


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

Chapter – 05
A First Course in Artificial Intelligence
Lecture – 46
A*: Pruning CLOSED and OPEN
Smart Memory Graph Search


(Refer Slide Time: 00:16)

When Frontier Search picks the goal node... ...it has a pointer to its relay node Relay 

Solve two recursive problems

1. From Start to Relay
2. From Relay to Goal

- Divide and Conquer Frontier Search (DCFS)



If $T(d)$ is the time complexity needed to find the goal at d steps then time complexity of DCFS is

$$T(\text{DCFS}) = T(d) + 2 \times T(d/2) + 4 \times T(d/4) \dots$$

If T is exponential, then $T(\text{DCFS}) = T(d) \times d$

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

So, this is a divide and conquer frontier search algorithm given by Korf and Zang. What it says is a throw away the close list, but at the halfway point maintain a layer of relay nodes.

When you find the goal node solve two recursive problems, one from the start node to the relay node that the goal node points to remember that is important. This relay node in this

diagram is a relay for the goal node essentially. So, we know that the path to the goal goes through that relay node.

So, you recursively solve the path from start to the relay node and from relay to the goal node and then that itself is solved recursively and so on. And we have this algorithm which does repeated searches to solve the problem. So, what we have achieved here is that we have managed to cut down on the closed space requirement, but at the expense of this extra cost which may be at most d times the original cost, where d is the depth of the goal node.

(Refer Slide Time: 01:24)

Smart Memory Graph Search (SMGS): Zhou and Hansen (2003)

Given that memory is getting cheaper and abundant, one *need not* make recursive calls when A* (without pruning) *could have* solved the problem!



Smart Memory Graph Search keeps track of available memory.

Only when it *senses memory is running out* it creates a relay layer.

In the process it *might create many relay layers*

.... or *even none* if the problem is small enough to be solved by A*.

At all points it identifies a layer of *Boundary nodes*

that can be *potentially converted into Relay nodes*.

The boundary nodes also *stop the search from leaking back!*



So, a little bit later another pair of researchers Zhou and Hansen, they came up with a new algorithm which they called as Smart Memory Graph Search essentially. Now, what was this motivation behind that the motivation is that given that memory is becoming cheaper, machines are becoming faster, and bigger as we said should it be rigid like divide and

conquer frontier search and always recursively keep breaking down the problem into two smaller problems.

What if A star for example without any pruning could have solved the problem, then we would not have a space problem, in the sense that we would have enough memory in our machine to solve A star.

What is the point of doing that extra work if you have enough memory and you are still trying to do this recursive divide and conquer process essentially. So, with this motivation Zhou and Hansen said that we will keep track of the available memory.

So, if the algorithm in some way can keep track of how much memory is available to the algorithm, it will decide whether to break down the problem into smaller problems or not to break it down into smaller problems. So, in that sense, they said that this was the smart memory algorithm. It kept track of how much memory is available to the algorithm and made its decisions appropriately.

So, only when it senses that memory is running out, it creates a relay layer essentially. So, this is a key aspect of the Smart Memory Graph Search algorithm is that it keeps track of how much memory is available. If it feels that the memory is running out, it says go ahead and rule some nodes. And like exactly like what Korf and Zang's algorithm did, and create a relay node and delete all the nodes which are behind in some sense the relay layer.

In this case, it will only behind; in the case of Korf and Zang you kept deleting them as you put them into close only keeping a pointer because you had decided that there will be only one relay layer at the halfway mark. So, you did not need to keep any other nodes on closed.

Zhou and Hansen's algorithm Smart Memory Graph Search says that we will delete closed only when we need to delete. And when we delete close we will insert a relay layer at that point of time. We will see shortly how that works. Now, because it is aware of what is how much memory it needs and it may be solving a very large problem in which case it may create

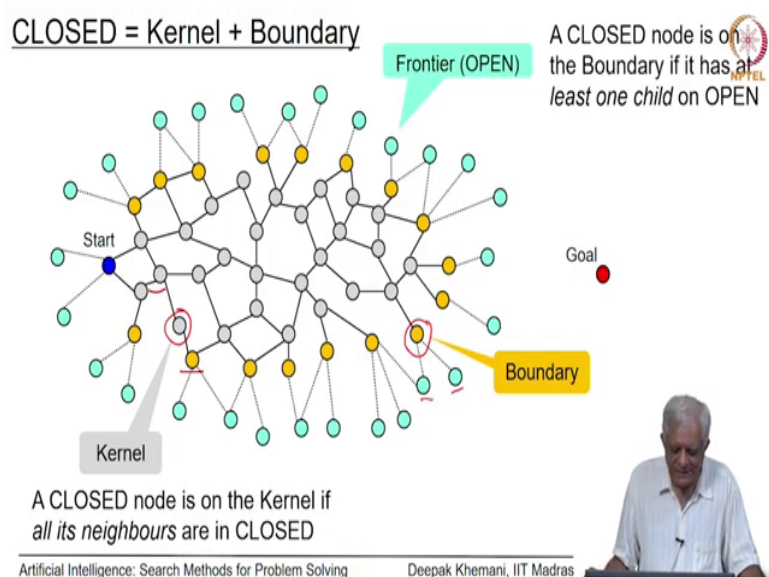
more than one relay layer. Remember it creates a relay layer as soon as it figures out that its running out of memory.

On the other hand, if the problem that you are solving was small enough, and it could be solved within the scope of the available memory, it may not create a relay layer at all, and just finished solving it like A star would have done essentially.

So, it has this flexibility and that is where the name Smart Memory Graph Search comes. It also changed the mechanism by which you post a search to go forward or search not to leak back, and it did that by creating a layer which they call as a set of boundary nodes.

And boundary nodes were only were those nodes of closed which were just next to the nodes on open, they delete everything else except for boundary nodes. And when they decide that it is time to prune some nodes they would convert those boundary nodes into relay node essentially. And at the same time the boundary nodes will serve the purpose of stopping the search from as everyone calls it by then leaking back into the closed case.

(Refer Slide Time: 05:36)



So, what is this boundary layer? Let us look at this diagram. In this diagram the nodes like have been colouring open nodes in cyan color are in the outer periphery of this graph. These are nodes which are being generated and which may be picked for inspection, and an expansion, and the standard process that we have. The rest of the nodes apart from the start node in the goal load are the nodes which are in closed essentially.

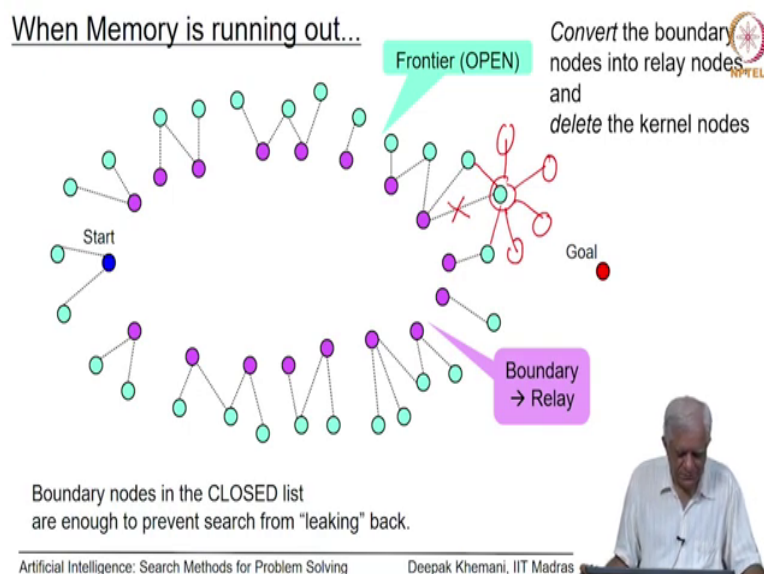
Now, in Smart Memory Graph Search CLOSED nodes are characterized into two different kinds of nodes. One which is called the boundary which is shown in orange which corresponds to what we called as closed earlier. So, a CLOSED node is on the boundary if it has at least one child on OPEN. So, for example, this node is on the boundary because it has got in fact two children on OPEN.

On the other hand, we say that the node is in the kernel if all its neighbours are in closed list. So, again if you look at this example that I have pointed to, it has two neighbours this is enclosed and this other node is also on the close. Remember that both boundary and kernel make up the closed in fact close the union of kernel and boundary and kernel and boundary are disjoint sets. So, CLOSED is equal to kernel plus boundary in some sense.

So, kernel nodes are those which have their neighbours already in closed, and therefore, there is no possibility of them being generated as a child of any node on open. Boundary nodes are those which have some child on OPEN, which means that it is possible that one of those children may be picked up for expansion in which case they would generate its neighbours which would include the boundary node, but the boundary node will alert us and saying that you are going back or the search is leaking back.

And exactly like we did in the early algorithms if you if the node is already on the boundary, then you it will not generate essentially, so that we would be pruned at the time of move gen is applied essentially at the place where move gen is applied. So, that is the role of the boundary. And the kernel is a one which is expandable in the sense that if necessary we can throw it away and save one space.

(Refer Slide Time: 08:11)



That is what the algorithm does when it finds that the memory is running out, it throws away the kernel nodes and so the graph looks like this. And what was called the boundary is now or less really essentially exactly like divide and conquer frontier search we have a set of relay nodes, and these relay nodes will be kept and we will keep pointers to this essentially.

So, what would happen in this particular example is that supposing this is a node that is being generated, then it would generate its children maybe this like this. And it would keep it like this; it would not make any change. The space behind the boundary nodes, the kernel has been deleted, but the space ahead of the boundary node is maintained just exactly like A star would have maintained then. So, we do not prune anything, we do not prune anything we just keep pushing forward.

And we keep identifying at each stage what is a boundary and what is the kernel that is the only thing that we do. That as we put more and more nodes into CLOSED, we identify them as kernel or as boundary based on the criteria that we have just specified.

The criteria being that kernel nodes have all their neighbours in closed, and boundary nodes have at least one neighbour in open node. And that process keeps go on but nothing is prune. So, it is a bit like A star beyond that point except that CLOSED comes in two use now boundary and kernel.

And again if SMGS finds that is running out of memory and again it will now delete the next set of kernel nodes, and it will convert the next set of boundary nodes into a next relay layer. So, as we said earlier it is possible that Smart Memory Graph Search may create more than one relay layer.

And that would happen if the problem that it is solved solving is much larger, and it requires much more space than we have essentially. Remember also that boundary nodes also serve the function of stopping search from going back.

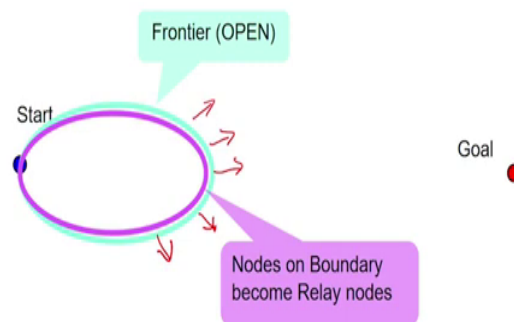
So, for example, if we had generated this node which I have just circled, then we would not generate its parent which is on the boundary as a child. So, the same notion of active nodes that we talked about in the divide and conquer frontier search would also be maintained.

It is just that the mechanism is different here. In divide and conquer frontier search, we remember with every node on open as to which nodes are bound, whereas in Smart Memory Graph Search we do not do that we do not modify the algorithm, we just keep the boundary nodes. And when we generate neighbours of a node on OPEN, we just check on the boundary node; if a node is present, we do not generate it. This is like the old way of using closed essentially. So, it maintains that ok.

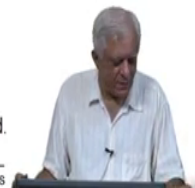
So, many finds that it is running out of memory, it converts a boundary layer into a relay layer and proceeds like A star till the next point where it has sensed that memory is running off, in which case it will create another relay layer and so on the process will continue.

(Refer Slide Time: 11:49)

A relay layer when needed

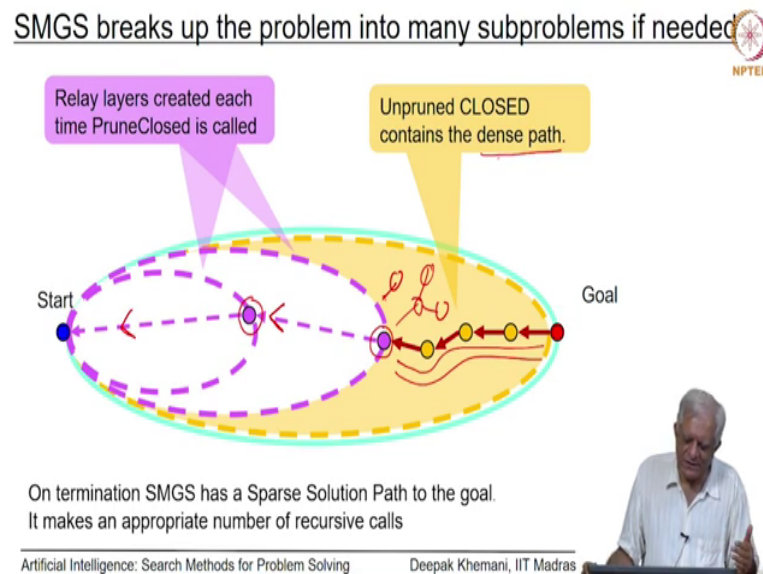


SMGS converts boundary nodes into Relay when prune module is called.



So, it creates a relay layer when it needs, and it deletes everything else inside the boundary layer or the relay layer. And the relay layer will stay essentially. As the search progresses further from here whichever the direction maybe, it will keep that relay layer, it will keep adding new nodes to closed, it will not delete them, but it will keep marking them as either kernel or boundary.

(Refer Slide Time: 12:26)



So, what could typically happens is that you may have more than one relay layer as shown in this diagram, so there are two relay nodes as you can see, one is here and the other is here. And is and the algorithm A star or its version Smart Memory Graph Search has picked the goal node.

It has nodes in closed which are shown in this orange here, brown colour. I have not drawn the nodes, but you can imagine that there are nodes which have been closed, which have not been deleted, and they would have been present and so on. But I have not drawn them I have just used the shaded region to depict the fact that these are nodes on closed.

And in that nodes on closed which have not been pruned, the goal node would have what is called as a dense path. Dense path is what we interested in where we want to know exactly

what is the parent of every node essentially, up to the last relay layer relay node which is here.

Then from the lastly relay node to the previous relay node it has what we call it as a sparse path which means that we know that we have to go to the previous relay node because there we have pointed to that, but we do not know the exact path we do not know the dense path essentially.

And likewise from the previous relay layer, we have a sparse paths to the start node. Start node is always in the path of every node. So, we have now three segments in the diagram that I have drawn here. In the third segment which is closer to the goal, we know the path already. In the first two segments, we have two layers and we have to recursively solve those problems.

So, unlike divide and conquer frontier search, the Smart Memory Graph Search may have many relay layers or it may have none essentially. And it would really depend on the problem that you are trying to solve. If the problem is large it will have many relay layers, and therefore, many recursive calls; if the problem is small, it will not have any recursive call.

So, it will not do the extra work that is needed. Even in this part, you can see in the last segment here, it is not doing extra work. It is just reconstructing the path because it is available to A it in form of the closed list which may have some nodes on boundary and some nodes on kernel which are not been deleted essentially.

So, on termination, this is the situation and it has a sparse solution path, not all of it is sparse part of it is already dense. And it makes an appropriate number of recursive calls to reconstruct the path. In this diagram that I have drawn there are two recursive calls.

And you can imagine that within those recursive calls, it is possible that it may not make any further recursive calls, because if you remember it made that relay layer only when it sends that it was running out of memory.

So, it is very likely that the recursive calls that Smart Memory Graph Search makes as a last recursive calls, and there is no nested recursion inside and that is because by then the problem has been broken down in such a manner that its memory that is available will be enough to solve that problem.

So, you can see that in that sense it is a marked improvement on divide and conquer frontier search. It will do this dividing of the problem only when it needs to. And when it does not need to divide, it will just solve it without doing any extra recursive work essentially. So, these are two algorithms that we saw which allow us to rule the closed list.

(Refer Slide Time: 16:19)



Next when we need we will talk about pruning OPEN. Because if we look at the general search tree in the breadth first version of it if we remember, closed was like this and open was

like this. And we had observed that for the tree which is growing exponentially open grows much faster than closed in general.

We are not talking about the common neutral problems like sequence alignment that we have talked about. But in general in search space open grows faster. And we saw that that was the case of best first as well which means that it will also be the case of A star because A star always has more nodes than best first.

So, typically the heuristic function is not perfect and that is why this happens. And the open list is often the problem and that is why we managed to convert that first search into DIFD which behaved like breadth first which means it went down layer by layer in this tree that we have drawn, but its space requirements were linear essentially because internally it was a breadth first search. So, I am just emphasizing the fact that very often open is a problem. And in the next session, we will explore how we can prune the open list essentially.

So, we will meet in the next time. See you till then.