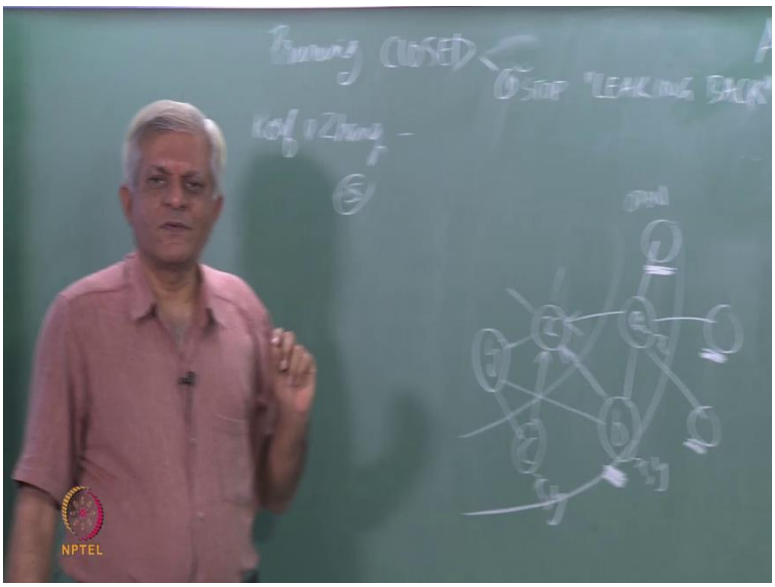
Artificial Intelligence
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture No - 23
1) Pruning the Open
2) Closed Lists

(Refer Slide Time: 00:11)



Our space saving versions of a star and today we want to first look at pruning the close list. So, just to recap assuming that you are working with a heuristic function which satisfies the monotone criteria or the consistency criteria we are worried about two things 1 is that as a community, now calls it stop the search from leaking back. This means basically just imagine the search frontier being pushed out into the search piece the open node should not come back from inside the search piece whatever what could have been in the closed list.

Essentially, that is the first objective and the second objective would be of course to reconstruct the path. So, let us first worry about the first objective is it how can you stop the search from going backwards. So, in 2000 or so Koror old friend who is been working on this for a while and his student Zhang.

They devise the salary sum which I will name in a little while, but it work has follows that imagine that you have this node x and you generate its children; let us only look at the forward children. So, let us say we expanding the search frontier here and let us just

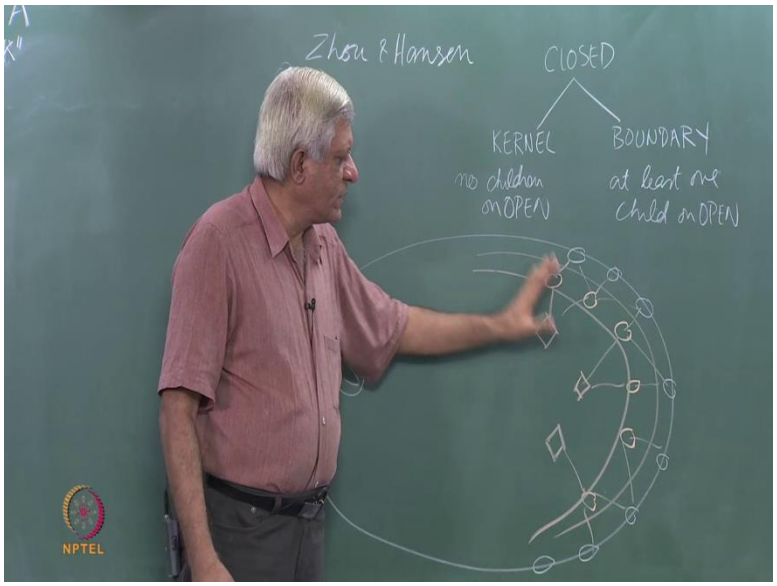
call them a b c essentially now what Korf and Zhang suggested was that you store along with these nodes. So, this is open this is only be the new open, so with every node in open you store it is parents list of its parents. So, with a for example, we will store x here with b also we will store x with c also we will store x what is the idea the idea is it this list forms a kind of a list essentially.

This means that when we in turn were to pick a or b or c for expansion we would not generate those children which we have listed in this list here. So, it is a very simple mechanism for search to be pushed only in the forward directions essentially. Now, for example, if there was a y here which was related to x and let us say this y also connected to this c and b essentially. So, either point when y was expanded then this x this could come x comma y this also will become x comma y. So, with every node in open, we maintain a list of nodes which will not be generated when that node is when this is called with that node essentially. So, it is like at abort list with every node essentially.

So, with this simple mechanism we can see it has a effect if you think about this that every edge is diverged only once while searching. So, this edge from x to a is only diverged in this direction or in other words a back pointer is put only in this direction a x can never become a child of a x can never become a child of b x can never become a child of c and. So, when a is generated some new nodes where we generated may be even b may be generated, but x would not be generated. So, that the children of a are going to be these nodes. So, this is 1 mechanism for stopping the search from leaking back which means that, so let us say this source is somewhere here and search frontier expanding.

So, when we when we generate children of a it will only be the forward looking children after a if we generate b then a will not be generated, but some other children of b would be generated and the search will only push in the forward direction. So, one task of getting into loops is taken care of by modifying the nodes in open list by them with extra information as to which nodes should not be generated.

(Refer Slide Time: 05:31)

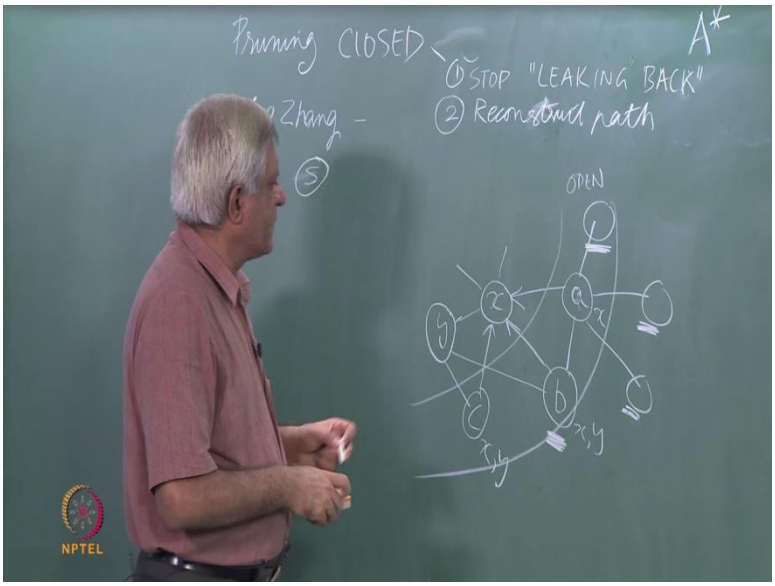


A little bit later 3 or 4 years later, another Chinese student Zhou working with Hansen supervisor produced a different algorithm in which they had a slightly different mechanism for search from leaking back. Their mechanism always follows that the set closed is partitioned into 2 sets 1 is called, the kernel and the other is called the boundary and the way to distinguish between the kernel and the boundary is that this has no children on open. This is a negation of this which means at least 1child; this is remembered this is the close list essentially. So, if I want to draw the close list if this is a start node then everything inside this is on the close list and everything on this frontier is open.

So, the boundary nodes which are if I can draw in this colour would be these nodes which are which have at least one child in open and all other nodes. So, let draw them as a numbers which has children which are all on closed would be the kernel essentially. So, they distinguished between closed and open and the idea here is that when you generate children of open you only need to look at the boundary and boundary serves the old function of closed which has to avoid the search from looping essentially. So, if a child is present in the boundary, then you do not generate it, otherwise you generate it.

It has a same function of pushing the search in the forward direction, so this boundary layer this intermediate layer of nodes the boundary layer and it basically stops the search from coming back any node will not generate a child in boundary and the boundary list.

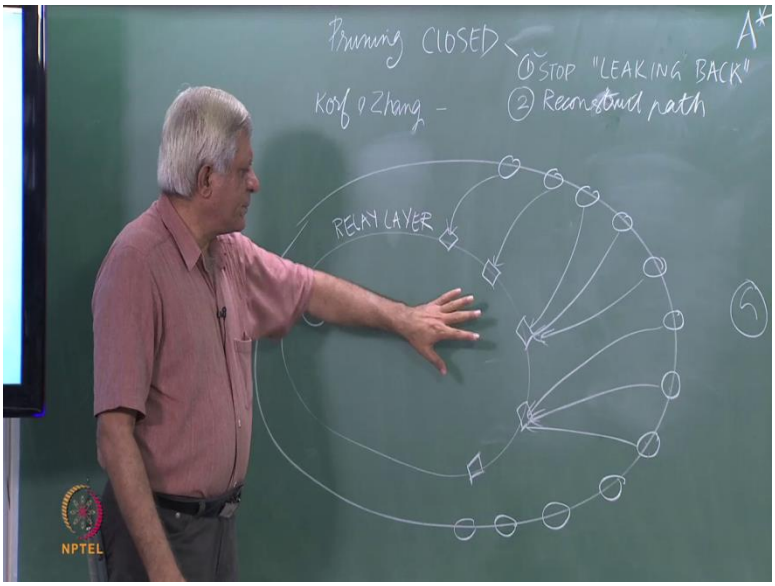
(Refer Slide Time: 08:29)



So, now let us address the second function of closed which is to reconstruct the path because it allows us to reconstruct the path from the start to the goal know how do we how to Korf and Zhang handle this. So, I hope this is clear this mechanism for pushing the search only in the forward direction and taking care of the first path because there is no looping which is going to take place taking care in this case by modifying.

The open list allows only successes, which are not in the closed, in this case why actually storing a pruned, closed list. They will see that this kernel is not necessarily stored, but the boundary has to be stored and the boundary is going to be only the edge of the closed essentially in some senses. Hence, if you only store those then you can stop the search from coming back, so let us look at the other problem of how to reconstruct the path. So, Korf and Zhang's algorithm actually maintains only the open list it does not maintain the close list at all essentially.

(Refer Slide Time: 09:52)



So, the algorithm the search face that this algorithm generates something like this, so I have this start list and i have this open list. I have a gold node here and that is all I have, only nodes on the open observe that these nodes are modified they store information about what nodes I do not want to generate. So, basically it has some extra information for every node there will be a list of few nodes which are tabor for it to be generated as children. Now, instead of closed what they maintain is a layer of nodes another layer of nodes which is roughly like this and this is called the relay layer and the relay layer is a list of nodes.

So, let me draw the relay with these numbers this time and we will keep it for the relay nodes only and every node on the open. So, remember that I a star we maintain the parent pointer that every node had a parent which we would if necessary if we found a better path around. Now, we are saying we will never find a better path anyway, so we do not need to worry about that, but we still and the function that the parent pointer did for us was to allow us to reconstruct the path when we found the goal now in this algorithm by Zhang and Korf. Every node on open maintains a pointer to earn ancestor which is in the relay layer.

So, every node will have 1 ancestor in the relay layer and so on essentially, so of course it is not pruning the close completely it is replacing close by a another layer essentially which is a relay layer essentially.

It is pruned everything else essentially now what it tries to do is that the relay layer is roughly at the half way mark I will just write half way mark here. So, I am not writing the details here we will give pointers to the papers as well as you can refer to my book and I have described the algorithm there initially when the search starts it, you do not maintain any pointer or you can say figuratively.

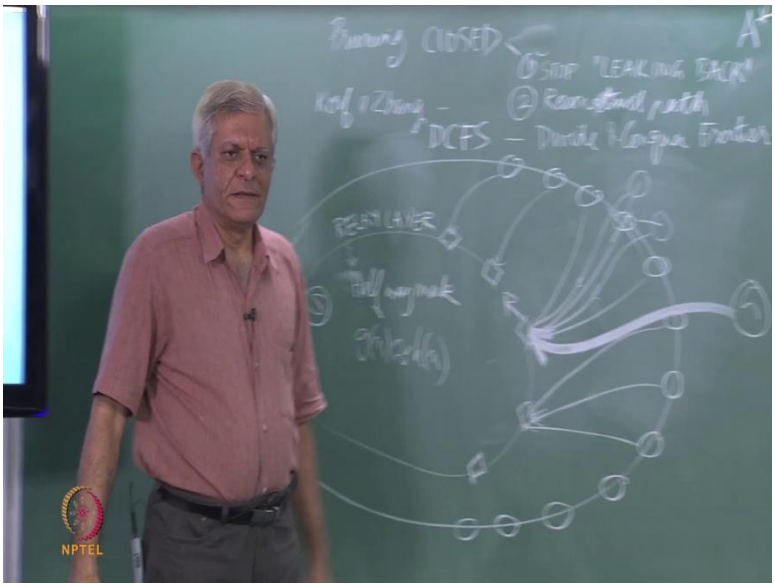
You maintain a pointer to the source node essentially an ancestor point know which do not really have to, but at some point it decides that a given node is at the half way marked from the start to the goal essentially. It says I will make this node to the relay layer, so the first question is relay node the first question is how you decide that the node is at the half way mark it does not have to be exact. Roughly, at the half way mark you look at its values what would happen at the half way mark at the f value g and h should be roughly equally. So, I will say g of n is roughly equal to h of n if your function is good, then it will be closer to me equal if the function is very conservative.

Then, it will end up setting up relay layer little bit earlier than actually it is requires, so maybe you can have a factor or something, but let us not get into those details essentially. So, this is how algorithm works it maintains one boundary sorry it maintains 1 search frontier or the open list and after a certain point in the search it maintains a layer relay layer essentially. Initially, you can imagine when the open is here there is no need for relay only when it has pushed beyond roughly the half way mark which is here. It starts constructing a relay layer and then it pushes forward essentially, so that is a basic search algorithm no closed, but this thing.

So, you can imagine that when a child when this node is expanded into these 2 nodes then this will be deleted this node will be deleted and the parent pointer would be pointed to this essentially. So, it will it will pass on the pointer to its children and so on, so at some point to the goal is picked essentially at some point the goal is picked and the goal will have some pointer to some relay node here. So, when you pick that goal you know what is the cost of reaching that goal and the optimal cost of reaching the goal because you know its g value because we are we are shown that a star finds an optimal path. So, when you pick the goal node you have you know the cause of the optimal path to the goal, but you do not know the path only know is that.

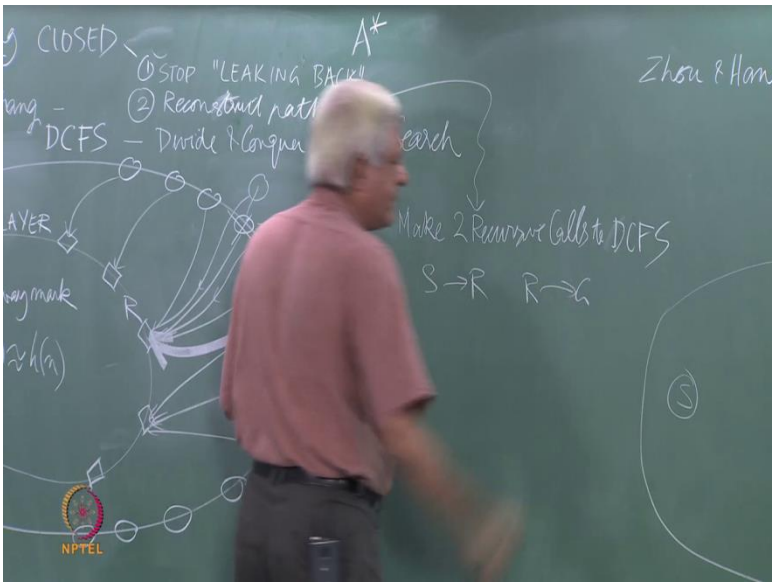
There is one relay node let us call it r which is an ancestor of the goal node on the path it is on the path and it is an ancestor of the goal node. Now, how what do you do you want the whole path you want all the nodes which take you from the start node to the goal node essentially. All we have is 1, it is like somebody tells you that if you are going from here to Delhi, then Bhopal is a relay node or something like that I do not know what distance is. Let us see that that you have to go to you have to first go to Bhopal, then you go to Delhi and you will get the optimal path, so let us first reveal the name of this algorithm it is called d c f s.

(Refer Slide Time: 17:20)



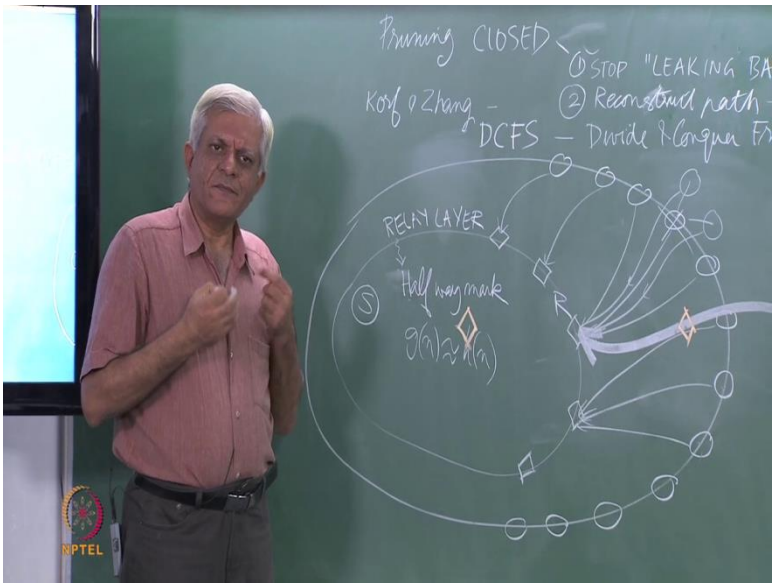
The expansion is divide and conquer frontier search, this will give you clue as to how do you reconstruct the path.

(Refer Slide Time: 17:34)



So, to reconstruct the path you make two recursive calls to divide and conquer frontier search 1 call goes from s to r and the 2nd call goes from r to you make 2 recursive calls and what would that give you?

(Refer Slide Time: 18:07)

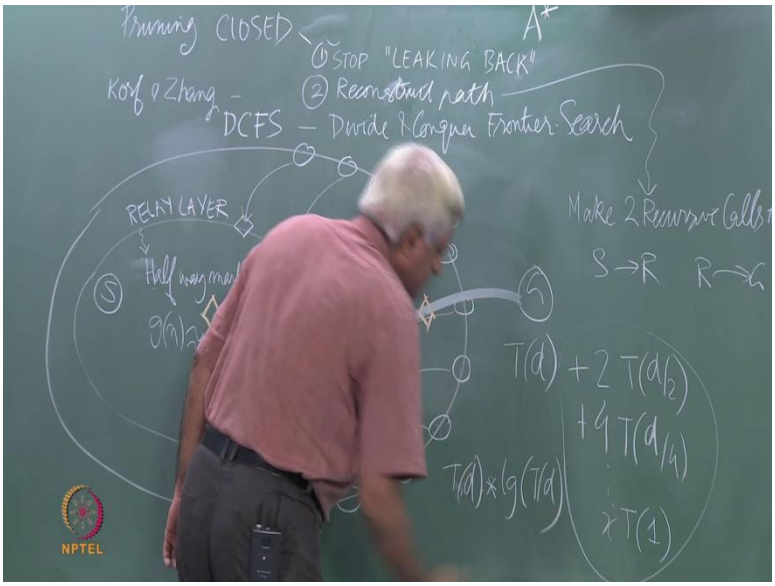


That would give you two more nodes somewhere here and somewhere here then you make 4 recursive calls from here to here and here to here, so all of this and you keep doing that till the problem has just become an edge that.

The next node is just a child of first node essentially that a base clause when you terminate the questions essentially. So, remember that once you have solved the first once you made the first call to divide and conquer frontier search you finish with all your memory requirements. All you know is that there is a start node there is a relay node and there is a gold node and then you making a fresh call which is to a smaller problem or roughly of half a size provided this is this holds that g is roughly equal to h. Otherwise, there may be a unequal number of size which means a as you can imagine it is like working with a unbalanced binary rather than a balance binary you may do more work in 1 half and less.

In the other half, but as long as you can divide it roughly half you will split the work half and half and you will keep doing that till f eventually reconstructed the path the full path. So, that is why this name divides and conquer frontier search, so this space requirement of the algorithm is only to maintain the open list or the frontier and a relay layer essentially and that is all it needs to do essentially. So, we are thrown away most of the close list and in the kind of problems that we discussed the sequence alignment problems it is close which is going faster. Then, essentially open is only going linearly close is going as quadratic of the size of the problem what would be the complexity of this.

(Refer Slide Time: 20:20)



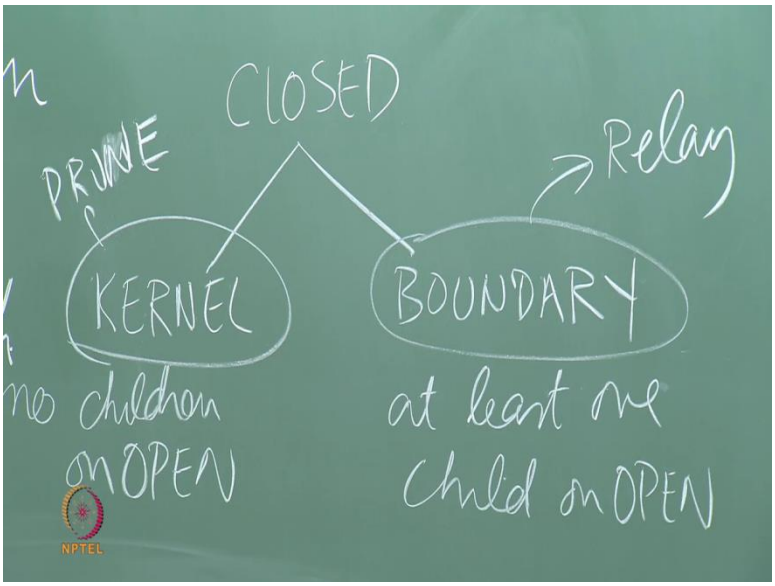
So, you can say that if the original problem of depth could be sized solved with prime complexity d whatever d is, it depends on the function rarely. Exponential function in

general plus the extra work that you are doing what is the extra work 2 into to f d by 2 plus 4 into t of d by 4 of depth 4 depth d by 4 and soon and so on till you solve this small problem of depth 1. Essentially, for some value x into c of depth 1, so all that is extra work you are doing, all the extra work is done to reconstruct the path essentially how much is the extra work.

So, you can solve this its multiplied by log to the base 2 from the depth of the complexity of td multiply take the log of that. So, if the td were to be exponential in nature be there as to d then this would be d times 6 log, so you are doing if you are finding a path of length then essentially you are doing d times extra work. So, if the path is of length 50, then you are having 50 times extra work to reconstruct the path, but in the process you are saving on space essentially let us see what Zhou and Hansen do.

What they say is why do we breakup the problem into half what is the rational for breaking up the problem into half. Of course, we know that divide and conquer strategy says that if you break it up into half then you can know solve it using this complexity. In this era of increasing memory available they say that you should do this pruning of closed only if you are running out of memory essentially.

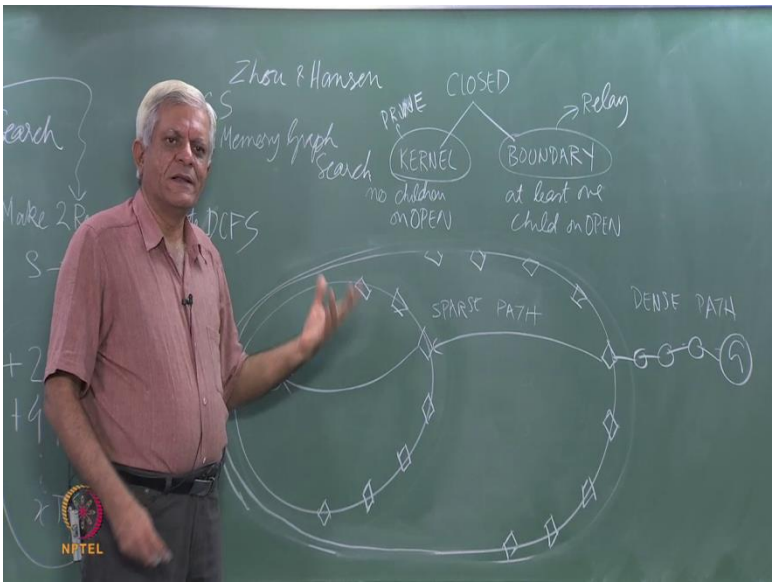
(Refer Slide Time: 22:51)



So, their algorithm is called SMGS and it expands to smart memory graph search. So, what do Zhou and Hansen do they say you just learn it like a star do not worry about pruning or something. You keep track of how many how much memory your algorithm

is using somehow and if you can at some point realize that you are running out of memory, then you prune essentially. What do you prune you prune the kernel you keep the boundary because you need it, in fact when you prune the kernel at that same very time you convert this into a relay. So, initially this algorithm is working with this layer boundary layer and the open layer going neck to neck open layer is moving forward the boundary layer is just following it at some point.

(Refer Slide Time: 24:09)



So let us say this is a situation this is a boundary layer this is the open layer. So, the outside 1 is the boundary layer the inner 1 is the sorry the outside 1 is the open layer and the inner 1 is the boundary layer and whatever inside the boundary layer is closed all the kernel. So, your counter something tells you that you are running out of memory, so what do you do you prune the entire closed and convert this into a relay layer and search progresses from there as before, so let us say its con here. It is got another boundary layer following it essentially, so at all points a boundary layer just follows this search frontier because it needs keep the search from leaking back.

Essentially, every time you generate children of open you check on the boundary layer if they are children or not and then this is the area between this curve and this curve is the kernel which you have not pruned essentially. Then, again some somebody tells you are running out of memory, so again you covert this into another layer. So, that is why it is called smart memory in the sense it is aware of how much memory it is using and

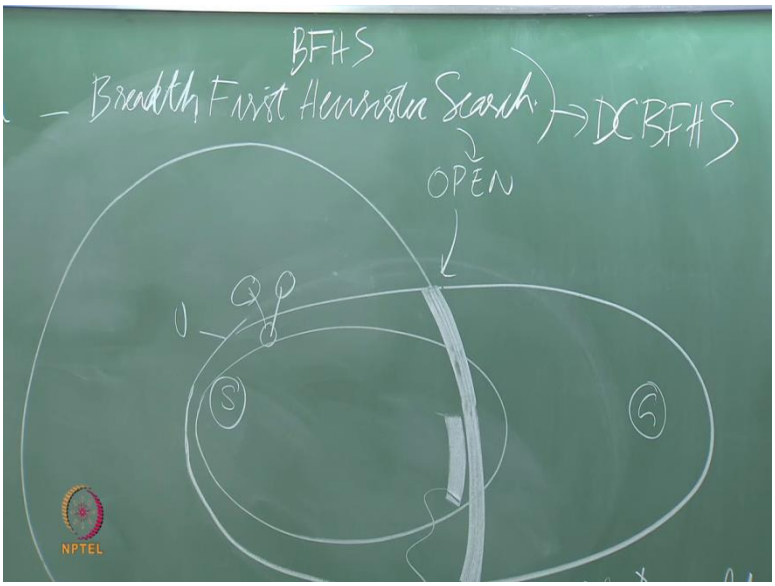
whether it is running out of memory essentially. So, unlike divide and conquer frontier search this maintains one relay layer roughly along the half way mark smart memory graph.Search maintains as many layers as required it could be 0 it could be 1, it could be 2, it could be 4 depending on how big the problem is and how much memory is available to you.

At some point, when it finds the goal it would have some path up to some relay layer which they call as a dense path and from this layer to another layer by a mechanism. You can workout it would have a series of ancestor pointers which they call it as a sparse path. So, in their terminology divide and conquer frontier search has 2 sparse paths 1 from start to relay and 1 form relay to goal in smart memory graph search if you are solving a very large problem, you may have a bigger sparse path. Then, of course you have to make that many recursive calls to solve each of them, so it may be the case that the first time around you make 5 relay layers.

So, from start to 1, 1 to 2, 2 to 3, 3 to 4 and 4 to 5, so you will make 5 recursive calls, but for each of the recursive calls is possible because it is a smart algorithm. You may not do many more recursive calls because you can imagine that it is close to what this memory can tolerate essentially. So, it is a little bit different from this in that sense that its aware of how much memory it can use and only prunes kernel when it is running out of memory. Remember that pruning kernel incurs this cost of reconstructing the path because once you are thrown away all these nodes you have to do all this recursive calls to reconstruct this path. You have to do recursive calls to reconstruct this path, but here you are not thrown away.

So, you just have to follow the back pointers and you can just get the path from there, so depending on the how much memory available this behaves in a mode smart way essentially now. So, let us move on to now pruning the close list essentially which is the work, in fact, carried these two groups all over again, so how does one prune the close list. What Zhou and Hansen showed and this was around 2004, so it is not so back in time compared to looking at.

(Refer Slide Time: 29:07)



They give some idea some which is called sounds like a curious name, but you can see what they are doing breadth first heuristic search it is it is sound like contradiction in terms. So, the basic idea behind breadth first and we are talking about pruning open, now most algorithms which prune the open list relay on getting some upper bound on the cost essentially. So, compute U upper bound on f of the problem that you are trying to solve what is the maximum possible value that the cost of solving the solution can be. Essentially, how do you compute an upper bound, one way is to use some ready algorithm to try to find the solution.

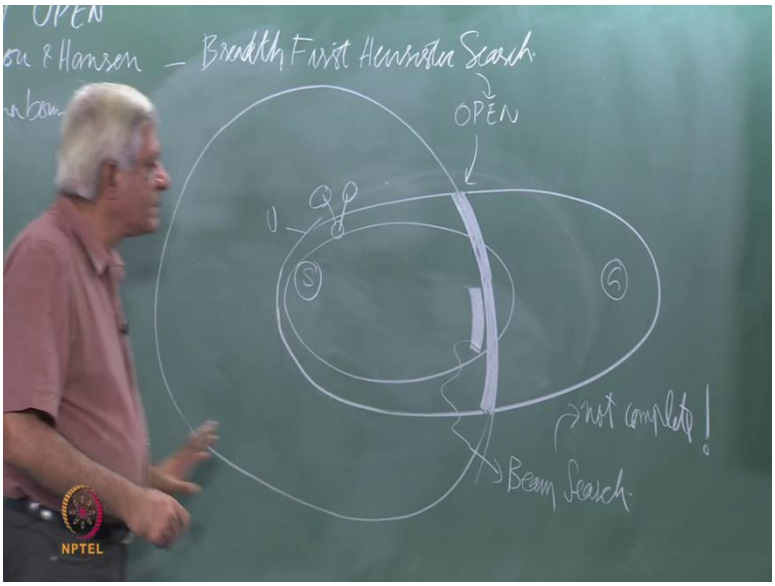
So, you do some beam search with very thick beam based and hopefully you will get some solution it may not be optimal, but it will give upper bound. Essentially if you know one solution the solution can be made the upper bound and that is a theme which runs in too many variations of these algorithms that we are going to see either as and when you find better solutions. You reduce the upper bound essentially and then what they say is the following that if this is your start goal.

This is a goal node and if you have a boundary which is the upper bound well that is not quite correct. So, the upper bound serves as a boundary which means that any node with f value greater than this value u you will not expand. So, if you generate a child here for example, then you will never expand these two children essentially, so that is the purpose that this boundary is serving.

So, we are only going to search within this hypothetical boundary, which is determined by the f value essentially. So, before expanding a node from open check whether it is less than u only then you expand it essentially now if you want to do a star like search. Then, the open at some point would look like this the node that you would, whereas if you do breadth first search keeping this upper bound in mind. Now, if you want to do blind breadth first search your search boundary would look like this. Assuming that you now costs are roughly equal for every edge essentially thus visualise the problem, but if you are doing this breadth first heuristic search which means you are guided by this upper bound which you have generated by some heuristic algorithm.

Then, your open is only going to be this much, so this is a open for this algorithm and basically empirically one can observe that the size of open for this breadth first heuristic search is smaller than the size of open for a star which is roughly like this. So, this should give a visual intuition, but this is do not take it at face value, but it just to allow you to give an intuition essentially now another variation.

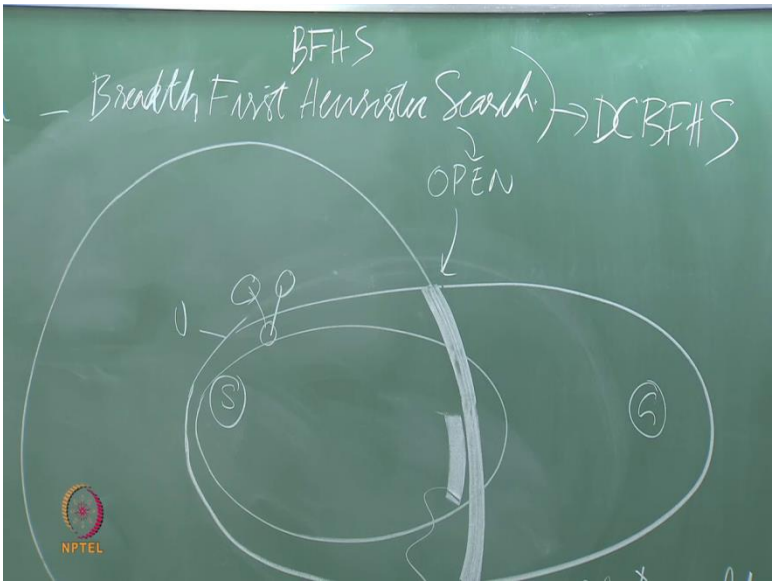
(Refer Slide Time: 33:43)



This is to prune this even for which is to keep it of constant width and you can imagine the algorithm is beam search. So, beam search and we have explode beam search earlier is the variation of beam search in the sense that you keep searching till you find the goal rather than hill climbing like beam search where you stop. If you do not find a better node here, meaning of beam search is that you maintain an open list of constant width.

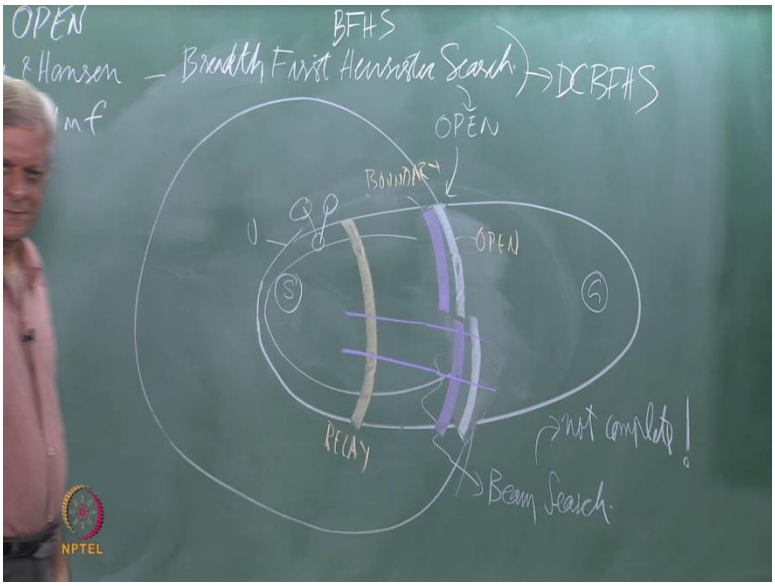
Then, you search towards the goal now; obviously, it is not complete because you are throwing away other nodes, see breadth first heuristic search is complete. It will find a path for the goal but beam search go off in this direction away from the goal and we never actually give you a path. So, it is not complete, so how can we make beam search complete before we do that Zhou and Hansen also gave us.

(Refer Slide Time: 35:01)



So, this is Breadth first heuristic search and you can convert this into divide and conquer breadth first heuristic search by using a divide and conquer mechanism of which we are by now familiar which means that along with.

(Refer Slide Time: 35:27)



So, in general if you want to draw this graph you would have an open which is to be seen like this followed by a closed not closed a boundary layer which is just behind it. So, you can imagine the search you know progressing in this direction and this open will slowly get converted into a boundary and the new open will move forward and also 1 layer somewhere in between. So, this is a relay, this is a boundary and this is open, so if you maintain this 3 layers of node you can convert breadth first heuristic search into divide and conquer breadth first heuristic search and you will have to do the same mechanism of reconstructing the path.

So, even algorithm which is not only saves on open it also saves on closed because you are no longer keeping the close you only keeping the boundary and then relay layers essentially exactly like what smart memory graph search would have done. Now, you can of course do the same thing with beam search beam stack search. So, we will visualise this algorithm as a search tree because it is easier to do it like that. So, just imagine that this is a search tree that that search same search algorithm generates and we saw this mapping.

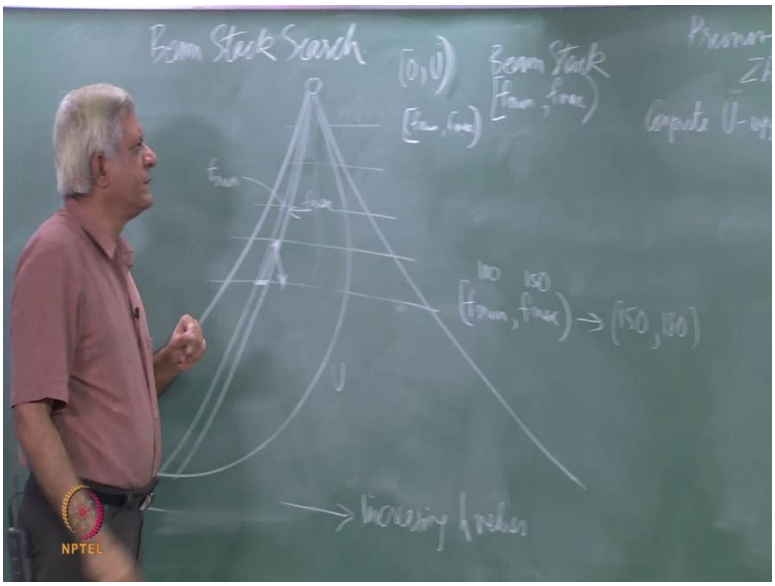
Earlier, we assume that the search tree is ordered, so the lowest heuristic values are from the left and the right and the highest at each layer is ordered. So, that this is increasing h just for the sake of visualization we assume that this tree is ordered by increasing h values.

In this case, the boundary that we are talking about their given by the u upper bound on would look something like this. So, you have to think a little bit about this and convince yourself that this is how a u value. Remember that this side is values lesser than u and that side is values greater than you and because u is on upper bound on cause that we have somehow figured out we know that we do not have to search that part of the tree. So, we have to only search this part of the tree essentially because it is a heuristic search. We can now assume that since we assuming that in this visualization the tree is draw in such a that heuristic values are increasing from left to right, so you can imagine the heuristic search also progress from left to right essentially.

So, the divide and the beam search is essentially beam search in this space, first of all it is a beam search. Now, the beam of course start searching from the left node that I have drawn it in the middle just to show what happen now beam search is incomplete we has observed essentially. So, you can make it complete by introducing back tracking that a little bit like what recursive best for search would have done no we are not talking about the backed up values just back trucking and retry. So, if it runs into this upper bound it back tracks and try something backtracks and try something.

So, something like back trucking behaviour we want to stimulate except that keep in mind that in this visualization which is actually my own idea, you would not find it in that paper this heuristic values are ordered from left to right. So, it is like a on this space it is like doing that first search from left to right, but how do we do this back trucking in how do we n practice implement that.

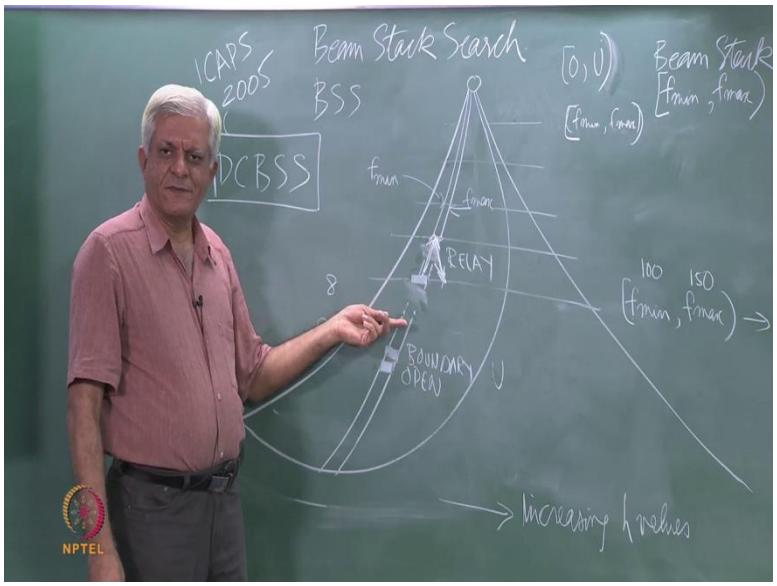
(Refer Slide Time: 41:10)



We do it by maintaining another data structure called the beam stack, where at each layer we store 2 values 1 is f min and the other is f max and it is a open interval on the right hand side and close interval on the left hand side. So, these two values are telling us as to where in this space you are searching, so the value of f min is here and the value of f max is here. So, it tells algorithm which part of the search space you are searching essentially and this it does.

For every layer, we have f min these two values, so what is that mean that if you are back tracking and you are coming back to this point in this layer you not found the solution. You have to come back to this layer you want to start a second search here, and then essentially these are the value f min at this level and f max at this level you go back to this layer and reset this value. So, let us say this is 100 and this is 1 50 let us say in some domain you replace it with 1, 50 and some other value whatever because we have the open list you generate the open list.

(Refer Slide Time: 43:15)



So, you go back to this parent here, generate the open list take children only whose values are greater that equal to this 1 50 or f max essentially and construct a new beam layers depending on something it could something like 1 80 or something like that. What is the initial value for this beam stacks every value will have 0 comma u. So, what is beam stack is doing is it helping with the back tracking process once you back track to a layer and you generate. So, you go to these nodes, generate their children again, so again which of those children do you want to now explode, this beam stack will tell you that you have already seen values up to 100 and 50 and look at value which are greater than 1 50 essentially f values.

So, generate only look at those values, so depending on of course how many there are this value could be 1 80 or it could be 1 90 or whatever essentially. So, you must convince yourself that maintaining this beam stack allows us to search completely in the search space. So, of course it is very difficult to visualize in this space, but in this space which is on ordered h value it is easy to visualize that the search will progress from left to right. If you can search this entire space inside this view, your algorithm is going to be complete essentially and this beam stack allows us to do that essentially, so this is called BSS beam stack search and the next step.

As you can imagine is divide and conquer beam stack search you can expand this you what used to this idea of DC BSS means what is this to this maintains only 4 layers like

those divide and conquer breadth first heuristic search. It maintains one open layer, one boundary layer, sorry one boundary layer here and some relay layer in the peak. It maintains only these 3 things beam stack search maintains all this everything which is inside this beam. So, the question I want to ask which i hope has occurred to you is in beam stack search you have the parents of every node. So, you could go back to the parent and regenerate the parent's children and take the next set of children in divide and conquer beam stack search, you do not have the open layer.

You only have the boundary layer and you only have the relay layer how can you back track. Now, I hope you see the problem isn't it may be I am going a bit fast here just because I am running out of time, how can how can this search go here? Retry this space, let us say the relay does not matter let us say this also relay or something does not matter how can it go here and how can it do this type because we do not have the parent of this relay node here. What is more we do not have the parent of the boundary node we cannot go back to it is parent and then this parent, so this was a paper published by Zhou and Hansen ICAPS 2005.

ICAPS is international conference on automated planning and scheduling and for this paper, they got the best paper award in the conference essentially. So, it is possible, so let me ask I want to do, I want to stimulate the behaviour of beam stack search which means I will go down searching down the beam and if I hit the boundary I will come back and try something else. I will come back and try something else trouble is if I throw away the close list or the parents, how can I come back and try something else we have 1 minute to answer this question the answer is that you do not talk of going back.

You regenerate from the source again and so suppose think you are at the ninth layer. You want to back track to the 8 layer what do you do you go to the start and generate 8 layers which children should pick the beam stack will tell you that of all the children that you are generating which once are the 1s. These are inside this beam that beam stack has this information, so you go to the eight layer and then you can generate of course there is a extra work again back tracking 1 step would have been just going to the parent and then retry.

Here, you have come from the source all the way to the parent and there from this parent to the next parent again you have to go all the way and so on. So, if have to back track

you have this extra work, but as a result of this if you do not covered to the memory required by the beam stack which goes linearly with it, but these a small value. Let us hope what is the space complexity of divide and conquer beam stack search, it is constant you are just keeping three layers of constant width, the open layer, the boundary layer and the relay layer. So, here starting with an algorithm a star which required exponential amount of space, we have an algorithm which practically requires constant amount of space.

In this era of huge memory sizes, you can keep the beam width as large as you can isn't it and it will work. So, I will stop here, I believe professor Shri Chaudary is waiting outside, He should be waiting outside and with this we will end the search part of it. We will look at problem solving from the slightly different prospective in the next class that we meet on Wednesday, which is you know looking from the goal towards the problems, essentially how can you move from the goal to the problem.