether the node is MAX or MIN. Because you are popping the node and you know whether this is MAX or this is MIN.

The the player parameter can take two values and you can check easily that the player is MAX because that is what you get out of the pop node essentially. So, just a small programming node before we look at an example.