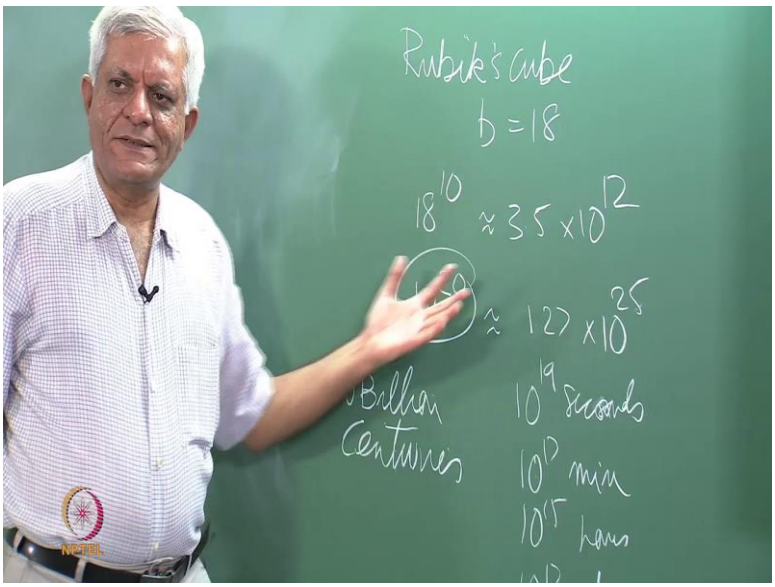
**Artificial Intelligence**
**Prof. Deepak Khemani**
**Department of Computer Science and Engineering**
**Indian Institute of Technology, Madras**

**Module No - 01**
**Lecture No - 09**
**Heuristic Search**

(Refer Slide Time: 00:14)




So, let us begin we continue our study of state space search and what we have just seen is that the size of the state space can be significantly large even for a small toys that we talk about. So, in Rubik's cube we saw where the branching factor is eighteen if you want to search up to depth 20, then you have to search about 10 based to 25 node essentially even at smaller puzzle which is the 8 puzzle and the 15 puzzle. If you look at the 15 puzzle where the branching factor is capped by 4, you can only move within one of four directions, it is about 10 raised to 13 states and the 24 puzzle is about 10 raised to 24 states.

So, even that small puzzles that you get can generate huge state space essentially and we have seen the 10 based to 24 states is not a number that you can slip that essentially it will take you huge amounts of time. So, what is happening essentially is that the algorithm that we have seen so far is blind. Essentially, that gives some state space given some start state, what is what this algorithm is doing with the goal that you want to achieve. It is using it only to test whether given state is goal state or not it is not using it

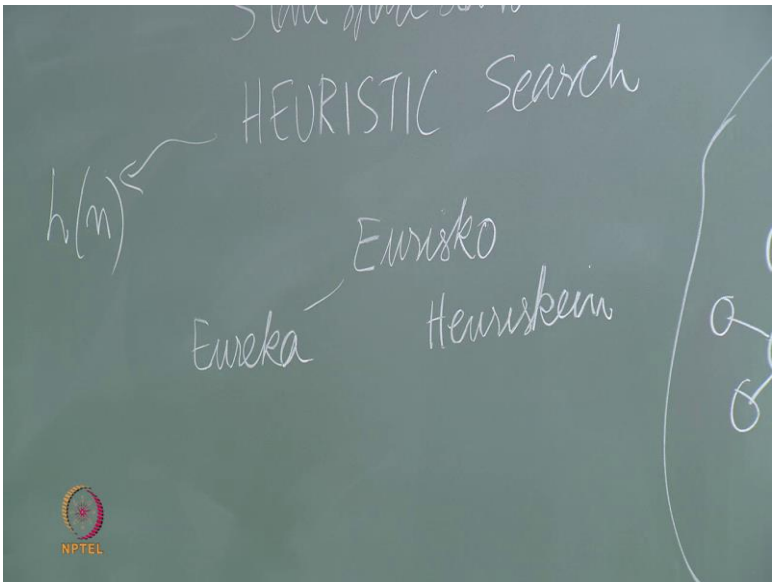
to guide at search in any manner at all depth first search as we said goes often some direction.

Then, it back tracks and tries other direction irrespective of where the goal state is breadth first search always circles around the star states and it will gradually expand, it search frontier to move away essentially. The goal step no role to play here essentially the next step that we would like to do is to somehow exploit what we know about the goal state to help guide the search. So, let us assume that this was a city map essentially and let us say this s stands for IIT and you have to go someplace essentially. So, let us say this is a goal state somewhere, so if you are consistent with the geography of Chennai, then this g could be may be Marina beach or something like that somewhere in the north east.

If you want to go of IIT to marina beach and you of course have to follow the roads the every junction is a node and every road segment is an edge essentially, so in some sense being at a junction is a state, so you are you are at IIT. So, you are at this state you want to be somewhere and here is the Gandhi statue in Marina beach that is a different state and you can move across the road to move to a different junction which is a different states and so on. Now, imagine what these two algorithms are doing they just go of in the predetermine manner essentially and eventually of course if somebody tells them we have reached Marina beach they will say I am done.

Essentially, what you really want what we really want is an algorithm which would somehow head in this direction somehow you know what do you mean by this? What we mean by this is that given a state given a start state and given a set of successors instead of choosing in a predetermine fashion, which successors to inspect or expand as a term we use next. Let us try to get some exploit whatever knowledge we can of the domain to help guide the search essentially.

(Refer slide Time: 04:29)

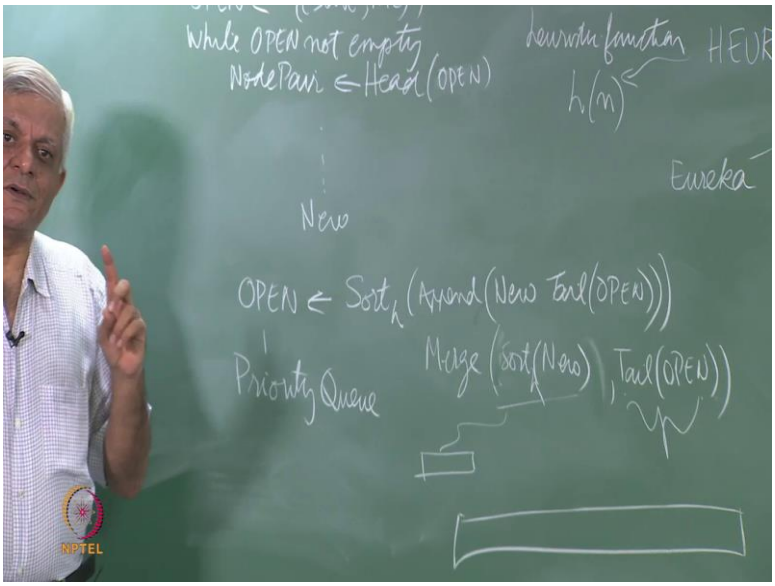



So, that is called as heuristic search, so the word heuristic comes from the word I mean with their various routes that I have read about, so Eurisko or Heuriskein and so on. All these are sort of Greek words of Greek origin and you might remember a word which is also related to Eurisko which is called eureka you know this story of argument is running naked through the streets saying eureka and so on essentially. So, what is eureka meaning it means that I have found it, I have discovered it, essentially and this whole idea of Eurisko heuristics is based on this knowing something essentially, so how can we get heuristic?

So, let me first discus how this heuristic knowledge would be used essentially, so we will assume that we have another function called h of n which will take a node or a state and give a value which is a measure in some sense. How easy is it to go from that state to the goal state, essentially so supposing we had such a function. So, the theoretical community would say we have an oracle, it tells you what is the complexity of going from this you know or it tells you which if somebody could tell you that of these successors.

In general, given open list which one to pick next then it would help it would be helpful essentially that because then they would just tell you if you go ahead if you go ahead if you go ahead you will reach the goal state essentially. So, first let us get this out of the way this mechanism of how we are going to exploit the heuristic function.

(Refer Slide Time: 07:03)




So, if you look at this algorithm that we have the same framework, we will follow open is initially made of this pair start. So, it should really be list of list essentially and then extract the first one extract the node all that is the same and we at some point generate new the set that we have of new nodes that we want to add to open, what we will do now at least conceptually is that. So, remember that so far we said that we can either add new at the head of the tail of open or the remaining part of open or at the end of the tail of open if it was at the head of open, it was like stack if it was tail of open, it was like a queue.

Now, we are saying is that something like this sort on happened new, so if I say append new tail of open that part is like depth first search essentially that new is added at the assuming append. That is what append does it sort of take this list and concatenate with this list putting them at this towards the head it is like that this thing. Now, I am saying after you are appended the two list sort them on the h value what is h value h value is this value of this heuristic function of the node essentially.

This means what simply that the best h value will come the head of the list we are still talking about it is the list and this same algorithm will pick it from the head of the list and then this thing. So, if we have such heuristic knowledge we can exploit it by simply modifying our algorithm to keep open as a sorted list. So, that when we remove the head element we always get the best element what do we mean by best? By best we mean best

according to what the heuristic function believes or what the heuristic function thinks essentially only as far as that essentially.

If we have a heuristic function then we can, this is called a heuristic function is should write it right, it returns a number which is a measure of how easy or hard it is to go solve the state from there essentially. So, this is the only chain we would need to make in our algorithm that we have to sort the open list every time we add new elements essentially. Now, obviously the data structure person inside you must be rebelling against this idea sorting this list again and again and again because sorting is expensive. So, you could you could do a other thing you could do for example, merge then sort new this tail open yeah that is a first thing one think.

So, this is not needed that because the number of new elements is going to be small and the rest of the tail is anyway sorted. So, just sort the new elements again an h of course, and then merge it with the tail open of course, which is a little bit better in terms of computational work, but that obviously is not the last word on this. So, just like we said that to make efficient use of close you must maintain it as table what about this open now how do we efficiently implement open. So, of course, that efficiency will only be from the computational point of view as far as the task is concerned of choosing the next successor.

It does not matter in what manner we manage to sort it how much time is spend sorting it essentially, but if you what if you were concern about that how should you manage open at the risk of turning this into a data sectors class we will spend one minute here.

Student: Sir, we have to maintain it.

So, you must maintain open as a priority queue of course, there are many different mechanisms to implement priority queues we will not go into that we will just make this observation that you must somehow implement open efficiently. So, what is really happening just imagine this that we have this tail of open this, tail of open is like a big.

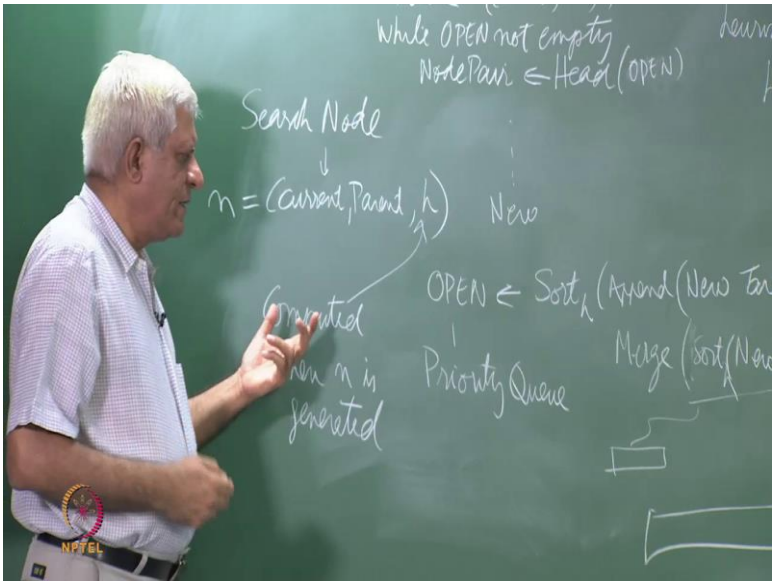
Let us say it is sorted list or something like that and then we have this new which is a smaller list essentially. So, this is already sorted and we have get this we have got this new elements and we have to sort of insert them into in the right place inside this sorted list. The best way to do it is by a priority queue one way to implement the priority queue

is use heaps, but there are other ways of doing it. So, you must if we are implementing heuristic search it is very important that you pay attention to how you manage open because that is going to affect your running time essentially things become a little bit more complicated.

When you remember that you also wanted to remove from the new list those things which are already on open essentially which is not for which the priority queue is not really the best we have doing things. So, I will leave this as a small exercise for you to as the data structure exercise you might say you want to do both these task you want to check for membership in open. That is a new node already present in open, which is based on by a hash table you also want to maintain open. So, always the best nodes will be at the head of open which is what a priority queue does.

How can you do both, can you do both in and if yes how it will be, but it is not our primary concern here our primary concern is to look at how heuristic search happens. This is basically what is happening essentially, now let us pay attention to the heuristic function itself how we get this heuristic function.

(Refer Slide Time: 14:34)

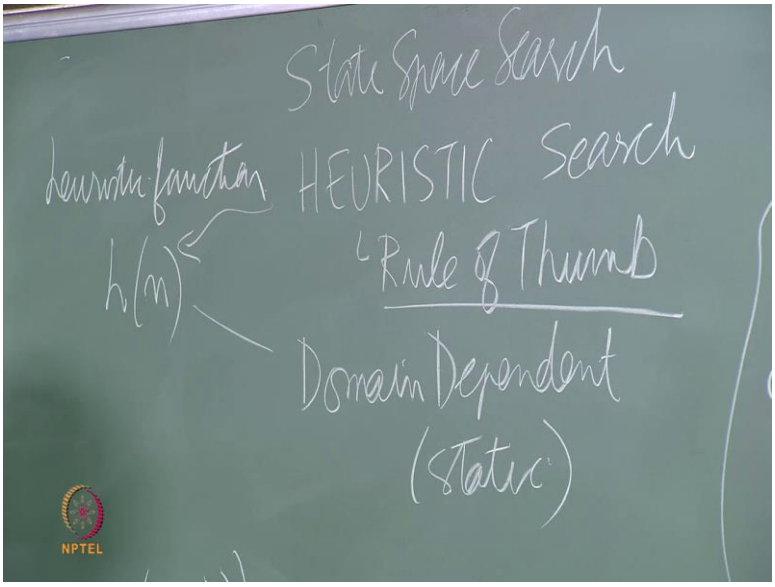



So, one thing that one can do is which part of operational zing is this whole process is at the search node which was originally a pair which has the current node the parent node. Now, we want to convert it into a triple, so it will become current parent and that is the call it h value for that node essentially. So, we want to also store that h value out of node

which means what that every time we generate a new node we will need to compute the h value. Let us say this is the n, n is some node and you want every time we generate a node n we want to compute the h value for that node.

Now, of course, you must keep in mind that this h value is not just the property of that node it is also dependent upon the goal node essentially for different goal nodes. The same node may have a different h value, but we are sort of losing over that here little bit essentially, so how do we how do we generate these heuristic functions?

(Refer Slide Time: 16:27)




So, there are two approaches to this one is domain dependent or static, so by static we mean that it only look as this current node and may be it looks as a goal node in some manner, obviously it will have to do that and gives us a value back essentially. So, let us take an example or couple of examples let us say this is a city map like we said that is this is IIT and that is Marina beach and so on and so forth. How can we give heuristic values to this node essentially, so the simplest thing is h of n, so city in a city map problem route finding problems. So, now, a days of course, we have all these algorithms which do route finding for you have to go from one place to another you must think about how they do this we can say h of n.

So, let us assume that the goal node is given to us, so what can we expect in the city map we can expect to get coordinates essentially. So, start state is from x start and y start in the two dimensional map and goal state is some x goal and y goal if the coordinates are

given to you and that you can that is reasonable to expect that the coordinates may be available in such a situation. How can we exploit this? How can we sort of get this? So we say for each node or each location in the map we compute the distance to the goal node and that is the estimate of heuristic that we will use.

So, one is Euclidean distance and that is a good estimate of, so how are you going to use this Euclidean distance we are going to say these are the different places that we can go to. So, remember that in a city map situation we are moving from one generation to the next that once you on a road you have to go to the next you have to go to next junction which has these nodes here. So, essentially you compute this distance for each of these n nodes each of this successor and in general for every node in open. To start with, this is only open that we have and then choose the one which has which seems to be closes to the goal essentially now this is the heuristic.

So, when we say heuristic we also use a term rule of thumb and if you go to the psychology today website they will say that it is a mental shortcut heuristic is a mental shortcut which we use to solve problems quickly. So, the emphasis the word quickly we are trying to fight this common neutral explosion and trying to use some knowledge in this case from the domain to guide our search. So, when you are using a domain dependent heuristic function we have now three domain function one is the move gen function that we already talked about and one is the goal test function.

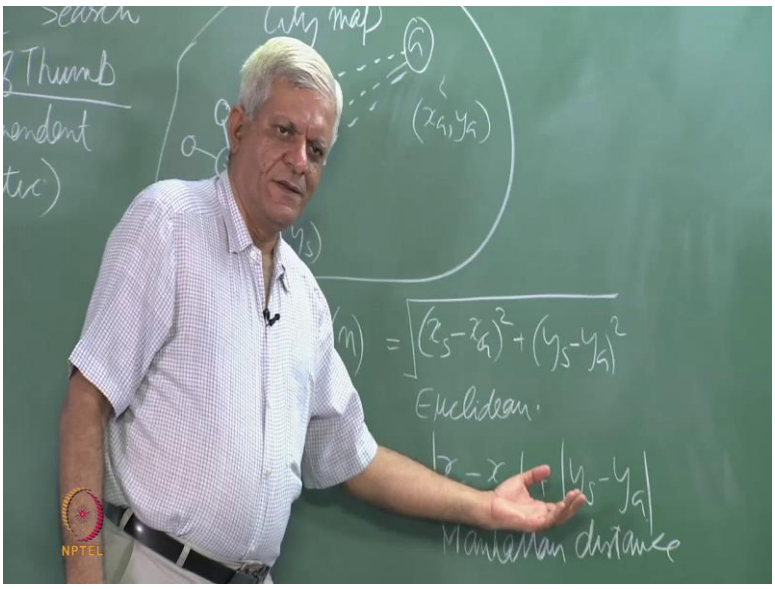
The third is now the heuristic function which means that if you are going to now create a new domain you must write the program which will compute h of n. So, that is the search algorithm can exploit that is why the search algorithm goes it only uses that h value to sort or maintain a sorted list or maintain a priority queue of the candidates essentially. The domain must now tell us what the h value is we will in the moment we will try to see whether is this can be done in a domain independent fashion at all because you see this whole idea that we are trying to pursue is to write this algorithms in a domain independent fashion.

We want to write these search algorithms without saying that I want to solve the city route finding problem or a Rubik's cube or a Robert movement problem or a factory scheduling problem. It does not care you have to find some general purpose algorithm and we would like to plug-in domains and just use those algorithms essentially.

Yesterday there was a talk about this mars Robert yes anybody attend, so this NASA, you know they had this mars Roberts. They used a lot of this kind of planning algorithms for finding paths for this and move because it is not just the path for the Robert to follow also what are the movements that.

For example, a robot arm must do if you have to pick up something from the ground, let us say piece of rock from the ground or something every move have to be has to be planned carefully and there we use the lot of these techniques. So, heuristic function of course, if the heuristic function is perfect which means it is like an oracle what the theoretical computer scientist would say. Then, it will just tell you which is the correct thing and then of course, you would find the solution in linear time essentially, you just go from along the path essentially, but in practice of course, it is not, so easy to find heuristic function.

(Refer Slide Time: 22:50)



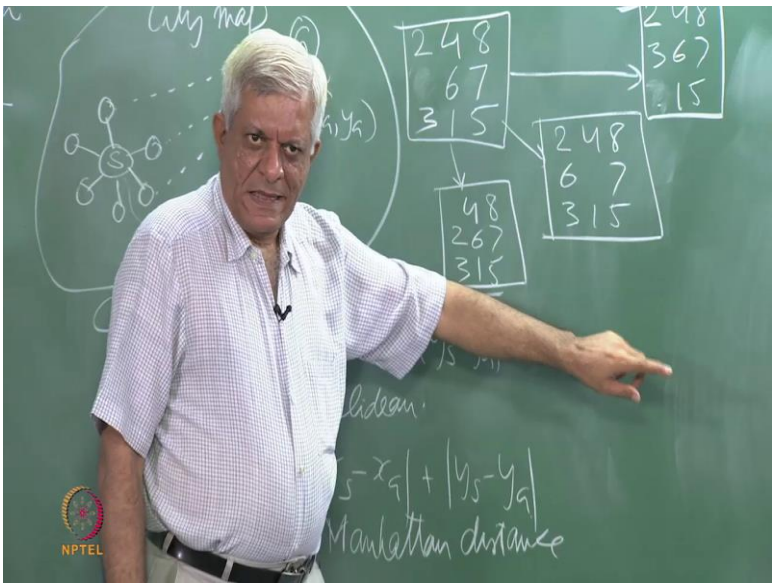

What is the other function that one can talk about, so another function that we use sometime is simply this x s minus x g, so what is this doing instead of measuring. So, the Euclidean distance is telling you the distance as we say as this is telling you a difference between the x coordinates and the difference between the y coordinates. What is the name of this distance function Manhattan function Manhattan distance also called a city block distance essentially if you have a nicely laid out city like Manhattan, then you know it is like on a grid then you can only go along the roads on the grid.

That distance that you cover would be basically this that if you know if you had if a grid on a grid you can only move along the x direction or on the y direction and then other distance measures that one can think of.

For example, one could take the max of this or max of this or max of this and this only and choose that or there are other something called Murkowski norm which we will not get into here. So, essentially what are we trying to do we are trying to devise a domain function which takes two inputs one is the node given node and the other is the goal node which we assume the during the ten year of the search is constant. So, we are not mentioning it here, but it is there somewhere in the background and it returns a value to us which gives some idea of how close one is to the goal node essentially.

So, in this context we can say a heuristic function is an estimate of the distance to the goal essentially. So, we are using the term distance explicitly here in this case essentially in general it is an estimate of how hard it is to solve a problem from the given state. By hard, we mean typically the number of moves that you want to make and later on when we see when we add cost to each move. So, for example, in the road we might add the length of a road all the conjunction on the road as a cost or something that we will come to that later. So far, our idea of optimal solution is simply to have a smallest number of moves essentially.

(Refer Slide Time: 25:36)



Let us look at another example which is this eight or fifteen puzzle, so let us say this is my given state and I have or let me choose one with three successors, so I have three successors from here one is that I can move this two down. So, I will get 4, 8, 2, 6, 7, 3, 1, 5 or I can get, I can move 6 this side, so 2, 4, 8, 6, 7, 3, 1, 5 or I can move 3 up. So, I have these three successors which i can make from here and I have to decide which one of them to choose so obviously I need a goal state. So, let me say that we have some goal state which looks like this and so let us say any state can be a goal state. So, let us say this is a goal state and we want to somehow arrange the tail, so we reach that state essentially how can we talk about heuristic functions can you think of heuristic function.

Now, what is the heuristic function or static heuristic function, so let us say we are talking about this state this middle state where we have moved 6 to the left hand side is this move that we should make? So, that is the question we are asking should we make this move or this move or this move essentially. So, we want to four each of this we want to compute the h value as it is called are the heuristic value which basically should look at this only. That gives us a number and let us say we were adopt the convention that the smaller the number the better for us essentially we going with the sense of distance or notion of distance the smaller the distance the better can you think of a heuristic function.

So, one function is let us calling it h 1 of n is equal to sum of the distance for each tile distance to goal. So, when I say distance to goal I mean the distance to the goal position no suggest keep in mind and what do we mean by distance here. Let us assume that we mean the Manhattan distance because this is the problem where actually distance is has to be it makes sense use manhattans because you can only move horizontally or vertically. So, what would be the heuristic value of this state for this state we look at two is here and two is there.

So, I will add one for two then four is here and four is there, so I need two steps to get to that. So, 2 plus for 8, I need 1, 2, 3 steps for 6, I need 1, 2, 2 steps for 7, I need 1, 2, 3 steps for 3, I need to move 1, 2, 3, 4 steps or one I need to move 1, 2, 3 steps and plus for 5 I need to move 0 steps. So, basically the sum of all these numbers, so let me remove them, it is confusing here, so I have all these numbers and basically the sum of all these numbers is the heuristic value for this state essentially. So, if we added up you will get a heuristic value likewise, you do for this and likewise you do for that and you would have

some way of making a guess which one is better. Can you think of any other heuristic function? It can be simpler than this, number of misplaced tiles.

That is another, so in this case h 2 of n, you simply count how many tiles are out of place essentially. So, you can say in this situation five is in place and nothing else is in place or if you want to count the blank tile as well then you can says the blank tile is in its place, but that will anyway add up to something essentially. So, it is a simpler measure essentially it is easier to compute, but is in a notion of better here is one heuristic function better than another supposing I want to tell you that these are the two heuristic functions to use.

What does your intuition say see there are two aspects to making this choice one is the cost of computing the function and second is the benefits that it gives us what is the benefit the benefit that we are looking for is that we should guide our search better. In other words, one way of counting this is that that if you let us say run 10,000 experiments on this 8 puzzle then you could compute something like this a measure its we call as effective branching factor equal to number of nodes seen divided.

So, there is a effective branching factor it is the property of the heuristic function and the way it is computed is that you run an experiment what is it mean you give some start state some goal state. Let the heuristic function guide search according to this algorithm that we have just written and then count how many nodes that the algorithms see and divided by the total length of the solution what is the ideal value here. Ideal value is 1, essentially even to heuristic function is perfect you will only see the nodes which take you to this goal station, so you will not see any other new terms.

So, the ideal value for this effective branch effective is one the branching factor of the problem itself is known to you which is 4. Essentially, let us say is bounded by four or it has Rubik's cube its I mean concentrating, but in this case it is not concern, but let us say it is. So, the effective branching factor will be some value between 1 and 4, the better the heuristic function. Of course, you will not do it for one experiment, you will run thousands of experiments just to even out all statistical variations that you come by choosing difference start states and that kind of a thing essentially.

So, then you have a notion of one being better than the other, but at the same time you have the cost difference between the two in this case these two functions. In general, they

may be a choice of more than two functions, so one has to choose the heuristic function essentially.

Now, if you just go back to this algorithm again it makes sense to use a heuristic function if now remember that the heuristic function you have to compute for every node that you generate every time you call the move gen function and generate the children. After removing the duplicates ones, you have to compute the heuristic function for that node essentially. So, the extra cost of computing the heuristic function must be much less than the total time save for this algorithm which means that how do you save time by seeing a fewer number of nodes.

Essentially, if the heuristic function is good then you will not go down some paths which are meaningless you would only go down the solution path essentially, so obviously it make sense to choose a versatile use a term inexpensive heuristic function. The simplest or the most inexpensive ones are the static functions which only look at given state and tell you a value essentially it shows a state. It will say this is a good state or a bad state in terms of some number which it will compute using one of those things why do we say that, so let us go back to this city map example for a moment.

What can happen which the heuristic function cannot foresee essentially, so if you know the topology of Chennai and if you know that you will go to the gate, you have no other option you go to the main gate and from there you have to decide where to go. So, this is the starting place that is the goal location what happens when there is a river running across this, we have this Adyar River and it even has water sometimes. A heuristic function which is only looking at the start state and the goal state will be oblivious of the fact that there is a river on the way.

So, in the mountainous situation it could be a paths for example, you need a paths to cross some mountain ridge here you need a bridge. So, let us say the bridge is one bridge is here and another bridge is here, but there is no bridge along the blowflies direction close the river. What will the heuristic function do it will drive the search in this direction because it will tell you this node is better than this node and this node is better than this node. In actual practice these two one of these two nodes is likely to be better why because this might take you to this bridge and this might take you to this bridge,

whereas this will take you to dead end and then you will have to go down this path and then go like this.

We will not be able to see in the future a heuristic function will make judgment only based on what information it has in this case it is a static function which means the current location and the destination location and based on that this appears to be the best. So, in that sense heuristic functions are fallible that is not necessary that they will give you the best thing. We will in the next class look at another example to see how a heuristic function can take, you can be misguided in some sense essentially and we will discuss that in the little bit more detail.

So, you have to devise some heuristic function which will give you a number for every node that you add to the open list and we are assuming here that the smaller is the better because we are talking about notion of a distance. Then, we will sort the open list according to the heuristic function and always choose the best node and the search will progress essentially. So, what is the hope the hope is that the search will go towards the direction of the goal, but sometimes it may go in one direction then it may have to back trap and try something else essentially. So, which one might say that you start with the search this thing go down some path and you can fill in some values so that we will go down this path here.

So, for example, this could be 40 this could be 60 and this could be 50 or something like that. So, which choose 40 and then likewise remember that all these are closed according to the algorithm and everything else is opened essentially. So, it is possible that the heuristic function will take it down this path and suddenly it will discover that the heuristic value has short of at this stage. So, this becomes the next directly, so the heuristic function is guided since its guided by the heuristic function it does not always explore the space in the same predetermined pattern, but it will depend on what is the goal that you have given to us.

If the goal of was instead of here is the goal was here, let us say station or something then the certain algorithm would have try to go in this direction essentially. So, its behavior changes with every goal that is a basic idea that it is trying to go in the direction where the goal will be achieved essentially. Likewise, for the other problem essentially I

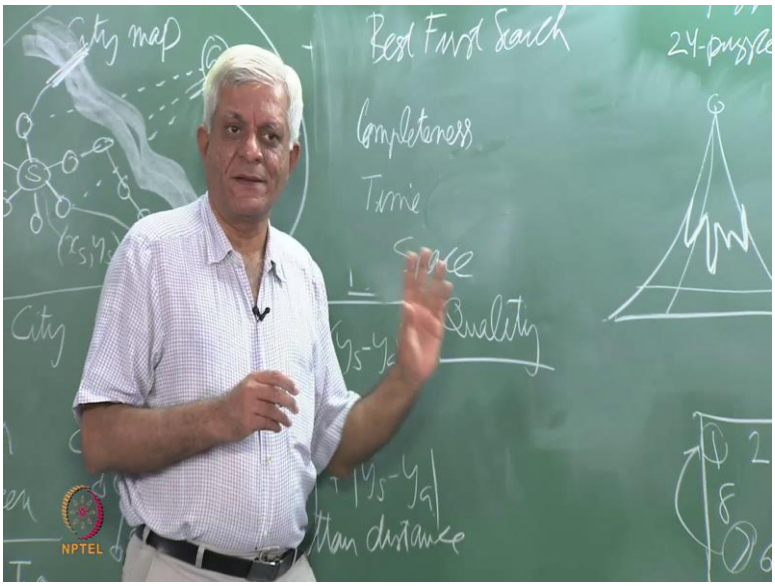
will come to the dynamic functions in the moment lets discus the properties of this algorithm I have gotten the four criteria that we have talked about.

Let us start with completeness, so where should I write this I think let me here, so let us give this algorithm and name by the way we haven't given it a name and this algorithm is called best first best first in the sense that of all the nodes in the open. It will pick the best one first and what do you mean by best the one with the lowest heuristic value, so best first what are the properties completeness let us assume that it is a finite space for simplicity, which is complete. It is complete, but how would justify your argument actually we did not actually give a very a little bit more formal argument even for the earlier search is, but you can give it now looking at this algorithm how can you say that it is complete essentially.

In every cycle, it picks one node from open inspects it and either put into close or succeeds or whatever the state space is finite which means open can have only all the entire state space in the worst case it will explore the entire state space. In every cycle, it will inspect one node from open, so it will terminate either when open is empty or when it found the goal state it will always find the goal state if it exist because it will at some point enter the open. It will always pick node some open and open is the finite list specially given the fact that we are prove we are moving the duplicates we are never adding the same nodes again.

So, the open have only the finite set to start with and in the worst case it will pick them one by one and in a recycle and remove it and say I cannot find the solution. When the solution exists at some point, it will be picked by open pick by this step here and it will say succeed what is the difference between depth first and best first. The only difference is that best first is sorting at least conceptually it is sorting the open list every time essentially though that is the only change, but it is going to pick every node once before it terminate, so completeness follows.

(Refer Slide Time: 44:45)




Let us talk about time and space in one breadth here very quickly what would you say time complexity, let us worry let us not worry about space. So, much in this moment, let us talk about time it really depends on how good the heuristic function is. Now, you can see that if the heuristic function is perfect, then the time complexity will be linear you will always go to the correct node linear in what linear in depth right. So, the depth is this search tree, you will always choose make the correct choice at every level and in linear time you will reach the goals.

In the worst case, it will exponential essentially if the heuristic function is bad the heuristic function could be actually malicious essentially like you know in some cities you find ask somebody how do I go there and then tell you the opposite direction. So, even in the worst case you will only do an exponential amount of search essentially, so it really depends upon the heuristic function likewise for space it really depends upon the heuristic function. Now, in the search tree that we were drawing like this solves the open list for breadth first search and the open list for depth first was like this. Now, it is really difficult to give you a feel of this exponentially going pen list of breadth first search, but you must keep in mind that every time when I go from one level to the next.

I am multiplying the length if you want to call it by b essentially where b is the bunching factor which is not really depicted in the diagram this is just kind of a schematic diagram, but every time it is going becoming longer by b. So, therefore, it is going exponentially,

whereas the for breadth depth first search the open is basically kind of proportional to length and its linear that we have argued it turns out that for best first search. Typically, the search frontier looks like this, now this is typical of course, which means it is not really linear and it generally tends to be more towards exponentially in nature in practice essentially.

There is one thing that you should consider if you have solved this kind of problems or if you have solved Rubik's kind of problems just try to imagine that sometimes you have to go against the heuristic function. We will come to this point later in the sense that if you are counting for example, the number of tiles and place or something like that then at some point you have to disrupt something that you have already done. So, for example, in the Rubik's cube you have done the top phase and then do to do the second layer you have to temporarily disrupt the top phase now heuristic function would be very upset about such things.

Now, you have made the top phase and then you are disrupting it will not reflect nicely in your search essentially. So, we will come to those issues later completeness time space and what else quality I will differ this discussion quality later just think about is do you think that this will give you an optimal solution or not and we will come to this later. I just want to spend the couple of minute on the other side which is the domain independent and without going into details I will just say this that domain independent heuristic function solves what we call a relaxed problem. So, it solves a relaxed problem and. So, what is the key difference the static function only looks at the given state and the goal state and gives you a value.

So, relaxed problem and when we look at planning we will see if we get time we will look at this in a little bit more detail, but the relaxed problem is modifying the original problem, so that it can be solve more easily essentially. So, to give you an example when you look at the eight puzzle like this, now if I have to go from two to if I if two has to go here or if two has to let us say seven has to go here in place of one. We want to move seven to one now in the real eight puzzle you have to first move eight out of the way, then you have to push seven up.

Then, you have to you know somehow create the gap by pushing, so we have pushed eight here, then you push 6 and then 5 and then 4 and then 3 then 2 and then 1 and then

you push seven up essentially you have to do a lot of moves in the real world. Essentially, imagine an eight puzzle in which you can slide on top of other slides or sit on top of other slides other tiles essentially which means I have modified my problem that I can make a move a seven can come here. Then, it can come here essentially, it does not have to be a blank tile to move into which is the real problem essentially the relaxed problem, you can move over on the tile and sit on top of another tile.

Now, you can see that if I am working with this relaxed problem I can easily find how many steps it take for me to go from seven to one this is two steps now which where we devise the static function. Also, we gave the same value remember that distance to the goal step he said the difference between the static. This dynamic domain independent function is that the static domain function was devise specifically for the eight puzzle whereas, when we learn how to sort of pose problems in the uniform manner. We can see that we can pose problems and then we can pose relaxed problems in a domain independent fashion and they can be solved typically in polynomial time that is a key thing.

So, what is a difference between a static function and a dynamic function or domain independent static function we assume is solved in constant time because it only looks at the given state. The goal state and gives you a value whereas, this domain independent function actually expose the space, but under different constraints which are relaxed from the original problem. So, it still may search up to the certain depth essentially, but problem has been relaxed to such an extent that to solve the relaxed problem it needs only polynomial time u that is a general idea. We will try and come back to that later, but it is done in a domain independent fashion I just illustrated it with the eight puzzle here essentially.

So, what happens now you have a heuristic function which is computed whenever n is generated with a static function this is computed in constant time. So, obviously, it likely to be helpful the question is if I have a polynomial time function sitting here is it going to help which is attractive. Then, I can do this in a domain independent fashion I do not have to say oh this is the city map. I can use a Euclidean distance or that is a eight puzzle I can or that is a Rubik's cube or I can count the red tiles and the blue tiles.

I do not have to do any of that reasoning I will say in some well defined manner, I will change the problem definition to such in such a manner that it can be solved in polynomial time and then the length of that solution I will use as a heuristic value here. So, you have plugged in a polynomial function here and the question is does it help you have plugged in this polynomial function inside this thing which is basically exponential in nature and the answer is that yes in practice. It helps and we will see that little bit later, but to generalize the idea of using a heuristic function is to use a function which will given the choices you have given the open list that you have.

It will tell you which node to inspect next rather than do it in a blind fashion like depth first or breadth first or for that method d f I d was using it is using some it is doing search in a more informed fashion and that is why it is called best first search essentially. So, we will take a best first search again and then we will try to improve upon this a little bit we will come back to this notion of quality and completeness and revisit them. Essentially, keep in mind that time and space complexity on the average are still exponential in nature they may be better exponential functions than the original, but that they still tend to be exponential.

Of course, in some domains, where the heuristic function is very nice easy to build in which case you will get lot of improvement. For example, if you are in Manhattan, then you can find the path quite easily I think, so we will come back to this in the next class.