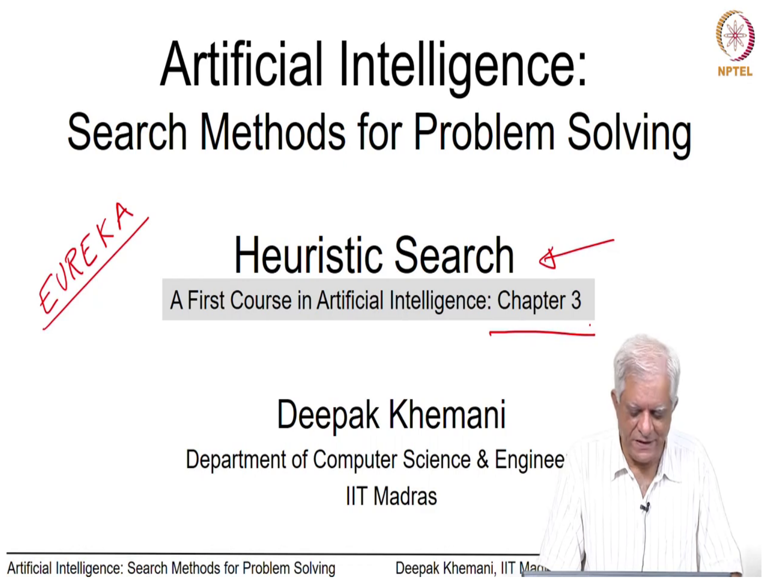
Artificial Intelligence: Search Methods for Problem Solving

Prof. Deepak Khemani
Department of Computer Science & Engineering
Indian Institute of Technology, Madras

Chapter - 03
A First Course in Artificial Intelligence
Lecture – 23
Heuristic Search

(Refer Slide Time: 00:11)



Artificial Intelligence:
Search Methods for Problem Solving

EUREKA

Heuristic Search

A First Course in Artificial Intelligence: Chapter 3

Deepak Khemani
Department of Computer Science & Enginee
IIT Madras

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

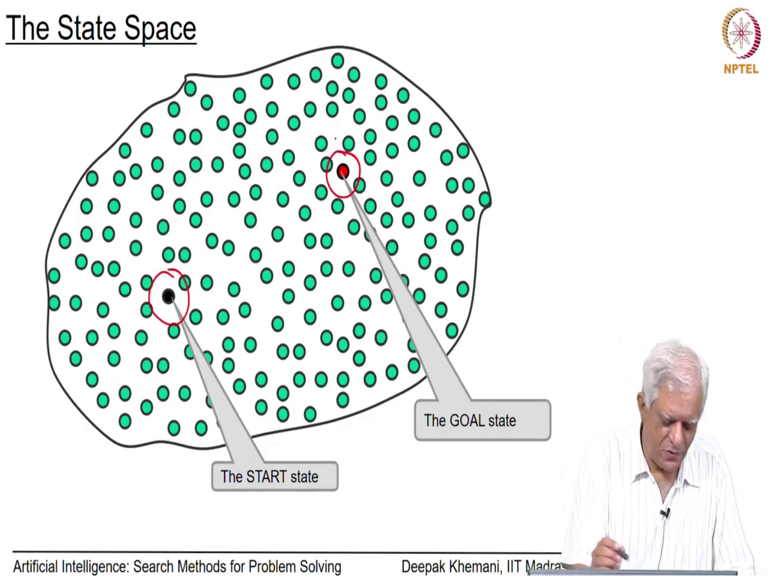
So, welcome back, we are just finished looking at blind search algorithms; we looked at depth first search, breadth first search and DFID. And we saw that, you know they always explore the search space in a systematic fashion; always doing the same thing irrespective of where the goal node is.

And the trouble with that these algorithms is that; because the search space is growing exponentially, the time that they require to run also grows exponentially in on the average. And therefore, we need to look for alternative methods to explore the space; and maybe with a sense of direction which is what the heuristic function gives us, and hopefully at least for some problems we able to arrive at the solution much faster.

So, this next set of videos that we will see, our lectures that we will see are on heuristic search; the word heuristic comes from the Greek word or they have the same source the word Eureka and there was a word called heurisca. You might remember the word eureka in the context of Archimedes when he ran out of his bath, naked on the streets saying Eureka Eureka; and that means, he had solved problem of how much is a loss of a body which is floating under water, the loss in the weight or the decrease in the weight is equal to the weight of the water that is displaced.

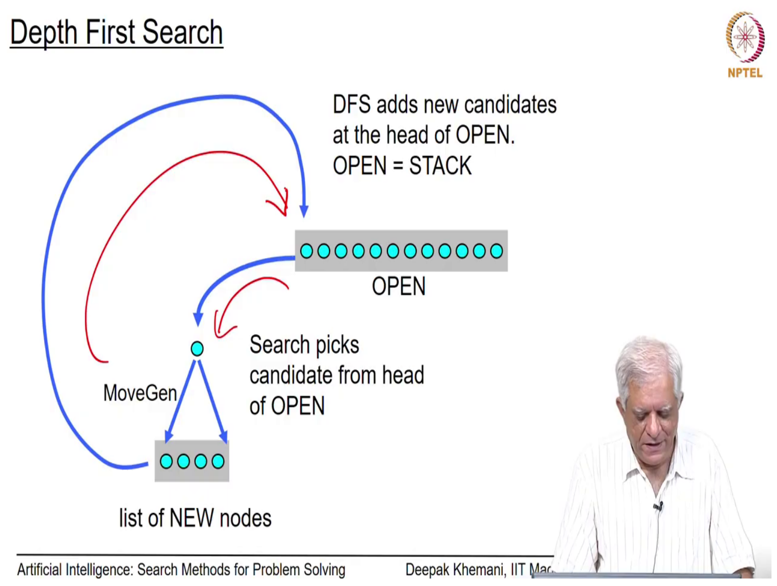
So, once he realized that he ran saying Eureka; and Eureka basically meant, I know, I know. And the word heuristic also comes from the same source which means that, in some sense the algorithm knows where it should go. If you are following my book, you will find all the material on chapter 3 essentially.

(Refer Slide Time: 02:19)


The State Space
NPTEL
The GOAL state
The START state
Artificial Intelligence: Search Methods for Problem Solving
Deepak Khemani, IIT Madras

So, let us just do a quick recap; we have a state space, in which we have a start node and a goal node.

(Refer Slide Time: 02:27)



Depth First Search

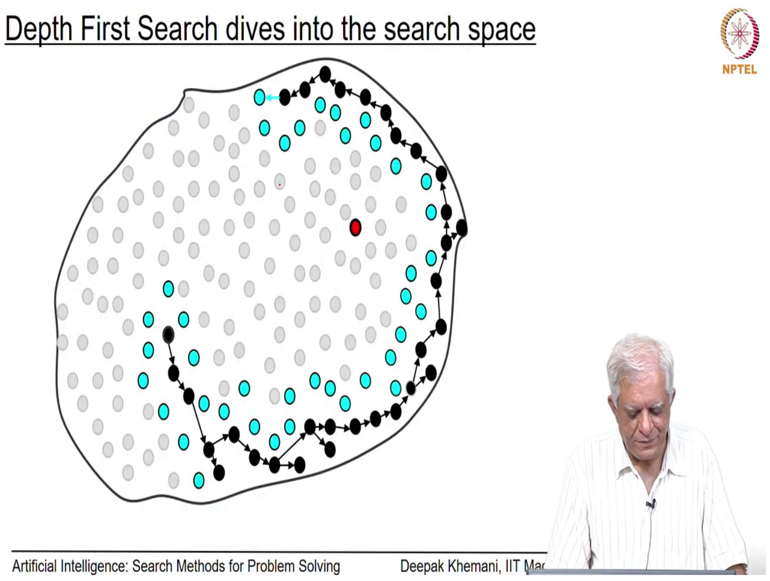
DFS adds new candidates at the head of OPEN. OPEN = STACK

Search picks candidate from head of OPEN

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

And we have studied depth first search, where we take out the nodes from the head of the list and then we add the new nodes back at the head of the list.

(Refer Slide Time: 02:39)

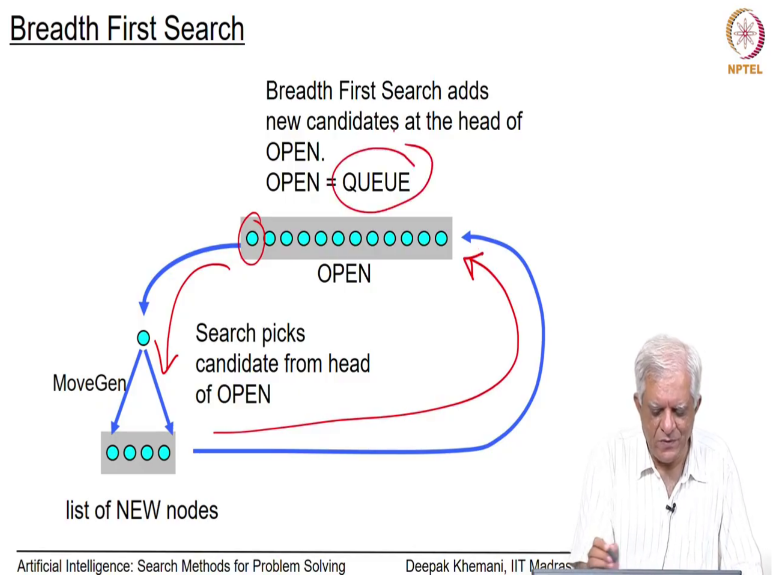


Depth First Search dives into the search space

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

Which results in this behavior of depth first search that we have seen that, it dives into the search space; it has a very impetuous nature, just goes off without you know looking here and there.

(Refer Slide Time: 02:52)



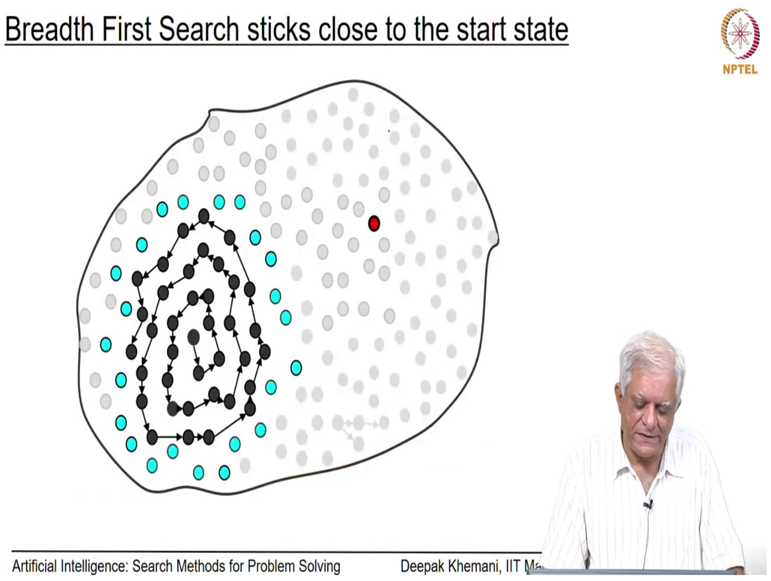
Breadth First Search

Breadth First Search adds new candidates at the head of OPEN.
OPEN ≠ QUEUE

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

The other algorithm that we saw was that, we again take out nodes from the head of the open list; but this time we add them to the tail of the list and therefore, we treat open as a queue.

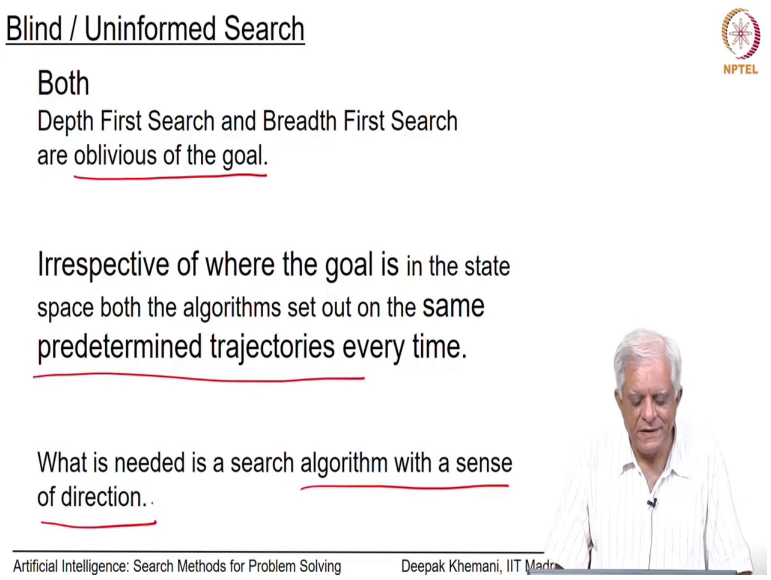
(Refer Slide Time: 03:05)



Breadth First Search sticks close to the start state

And as a consequence, the behavior of the search algorithm is startly different; it tries to stay as close to the source node as possible and you can think of it as a conservative search algorithm. And as a result of its conservative behavior, it also gives us the shortest path; which means the least number of moves in the solution.

(Refer Slide Time: 03:26)



Blind / Uninformed Search

Both
Depth First Search and Breadth First Search
are oblivious of the goal.

Irrespective of where the goal is in the state space both the algorithms set out on the same predetermined trajectories every time.

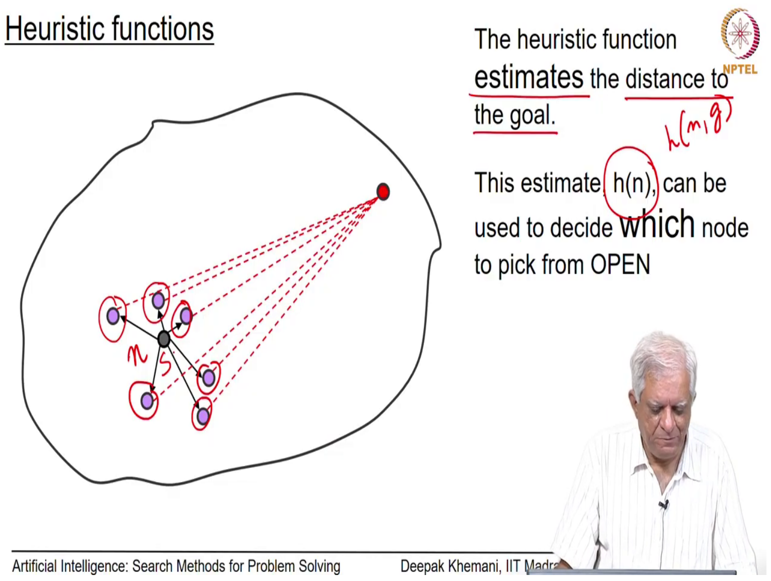
What is needed is a search algorithm with a sense of direction.

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

Now, both of them are oblivious of the goal, they are just not aware of where the goal is; ideally the goal, they should try to move towards the goal. So, irrespective of where the goal is, they both the algorithms follow the same predetermined trajectories every time essentially.

Now obviously, that is not intelligent behavior, but it could still form the basis or a foundational layer of a higher level algorithm which uses them; but even for this basis of foundational algorithm, we can try and do better. So, what we need is a algorithm with a sense of direction, and that is what we will start looking at in this sequence of lectures, ok.

(Refer Slide Time: 04:13)



Heuristic functions

The heuristic function estimates the distance to the goal.

$h(n, g)$

This estimate $\boxed{h(n)}$, can be used to decide which node to pick from OPEN

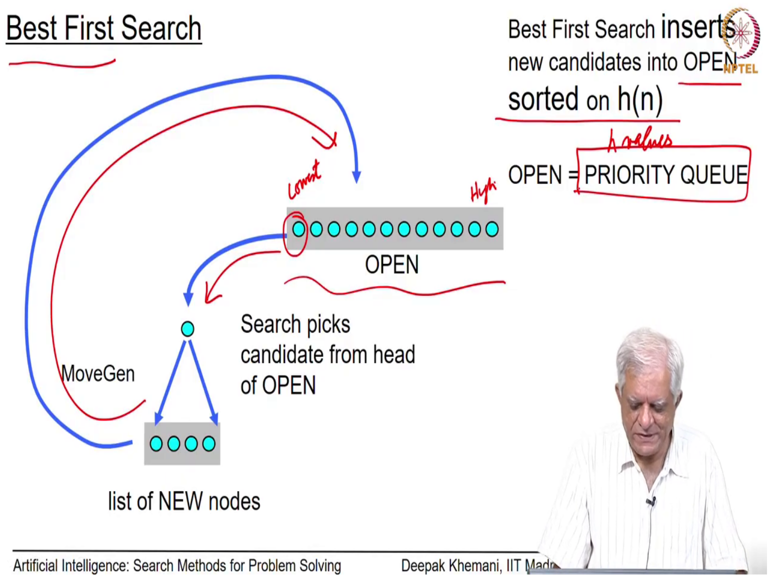
Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Madras

So, we have a notion of a heuristic function. The heuristic function which is a function of a node n; in this case this is the node n, let me draw that again n. Traditionally we write it only as a function of the node n, though in reality it is a function of the node n and node g or the goal node; but since very often you may not specify a goal node as we saw in the water jug example little bit earlier, you may specify the conditions for the goal to be true which may be true in a set of nodes.

So, we do not tend to express that goal part here and it is kind of implicit that what the heuristic function does is that; it estimates the distance to the goal essentially. For sake of simplicity in this diagram we have drawn one goal, but if there were many goals; then for each node that we are interested in, we would choose a distance to the nearest goal essentially.

So, as n varies over these different nodes, we can which are the successors of that start node s; we can compute the heuristic value, and then based on the heuristic value we decide which node to open. Unlike depth first search and breadth first search which always chose the nodes in the predetermined fashion; heuristic search will choose the nodes based on the estimated distance to the goal, and that is why it is called a heuristic search algorithm.

(Refer Slide Time: 06:01)



Best First Search
Best First Search inserts new candidates into OPEN sorted on h(n)
h values
OPEN = PRIORITY QUEUE
Lowest
High
OPEN
Search picks candidate from head of OPEN
MoveGen
list of NEW nodes
Artificial Intelligence: Search Methods for Problem Solving
Deepak Khemani, IIT Madr

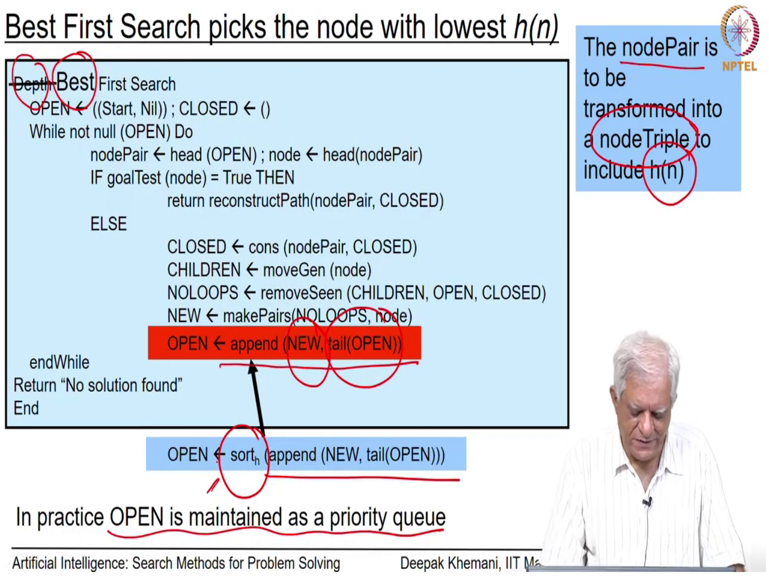
The algorithm that we are talking about will be called best first; in the sense that it always chooses the best node first. Now essentially you can imagine that you have a set of candidates which are stored in open list, and for every candidate we know what is the estimated distance to the goal; and we want to choose the one which has the lowest estimated distance to the goal.

The way to implement this which is consistent with what we have done earlier is that, we will still extract the node from the head of the open if it is a list like this; but we will keep this open sorted on the heuristic values. So, sometimes these are called as h values.

So, this open list will be shortest and the on the h values; so the lowest will be here, and the highest will be here essentially. So, when we pick the node from head, it will always pick the one with the lowest h value; and when we had done generating the new nodes, we will inserts them at an appropriate place essentially.

In practice of course, we do not sort the list; in practice we would maintain open as a priority queue, which is just an efficient way of making sure that you always extract the node with the lowest heuristic value essentially.

(Refer Slide Time: 07:36)



Best First Search picks the node with lowest *h(n)*

```
~~Depth~~ Best First Search
  OPEN ← ((Start, Nil)) ; CLOSED ← ()
  While not null (OPEN) Do
        nodePair ← head (OPEN) ; node ← head(nodePair)
        IF goalTest (node) = True THEN
              return reconstructPath(nodePair, CLOSED)
        ELSE
              CLOSED ← cons (nodePair, CLOSED)
              CHILDREN ← moveGen (node)
              NOLOOPS ← removeSeen (CHILDREN, OPEN, CLOSED)
              NEW ← makePairs(NOLOOPS, node)
              OPEN ← append (NEW, tail(OPEN))
  endWhile
Return "No solution found"
End
```

OPEN ← $sort_h$ (append (NEW, tail(OPEN)))

The nodePair is to be transformed into a nodeTriple to include h(n)

In practice OPEN is maintained as a priority queue

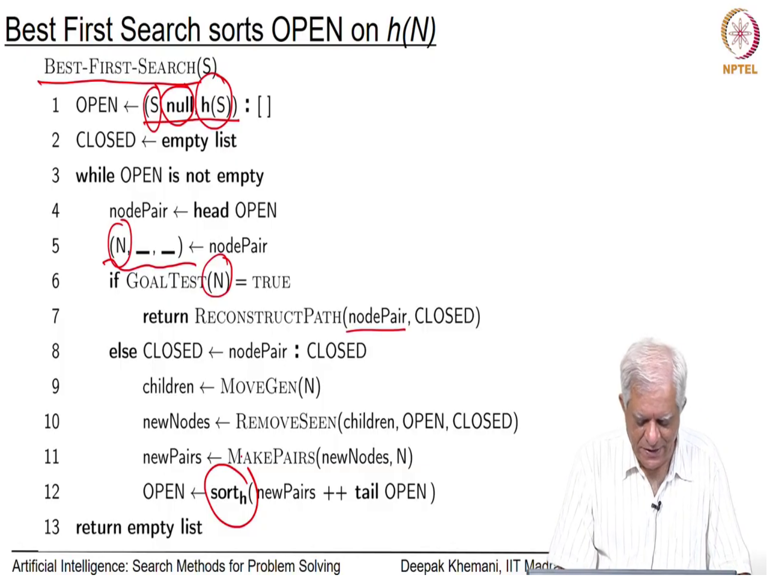
Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Ma

So, here is a variation of our algorithm that we had written for depth first search. So, we are not doing depth first search, but we are doing best first search. And the only difference that we are making here or only difference that we are doing here is that; instead of saying that you are going to append the new nodes to the tail of open, we are saying we will do that, but after that we will sort the nodes on heuristic value.

Of course, a little bit of extra change that we also need to do is that; instead of the node pair, we will have to make it a node triple and include the heuristic value as part of the this thing, as part of the node representation.

So, in the node pair we have a current node and the parent node; in the node triple we will have a current node, the parent node and the heuristic value for the current node essentially. So, as I said little while ago, in practice actually open is maintained as a priority queue; but we can conceptually think of it as being sorted essentially, which is doing the same thing that always giving to us, so node with the lowest heuristic value.

(Refer Slide Time: 08:52)



Best First Search sorts OPEN on $h(N)$

```
BEST-FIRST-SEARCH(S)
1  OPEN ← (S, null, h(S)) : [ ]
2  CLOSED ← empty list
3  while OPEN is not empty
4      nodePair ← head OPEN
5      (N, _, _) ← nodePair
6      if GOALTEST(N) = TRUE
7          return RECONSTRUCTPATH(nodePair, CLOSED)
8      else CLOSED ← nodePair : CLOSED
9          children ← MOVEGEN(N)
10         newNodes ← REMOVESEEN(children, OPEN, CLOSED)
11         newPairs ← MAKEPAIRS(newNodes, N)
12         OPEN ← sort_h(newPairs ++ tail OPEN)
13 return empty list
```

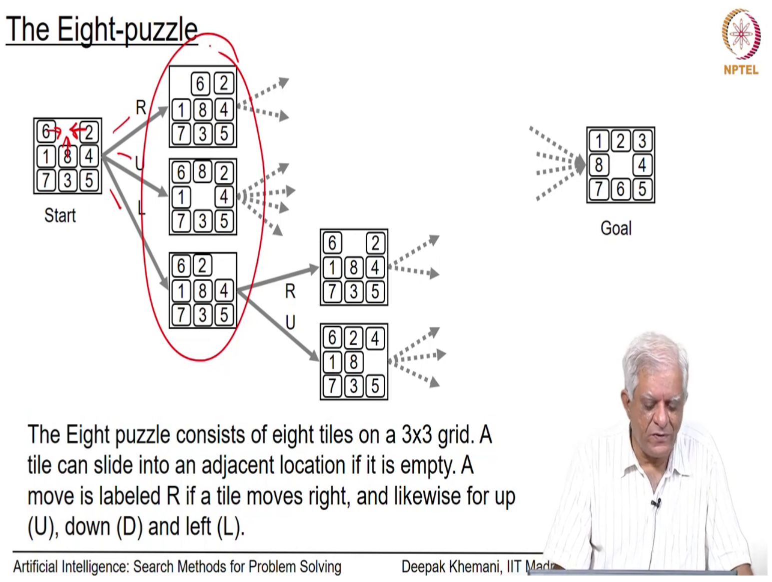
Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madr

If you were to do this, then we have our modified algorithm best first search here; and observe that we have a triple here, where we have a start node, it has a parent which is null and the heuristic value of the start node which is the estimated distance to the goal essentially.

And at every point we extract a trickle triple; and the first node n is what we are interested in, we apply the goal test to that n. And again we have called this node, node pair here, this is simply for the sake of reusing the earlier code; but you could have called it node triple, after all there is nothing in the name essentially.

So, everything else remains the same; there except that we have changed the representation to include the heuristic value and conceptually at least we are sorting this open list. We say I will again repeat to say that, in practice you would want to do it as a priority queue which is implemented as a heap.

(Refer Slide Time: 09:53)



The Eight-puzzle

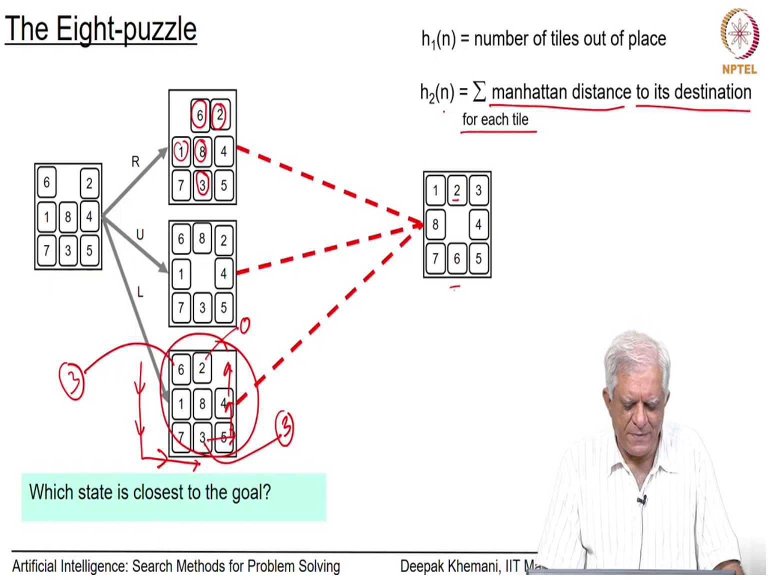
The Eight puzzle consists of eight tiles on a 3x3 grid. A tile can slide into an adjacent location if it is empty. A move is labeled R if a tile moves right, and likewise for up (U), down (D) and left (L).

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

Now let us look at this problem that we have seen earlier; the eight puzzle problem, in which you have to move around tile. So, either you can move this tile here or you can move this tile here or you can move this tile here; resulting in these three moves that we can see here. And then once having made a move, you can choose; you have to choose again what is the next move you want to make and so on.

So, the thing that you want to focus on is, how do we decide that for example, between the three of these nodes; what is the node that we should pick? How do you estimate the distance to the goal essentially?

(Refer Slide Time: 10:35)


The Eight-puzzle
h1(n) = number of tiles out of place
h2(n) = ∑ manhattan distance to its destination for each tile
R
U
L
Which state is closest to the goal?
Artificial Intelligence: Search Methods for Problem Solving
Deepak Khemani, IIT Ma

So, we can think of a couple of heuristic functions; one function which simply says that, count the number of tiles which are out of place. So, if you look at this for example, the first successor here; we can see that 6 is out of place, because it should have been here, 2 is out of place because it should have been here. 1 is out of place, 8 is out of place; but 4 is in place, 7 is in place and 3 is out of place.

So, you can see that five tiles are out of place, three tiles are in place. So, we can think of this as a distance of 5, which says intuitively that you have to change the position of 5 this thing essentially.

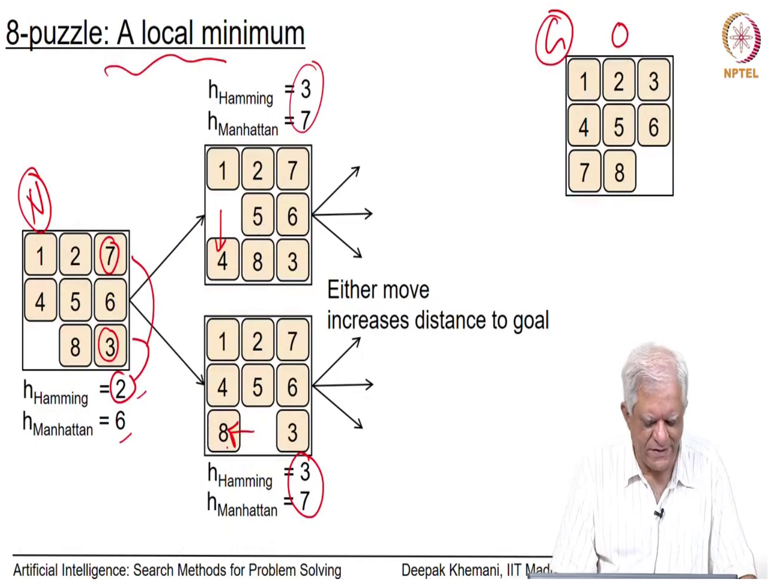
The other measure that we can use is the notion which is close to manhattan distance and what you do there is; that you found the distance to every tiles destination for each tile essentially.

So, if I look at this third example here; then you can see that for 6 to go to it is destination, it has to make one move here, a second move here and then the third move here.

So, the distance of 6 is 3, the distance of 2 is 0; because it is in place. Let us distance of 5 is 0, 4 is 0 and if you look at 3; then the distance of 3 is also 3, because it has to come here, then it has to go here, then it has to go here this 3. So, in this manner we compute the distances for each tile and the sum of the distances will give us an estimate of distance to the goal.

Now, both of us in some sense try to tell us, both of the; both of the heuristic functions try to tell us which node is closer to the goal; the first one is kind of more coarse, because it simply says how many tiles you have to get into place. The second one is a little bit more refined; because it tells you how much each tile will have to move to reach the goal node essentially.

(Refer Slide Time: 12:58)



8-puzzle: A local minimum

The trouble with heuristic functions is that, it is very difficult to define heuristic functions which are accurate. So, here is let us look at a another example, where this is the node n that we are looking at; and you can see that the first heuristic function which is also called a hamming heuristic function, because it tells you how many things are out of place. So, in this case we can see that there are two things out of place; 1 is in place, 2 is in place, 7 is out of place, 4, 5, 6 are in place, 8 is in place and 3 is out of place. So, that gives us this count of 2 essentially.

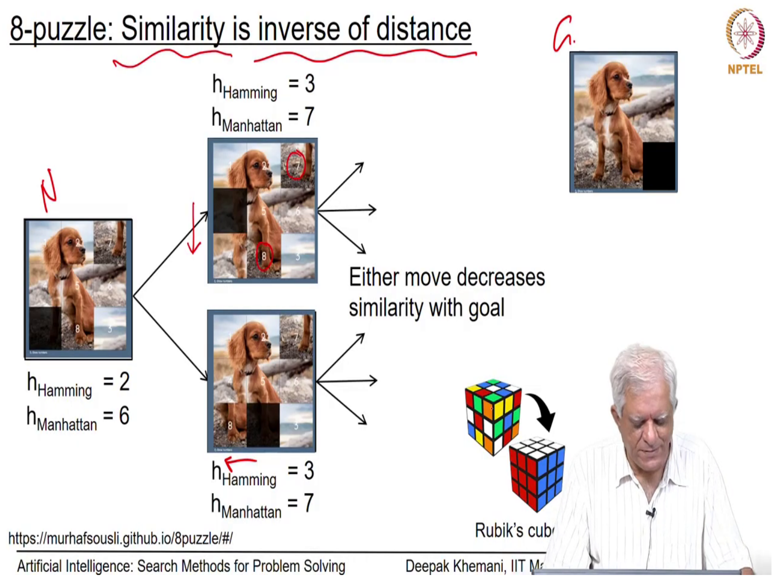
If you have counted the distances, then this distance would be 6 for this node. Now if you look at the two successors that this node has; you can move 4 down that is what you have done here and we get this thing. And now 4 was in place earlier and now it has gone out of place; so all these values have gone up by 1 here essentially. Alternatively you could have moved 8 here in the second move, and again 8 was in position and it is gone out of place, so the heuristic values are gone up.

So, this kind of illustrates of act that heuristic functions are not always accurate. Now clearly if you want to go from this node n to the goal node g, you will have to go through one of these through children nodes. But the heuristic function is telling you that both these children on both the heuristic functions are worse than the current node, and that is a phenomenon of local minima in this case, we will explore this a little bit more.

That is a problem with heuristic functions that, you often end up with local minima or maxima, where essentially what you want to reach is a value of 0 and you are starting with a value of 2 in the case of first heuristic function or 6 in the case of second heuristic function; but in both the moves the heuristic value is going up essentially, now which does not look very encouraging, but it has to go through one of those nodes.

Very often if you have played this puzzle as a child, you would have seen some pictures which are trying to, which are placed instead of these numbers. And in from that perspective we can think of this as similarity that, you try to move to that successor which is more similar to the goal node, ok.

(Refer Slide Time: 15:42)



8-puzzle: Similarity is inverse of distance

G.

N

$h_{Hamming} = 3$
$h_{Manhattan} = 7$

$h_{Hamming} = 2$
$h_{Manhattan} = 6$

$h_{Hamming} = 3$
$h_{Manhattan} = 7$

Either move decreases similarity with goal

Rubik's cub

https://murhafsousli.github.io/8puzzle/#/

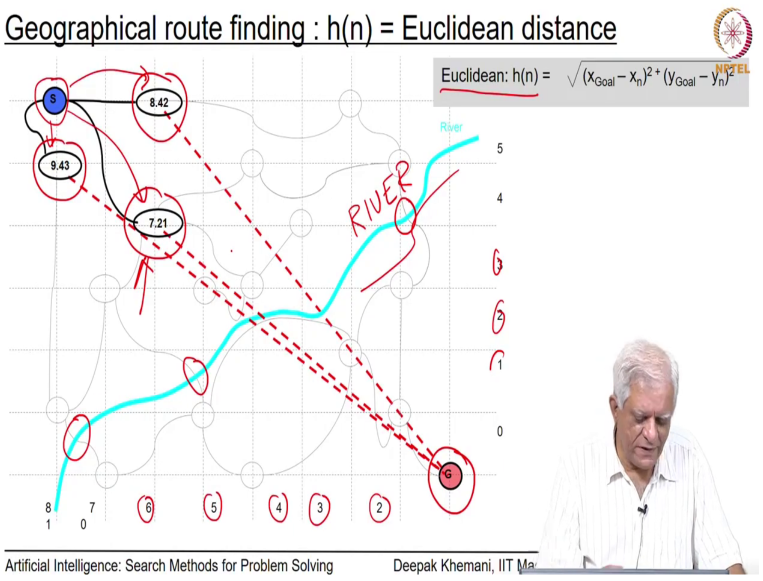
Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Ma

So, in this example either move increases the distance to the goal; but if you are working with a puzzle which had pictures on it, then you can see that the node n is more similar to the goal node g, then either of the two successors. And you can think of similarity as the inverse of the distance; because the closer you are the smaller is the distance, the more is the similarity. So, that is another way of looking at it.

And this also illustrates the fact that, if you make either move; that if you move this tile here or if you move this tile here, which is what we did in the same, the same because it is the same puzzle actually with the figures of the with the picture of the dog shown in, the location is still the same. If you were to kind of look at this slide carefully; you would see the numbers that we were talking about 7 is here, 8 is here and so on.

It is a just that we have pictures and therefore, we can think of similarity essentially. We can see a similar phenomenon for Rubik's cube which some of you must have solved is that; very often you have to move from a state which is fairly similar to the goal state, but where any move makes you the little bit less similar. So, this problem we will look at again later.

(Refer Slide Time: 17:05)



Geographical route finding : h(n) = Euclidean distance

Euclidean: h(n) = $\sqrt{(x_{Goal} - x_n)^2 + (y_{Goal} - y_n)^2}$

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

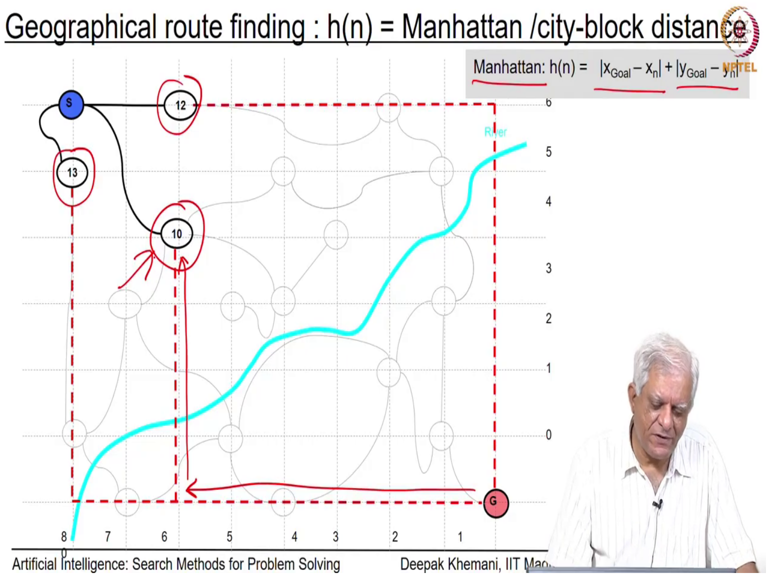
Here is another problem, supposing you want to do geographical route finding; for example, Google maps does it for you nowadays. And the problem here is that, you are at the start location here and from this start location, you can either go to this successor or this child or this one or this one, and you have to choose which of those three is the right move to make, if you want to reach this goal node here. So, this dash red line that you see, these are Euclidean distances; because the map is on a grid and you can see that.

Let us say this is in kilometers; so we are 2, 3, 4, 5 kilometers away and likewise in this direction 1, 2, 3 kilometers away. So, one way of estimating the distance to the goal is to measure the Euclidean distance which is the distance as the crow flies, which you know is the formula which you can compute by taking the differences adding and taking the squares of the differences, adding them and taking the square root.

And this red lines that you see here, essentially are the distances as the crow flies; and the numbers you see inside this are the actual computed values, so 8.42, 9.43 and 7.21. So, clearly this is the best move to make for you, and you can see that this also gives you a sense of direction.

The problem is, the heuristic function is only looking at difference in coordinates. So, it is not aware for example, that there might be a river running across the path and the only bridges which happened are here and here and here; but you know it will try and drive you closer in the direction of the goal and we will see a real world example in a moment.

(Refer Slide Time: 19:06)


Geographical route finding : h(n) = Manhattan /city-block distance

Manhattan: h(n) = $|x_{Goal} - x_n| + |y_{Goal} - y_n|$

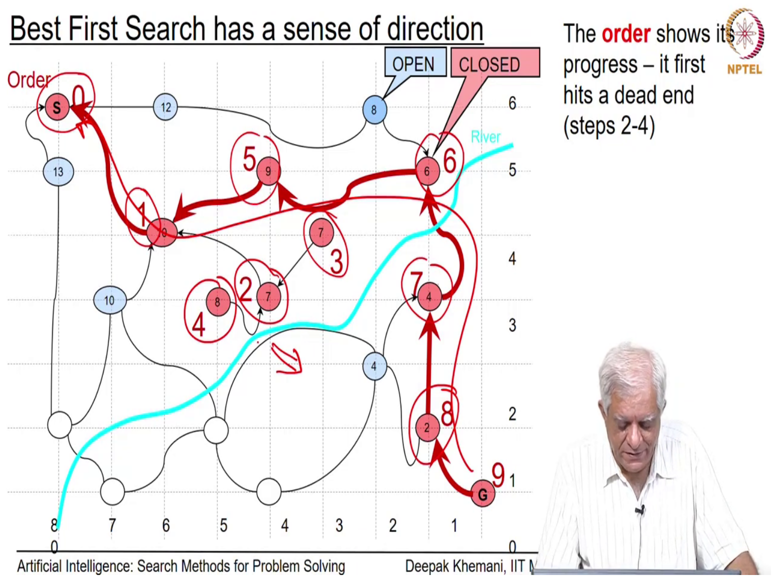
Artificial Intelligence: Search Methods for Problem Solving

Deepak Khemani, IIT Mad

Another estimate of distance is the Manhattan distance, which as you can see simply the modulus of the differences of the x coordinates and the y coordinates. So, this is called Manhattan distance, because it is the kind of distance you would have seen in a well structured town or city like Manhattan; where if you want to go from one point to another, you have to walk along one road certain number of blocks and then walk along a perpendicular direction.

So, the distance that you walk along is what you count. So, for example, in this case the value is 10; because you walk 6 units this side and then 4 units up, and that is a estimate of distance. So, this is 10 and this is 12 and this is 13; and as you can see again this also tells you that, this node is a better node to, better node to go to.

(Refer Slide Time: 20:05)



Best First Search has a sense of direction

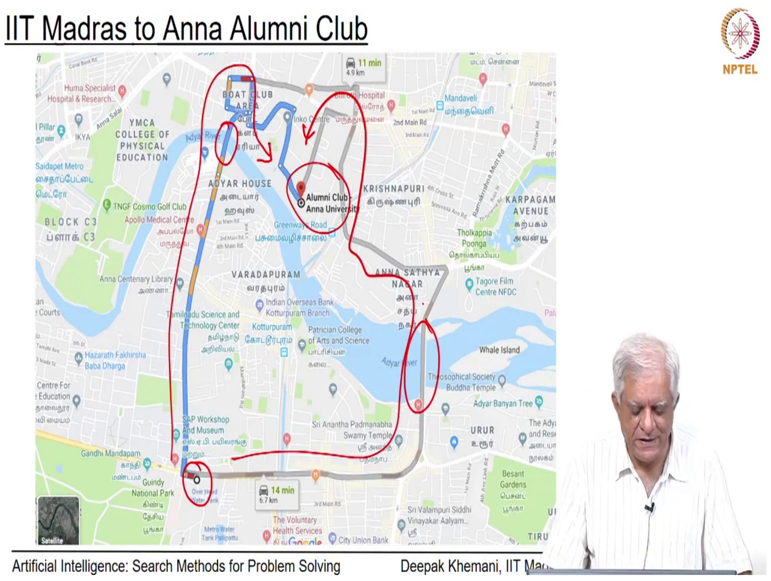
The **order** shows its progress – it first hits a dead end (steps 2-4)

Now if you want to learn best first search algorithm on this problem; then you would find that this is the kind of exploration it would have done, it would have the 1st node or the 0th node would they have been start. Then as we just saw, it would have gone to this node number 1; then you would have gone to this node, because it appears to be closest to the goal.

And then it would be forced to go here and then to node number 4; till after which eventually it finds a path through node numbers, nodes which is it sees in the 5th turn, 6th turn, 7th and 8th turn and this is the path where it is found to the goal.

So, this is the behavior of the best first search algorithm, it tries as far as possible to go towards the goal; but if there is a obstacle on the way or in this case if there is a river, then you cannot essentially.

(Refer Slide Time: 21:03)



IIT Madras to Anna Alumni Club

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

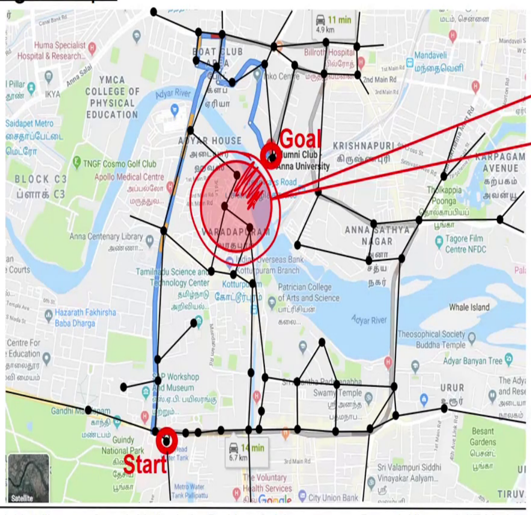
So, here is a real world example, I was looking at a couple of days ago is that; if you are here at IIT Madras and if you want to go to the Anna University Alumni Club, which was hosting, which is in fact hosting a bridge tournament and how do you get there?

So, if you consult Google maps, then you can see that, that Google maps gives you this as the best path and it gives you some alternate paths which you can go like this; but it has figured out that there is a bridge here and there is a bridge here, so the path has to go through those bridges.

And that is a thing that any search algorithm will do for you; as long as it is a global search algorithm, it will investigate all possible paths and take you to the path which, it will find a solution for you essentially.

(Refer Slide Time: 21:59)

Google Maps

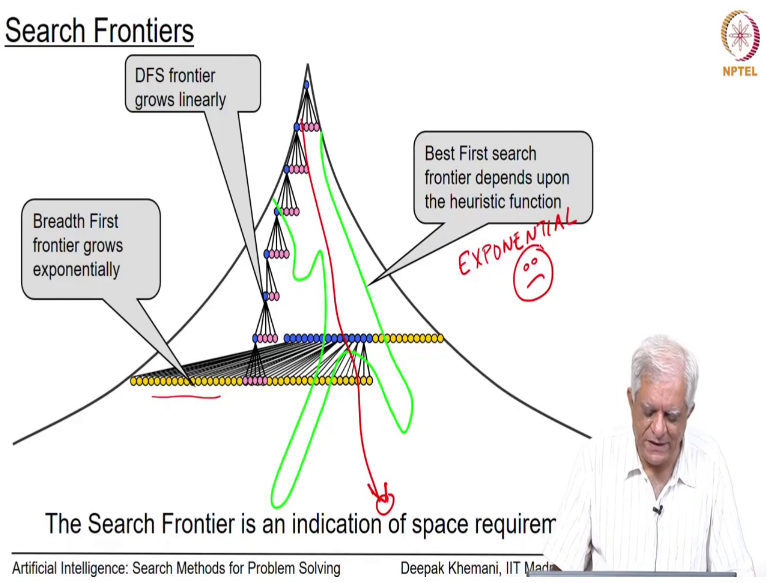


A heuristic function would drive the search towards *these* nodes and then would have to *move away*

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

Now, if you were to superimpose a graph on this same problem that we saw with the start node and the same goal node; you will see that the search algorithm will drive you towards this set of nodes which appear to be closest to the goal node essentially. But because the algorithm is complete; because it maintains the open list all other candidates that it has not yet inspected, it will eventually find a path to the goal node essentially.

(Refer Slide Time: 22:31)



Search Frontiers

DFS frontier grows linearly
Best First search frontier depends upon the heuristic function
EXPONENTIAL
Breadth First frontier grows exponentially
NPTEL

The Search Frontier is an indication of space requirem

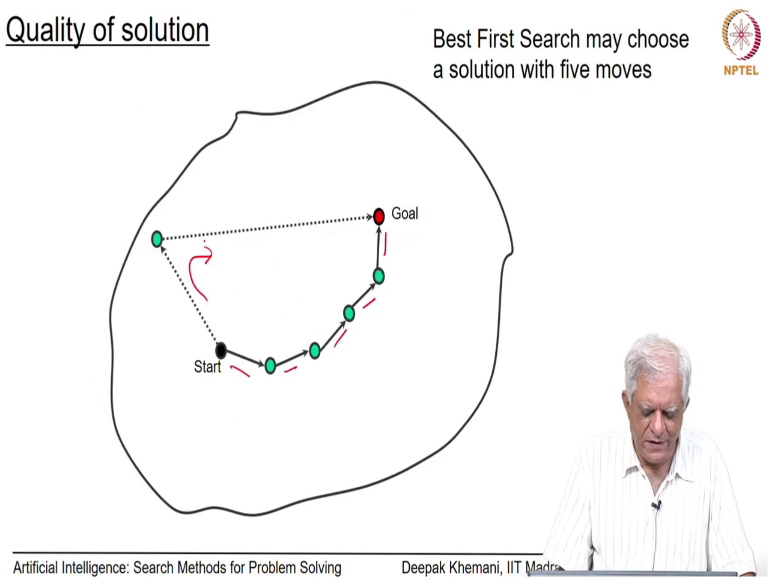
Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Mad

So, how does the search frontier look like? We have already seen earlier that the search frontier for depth first search is the set of pink nodes that we have we are seeing here, which is growing linearly; this search frontier for breadth first search was the set of yellow nodes that we have seen here.

In practice because the performance of best first search depends on the heuristic function and as we have seen that heuristic function can sometimes be misleading, it can take you towards the riverfront when you have to go from there to somewhere else; so in practice so such frontier tends to be quite erratic in nature.

But it has been a empirically observed that, it is often tends to be exponential in nature. If you had a perfect heuristic function which is like an oracle which tells you what is the right choice to make it every place, then the search frontier would simply be linear essentially.

(Refer Slide Time: 23:41)

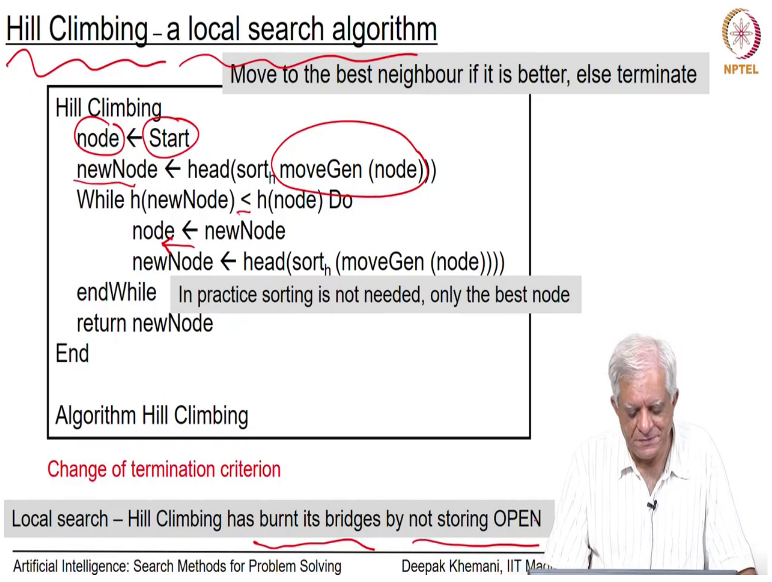

Quality of solution
Best First Search may choose a solution with five moves
NPTEL
Goal
Start
Artificial Intelligence: Search Methods for Problem Solving
Deepak Khemani, IIT Madr

If you just go, go off in some direction which takes you to the goal node essentially; but unfortunately our heuristic functions are not perfect and hence the search frontier tends to be a little bit larger than a linear function.

Whatever the quality of the solution found, so let us take this slightly contrived example; if you are at the start node here and you want to go to the goal node, then the there are short segments of roads which take you to a path which is heading towards a goal, but the number of segments is five, whereas there is a longer way of reaching the goal which has only two segments.

So, if you are interested in the number of hops only, then you can see that best first search does not find the shortest path. You will see variations of best first search which will also give you shortest paths later essentially; but at the moment we are focused more on finding the solution quickly and that is what the heuristic function is trying to do for us.

(Refer Slide Time: 24:33)



Hill Climbing - a local search algorithm

Move to the best neighbour if it is better, else terminate

```
Hill Climbing
  node ← Start
  newNode ← head(sort_h(moveGen (node)))
  While h(newNode) ≤ h(node) Do
        node ← newNode
        newNode ← head(sort_h (moveGen (node)))
  endWhile
  return newNode
End

Algorithm Hill Climbing
```

In practice sorting is not needed, only the best node

Change of termination criterion

Local search – Hill Climbing has burnt its bridges by not storing OPEN

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

Now as I said the search frontier for this thing also tends to be exponential in nature, which is a bit unfortunate. So, we have to look for other algorithms; because you know we do not as computer scientists, we do not like things which are growing exponentially. So, will start looking at variations which require less time and less space.

And we start with an algorithm called hill climbing, which is a local search algorithm. It is local in the search, in the sense that it only looks as the local neighborhood of a current node; as opposed to best first search which looks at a all the possible candidates that we have ever generated which are kept and stored in the open list essentially.

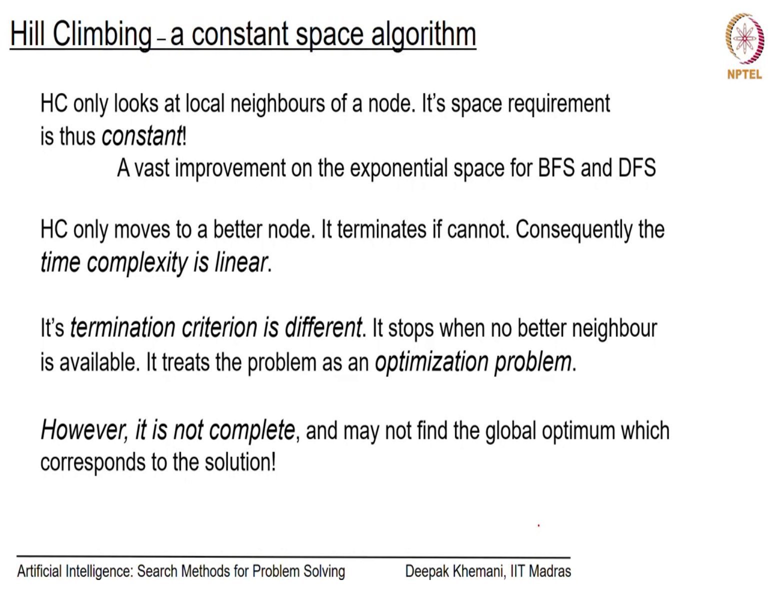
Hill climbing algorithm just throws away all the earlier candidates, and the algorithm is can be seen as following; that you are at some given node which is initially the start node and you go to a neighbour which is the best of all its children. So, you do moveGen on that, sort it, pick the head that is the best node; and if this new node is better than the old node, which is given by the fact that it is heuristic value is less, you move to the new node which is shown here and then look at the next node which is the one of the children of the, one of the neighbours of the node essentially.

So, we just keep repeating this process of moving to the best neighbor, if it is better else you just stop. So, while h of new node is less than h of node, you do this; otherwise you exit from there. In practice we do not even need to sort the children of any given node, because sorting is still quadratic in nature or at least n log n in nature; whereas you just want the best nodes, so you can just scan it once and get the best node in linear time. But that is a matter of implementation; we should not worry too much about, ok.

One thing to notice is that there is a change in the termination criteria; we simply said that if you cannot find a better node you terminate, which is very different from. What we said earlier is that if you found a goal node, you terminate or if there is no path to the goal node, then you terminate.

Here it is neither, it simply says if you cannot find a better node, you terminate; and that we will see results in certain change in behavior. The point to note is that, as a local search algorithm it has burnt its bridges by not storing the open list. So, it can go and get stuck in the local optima.

(Refer Slide Time: 27:53)



Hill Climbing - a constant space algorithm

HC only looks at local neighbours of a node. It's space requirement is thus *constant*!

A vast improvement on the exponential space for BFS and DFS

HC only moves to a better node. It terminates if cannot. Consequently the *time complexity is linear*.

It's *termination criterion is different*. It stops when no better neighbour is available. It treats the problem as an *optimization problem*.

However, it is not complete, and may not find the global optimum which corresponds to the solution!

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

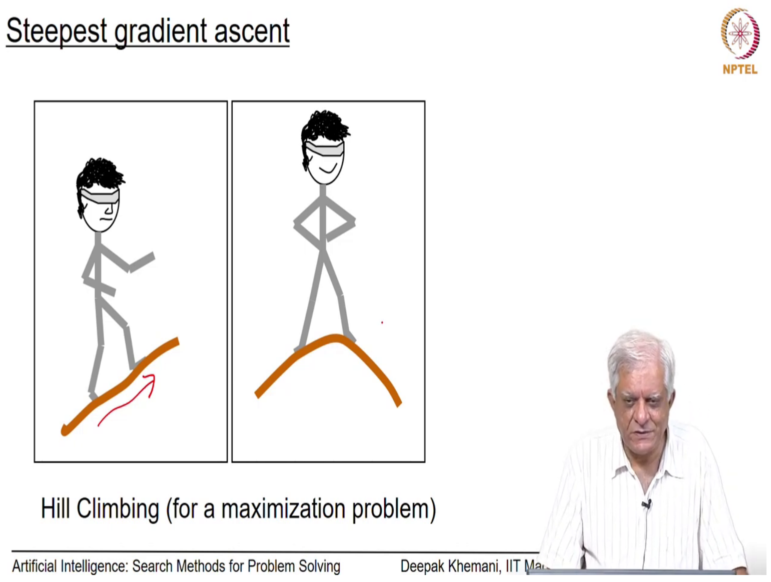
The nice thing about hill climbing is that, it is a constant space algorithm, it only looks at the neighbours of a node; it is space requirement is just constant, which is a vast improvement on the exponential space for breadth first search and depth first search that we need it. Overall space requirement, its behavior is it only moves to a better node; which means it will terminate if it cannot move to a better node.

So, consequently it is time complexity is linear in the depths essentially; that it goes to a certain distance and stops and you know, but it does not spend time looking here and there. Its termination criteria as we just mentioned is different; it stops when no better neighbour is available.

So, in some sense it reads the problem with an optimization problem; it says give me the best value of h that you can find, instead of saying give me a path to the goal essentially. So, it is like thinking of it as an optimization problem and it is not complete essentially, ok.

As we will see that it may not find the global optimum which corresponds to the solution. So, if you remember the eight puzzle example and city map example that we considered a little while ago; you would notice that hill climbing would have you know stopped much before reaching the goal essentially.

(Refer Slide Time: 29:22)



Steepest gradient ascent

Hill Climbing (for a maximization problem)

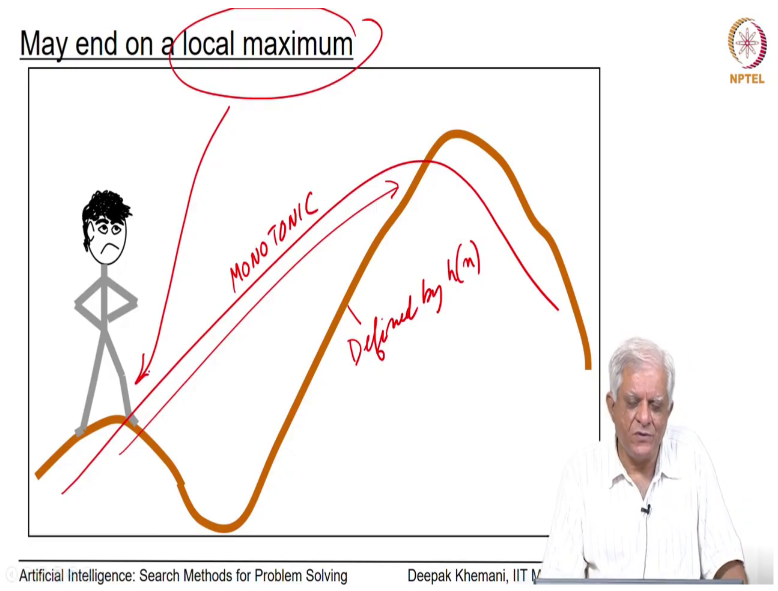
Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Ma

So, hill climbing is also known as a steepest gradient algorithm or gradient descent algorithm or a gradient ascent algorithm as a case may be; it really depends on whether you want to maximize the heuristic function or minimize the heuristic function.

And if you think about this maximization and minimization are just two sides of the same coin. So, in the eighth puzzle for example, if you want to minimize the number of tiles which are out of place; it is equivalent to saying that maximize the number of tiles which are in place essentially.

So, if you are talking about the maximization example; then hill climbing is essentially saying that you go in the direction of the steepest gradient, you just keep climbing. And you stop timing when you have reached the point where no neighbour is better than another neighbor. Now as you can imagine if you were doing this; if you were blindfolded and put on a hillside and you would do something like this and not surprising with this algorithm is called hill climbing.

(Refer Slide Time: 30:34)



May end on a local maximum

You would reach a place where you thought was the top; but if you were to open your eyes, you would see that this spot is called as a local maximum. And you have not found a solution, but the algorithm will terminate at this place essentially.

So, what do we do, how do we address the problem of not getting stuck in local maxima? This is something that we will do in the next few classes; we will start off by looking at some deterministic algorithms to avoid local maxima or we could look at some stochastic algorithms as well. And we also need to get an insight into the fact that, the surface that we are talking about is defined by the heuristic function.

That if you could somehow define a heuristic function where the surface is monotonic; in the sense that there as only one maxima or only one minima is the case may be, then. For example, if the surface was like this, then you can see that you would have reached the global maximum. And we will see, we will get some insight into this as we go along; that the surface is defined by the heuristic function.

So, if you were to choose a different heuristic functions, so we saw there are two possibilities in the case of eight puzzle; there were two possibilities we considered in the city planning.

And in the next class we will take a more concrete example, where we will look at the blocks world domain; and we will see that there are certain heuristic functions which define a monotonic surface, which would take you to the the goal node. But jagged surface like the one that we have started here with, you could end up on a local maximum, ok. So, we will do this in the next few classes essentially.