

**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 – 36**  
**Finding Optimal Paths: Algorithm A\***

So, welcome back and we have been looking at algorithms to Find Optimal Solutions or Optimal Paths in a graph and today we have arrived at this major Algorithm called A star which is in many ways the highlight of this course.

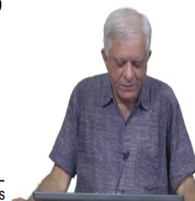
(Refer Slide Time: 00:40)

Algorithm A\*



- [Peter Hart](#), [Nils Nilsson](#) and [Bertram Raphael](#) of Stanford Research Institute (now SRI International) first published the algorithm in 1968.
- It can be seen as an extension of Edsger Dijkstra's 1959 algorithm.
- A\* achieves better performance by using heuristics to guide its search.

- [Wikipedia](#)



It is not a new algorithm, as you can see from the Wikipedia page this was devised by Hart, Nilsson and Raphael I think Raphael was a Ph.D. student who did the work at the Stanford Research Institute which is now SRI International and it was published in 1968 essentially.

So, it is quite an old algorithm, but it is such a important algorithm that it is used in many places where you have to find optimal solutions and we will look at this in considerable detail.

We will see that it can be seen as an extension of Dijkstra's algorithm for shortest path except that Dijkstra's algorithm as we will see again does not have a sense of direction and it is in fact designed to find shortest paths to all nodes in the graph.

We will also see today that we are not given a graph in when we are doing AI search algorithms we are not given the complete graph, the graph is generated implicitly by the Mogen function. So, as and when a node is inspected its neighbours are generated and added to the graph. So, the graph goes dynamically and hopefully we will get a feel of this in the this session and in the next one.

Now, A star is better than Dijkstra's algorithm simply because it uses a heuristic function which means that the heuristic function guides this exploration towards the goal node whereas, Dijkstra's algorithm was searching in all possible directions, it was not particularly concerned about the goal node.

The branch and bound relation we saw was concerned about the goal node, but it did not use the fact that it had to find a path to the goal to influence its behaviour and A star incorporates that heuristic function behaviour into that algorithm.

(Refer Slide Time: 02:38)

**Branch & Bound (recap)** Maintains distinct paths as separate nodes in the search space. Ends up expanding same state space nodes again and again.

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

So, just a quick recap, branch and bound maintains distinct parts as separate nodes in the search space. So, what you see on the left is the search space and the graph that we are searching is shown on the right essentially. So, you can see that even though a node occurs only once in the graph as should be the case in the search space a node occurs many times.

So, for example, the load A occurs here and at this place it represents a partial path every node in the search space represents a partial path. What is the partial path, that this node represents? It says that you have started from S and you have come to A; so it is the path E.

On the other hand if you look at this load A it represents a different partial path. The partial path this node represents is that you started from S, then you went to B, then you went back to

S and then you came to A and you can see that the cost of this path is 12 whereas, the cost of the first part which has a direct edge was 6.

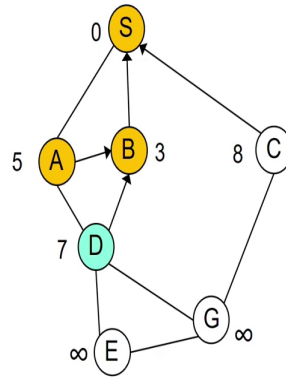
Now, there are other instances of A for example, from S you can go to B and then you can go to A and if you inspect the graph you will see that this in fact, is the shortest path to the node A from the start node, but our algorithm will not give up so easily, it will even explore this node where you have gone from S to B to A and then again B to A back incrementing the cost by another 2 units 4 units 2 plus 2 and the cost is 9.

The reason why branch and bound explores this node is because it is waiting to find the shortest paths through the goal node and the goal node as we saw when we looked at branch and bound has a shortest path of length of length of which goes from S to B to D to G. And because 13 is more than A branch and bound does not know that there is not a shortest path from any of these partial paths that it is explored.

So, branch and bound ends up extending all these partial paths till they have become longer than the path to goal that you have found and we saw when we looked at an illustration of this algorithm that even before it picked this particular path to the goal node which is of cost 13 it had found other parts to the goal node and it had ignored those paths essentially.

(Refer Slide Time: 05:24)

### Dijkstra's Algorithm (recap)

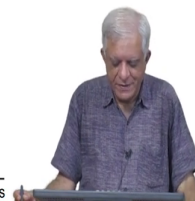


Keeps only one copy of each node, and adjusts the parent pointer when it finds a better path to a node.



Feature adopted by A\*

Initialize Start cost to zero, and rest to infinity  
Colour cheapest node and relax the connected edges  
Has no sense of direction



So, branch and bound does a fair amount of extra work, but it is focused towards the goal unlike Dijkstra's algorithm which we also saw. So, just to recap the Dijkstra's algorithm you initialize all nodes you colour all nodes white as they call it in Dijkstra's algorithm and initialize the start node to the cost 0 and the rest of the cost infinity and the algorithm says colour the cheapest white node and relax the connected edges.

So, here we have shown that we have coloured some nodes in this case S and A and B and this was during the execution of Dijkstra's algorithm, but all the neighbours of the nodes that we have relaxed are have been updated with better paths.

So, D for example, does not have a path infinity it has a path 7, C also does not have a path infinity it has a path 7, because these are neighbours of nodes which have been coloured black in our case orange. But nodes E and G are still labelled infinity because we are not relaxed it

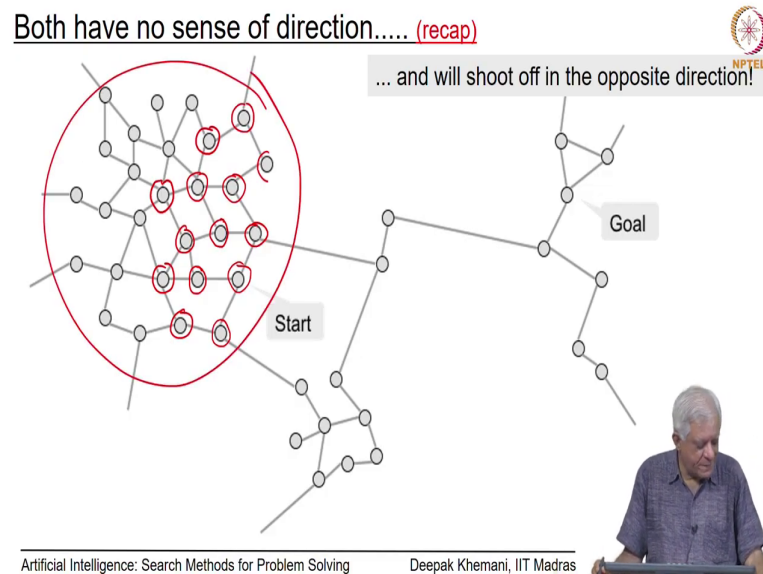
is any of its neighbours which is D or C and at this stage of the algorithm we can see that the algorithm was about to relax a node D.

So, the nice thing about Dijkstra's algorithm is that it keeps only one copy of each node and adjust the parent pointer when it finds a better path node. So, no point keeping longer paths to a given node for example, the node D the best path is from B and D.

So, no point keeping a path from S to A to D or S to A to B to D or S to B to A to D, S algorithm only keeps the best path it is found to load and because from this node onwards the search is independent of what it is found so far.

This is a feature that A star algorithm also adopts essentially. So, instead of maintaining node pairs actually as we did so far, we will now maintain only the nodes, but we will also maintain parent pointers to the best paths or to the best parent from where the best path comes essentially.

(Refer Slide Time: 07:43)



Now, both branch and bound and Dijkstra's algorithm do not have a sense of direction we have been repeatedly saying this because they do not look at where the goal is they just try to find the shortest paths to anywhere essentially.

So, given that this is a start node both algorithms will look at the neighbours of these start nodes which are these three neighbours and we will try to see whether they have found a path to the goal or not in the case of branch and bound in the case of Dijkstra's algorithm it will continue till it explore the entire graph. So, we assume that the graph is given in Dijkstra's algorithm, but in any case they explore the paths which are closest to the goal sorry closest to the start state.

So, after these 4 nodes have been explored perhaps this is the next node that would be explored and maybe, then this would be the next node and maybe this one and maybe this one

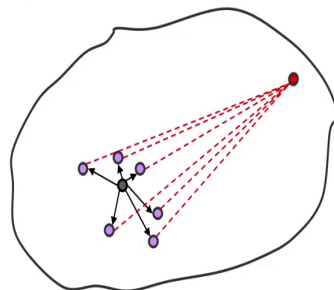
and this one and then this one and then this one and then this one and this one and so on. So, you can see that this search is going in a totally wrong direction it started with the start node and it is going towards the left as we see.

Whereas the goal writes on the lies on the right hand side of this particular graph essentially and the reason because this happens is because they always extend the cheapest path essentially.

So, they do not look towards the goal they only look behind at to what is the path that they have discovered to all the nodes on OPEN and then they picked that node on OPEN which has the cheapest partial paths found so far. So, they do not have a sense of direction both the algorithms will end up searching all these nodes before they will turn their attention towards the goal node essentially.

(Refer Slide Time: 09:39)

### Best First Search (recap)



A heuristic function estimates the distance to the goal.



This estimate,  $h(n)$ , is used to decide **which** node to pick from OPEN

### Best First Search

Candidates that appear to be *closest to the goal* are best  
Chooses the candidate with the *lowest h-value node* in the hope of finding a solution *sooner*.





So, which is what A star does not do and A star is good. So, A star adopts this goal directed behaviour from best first search which is an algorithm that we have also studied. So, if you remember best first search maintains for every node on OPEN a heuristic value which we call  $h$  of  $n$  and the heuristic function is a function which somehow estimates the distance to the goal or the cost of reaching the goal and it does this by simply looking at the given situation.

So, it does not explore it does not look ahead to find out a path to the goal it just says these are the different nodes and maybe by looking at the coordinates if you are finding paths in the spatial domain it will say that this node is closer to the goal.

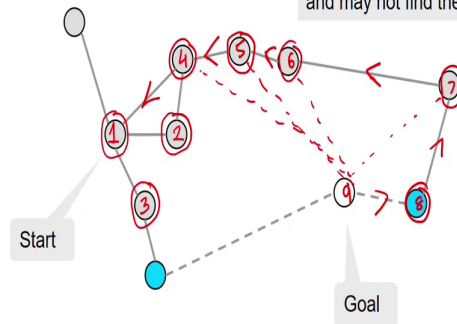
So, maybe that is the one that you should explore and the strategy behind best first searches that candidate said appear to be closest to the goal or the best. It does not care about how much expenditure or cost was made for reaching that particular node, it only looks at all the candidates and says this is the one which is closest to the goal and expands that. So, it chooses the candidate with the lowest heuristic value and in that way pushes the search hopefully towards the goal node provided that your heuristic function is a good function.

(Refer Slide Time: 11:12)

### Best First only looks ahead (recap)



Best First only looks at the distance to goal, and may not find the shortest path



Algorithm A\* combines the best features of all three algorithms!



Now, that this behavior that it only looks ahead it only looks forward towards a goal node it may often find a path which is not the optimal path. So, if you look at this particular small instance of a graph and remember that you are always looking at the node which appears to be closest to the goal.

So, if you were to start with the start node and this was the first node that we explored then it has 4 neighbours as you can see and out of those 4 neighbours one of them appears to be closest which is in this neighbour 2. So, it will explore that next.

So, let us circle the nodes to represent the fact that they have explored those nodes then we will look at the remaining neighbours. So, at this point there are 3 neighbours which are remaining children of both node 1 and 2. And let us say that this is a third node that we

explored and then we generate this one blue node which is just one step away from the goal node, but that that one step is a long step as determined by the heuristic function.

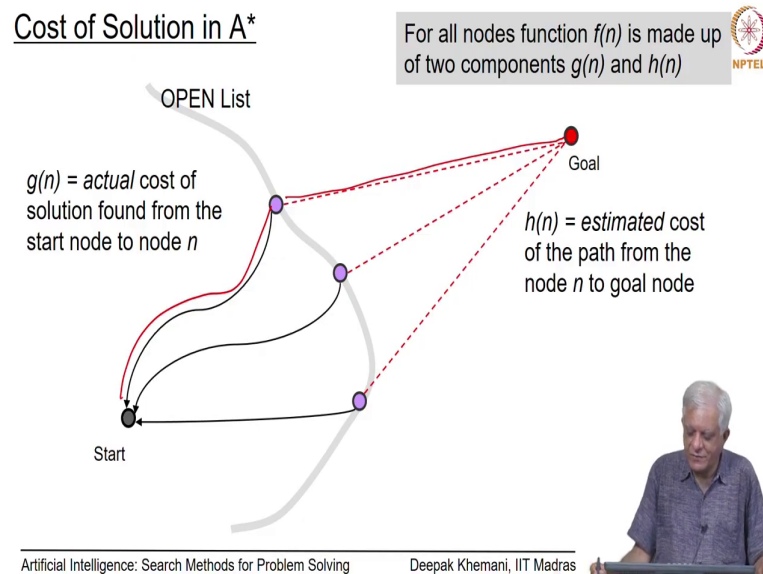
On the other hand this is likely to be the next node because this distance we imagine is shorter than the distance from the blue node which would have been the shortest path and once it generates this its neighbour which is the next node is even closer to the goal as you can see here. So, it explores this as a 5th node and this process continues and it finds shorter end nodes which are always closer to the goal than the path on the shortest path.

So, this is 8 and then it 9. So, the path it found if you remember the if you denote the back pointers would be this one. This is apparently not the shortest path and the reason why this happens for with A star with best first search is that it was only concerned by saying that which node appears to be closest to the goal node and that if you recall the motivation for doing that was to find the solution faster.

The motivation was not to find the optimal solution. So, motivation also find the solution faster and therefore, it made sense to say that the node which appears to be closer to the goal let us explore that node. Branch and bound and Dijkstra's algorithm on the other hand their motivation was to find the shortest path.

So, therefore, they always extended the shortest partial paths that they have found and now we are reaching this algorithm A star which combines the effect of all these three algorithms all the best features of all these 3 algorithms and that is why it is a seminal algorithm in this AI search domain.

(Refer Slide Time: 14:05)



What does A star do? Therefore, each node on OPEN you can see that this OPEN list there are 3 nodes coloured in purple and let us say these are the 3 candidates for each node it maintains 2 parameters; one in A star terminology is called  $g$  of  $n$ . And  $g$  of  $n$  is the actual cost of the solution found from the start node to the goal node. So, in this graph  $g$  of  $n$  is represented by these curved lines for example, this one and whatever is the length of this path is  $g$  of  $n$ .

The other feature that A star uses is  $h$  of  $n$  which is the estimated distance to the goal and A star essentially uses the combination of both these distance functions and uses a combined function which we call as  $f$  of  $n$  essentially.

(Refer Slide Time: 15:02)

### Algorithm A\*



Each candidate is tagged with an *estimated cost of the complete solution*  $f(n)$ ,

$$f(n) = g(n) + h(n)$$

where,

$g(n)$  is the *known* cost of path found from Start to node  $n$

- as used by the Branch & Bound algorithm

and  $h(n)$  is the *estimated* cost from node  $n$  to the goal.

- as used by the Best First Search algorithm

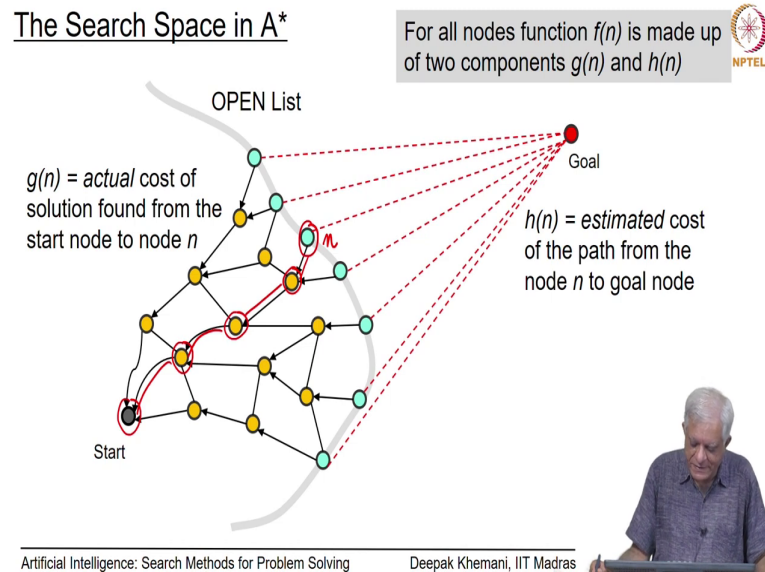


So, this is algorithm A star, each candidate is tagged with an estimate of its cost of the complete solution. So, in general if you remember this has been our strategy that you estimate the cost of all partial solutions and when you say estimate the cost we mean estimate the cost if that partial solution was to be completed, but of course, these are only estimates and then refine the best looking path that was a high level strategy that we are following.

In branch and bound the estimated cost was only  $g$  of  $n$ , in best first search the estimated cost was only  $h$  of  $n$ , but A star uses the combination  $f$  of  $n$  is equal to  $g$  of  $n$  plus  $h$  of  $n$  it looks behind as well it looks ahead.

So,  $g$  of  $n$  is a known cost of the path found from the start node to the goal to the node  $n$  and this is what was used to branch and bound and  $h$  of  $n$  is the estimated cost from the candidate node  $n$  to the goal node and this is what was used by best first search essentially.

(Refer Slide Time: 16:12)



So, at any point the graph that it is searching which is an adaptation of the figure that we just saw looks like this these nodes in orange are the close nodes that have been explored and the nodes in cyan here are the OPEN nodes that are the candidate nodes, the dashed red lines are the estimated distance from each of these nodes on OPEN to the goal node and back pointers from the nodes are OPEN are the best path found so far.

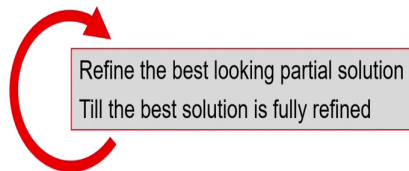
So, for example, if you are looking at this node then the best path found is from here to here and from here to here, here to here and here to here. So, sum of these edges would be the cost

of the path found to this particular node  $n$  if you say this is  $n$  and the dash edge would represent the estimated distance to the goal and A star uses are sum of these two estimates.

(Refer Slide Time: 17:10)

### Algorithm A\*

- Maintain a priority queue of nodes sorted on the  $f$ -values  
 $f(n) = g(n) + h(n)$
- Pick the lowest  $f$ -value node, and *check* whether it is the goal
- Else generate its neighbours, and compute their  $f$ -values
- Insert new nodes into the priority queue
- For existing nodes *check for better path* (like Dijkstra's algorithm)



So, just to recap and repeat we are looking at this overall strategy says that refine the best looking partial solution till the best solution is fully refined, both these steps are important essentially that you keep refining partial solutions till and you keep picking the best of all candidates. And you stop only when the best candidate that you pick up happens to be fully refined and that is going to be the algorithm A star as well.

So, we would maintain a priority queue of nodes sorted on their  $f$ -values where  $f$  as you remember is  $g$  of  $n$  plus  $h$  of  $n$  and from this priority queue we pick the lowest  $f$ -value node and check whether it is the goal. So, this is the second step in our high level algorithm, which says till the best solution is fully refined.

So, we are picking the best solution which in our case is the lowest  $f$  - value and checking whether it is the goal. If it is a goal we would terminate at this stage, but if it is not the goal, then we would generate its neighbours and compute their  $f$  - values in turn and add them to the set of candidates.

So, insert new nodes again into the priority queue, but as you will see in some examples as well that you may find better paths like Dijkstra's algorithm did 2 nodes which already exists in the priority queue. And we may just need to check for a better path and update that like the Dijkstra's algorithm did as well.

(Refer Slide Time: 18:44)

```

A*(S)
1  default value of g for every node  $s + \infty$ 
2  parent(S)  $\leftarrow$  null
3  g(S)  $\leftarrow$  0
4  f(S)  $\leftarrow$  g(S) + h(S)
5  OPEN  $\leftarrow$  S : []
6  CLOSED  $\leftarrow$  empty list
7  while OPEN is not empty
8    N  $\leftarrow$  remove node with lowest f value from OPEN
9    add N to CLOSED
10   if GOALTEST(N) = TRUE
11     return RECONSTRUCTPATH(N)
12   for each neighbour M  $\in$  MOVEGEN(N)
13     if  $g(N) + k(N, M) < g(M)$ 
14       parent(M)  $\leftarrow$  N
15        $g(M) \leftarrow g(N) + k(N, M)$ 
16        $f(M) \leftarrow g(M) + h(M)$ 
17       if M  $\in$  OPEN then continue
18       if M  $\in$  CLOSED then PROPAGATE-IMPROVEMENT(M)
19       else add M to OPEN  $\triangleright$  M is new
20  return empty list

```

**Algorithm A\***  
NPTEL

This variation courtesy  
S Baskaran

In the style of Dijkstra's algorithm


Priority queue

Better path found?

Case 2: Neighbour in OPEN

Case 3: Neighbour in CLOSED

Case 1: Neighbour is new node



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

So, here is a version of the A star algorithm which is a being devised by my colleague S Baskaran who is been helping me with this course all along ah. It is a very neat combination of Dijkstra's algorithm and best first algorithm and like Dijkstra's algorithm we initialize



every node with a  $g$  - value of infinity and this means that whenever you find a new path to that node you will always have found a better path.

So, we have a uniform treatment as we will see as we go down this algorithm. So, we initialize everything to infinity like Dijkstra's algorithm does we say that the start node  $S$  has no parent its parent is null. We say that the start node  $S$  has a cost  $0$   $g$  of  $S$  is  $0$ , we compute the  $f$  - value of the start node which is  $g$  of  $S$  plus  $h$  of  $S$  which will be  $h$  of  $S$  because  $g$  of  $S$  is  $0$ , then we put  $S$  into the priority queue and we maintain that as sorted list let us say.

So, if you remember your list operations then essentially what this is saying is that make  $S$  into a list or put this node  $S$  into an empty list. So, it becomes a list containing only  $S$  at this moment and then you add you create a CLOSED list and CLOSED list will be as before nodes that we have visited and the algorithm is similar to what we did for best first and best first search for that matter or breadth first search that until OPEN is not empty you remove the node from the head of the OPEN.

Remember that we are maintaining OPEN as a priority queue and this in this algorithm has been expressed as remove the node with the lowest  $f$  - value because in a priority queue the node with the lowest  $f$  - value would come to the head of the queue.

So, we remove it from the priority queue and add it to close then we check whether it is a goal node, if it is a goal node, then we go back and reconstruct the path and that as you can imagine is done by tracing back the pointers from that parent pointers from the node till you reach the start node.

If it is not the goal node then we see whether we have found a better path to each neighbour  $M$  of this node  $N$ . So, we call MOVEGEN with the node that we just inspected and for every neighbour  $M$  of this name node  $N$  we found we find we check whether we have found a better path.

Now, given the fact that we have initialized all  $g$  - values to infinity, every time we find a new node we would have found a definitely a better path and then we would update the  $g$  - value of that node to the  $g$  - value of the parent plus the edge cost from  $N$  to  $M$ .

And we will also adjust and say that parent of  $M$  is the node  $N$  may be just came from and compute the  $f$  - value by adding up the  $g$  - value and the  $f$  - value remember that the  $f$  - value is just a property of that node and it is a static function which looks at the node and tells you what how far it is expected to be the distance from the goal.

Now, just a small point here that is traditionally all literature uses simply  $h$  of  $M$  or  $h$  of  $N$  as the heuristic value of a node, but you must keep in mind that it is really a function of the given node  $M$  and the goal node  $g$ , but for some reason we do not use the fact that  $g$  is a goal node in the heuristic depiction and that partly because you know there may be many different nodes which are goal nodes and the goal is often specified by some predicate.

So, for example, if you have decided that you want to go to city and eat in let us say in Chennai Saravana Bhavan then there are more than one restaurants which are in this chain and any one of them could have been a goal node and essentially your A star algorithm will try to find your path which is closest to the one restaurant in this chain.

And by closest we would mean least cost and least costs could include anything, it could include the length of the road or it could include the traffic on the road and you could have all kinds of cost functions and then try to optimize on that.

So, anyway so for every node  $M$  of which is a neighbour of  $N$  we estimate we update the cost the  $g$  - value if it has been if a better path has been found and there are three cases which happen. The simplest case is when  $M$  is a new node if  $M$  is a new node we just simply add it to OPEN and carry on if  $M$  the case 2 is when  $M$  is already on OPEN.

Now, if it is already on OPEN and if we have found a better path remember we are doing this check that  $g$  of  $N$  plus  $k$  of  $N$ ,  $M$  which is a new path is it better than the older path that we had found to that node on OPEN.

For nodes which are new  $g$  of  $M$  old was infinity, but for nodes which are OPEN you already have some path well the question is have you found a better path. If you have found a better path, then in this version of algorithm given to me by Baskaran you have already updated the better path and updated the parent pointer and everything. So, you just continue in case 2 and you do not have to do anything else.

Case 3 is when the neighbour is on CLOSED which means you have already inspected it and you have already generated its children because the node is on CLOSED only when it has been expanded and if it has been expanded it would have its own children or own neighbours and then in case 3 we need to propagate the improvement if we have found, remember that all this is happening in the case when this if condition is true.

(Refer Slide Time: 25:28)

### Algorithm A\*: Case 3 – if better path to node on CLOSED found



This variation courtesy  
S Baskaran

```
PROPAGATE-IMPROVEMENT(M) CLOSED
1 for each neighbour X ∈ MOVEGEN(M)
2   if (g(M) + k(M, X)) < g(X) Better path found to X?
3     parent(X) ← M Update
4     g(X) ← g(M) + k(M, X)
5     f(X) ← g(X) + h(X)
6     if X ∈ CLOSED Recurse
7       PROPAGATE-IMPROVEMENT(X)
```



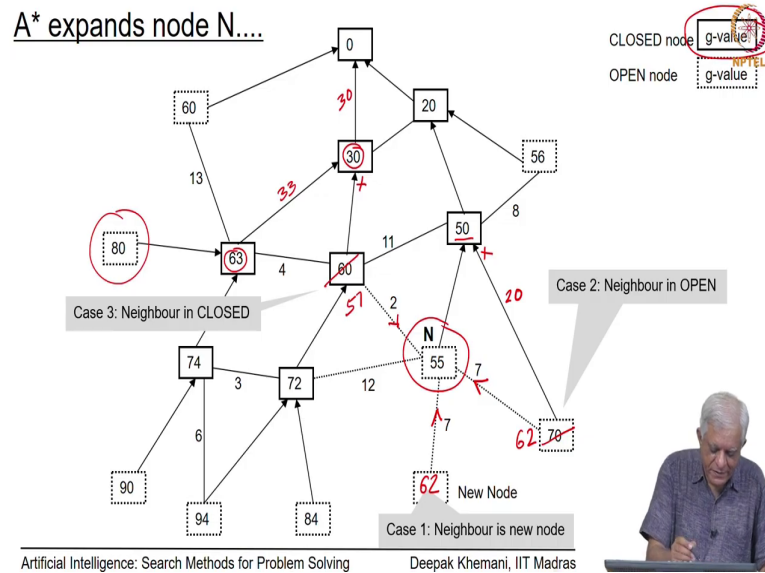
So, what is this propagation doing M is a node that which is on CLOSED remember this is on CLOSED and we have found a better path to this and now we want to propagate the improvement in the cost.

So, we look at every neighbour X of M and we asked the same question again have you found a better path to X if we have found a better path to X we update that the parent of X is M, remember that X was a child of M, M was already in CLOSED and X may have been a child of some other node, but now we have found a path to X through M.

And this happens to be a better path and therefore, we say parent of X is M which means that you will reach X via M we update the g - value as well and then if X also happens to be in CLOSED then we propagate this improvement recursively again essentially. So, that is algorithm A star what we have here is a slightly neater version then that was originally

postponed in which the three cases were dealt with explicitly, but the algorithm is still the same.

(Refer Slide Time: 26:42)



So, to understand this whole process let us look at a small instance and if this is a graph that has been generated by A star so far and the nodes that you see in bold rectangles these are the nodes which are already on CLOSED nodes and as mentioned here they are labelled with their g - values which is the path found till that node.

So, for example, this node is labelled with a value of 30 and we are not labelled that particular edge, but you can infer from this that this edge must have been 30 because that is the cost of reaching that node and likewise you can infer that because this value is 63. So, this would be 30 plus something and that has to be 33. So, this edge would have been 33. So, we have not label everything, but all the information that you need is here.

So, this shows the CLOSED nodes and the dashed rectangles like this one for example, they represent OPEN nodes essentially and in particular one OPEN node that is about to be expanded is this node which you we will call N and remember that OPEN nodes are picked based on f - values.

We are not depicted what the h - values here are we assume that there is some heuristic function which determines that, but based on that we have said decided that N is what we are going to expand next. And what we want to illustrate here is that when you expand N what are the changes that happen in the graph and the changes that happened are only in g - values and parent pointers the h - values are anyway a static function of every node essentially.

So, here is this node we are about to expand A star is about to expand and you can see that it has got five neighbours and one of them is the parent from where it came. So, there of course, we do not be doing much, but after other four neighbours there are these 3 cases that we talked about case 1 is when it is a new node it was not there on the graph at all.

So, we simply compute its g – value, what will its be g - value would be 55 plus 7 which is 62 and we would put a parent pointer to this and say we have added to the to the graph that is case 1. Case 2 is when the node was already on OPEN so, this node as shown here was an OPEN it came as a child of the node which was a parent of N and it had a cost of 70. So, presumably the edge here was 20 so because 50 plus 20 would be 70.

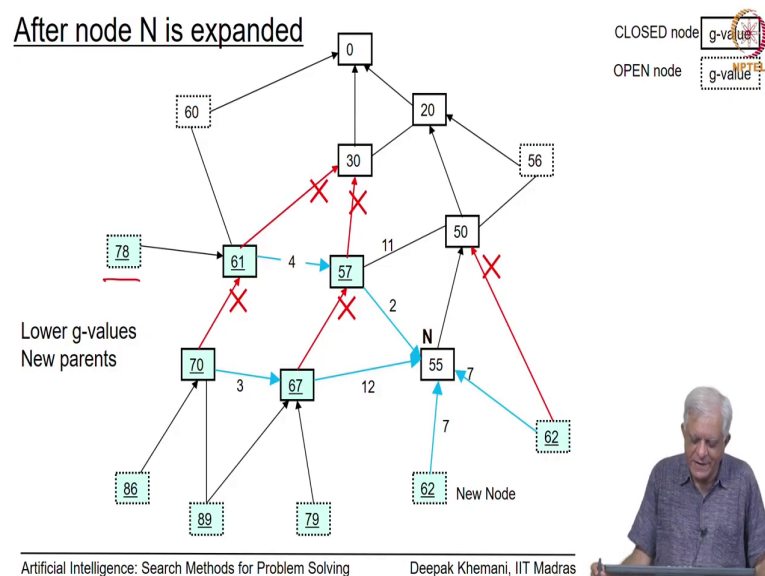
But now after we have expanded this node N whose cost was 55 and the edge connecting them has been discovered when N is expanded and its cost is 7. So, we can see that this new cost of this node we should revise.

So, instead of 70 this should become 62. And we should remove this parent pointer from here and instead put a parent pointer here to see that the best cost of reaching that particular node is 62 and the path comes through the node N which is the node that we just expanded.

Now, because this is an OPEN we do not have to do anything else and that is what the algorithm said that we just continue, but in case 3 when the node is on CLOSED, then also we have to make the change. So, this node had a cost of 60 and now from 55 we have an edge of length 2.

So, this will become 57 55 plus 257 and we would again want to remove this parent pointer from here and make it point 2 the current node and because this is on CLOSED it already had children it already had neighbours and we need to explore whether we have found better paths to that those neighbours and keep propagating as long as we are finding better paths so now, then CLOSED.

(Refer Slide Time: 30:54)



So, by the end of this process the graph now looks like this the you can see that that 5 parent pointers have been removed and reassigned either to the node N or to some new child of N and many g - values of this of this graph node nodes in this graph have been updated.

And what we are showing here are the updated g - value. So, if you go back to this you can compare for example, that this value of 63 or this value of 80 at the has reduced to 7 78 and that is because there has been a reduction in the cost of 61 to 59 and so on or from 63 to 61 and so on.

So, this is a key factor of algorithm that it with which it has borrowed from Dijkstra's algorithm to some extent because Dijkstra's algorithm basically maintains the best partial paths found. So, far and A star algorithm also does the same thing the only difference between Dijkstra's algorithm and A star algorithm is that of all the nodes on OPEN.

For example here it will pick the node with the lowest f - value, we do not know which one is the lowest f - value here because we have not specified the heuristic function, but if there was a heuristic function then that particular node would be picked and hopefully the search will move more towards the goal node ok.



(Refer Slide Time: 32:27)



Next

An illustration of how A\* searches the implicit graph,  
guided by the f-values,  
generating the nodes on the fly,  
till it picks a goal node.



So, I hope this gives you a clear understanding of this algorithm A star ah, what we will do next is to look at an illustration of how A star searches an implicit graph I keep repeating that the graph is not given to the algorithm what is given is the start node. The goal conditions are the goal node as the case may be and move gen function which implicitly represents the neighbours of every node in the search space and therefore, generates the graph as and when needed.

So, it generates it on the fly the algorithm A star is guided by f - values and it continues doing that till it picks the goal node. So, in the next session we will look at the detail example and hopefully that will make the algorithm A star behaviour very unambiguously clear hm. So, we will do that in the next session.

