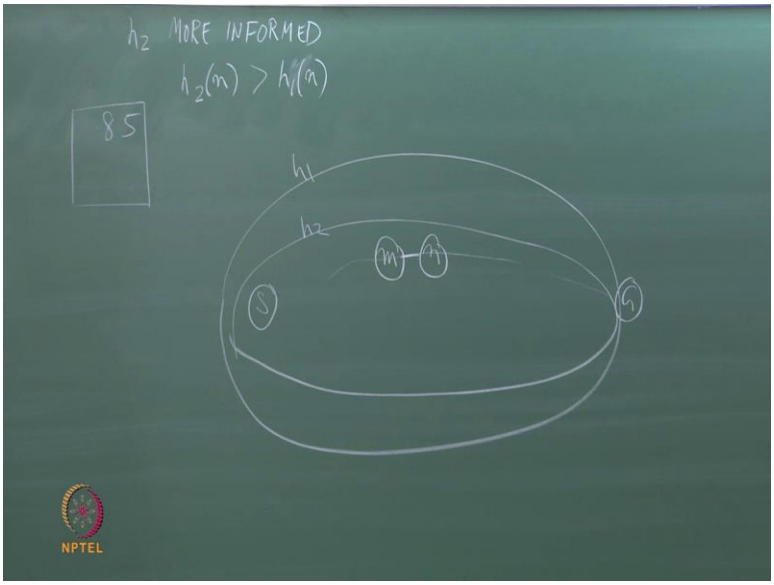
**Artificial Intelligence**
**Prof. Deepak Khemani**
**Department of Computer Science**
**Indian Institute of Technology, Madras**

**Lecture - 21**
**A* Monotone Property, Interactive Deeping A***

(Refer Slide Time: 00:13)



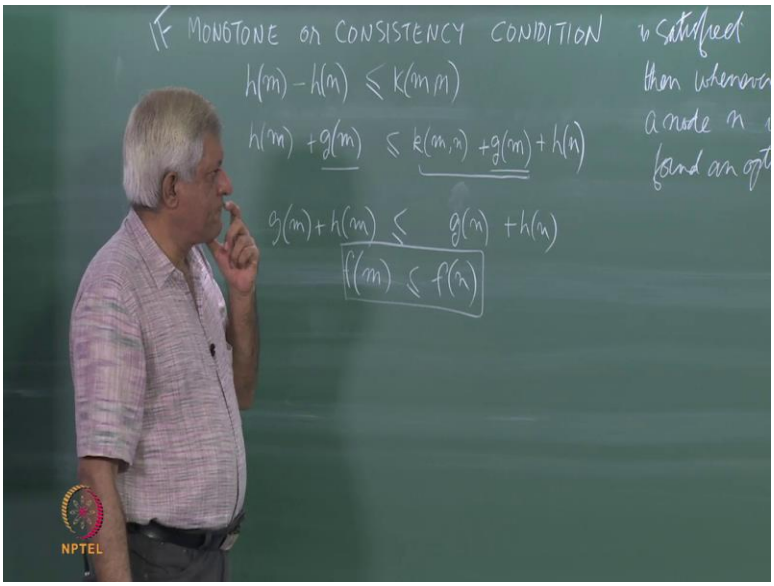

We are looking at the A star algorithm. In the last class, we saw that the algorithm is admissible, provided the heuristic function underestimates the distance with the goal, and provided, every edge has the cost, which is greater than the some small value, epsilon, and there is finite branching. We also saw that if two nodes, if their two heuristic functions, one of them is more informed than the other, which means h 2 of n is greater than h 1 of n, for every n. Then, we said that algorithm a 2, which uses h 2 or a 2 star, which uses h 2, will explore a smaller search space. If you can depict as follows; this is the start node and this is the goal node. Then, one algorithm will search something like this, and the other algorithm will search something like this. So, this would be h 2 and this would be h 1; this is what we expect, that the more informed heuristic function will be more focused towards the goal. So, it will search less away from the goal, whereas, what branch and bound would search, would have been full circle around of that length.

So, the fact that using a heuristic function actually, focuses a search towards the goal and more informed heuristic function is, the more it focuses towards a goal; which means, search becomes

narrow and narrow. With a perfect heuristic function, the search would be just one pass that would take place, essentially. Unfortunately, we do not have perfect heuristic functions. So, we try to look for as good functions as possible, which means, that the heuristic value must be as high as possible, but it should not cross the level of the optimal value, especially. For example, in the 8th puzzle, it has been observed, that if there two tiles, let us say, 8 and 5; and they should be reversed, let us say, in the whole situation; 5 and 8 are here, but they should be 5 and 8, instead of 8 and 5. Then, one way, one of the functions that we saw was; simply count how many tiles are out of place, in which case, two tiles are out of place; both 8 and 5 are out of place. The other function that we saw was the Manhatten distance function, which said that you need one step to take 8 to this position, and you will take one step to get 5 to this position. So, the cost would be 2, essentially, or the contravention of these two tiles would be two, essentially. Now, if you have solved this 8 puzzle kind of thing, you know that you cannot exchange them, essentially.

If you want to interchange the position of two tiles, you have to do some round about movement. So, you will need to add a certain value to that, essentially. About 7 to 8 years ago, somebody enhanced the heuristic function to count for such things, that they are in the correct row, but in the wrong order. Then, you add a certain amount, and that search was able to find optimal solution such faster for bigger problem, which means for the 15 puzzle and the 24 puzzle as well, essentially. Now, today we want look at the other property, which is that; consider two nodes, we consider some path to the goal node, and two nodes m and n, on the path will (Refer Time: 04:48).

(Refer Slide Time: 04:50)




If you take any such two nodes in the search place, and if they satisfy this property, that h of m minus h of n, is less than equal to k of m n, in a sense, you can think of this, saying that it under estimates the cost of every edge, which is on the path to the goal, essentially. So, h of m says that it is an estimate to the, from m to goal; h of n says the estimate from n to goal, and h of m minus h of n in some sense, is the estimate of the edge cost from m to n. If this is less than the actual h cost, in some sense, we are saying that it is underestimating the edge, every edge cost, essentially. This condition is called monotone of consistency condition, and what we want to show is that if this condition is satisfied, which means that it is the property of the heuristic function; a heuristic function is such, that this property is satisfied for any two nodes, which connected like this. Whenever, A star picks some node n, because already found an optimal path to n. You want to show that, you will do that in a moment, First, let us look at that what this is saying; what is the implication of this, essentially? We are saying that, under certain condition, which is this fact that heuristic function satisfies this condition which is called monotone or consistency condition.

If this is satisfied, if the heuristic function said that it satisfies this property, then the algorithm which is using that A star, which is using that heuristic function; whenever, it picks a node, any node n; picks meaning, it picks some open, right; it has already found an optimal path to n. What is the implication of this on our algorithm? If you look at, if you think about the A star algorithm,

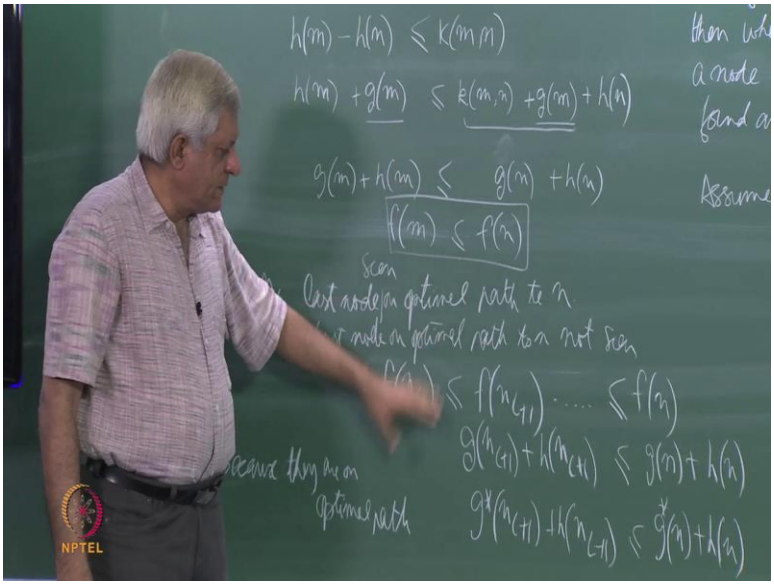
what does it do after it picks a node and it takes for, whether it is a goal or not; if it is not a goal, it generates its children, and the children are of three kinds, either new nodes or nodes on open or nodes on closed. For nodes on whichever, is open and closed, it checks for, and found a better path. For nodes on closed, if it is found a better path, has propagated that improvement to its children, essentially. So, the application of this statement, if this statement were to be true, it means that it, the moment it puts a node into closed, it has already found a better path, best path to it; optimal path to it, which means you have to revise that path, so that, the third stage of that revision process, we can do away with, essentially.

You can be showed that if node is in closed, you already have the optimal cost to it, essentially, which of course saves a lot, in terms of a this propagation that you have to do, and in later classes, which is in next class, you will see that this allows us to design some algorithm, which can save on spaces, but that we will do in the next class. Now, let us try to prove this property, essentially. What are we saying? We are saying that when it picks node n from the open list, at that point, g of n equal to g star of n; this is the optimal value. We can rewrite this condition as follows; h of n is less than equal to, plus h of n; I will take just h of n to that side and then, I can add g values; plus g of m. So, I can add this g of m to both sides; same value I am adding to both sides, so that does not change things. But this one, g of m, if you look at the figure; g of m and k of m equal to g of n. Let me rewrite this in this order. In other words, f of m is equal to f of n; that is a first observation we make.

What are these values saying here? When we are looking at this criterion, we are saying; is that the farther we have from the goal; the less accurate our estimate is, and the closer we go towards the goal, the more accurate our estimate becomes. As we move from node m to node n, our f value actually increases, essentially. Now, remember the f value is an estimate of the cost of the path, going through the node. Since, the path is going through both f, and both m and n, the estimate should be ideally the same. But when you see from m's perspective, it is actually less and when you see it from n's perspective. What is the difference; that you have n is little bit closer to the goal. So, that is one of the reasons for, this is called a monotone criteria. A monotone condition is that, as you move closer to the goal, the f value monotonically increases. Since, this is, this goal, as go closer to the goal, the f value monotonically increases. That is one observation, which holds for any, by transitivity, this will hold for any two nodes on a path from

to the goal, essentially. Now, let us assume that there is a node n which, let us remove this, that depicts a search space that we are exploring. There is node n which A star is about to pick, which means, it must be on open; this is open at that stage of time, or that stage of the algorithm, and n is about to pick node n. We want to make this clean that if A star is about to pick that node n, it must have found that optimal path to n. So, we will do this proof by contradiction. We will say that assume, that when it picks this node n, it has not found optimal path. So, assume that g of n is greater than g star of n. At the point, where it is about to pick n and we will show that this leads to a contradiction.

(Refer Slide Time: 13:51)




Let this be the optimal path. Let this be the optimal path to the node n, and let us say that this is in closed; this is in closed. Let us say that A star has found some other path so, this is open. And let us say that A star is about to pick this node n, and let us assume that at point, g of n is more than the optimal cost, and let n l be the; so, this optimal path to n, let us say it is this, and n l is the last node, A star has inspected on that optimal path, and n l plus one is the node which is on open, which is child of this node. But A star has not expected. So, what we are saying that A star has not found this path, which is the optimal path, but A star has found this path, which may not be an optimal path; we are trying to say. So, n of l, last node on optimal path to n, last node seen, and n of l plus one is the first node on optimal path, which is not seen. So, I am saying that this is

the optimal path. This of course, is one edge and then, they added some more; that is the optimal path to n.

We are assuming A star has found some other path to n, which is not the optimal path. Now, we can apply this criteria here, f of m, sorry. This monotone criteria that we observed, assuming that heuristic function satisfies this property from any path to the goal, this property holds that, as you go closer towards the goal, the f value increases. As we move from f of this last node n l to n l plus one, and so on to n, the f value will increase, or at least not decrease. In particular, we can write that g of n l plus one, plus h of n l plus one, is less than equal to g of n plus h of n. I am just expanding this one here, and that one there, and I am skipping the transitive steps, and because they are in optimal path, we can replace g with g star. So, we can write g star of n l plus one plus h of n l plus one less than equal to; so, we can put this star, because we are assuming that these nodes, these three nodes that we are talking about. These two nodes, n l and n l plus one, are on the optimal paths to n. So, on this path, these properties will hold, essentially. Now, if this is the case we can write this as, I am just adding plus h of n to both sides, essentially. So, g of n plus h of n greater than g star of n plus h of n. Anyway, that is not so important. What is really important is that f of n is less than equal to f of n l plus 1, why because we are said that A star is about to expand this node n. This whole property, we want to say that when is about to expand n, it must have found an optimal path to that, essentially. So, this we can write as g of n plus h of n less than equal to g of n l plus one plus h of n l plus one. So, I want to look at this one, and this one, see I want basically, this g of n is less than equal to g star of n using these two, but I have a star sitting there.

So basically, I have to put this star here, because we have assumed that in this graph, this is optimal path. So, this g of n l plus 1 is equal to g star of l plus one; this is what I did here. I said I could replace this by the optimal path. I can do it here as well. Now, of course, this becomes the same, which becomes the same as this. So, this is less than this, and if you remove h of n, you get this is less than this. What is this saying? This saying that g of n less than the optimal cost, less than equal to the optimal cost, but that can only be the case, if g of n equal to g star of n. By definition, it cannot be less than optimal cost; it must be the optimal cost.

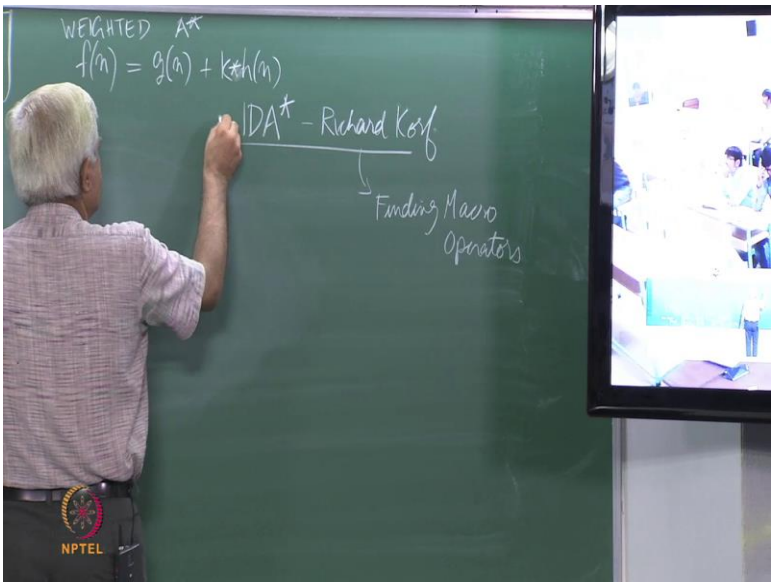
So, what we are said? There is a contradiction between this and this, and this and this. If you do not even assume this, you can simply show that g of n is less than equal to g star of n, which you

can interpret the same; g of n equal to g star of n, which is the statement we are making here. Whenever, it picks this node n, it has found optimal path to that node, which means g of n is equal to g star of n. The implication of this, as I said is that you do not have to revise the cost of those nodes, which have been put into closed. Once you put a node into closed, you know that you have the optimal cost, essentially. We have shown a serious of properties about this A star algorithm. The conditions are that the heuristic function must under estimate the cost to the goal h cost, must be more than some specified amount, some small amount, then A star will always find optimal paths to the goal node; that is what we showed in lemmas 1 to 6, I think, and this is actually in lemma 7; the last of the statements that you want to make. In addition of this condition is satisfied, then you do not have to keep revising cost to the nodes in close; the moment you pick a node from open and put in close, at that moment, you found the optimal cost to the node.

If you go, take this figure seriously, then essentially, what you are saying is that both these costs are initially of the same, this thing, whether this path or this path, it is the same cost; g value must equal to the optimal g value. So, let us compare this with the earlier algorithm we have seen. In terms of the four parameters, we talked about time complexity, space complexity, quality of solution and completeness. As far as the last two of these are concerned, the quality of solution and completeness, we have proved today, that A star will find a path to the goal, if there is a little, find an optimal path, always. That leaves the question of time and space complexity. Again, like we said in case of best for search, time and space complexity, both depend upon the quality of the heuristic function. The better the heuristic function is, lesser the amount of space, that your algorithm will explore, and that is what we said here.

Now, sometimes what people do is now, unfortunately, both space and time requirements have been observed to be large in nature, essentially. We will look at the examples where, space is cognatic in nature, for example, city, if you imagine a 2-dimensional city, then the farther you are go away, you can see the area that you have to explore grows as a square, essentially. But the combinations make low exponential, essentially, and we have seen this property, that the higher the heuristic value, the heuristic function that we are using, the better for you. But if you want to guarantee admissibility, the heuristic function must be less than the optimal heuristic value, essentially.

(Refer Slide Time: 27:09)



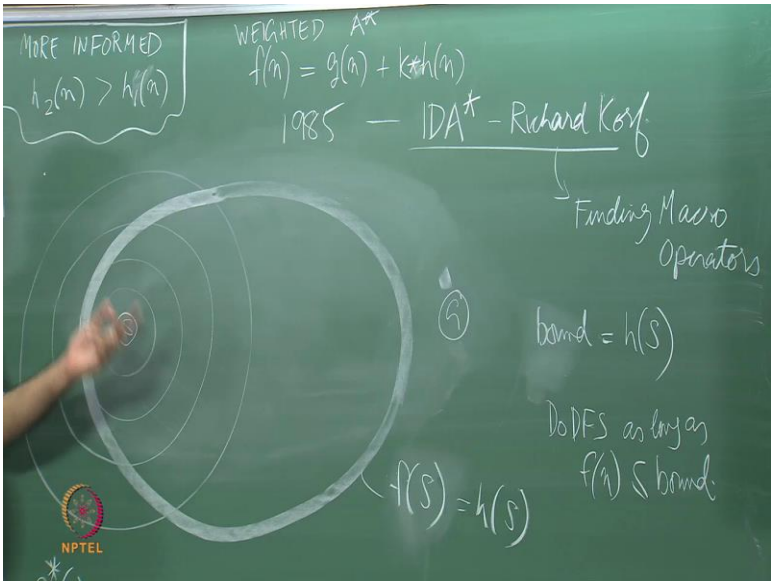

So, what people have often done is that, they have tend to use a function like this; f of n equal to g of n plus k times h of n; this is known as weighted A star, essentially. So, you use a parameter k to decide, how much influence the heuristic function has. Now, notice that there are two influences on A star algorithm; g of n is trying to keep it, because we always going to pick one with a lowest f value. So, low g means, it is close to the goal; low h means, it is close to the; sorry, low g means it is close to the source; low h means, it is close to the, at least, thinks it is close to the goal. The effect of g is that tries to keep the algorithm like branch and bound, as close to source as possible. The effect of h is like, best for search to pull it towards a goal, without any regard to what was the time spent in reaching that node n. If you use a parameter like k, you can actually control the effect of heuristic function versus g of n. If you put k equal to 1, then we have the A star algorithm to just describe, if you put k equal to 0, for example, then you have, simply, branch and bound. You do not even look at the effect of h. If you put k as very high, which I am also saying, that j is 0, there is like best for search. It only looks forward; it does not look behind. But if you put a value of k greater than 1, then you are giving more emphasis to the heuristic function, which means the search will become narrower and narrower, essentially. It will be given more by the heuristic function and less by the tendency to keep track of, whether you are finding optimal paths or not.

So, obviously, the moment you put k greater than 1, you are losing admissibility, but you can expect that your solution would be close to optimal solution, and depending on some trade off that you may have to make, you can choose the value of k higher than 1, essentially. Very often, people tried 4, 5, 6, values of k and seeing, that their algorithm runs much faster. This is simply to guarantee faster running time. So, what I want to do is to, now, focus on saving space. This is what people have done. After A star was discovered, it is quite a well-known algorithm. We want to look at how we can save on spaces. One of the first ideas that was explored, is an extension of an idea that we have seen earlier, which is; let us do depth first search. Instead of best first search, do depth first search. Now, if you remember the algorithm DFID, what DFID does is that it does the sequence of depth first search, with increasing that bound, but there, we have the motion of level, because we had no cost associated with edges. So, all edges were supposed to be of equal cost. Since, we have graduated to h cost and we looked that algorithm like, branch and bound, and so on; we need a variation of that and that variation is called IDA star. It was given by a guy called Richard Korf. I do not remember, whether we have discussed cost work earlier, but he has done a lot of work in search, essentially. His PHD thesis, which was in the earlier 80s, was finding macros, operators. In fact, his PHD thesis is also available as a book at some point of time. If you remember the puzzle like Rubics cube, in fact, he was working on Rubics cube and the eight puzzles. We have seen that it is very difficult to devise heuristic function, which will drive actually, climbing like, algorithms to solutions. So, what we tend to do is that we have the set of macro moves, which says, if you have done the top layer, then if you want to; let us say the next objective, which is to get; Let us say one cube, let into place in the middle layer; you say do this sequence of moves; left, right, left up down, whatever, we have some notation for that. That is the macro move.

A macro move is a sequence of moves, packaged into one abstract move. The question is; how do you get macro move? Of course, most of us learned from friends, at what cost rate for this PHD work was that he wrote an algorithm, which will search in the Rubics cube problem and the eight puzzles problem, and tried to learn macro move, essentially. His algorithm actually, built a macro table, which of course, once you have a macro table like, we saw all the Rubic cubes and we do not do any search; we say, ok, I will do the top layer first; then, I do the second layer and then, I will do the third layer. Korf's PHD thesis was to build that table, which listed all the macro moves, essentially, But this IDA stuff is also by Richard Korf, and we see a bit more of

him, as we go along. This was in 1985 also. And IDA stands, IDA star stands for iterative datening A stars. It still, A star in the sense, guarantee the optimal solution, but is iterative datening like, DFID essentially.

(Refer Slide Time: 33:44)

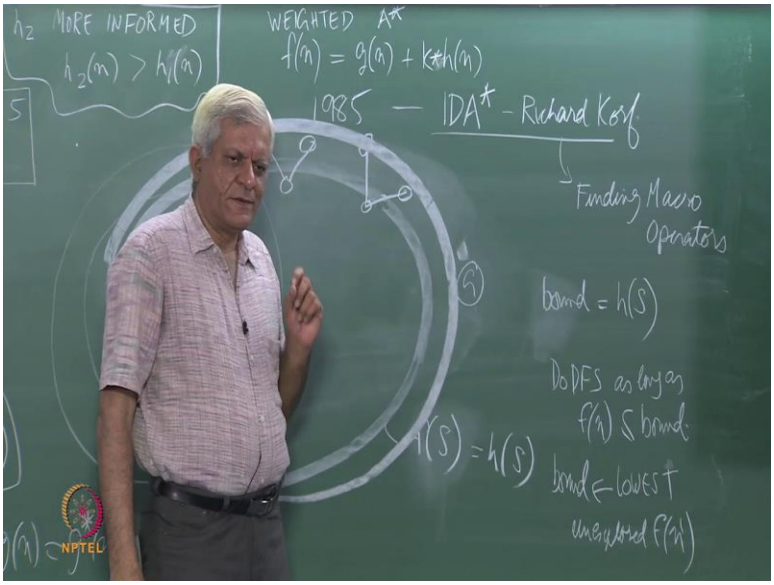



The algorithm idea is very simple. What it says is that if we were the start node, you create a boundary. Now, first, let us see, what DFID would do. Ignore, because it does not say cost of the edges; it basically, sees it as level by level. So, it would first do depth first search up to some level, then up to some next level, then up to next level; so we are assuming that edges are sort of roughly similar in cost in source.

Even if we have these edges of equal cost, IDA star has some value. So, they finally, would have basically expanded it search space gradually, and if this is the goal node here, it would have found eventually, at one of the expansions, it would have found a path to goal and since, it did not find it in previous round; it means, its new path must be the shortest path, because it is only in this last situation, that you extended this step by 1, but it uses linear space, because it does the first search, essentially. IDAs are basically variation of this. It says that you draw a boundary, which you use for controlling a search, and the boundary is essentially, those nodes. Well, it should not come here. It falls short of this, actually, something like this; just would depict this, essentially. This is the curve which is defined by f star of s. This is the locals of f star of s, which

means, all the nodes on this boundary have value equal to f; f value equal to optimal cost, essentially. Sorry, not f star of s; f of s. The estimated, because you do not know f star of s; you know f of s. That is the estimated cost from start to goal, as which is equal to h of s, because g of s is 0. So, these are the nodes whose f value is equal to that value h of s, and nodes inside will have lesser value, and the nodes outside have greater value. Now, one of the things that you should observe is, that is boundary is ellipse towards the goal and that is, because it is using the heuristic function, essentially. Nodes which are closer to the goal will have lower heuristic value, whereas, nodes which are away will higher heuristic value, essentially. So, what IDA star is, that in the first situation, you said that bound, well, depth is not the value.

(Refer Slide Time: 37:02)




Let us say a bound, its starts with saying, bound equal to h of s, and it does DFS, as long as h value is, f value is less than the bound. So, it is like a boundary it is drawn for edges, and it is saying that within this boundary, I will do the depth first search. So, let us say, it explores a node here; we have some children here; another node, it explores here, somewhere here, let us see. It may have some child here. Another node is here, which may have some child and so on. All of them will have their f values that you would have completed. If it does not find goal in this depth first search, now, this is depth first search with a bound; that it cannot go of in this direction, and it cannot go from this direction; only has save within this boundary. It augments this boundary by another boundary, which is little bit bigger than this; so, increment. For the lowest f value of a

node, which it is not picked, which means, it beyond the boundary, it increments a bound to that and then, puts a simpler loop, essentially.

So, you can see this similarity with DFID. The DFID, if you did not find a path to the goal, you would increment the level by one and then, try the DFS again. What this is doing is it increments the bound to the next. So, with all these nodes, which it has; this is explored; this is explored, but all these nodes, which is not explored, generated but not explored; it keeps track of a what is the lowest f value and increments the bound to that in the next round, and does another depth first search. It repeats its process till it finds a goal. What do you have to say about this algorithm? First, we should convince ourselves that it is admissible. It is worthy of that star on top of that name, that it will guarantee an optimal solution. Can you argue for that? Well, when it starts, it starts with a bound which is equal to h of s, and given that h of s is an underestimating function; it is not possible that a path of length greater than optimal path, will exist within h of s, because h of s is an under estimative function.

If the actual cost of going to the goal is h star of s, h of s is less than h star of s. So, this boundary will never take you to a node, which is more expensive than the optimal cost. If it does not find the value, if h of s was perfect, then within the first situation itself, it would have found a path to the goal. But if h of s is not perfect, it would just be short of goal little bit and then, you are going to increment the bound. Looking so, I have not written the word incremented, but it is the lowest unexplored, if you can read this word; this word must be unexplored. The node unexplored, f of n prime where, n prime are all these nodes, which have not been taken, essentially. So, because it is only incrementing the boundary to the lowest unseen f value, if it finds a goal of f that value, then it will be an optimal cost. Because that is only making very conservative increments; it is guaranteed to find the optimal cost. But you can imagine that in the search space like this, the number of iterations that, it will have to do is going to be very many, essentially.

So, it is going to be optimal; it is going to save on space; why? Because it is a depth first search; it is a sequence of depth first searches, which only requires linear space, but its time complexity is going to go up, by many times, because it will do many iterations, essentially. So, some variations that people have tried is that; instead of incrementing it by to the next lowest unseen value, increment it by a fixed cost, that you are willing to bear, essentially; some delta. So, instead of this, if I write bound, it means that I am making a bigger jump in this, essentially. That

means, let us say, this is my next value. So, let us say this is plus delta. So this was the original bound, and this was a bound plus delta. What is the danger here? The danger here is that there may be a goal node, just beyond this, but there may be a goal node, here. Because it is doing depth first search sweeping the space like this, it will find this goal node; but it will not find this goal node. This goal node is cheaper, I mean, if you just assume that this is kind of two scale, in some sense. This node is cheaper than that node, because it being depth first search, sweeping the space like this, it will find this node. But this, of course, this delta allows you to control; how much sub optimal you are willing to go essentially? So, this distance, this is delta that you have increased the boundary by. Basically, says that in the worst case, you may have optimal cost plus delta, but in this big jump, you would cover many nodes, so the number of iterations that you do would drop, essentially. So, it is nice in the sense, that it takes linear space, and it gives currently, optimal cost so, it is an extension of DFID, in that sense, which uses this fact that there are edge cost and there is heuristic function and things like that. What is not nice about this algorithm is what was not nice about the original algorithm; it is a blind algorithm, now essentially. Except of course, it retains a boundary that it draws, which is determined by the heuristic function, it is uninformed; it does not move towards a goal, which is why, we started with best first search in the first list. This is why, we used the heuristic function in the first list, but this is not exploiting the heuristic function; it uses the heuristic function only to determine this boundary.

In the next class, we will look at another variation, which exploits the heuristic function, and which is also linear space algorithm. That algorithm was also given by Richard Korf, may be, if you want to think about that little bit, essentially, it is a little bit like saying, that it is hill climbing with back tracking, if we can think of it along those lines. The algorithm is like hill climbing, but allowed to back track, essentially, and then, try another path; which means, it will not have exponential space; it will have only one path always in the main way. We will look at that algorithm in the next class and then, we will look at some more recent algorithms, which have come in this century, I should say, which are space saving algorithms, which are quite interesting, essentially.

So, I will stop here for today.