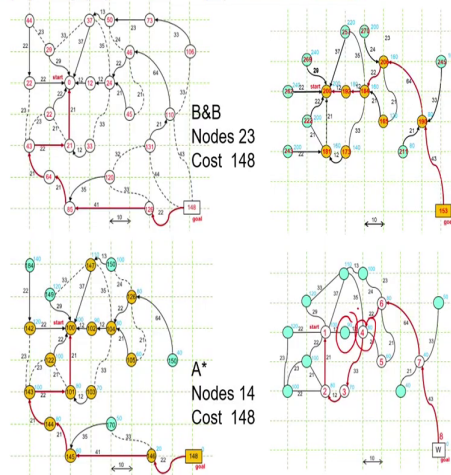


Artificial Intelligence: Search Methods for Problem Solving
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

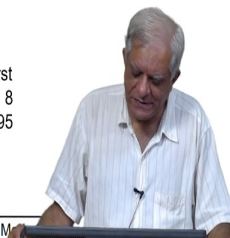
Chapter – 05
A First Course in Artificial Intelligence
Lecture – 47
A*: Pruning CLOSED and OPEN
Variations on A*: The story so, far

(Refer Slide Time: 00:14)

$f(n) = g(n) + w \times h(n)$: B&B (recap)



As the weight of $h(n)$ increases search requires less space, but, eventually becomes inadmissible



So, welcome back we have been looking at space saving versions of A star. In the last session, when I was showing you the impact of the weight on the evaluation function f of n is equal to g of n plus w into h of n ; there were some little bit of a problem with the slides and which has been now rectified.

So, I will start from there; quickly go over the bit in which we were talking about pruning closed and then move on to pruning open today, that is the main topic of today and that is in fact, the last topic of this algorithm A star ok.

So, if you generalize A star and best first search and branch and bound and weighted a star; all of them can be captured by this one function g of n plus w into h of n . And as w varies, the behavior of the algorithm changes and as w increases, it requires less and less space, but as we saw in the case of w A star it becomes eventually inadmissible. So, this is what we were learning to see the case where w equal to 0 is branch and bound, the algorithm explores the entire graph of 23 nodes in this case.

And find the solution which you can see in thicker lines in the bottom left of cost 148. Then, we increase w to 1 which makes it A star; A star uses g of n plus h of n and what we get is slightly smaller graph that is explored.

You can see that A star has explored only 14 of the 23 nodes. But it has found the optimal path also in fact, the same path is what branch and bound found and this of course, we are not surprised because we have shown that A star is admissible.

Then, when we increase the weight to 2; what we got was w A star and you can see that w A stars saw much smaller number of nodes which is 9, but it found a more expensive path which had a cost of 153. In fact, w A star never ventured into the bottom left half of the graph whereas, A star did explore a little bit of the top right but decided that bottom left was better and then moved on to that essentially; whereas, w A star being a little bit more influence by the heuristic function.

Once it found a path which seemed to be going closer to the goal, it headed into that path. The paths found by w A star was better than the path found by best first that is the algorithm we started search with.

And we can see that best first sees only 8 nodes; these are the 8 nodes which are not colored, but which have numbered in them 1, 2, 3, 4, 5, 6, 7, 8. But it found a much longer path and the reason for that was that it did not explore this node because its competitor so to speak was much better.

It was closer to the goal and therefore, it chose a node which we have labeled as number 4 and never discover the path which wA star found essentially. Then of course, we decided that we still want to save on space and we looked at IDA star and we looked at RBFS.

(Refer Slide Time: 03:44)

The Monotone Condition (recap)



The *monotone property* or the *consistency property* for a heuristic function says that for a node n that is a *successor* to a node m on a path to the goal being constructed by the algorithm A^* using the heuristic function $h(x)$,

$$h(m) - h(n) \leq k(m,n)$$

The heuristic function
underestimates the cost of each edge

For A^* the interesting consequence is that every time it picks a node for expansion, it has found an optimal path to that node.

We can design algorithms that never look back!



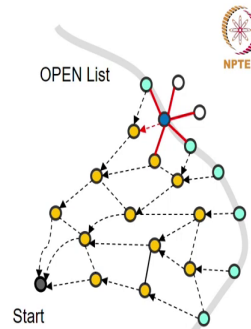
And then we looked at situations where in the monotone condition you can afford to ruin the close list and we explored that in the last session. I will quickly go over these slides, so which will also serve as a recap.

(Refer Slide Time: 04:00)

The role of CLOSED in Search

We needed to maintain the CLOSED list* of nodes for two reasons

- One, to avoid getting into infinite loops, which happen because a node on CLOSED may be a neighbour of the node being expanded.
 - We will look at other ways to avoid the search from "leaking back"
- Two, to reconstruct the path once the goal node is picked up by the algorithm.
 - We will look at a new mechanism to do so

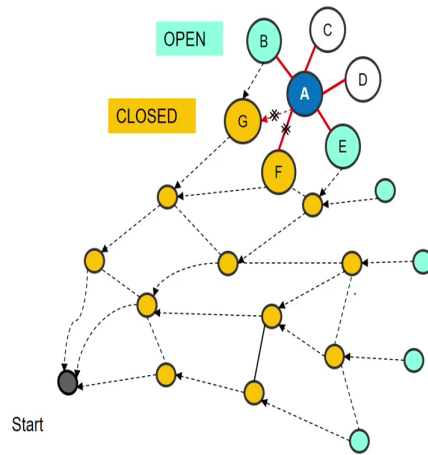


* In practice CLOSED should be implemented using a hash table for efficiency

So, the idea is that because closed has two purposes; one is to avoid infinite loops and the other is to reconstruct the path. To find alternate mechanisms for doing both these things essentially; how to avoid going back to the closed nodes; if we are not maintained close then how can we check whether we have these closed and the other is to reconstruct the path.

(Refer Slide Time: 04:24)

Korf and Zhang, 2000: Frontier Search



Let us say node A is about to be expanded.



Its *active* neighbours are
C,D (new nodes)
B,E (on OPEN)

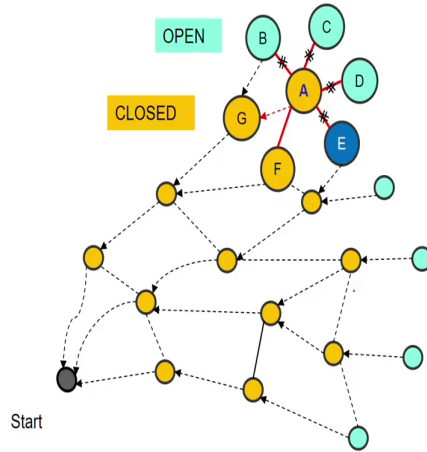
Nodes G and F
(on CLOSED) are barred
(see next slide)



What Korf and Zhang showed was that if you modify the open list to keep a memory of where you came from and bar those nodes from where you came, then you do not have to regenerate them. So, in this example here when A was being expanded; its presence in closed G and F were barred and they were not regenerated. So, the search only looked forward towards B, C, D and E and in that way, avoided leaking back into the closed list.

(Refer Slide Time: 04:55)

Korf and Zhang, 2000: Frontier Search



A is added to CLOSED and its *active* neighbours C, D, B, and E are barred from generating A when *they* are expanded



Remember A* has already found the *optimal path* to A

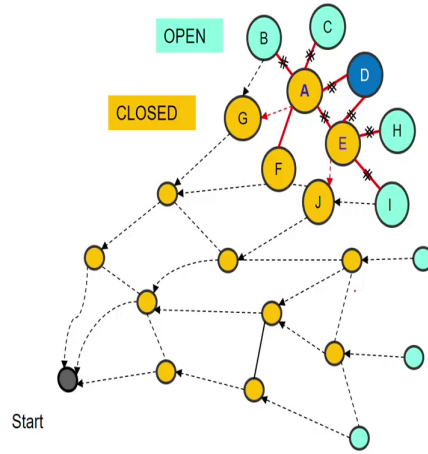
Let E be the next node visited



The same thing we continued when you looked at expansion of node E, when we expanded node E.

(Refer Slide Time: 05:04)

Korf and Zhang, 2000: Frontier Search



When E is added to CLOSED its active neighbours D, H, and I are barred from generating E when they are expanded

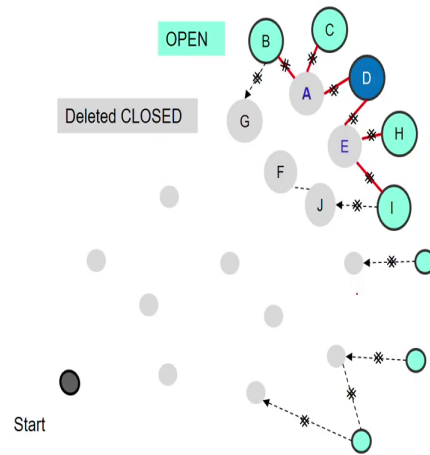
So D will not generate either of A and E



Next, again it only looked forward; it only looked at nodes D, H and I; it did not even look at node A from where it came.

(Refer Slide Time: 05:21)

Korf and Zhang, 2000: Frontier Search



Frontier Search:
Keep only the OPEN nodes,
along with list of barred (tabu)
nodes with each node in OPEN.



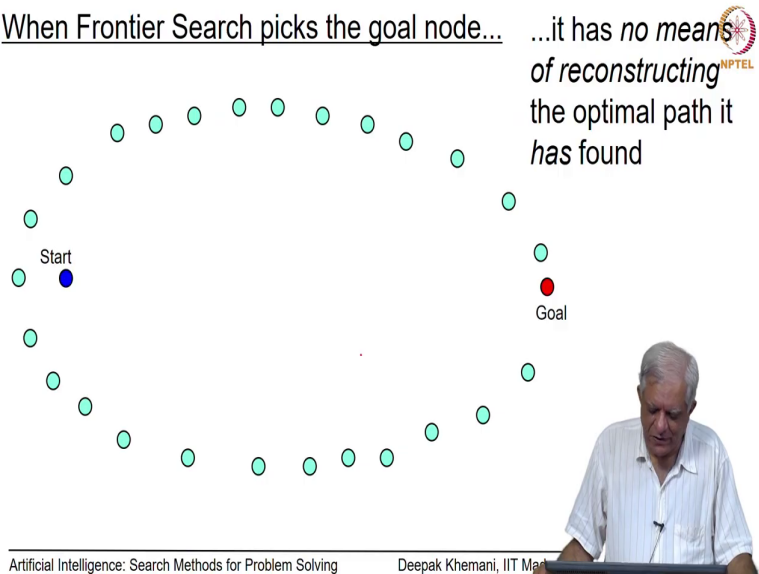
Search only moves forward,
without "leaking back"



And this process continued and we saw that what you could do is maintain only the open list which is shown in blue or cyan nodes with the blue node being the one about to be picked reasonably. Whereas the closed list is deleted entirely essentially and problem remained of how to find the paths to the goal.

(Refer Slide Time: 05:37)

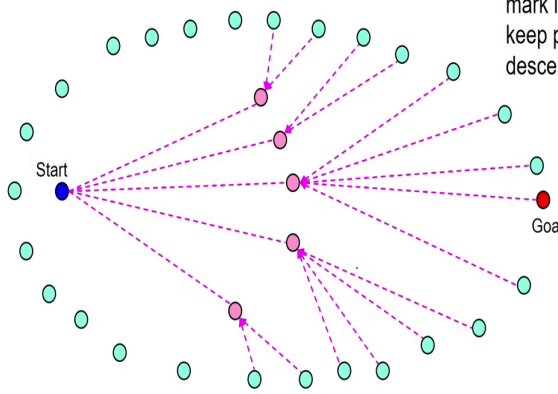
When Frontier Search picks the goal node... *...it has no means of reconstructing the optimal path it has found*



(Refer Slide Time: 05:43)

Relay nodes: at roughly the half way mark

When a node on OPEN has $g(n) \approx h(n)$ mark it as relay and keep pointers from its descendants on OPEN



Start


Goal

Artificial Intelligence: Search Methods for Problem Solving

Deepak Khemani, IIT Madras

And the solution offered by this algorithm called divide and conquer frontier search was to maintain a relay layer of nodes at roughly at the halfway mark and the idea there was that once you have found the goal node, then you can break up the problem into two sub problems and solve them recursively.

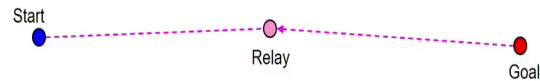
(Refer Slide Time: 05:58)

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 $T(\text{DCFS}) = T(d) \times d$



What is the advantage here? The advantage is that we need less space. We have already thrown away closed and as we solve smaller and smaller problems, we will need less space. and we had seen that by solving this we may be doing some extra work, but that could well worth be the trouble worth well worth the trouble especially of doing that extra work.

(Refer Slide Time: 06:26)

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!*

For accessing this content for free (no charge), visit : nptel.ac.in

Artificial Intelligence: Search Methods for Problem Solving

Deepak Khemani, IIT Madras



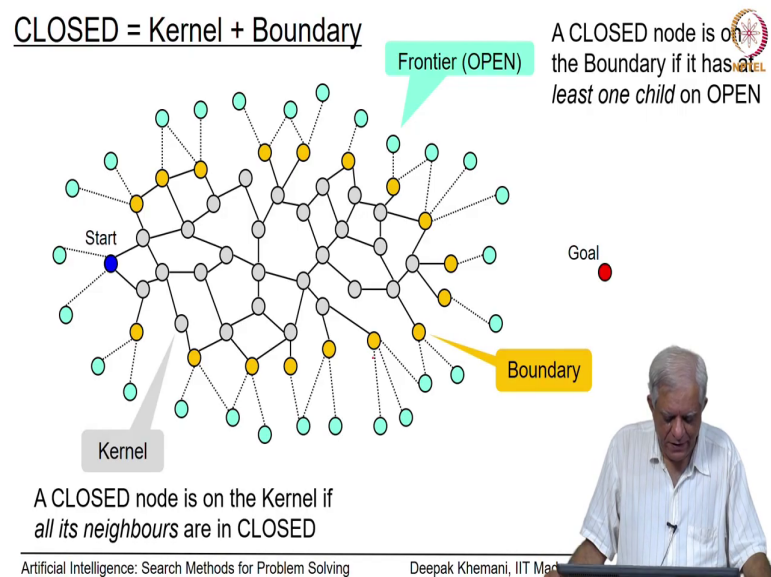
Then, we looked at an algorithm called smart memory graph search which said that why do you want to keep breaking up the problem into smaller parts because there may be the situation where the memory that you have is enough to solve the problem completely. So, do not be rigid about doing only recursive break breakup and solve.

Do that only when there is the need to do that and in that sense, this algorithm earned its name of being a smart memory search algorithm or smart memory graph search algorithm. And what it said was that keeps sensing how much memory is available and only when it senses that memory is running out, it creates a new layer essentially. And we also saw that in this process it might create many layers or it might create none essentially.

So, that would depend on what is the size of the problem that you are solving, if you are solving a very large problem; it might possibly create many layers, if you are solving a small

problem; then it may not create any relay layers and just solve the whole problem in one go itself which of course, would be faster and would fit into the space available that we have essentially. Remember, that our main concern here is to how to manage with the space that is available to us.

(Refer Slide Time: 07:44)



So, we saw this algorithm; this was given to us by Zhou and Hansen and they essentially broke up the close list in to two sets kernel and boundary. Kernel was something which could be deleted whenever required and boundary was kept and boundary was kept to avoid the search from leaking back essentially. So, boundary essentially serve the original purpose of close which is to check whether you are going back and kernel could be deleted.

(Refer Slide Time: 08:17)

When Memory is running out...

Frontier (OPEN)

Convert the boundary nodes into relay nodes and delete the kernel nodes

Start

Goal

Boundary → Relay

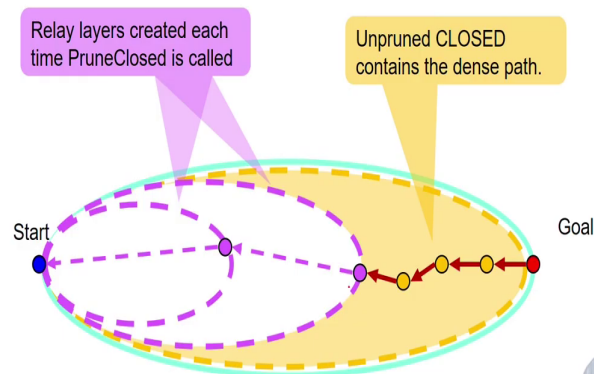
Boundary nodes in the CLOSED list are enough to prevent search from "leaking" back.

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

And what they said was that when you are running out of memory, you just simply delete the kernel and convert that boundary nodes into a relay node and then carry on in A star like fashion again except that; you keep identifying which of the close are in the kernel which of the close are in boundary and if need we create another layer by deleting, another set of kernel nodes.

(Refer Slide Time: 08:42)

SMGS breaks up the problem into many subproblems if needed



On termination SMGS has a Sparse Solution Path to the goal.
It makes an appropriate number of recursive calls

Artificial Intelligence: Search Methods for Problem Solving

Deepak Khemani, IIT Madras



And it just did this process and it would create as many relay layers as required by the problem and eventually solve the problem by making a certain number of recursive call. We also observed that because it creates relay layers only when it is running out of memory, one can imagine that in the recursive call; it will not create any further relay layers and most likely would solve the problem in one go essentially.

So, there is the level of nesting that happens in smart memory graph search would be much lower than that of the divide and conquer frontier search, which kind of very rigidly keeps recursively solving till you have reached an edge as the smallest problem to solve.

(Refer Slide Time: 09:28)



Next Pruning OPEN



Artificial Intelligence: Search Methods for Problem Solving

Deepak Khemani, IIT Madras

Now, we said that the next thing that we want to study is pruning the close list; oh pruning the open list. The reason for that as we have been repeatedly saying is that in a general search tree which is growing exponentially, it is the open list which grows faster than the close list. And we saw that and that is why we could implement algorithms like DFID which behaves like breadth first search, but which use the memory of depth first search which is linear.

But as a expense of revisiting the close node again and again and then we said argued that the amount of work you do in revisiting those closed nodes can be ignored or it is not significant essentially.