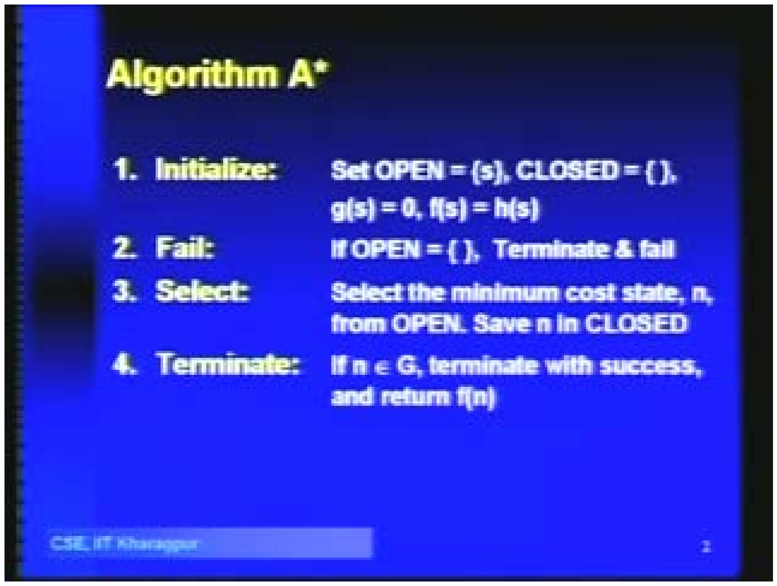
Artificial Intelligence

Prof. P. Dasgupta
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Heuristic Search: A* and Beyond
Lecture - 5

We will continue with our discussion on A* and heuristic search engine, from this class onwards. So, the topic of this lecture is heuristic search A*, and beyond. Quickly, to recap what we had done in the last class: we studied the algorithm A*, which maintains 2 lists- open and closed, and also 2 functions. One is the g value, which computes the distance of the state from the start state, and the h value, which is the heuristic estimate of the distance of that state from the goal state, and fs is the sum of gs and hs, and that gives us the estimated cost of a solution which goes through the node n. So, the first step was- if open is empty and we have still not yet found the goal, then we terminate with failure; otherwise, we select the minimum cost state n from open and save it in closed. If the selected state is a goal state, then we terminate with success and return the f value of that state as the cost of the goal.

(Refer Slide Time: 02:10)



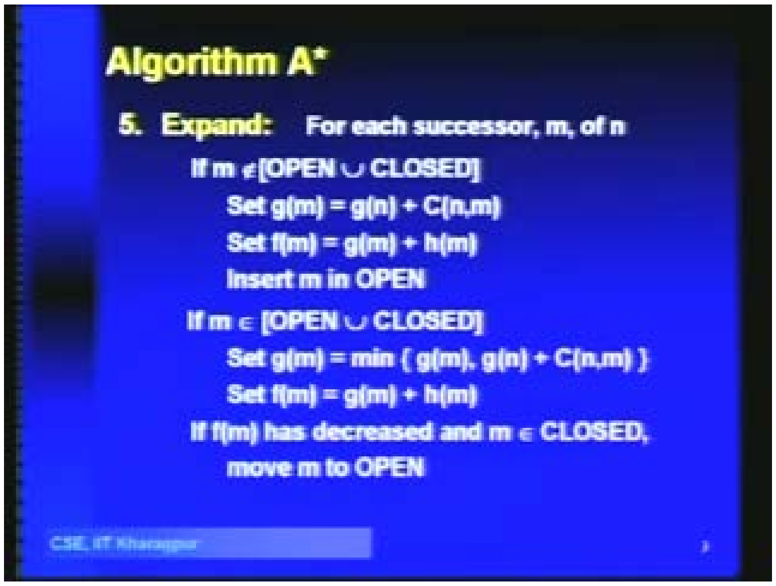
Algorithm A*

1. **Initialize:** Set OPEN = {s}, CLOSED = { }, g(s) = 0, f(s) = h(s)
2. **Fail:** If OPEN = { }, Terminate & fail
3. **Select:** Select the minimum cost state, n, from OPEN. Save n in CLOSED
4. **Terminate:** If $n \in G$, terminate with success, and return f(n)

CSE, IIT Kharagpur 2

Otherwise, we expand the node n, and to generate the set of successors and for each successor m, we compute its cost, based on the g value of that node and the h value of that node. And if the node already belongs to open and closed, we update it only if the cost is decreased. And, if the node is already in closed and its cost has decreased, then you must bring it back to open. Now, in uniform cost search, we had seen that if you have only positive cost, then you cannot have a case where a node comes back from closed to open.

(Refer Slide Time: 02:49)



Algorithm A*

```
5. Expand:  For each successor, m, of n
    If m ∉ [OPEN ∪ CLOSED]
        Set g(m) = g(n) + C(n,m)
        Set f(m) = g(m) + h(m)
        Insert m in OPEN
    If m ∈ [OPEN ∪ CLOSED]
        Set g(m) = min { g(m), g(n) + C(n,m) }
        Set f(m) = g(m) + h(m)
    If f(m) has decreased and m ∈ CLOSED,
        move m to OPEN
```

CSE, IIT Kharagpur

Now, why are we so concerned about nodes coming back from closed to open? Because what we want is: that we want that when we expand a node, we expand it only once. If we can ensure that we expand nodes only once, then there is a set of states which any admissible algorithm must visit. As we had seen in the last class, that if you look at the set of states whose f value is less than the cost of the goal, then any admissible algorithm, any algorithm which guarantees to find the optimal solution, we will definitely have to expand those nodes, right?

So, the complexity will be linear in the number of expanded nodes, if we can allow that no nodes are re-expanded. If you can ensure that every node is expanded only once, then, we are assured that if s is the set of nodes which must be visited by any admissible algorithm in, then our complexity is linear in s. Let us see. So, this brings us to a kind of notion of optimality. We say that, okay, let s denote the set of states, such that it denotes the set of states n, such that fn is less than C*, where C* is the cost of the optimal goal. We know that any algorithm which is admissible, or any algorithm which guarantees to give us the optimal solution- we will have to expand this set of states, right? If we can have our algorithm which is linear in the size of s, then that is something which is asymptotically optimal, right? Because any algorithm which is admissible and guarantees the optimal solution will have to expand that at least s nodes.

If you are linear in s, then, that means that we have an asymptotically optimal algorithm, but if we end up re-expanding those nodes, then this guarantee disappears. Then it could be quadratic in s. It could be exponential in s. We do not know, right? What we have seen is that, in uniform cost search, there is no such re-expansion, because a nodes never comes back from close to open. Once we have expanded it and put the node in closed, it is done. We do not re-expand that anymore, right, but in the case of heuristic search, we can have scenarios where a state is re-expanded, and this is there because of what we call non-monotonicity of the heuristic function.

I will give you an example to show where a node will come back from closed to open. So, the example that I will show is like this: that we have the state one, and let us say that the costs are 3, 2... This is the graph. Now, let us say that the heuristic functions are as follows: at every state, the heuristic function will give us an estimate of the cost to the goal. We will assume that the heuristic function h that is given to us, always under estimates, so it always gives us a lower bound, right? Now, so, but even if it underestimates, you can have an accurate heuristic and an inaccurate heuristic.

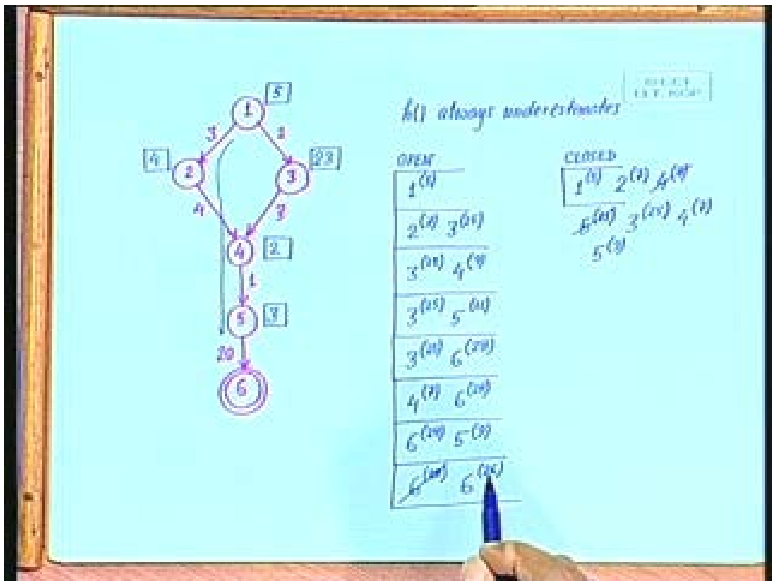
So, suppose that that this heuristic function gives us a pretty accurate estimate here. If you can see, the cost of this is 24. The optimal cost solution from 3 is 24, right? Where 6 is the goal and we have 23 here. Now, that is a pretty accurate estimate that we have here. Unfortunately, in the other states, s gives us a pretty weak estimate. So, let us say we have this kind of scenarios, right? Now, let us run A* on this and see what happens, and I will also maintain closed, to show that some nodes will come back from closed to open. Intuitively, what will happen is, we will- because of this heuristic, this node will have a pretty accurate cost value, and therefore, will not be expanded, until we have expanded this whole path. And then, we are near about here, we will realize that what we had all along this path was an inaccurate heuristic.

Then, we will again re-expand this path. So, let us see how we go about doing this.
Initially, we have 1 with a cost of 5 in open, right? The first step: we will expand that, and that is going to give us 2 states. One is 2 with a cost of 4 plus 3, 7, and 3 with a pretty accurate cost, which is 23 plus 2, 25. In the next step, you are going to put 2 with a cost of 7. So, 3 with a cost of 25 which is going to remain in open, and we are going to have here 4 with a cost of 4 plus 3, 7, plus 2. So 9, right? In the next step, we are going to pick up 4 with a cost of 9, and we are going to have 3 with a cost of 25 here, and 5 with a cost of 3 plus 7, and 8, 11, right?

In the next step, we are going to pick up 5 with a cost of 11 and then we have 3 with a cost of 25, and 6 is going to come with a cost of 20, 28, because this is the path; it is this path, which has a cost of 28. See, now we find that 3 has a lesser cost. See, at this point, though the goal is there, we cannot pick up the goal, because there can be better path to the goal. So, we cannot pick up and declare that we have found that solution. No. So, we will expand 3. When we expand 3, now, see what happens: we get 4, but now 4 comes with a cost of 5 plus 2; 7, right? So, the cost of 4 has decreased. We will take it out from closed and bring it back to open with a cost of 7, and then we have 6 with a cost of 28. Next step: 4 with a cost of 7 is picked up, right?

So, we have 6 with a cost of 28, and 5 now comes in, with a cost of, how much? 5, 6, 7, 8, 9, right? And this is better than the cost of that with which it was in closed. We have brought it back from closed to open, right? Now, we pick up 5 with a cost of 9, and that gives us 6 with a cost of 26. So, this gets replaced with 6 with a cost of 26, and we pick up 6 finally and find out that the optimal cost is 26, right?

(Refer Slide Time: 12:30)

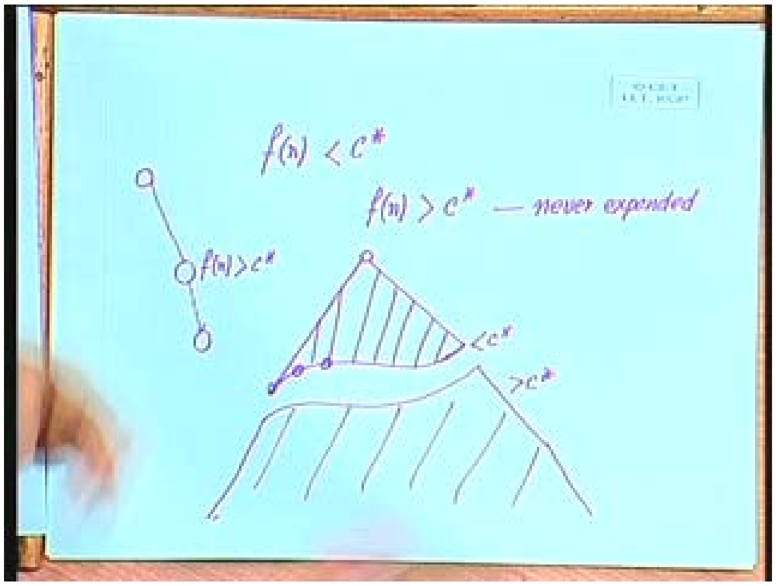


So, we see that we had to do a lot extra work, because our heuristic values were such that along one path it was well informed; along the optimal path it was well informed, and along the sub-optimal paths, it was not well informed. So, we had to expand the sub-optimal path up to a large extent, right? Now, there are several cases where we can make some improvements based on this notion. We are going to come back to that, but before that, let me let us study a few properties of A*. Firstly, we will say that a heuristic is called admissible if it always under estimates.

That is, we always have hn is in lower bound on the actual cost of the goal, from that state where A* denotes the minimum distance to a goal state from the state n. This is what we have been talking about all the time. The heuristic is called admissible if it underestimates. What happens if it overestimates? See, if it overestimates, then we do not have that nice property. We had talked about the property, that whenever you have a node whose fn is less than C*, then this is the set of states which are expanded, and all states which have the value fn greater than C*- these are never expanded, right? So, that means, if you look at the state space tree, then all the states which have cost less than C*, that is, all the states in this region, they are going to be expanded. And the ones which are greater than C*, this whole region, these states are never expanded.

These are all greater than *, and we never expand this. So, this gives us a nice bound; depending on the accuracy of the functions that you have, we have this nice bound, which tells us that these are the set of states that are that have to be expanded anyway, right? Now, the moment you do not have underestimating heuristics, what can happen is, along some path, you have the heuristic value overestimating. At this state, you will have fn greater than C*, but because the heuristic has overestimated, it is still possible, that below this, somewhere, you have you have a goal which has cost less than C*. That possibility cannot be ruled out. In this case, when we have underestimating heuristics, you know when the cost has exceeded C*, then the actual cost can only be more. If you look at the states in this frontier, you know that the actual cost of these states can only be more.

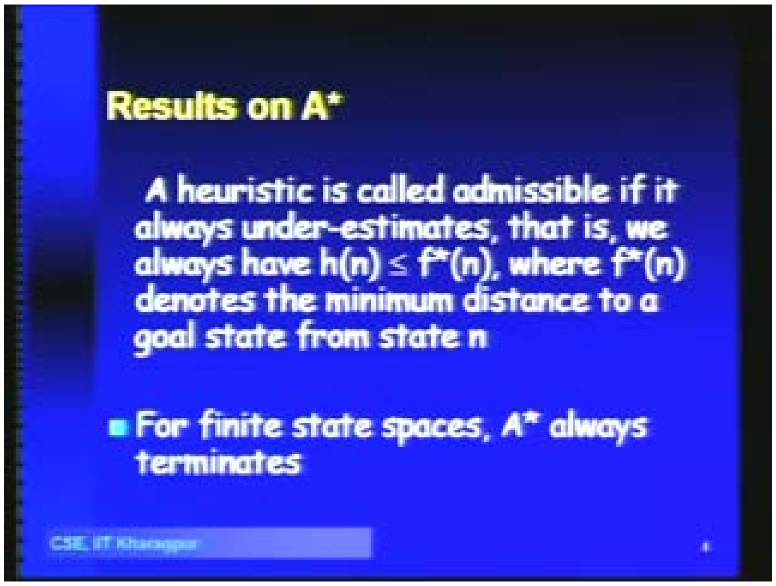
(Refer Slide Time: 16:13)



So, any goal that you find beyond this, will also have cost greater than C*. So, there is no point in expanding those nodes, but the moment the heuristics overestimate, it is quite possible that there is another goal beyond this, which has cost less than C*. And we have this f value greater than C*, because our heuristic has given an overestimate, right? As soon as we have overestimating heuristics, this nice property will disappear, and then, that will mean that even if you find that the cost of the node, even if you find that you have found a goal, you still do not know that the nodes that are there in open. If you expand them, you can still find the lesser cost goal, right? Because cost can decrease beyond that point.

That is why, though this property will not be there, but there are some advantages also, of using overestimating heuristics, which I will just briefly touch upon later. And when you actually do the experimentations, you will find that there is some merit in using overestimating heuristics. So, as of now, we are assuming that the heuristics are admissible, that is, they are underestimating heuristics. So, for finite state spaces, A* always terminates.

(Refer Slide Time: 17:43)


Results on A*

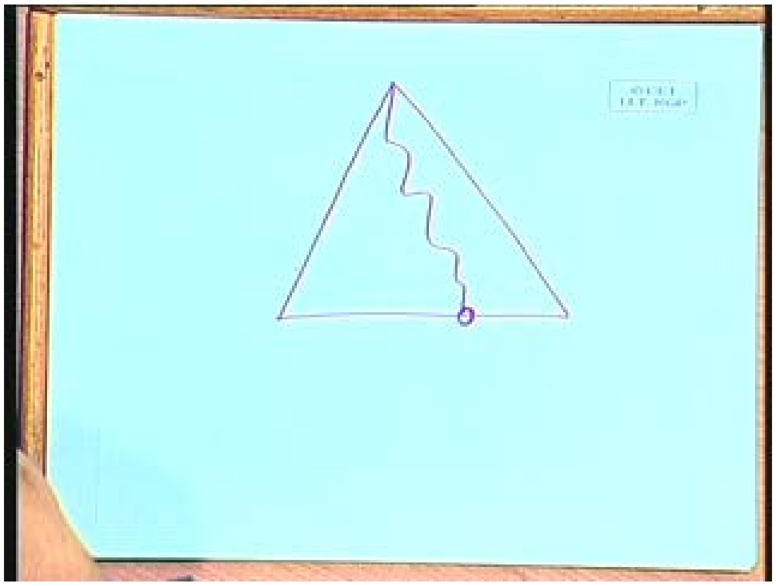
A heuristic is called admissible if it always under-estimates, that is, we always have $h(n) \leq f^*(n)$, where $f^*(n)$ denotes the minimum distance to a goal state from state n

- For finite state spaces, A* always terminates

CSE, IIT Kharagpur

We need not consider the proof of these; this is very straightforward, right? Because we are going depth first, based on the cost criterion. So, if you have a finite state space under finite cost, you will eventually reach the goal. At any Time: before A* terminates, there exists in open, a state n, that is known as optimal path from s to a goal state with fn less than f*s. Why is that the case? Suppose in the state space, let us say this is our goal, and let us say that this is the optimal cost path to the goal. Our objective is to find this, right? Now, until A* terminates, we are claiming that at least one of these states will always there be in open. (Student speaking). But, until we found this state, or the best path to this state, until A* terminated, there was always some state which was there in open.

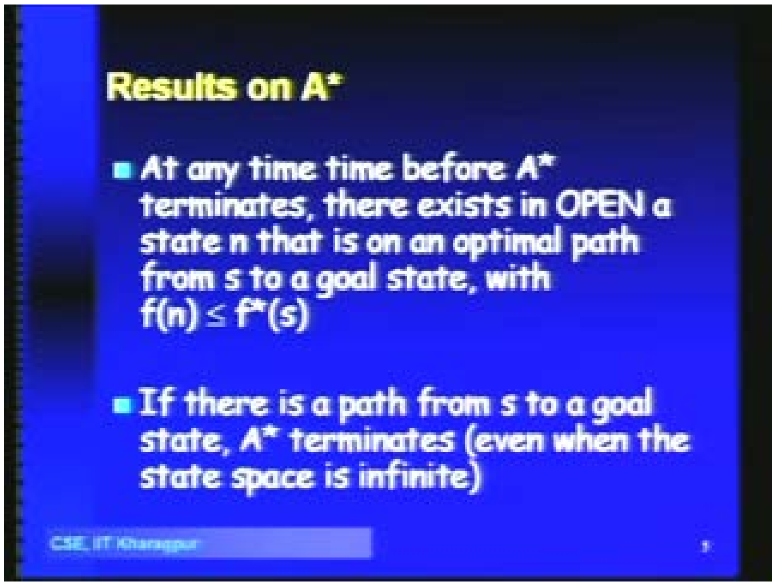
(Refer Slide Time: 19:46)



So, now let us look at this example once again, just to get this idea clear. See, the optimal path was here. 3 was that state which was there, right here, right? And then, when 3 got expanded, 4 was the state which was on the optimal path. Then 4 got expanded, then 5 was there on the optimal path, right? So, at every point, see, here- 1 and 3 and 5, and then we find the goal, right? Now, you can prove this by induction, where initially it is the start state which is on the optimal path, which is there in open, right? When you expand open, if, when you expand the start state, you are going to have a set of states that are the successor of the start state, right, one of these successors is on the optimal path. That fellow will be there in open. If you expand that state, if you as long as you do not expand that state, right, you will not terminate, because that state is always going to have a cost less than C*.

You will have to eventually expand it. As long as you do not, that state is the one which is there in open. When you expand that, then its successor will come into the- one of its successors on the optimal path will be there in open. So, in this way, it will continue, until you have expanded the goal or picked up the goal for expansion. There will be one of those states always there, right? (Student speaking). You design the heuristic in such a way where it is admissible, like, for example, if you look at the heuristic that- the Manhattan mode heuristic, for the 15 puzzle.

(Refer Slide Time: 21:54)



Results on A*

- At any time time before A* terminates, there exists in OPEN a state n that is on an optimal path from s to a goal state, with $f(n) \leq f^*(s)$
- If there is a path from s to a goal state, A* terminates (even when the state space is infinite)

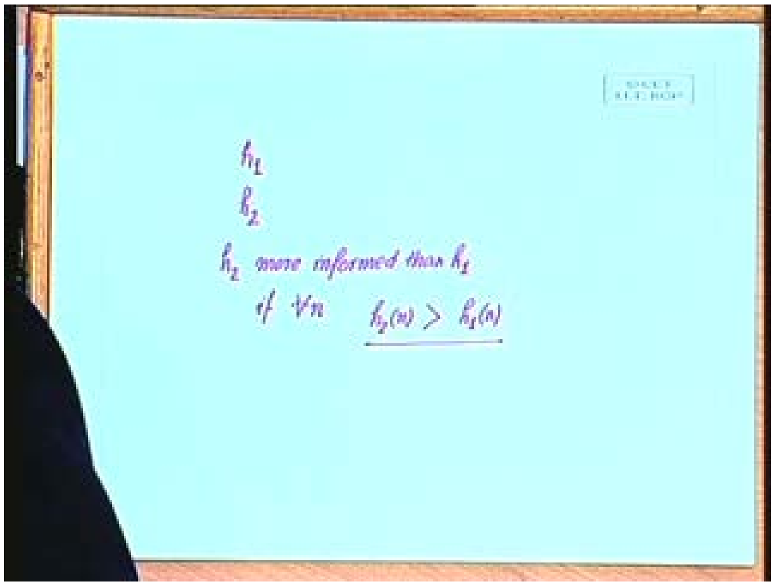
CSE, IIT Kharagpur 5

We said that at least that many moves will have to be taken for each tile. So, if you just add up the Manhattan distances of each of the tiles from their final position, then you get an underestimate. Likewise, when we talked about the minimum cost spanning tree heuristic for traveling salesperson problem, we know that the minimum the cost of the minimum spanning tree is an underestimate on the minimum cost tour. (Student speaking). Or, usually, you will be able to. If you do not have, see, the worst case scenario is, you do not know anything about the problem, in which case your heuristic function is zero everywhere. So, you boil down to uniform cost search. If you do not have a good underestimate but have a pretty tight overestimate, then it is better to go for depth first branch and bound. If you do not have a good overestimate and you have a good underestimate, then you can go for heuristic search, like A*.

The designing of the heuristics is, by itself, an interesting topic. And there are algorithms which will approximate, and will always keep the approximate solution to be of cost- you know you know that it is going to be it is going to be an underestimate of the actual solution, right? For example, there are approximation algorithms, which are going to guarantee you some k times optimal. So, if you find, apply that algorithm, and divide it by k- divide the solution cost by k- you know you have an underestimate. So, there are various ways of designing a heuristic which is an underestimate. When we work out some problems, you will see that, how to design such heuristics. Okay? Then, let us continue with this.

So, we say that algorithm A* is admissible. That is, if there is a path from s to a goal state, then A* terminates by finding an optimal path. Try to look at the proof of this result from the book of Nielson. Nielson's Principles of AI- proof is given there, right? Now, here is an interesting question. This is with respect to the accuracy of the heuristics. Suppose we have A1 and A2, and their 2 versions of A*, such that A2 uses a heuristic which is more informed than A1. What do we mean by more informed? Suppose the heuristics are all admissible. Suppose h1 is admissible. h1 is admissible; h2 is also admissible, right? So, they both underestimate. So, when should we call h2 more informed than h1? If for all states n, h2 greater than or less than? Greater than, right? If h2 is greater than h1, then this is more informed than this one, but remember that though h2 is greater than h1, at every state, it is still the case, that both are underestimates of the optimal solution. So, h2 is tighter than h1.

(Refer Slide Time: 26:33)



h_1

h_2

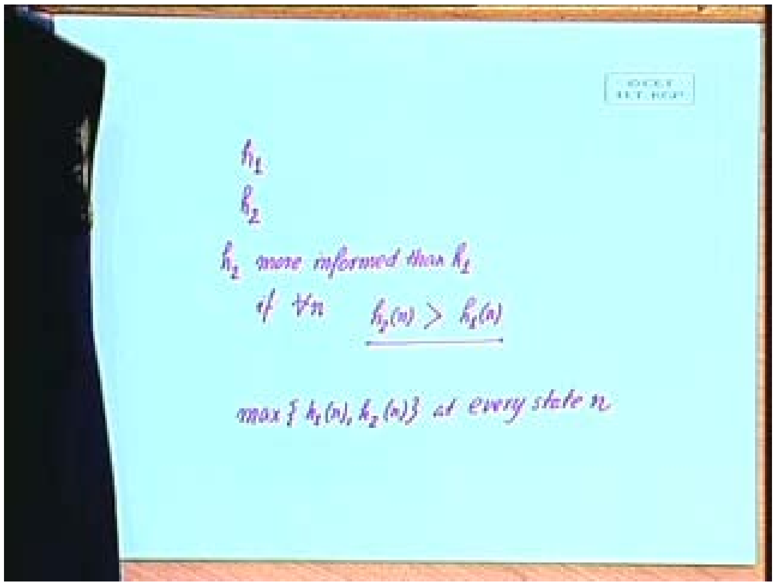
h_2 more informed than h_1

if $\forall n$ $\underline{h_2(n) > h_1(n)}$

So, what this result tells us- slides- is that, if A1 and A2 are 2 versions of A*, such that A2 is more informed than A1, then A1 expands at least as many states as A2. Now, that is very easy to see from that result which we saw, that fn less than C* is the set of states that will always have to expanded. So, the set of states which have fn less than C* will be more, when we use the less informed heuristics, than the one where we used in more informed heuristic. In the cases where we used the more informed heuristics, some of the states will now have costs greater than C*, and there will not be expanded, right? But if I have both of these heuristics, h1 and h2, and we do not know which is more informed than the other, then, what heuristics will we use for A*?

Suppose we have h1, we also have h2. Both, we know, are good heuristic functions. Both do not have much overhead of computation, and we want to run A* to the solve the problem. What heuristic function will we use? Average? Why average? Of the 2, yes.
So what if we have h1 and h2? Then, it makes sense to use max of h1n and h2n at every state, right? Because max of h1 and h2 is also going to be an underestimate and it is also going to be as informed as h1 and as informed as h2, perhaps better.

(Refer Slide Time: 28:49)



h_1

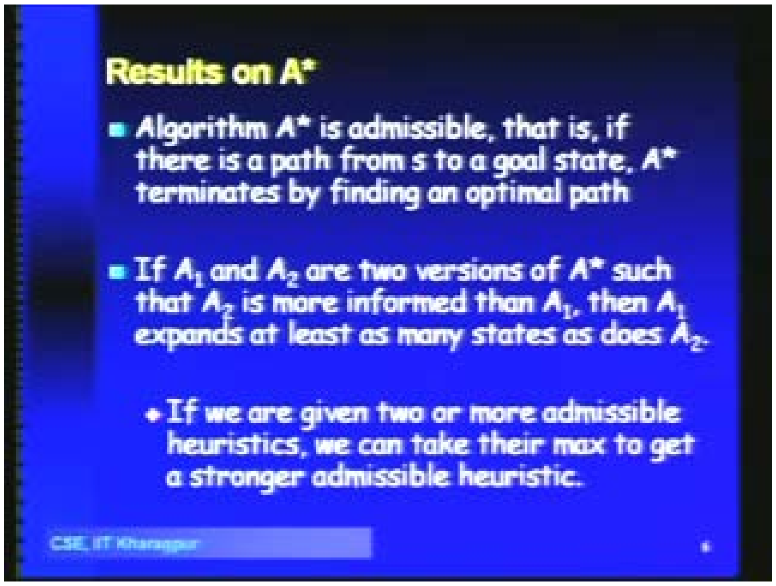
h_2

h_2 more informed than h_1

if $\forall n$ $\quad \underline{h_2(n) > h_1(n)}$

$\max\{h_1(n), h_2(n)\}$ at every state n

(Refer Slide Time: 29:01)

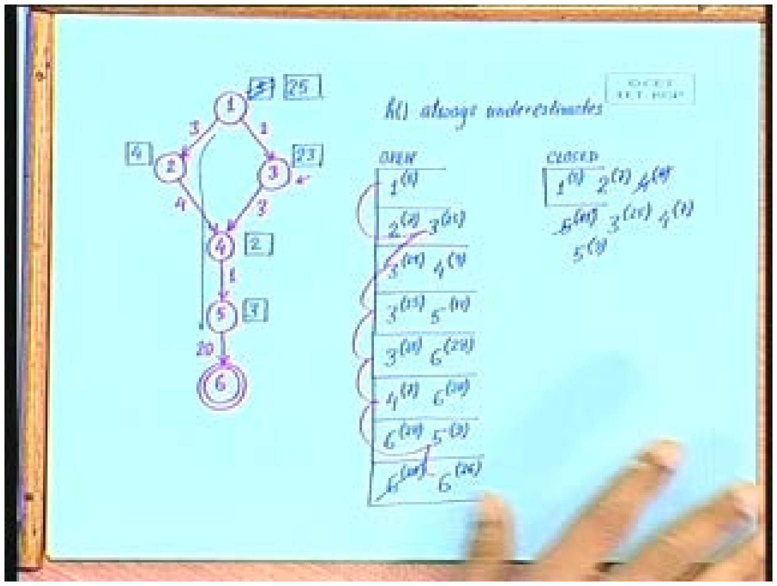

Results on A*

- Algorithm A* is admissible, that is, if there is a path from s to a goal state, A* terminates by finding an optimal path
- If A_1 and A_2 are two versions of A* such that A_2 is more informed than A_1, then A_1 expands at least as many states as does A_2.
    - If we are given two or more admissible heuristics, we can take their max to get a stronger admissible heuristic.

CSE, IIT Kharagpur

Yes. But in practice, we also have to look at the actual overhead of computing the heuristic. In some cases, see, there is a balance that one has to strike; in some problems, the state space is so bad, that unless you use a very good heuristic, you will run into trouble. In those cases, you do not mind spending more Time: in computing a better, accurate heuristic, and then using it. In other problems, wherein optimal goals are easy to find; in those problems, you may just go with a heuristic which is easy to compute, right? Now, we come to the issue of monotone heuristics. As we had seen previously, in that example, that one of the main problems that we had here, in the text, that we had some state in between, which had which was well informed, but the others were not well informed, right? Now, suppose, let us look at a slightly different scenario, where instead of 5 here, we have a more informed heuristic here, which gives us 25.

(Refer Slide Time: 30:40)



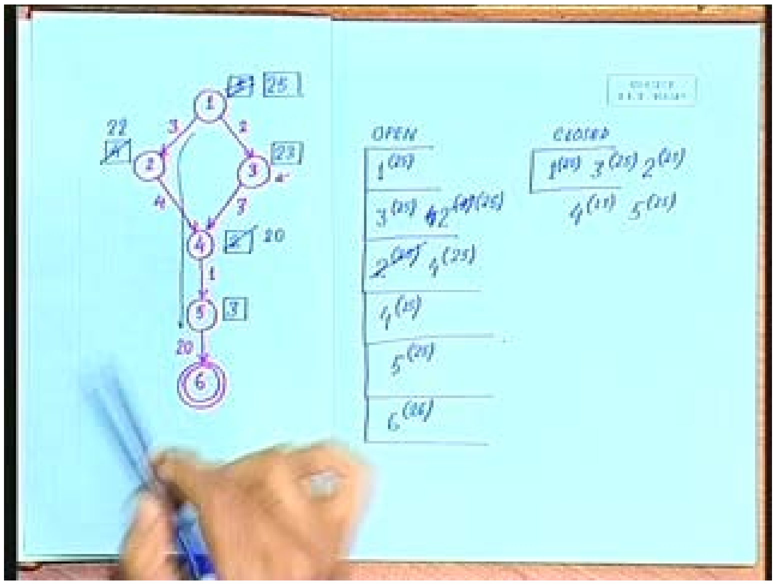
So, the only difference with what we had done previously is that, we now have 25 here, okay? Now, let us see what happens. We have open and we have closed. Now, when we have 1 out here, it is going to be there with a cost of 25. In the first step, we are going to take 1 and expand that. Now, I note one interesting thing. At what do we generate here? We have 3 with a cost of 25, as we had before, and now, if you just simply use for 4, if you just use the hn plus gm, then you will end up having again, as in as before, you will have 4 2 2s, rather, 2. Sorry, 2 with a cost of 7. Now, that is bad. Bad, because we know that from 1, any solution path is going to cost you 25, at least, and then this 2 as “ “ . From here, you can reach the goal in 7, with cost of 7.

Now, we clearly know that that is not possible. That is not possible, because we already know that from 1, we require 25. So, if you could do from 2 with a cost of 7, then from 1, you should be able to do with a cost of 10. But the heuristic, which is an underestimate, tells us that at least 25 is required. So, what we can do is, we can say that see, this is clearly an underestimate, which is inaccurate. So, we can safely upgrade this to 25. Why? Because we know that if it is 25 from here, right, and this edge cost 3, then it is at least 22 from here, right? So, we can safely replace this fellow with 22, and then the new f value- so 22 plus 3, which is 25.

Okay. Now, we have 2 states, one with a cost of 25, another also with a cost of 25. Now, recall, that when we have 2 states with the same cost, we choose the one which has lesser g value, because the other one has already incurred more cost. And the remaining part is in the heuristic. So, we will pick up 3 first, because it has less g value. So, we will expand 3 with a cost of 25, and then we will have 2 with a cost of 25 here, and 4 with a cost of how much? Again, we will use the same thing. See, if we just used 3 plus 2 plus 2, you know, that is again not accurate. So, 4 with a cost of 25. By upgrading this heuristic value to 20, right, so we upgrade this to 20.

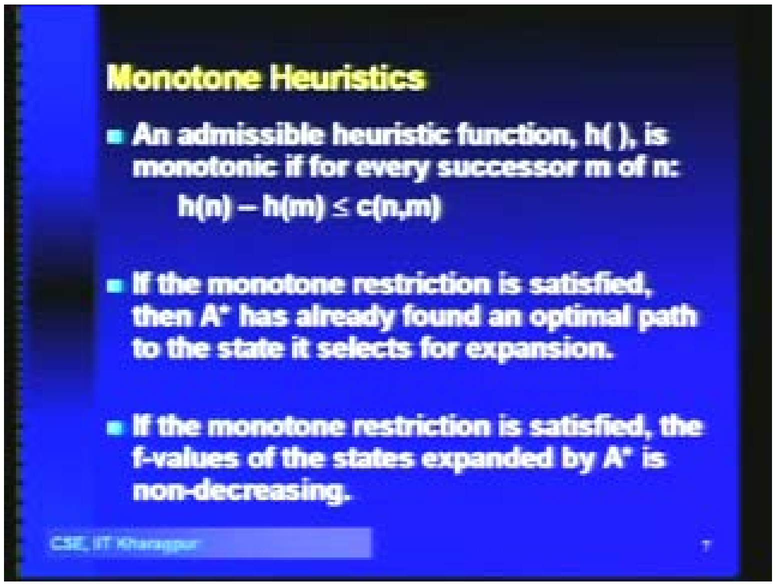
Now, 2 will be picked up, because it has lesser g value. So, pick up 2. 2 with a cost of 25 and what we have here is 4 with a cost of 25; and now, because of the expansion of this, we will again have 4 with a cost of 25, but, if we take this path, then it comes to 4 plus 3; 7 plus, so 27. So, 27 is larger than this. So, we will ignore that, and just stay with 4 with a cost of 25. And then, you can see that we will pick up 4 with a cost of 25. We will get 5 with a cost of 25, and then you will expand 5 with a cost of 25, and you will get 6 with a cost of 26, right? So, that was nice, because we have upgraded the heuristics nicely and avoided all those re-expansions, right?

(Refer Slide Time: 35:35)



So, this was proposed in, okay- so, we define a heuristic function to be monotonic, if we do not have this problem. That is, if we have a successor for every successor hn minus hm is less than or equal to cnm, right? If that happens, then it is monotonic, right, but clearly, that was not what was happening here, in our example. And if this if this does not hold, then we can update hn to have the value hn minus cnm, okay? Now, it so happens that if the monotone restriction is satisfied, then A* has already found an optimal path to the state it selects for expansion. If the monotone restriction is satisfied- because if the monotone restriction is satisfied, then along every path, you will update it with the least cost. We will just skip off these properties.

(Refer Slide Time: 37:16)



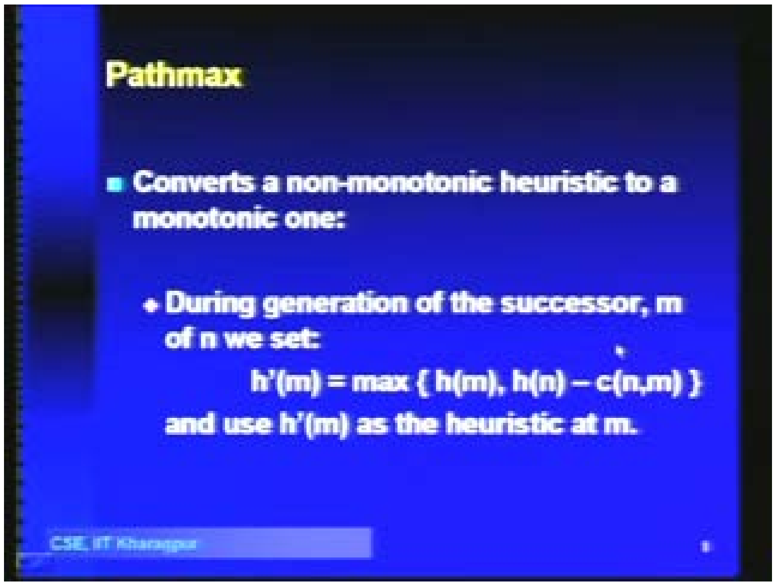
Monotone Heuristics

- An admissible heuristic function, h( ), is monotonic if for every successor m of n:
  $h(n) - h(m) \leq c(n,m)$
- If the monotone restriction is satisfied, then A* has already found an optimal path to the state it selects for expansion.
- If the monotone restriction is satisfied, the f-values of the states expanded by A* is non-decreasing.

CSE, IIT Kharagpur 7

You can just check out whether these properties hold, or just convince yourselves, by trying to prove them. Let us go over to what we call pathmax. Pathmax is what we just now discovered. What we do is, when we generate the successor m of n, we set the heuristic value to the maximum of hm and hn minus cnm. So, hm is the heuristic value that we computed that state, and because n is the parent, so, hn minus cnm is at least some amount- the amount of cost that will have to be incurred, because if you can reach the goal state with a cost less than this, then you are able to do better than hm, right? But because hm is an underestimate, you can actually never do better than hm.

(Refer Slide Time: 38:22)



Pathmax

- Converts a non-monotonic heuristic to a monotonic one:
  - During generation of the successor, m of n we set:
    $h'(m) = \max \{ h(m), h(n) - c(n,m) \}$
    and use h'(m) as the heuristic at m.

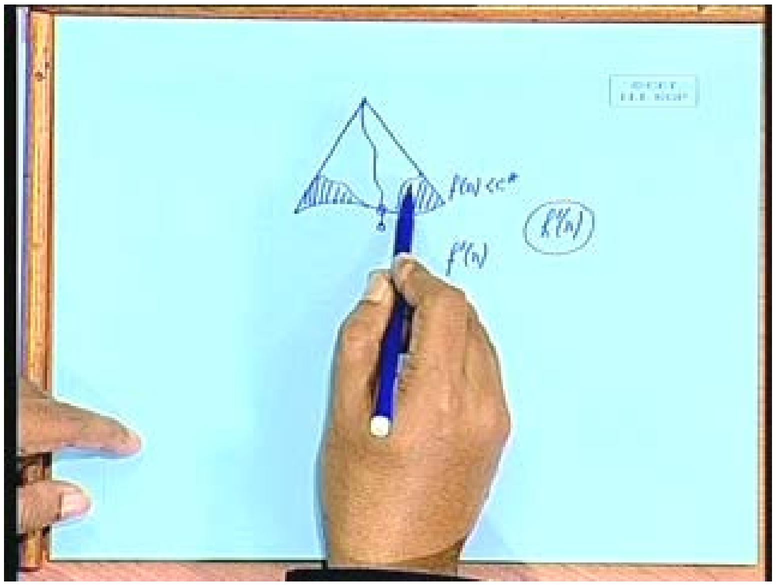
CSE, IIT Kharagpur 8

So, coming back to our example, you see that what we had done here was exactly that. We had hn as 25; hn was 25, hm was 4. So, we took max of 4 and 25 minus 3, that is 22, right? We took the max of these 2, and the max of these 2 was 22, which is what we updated the heuristic function to be, right? So, what we did here is exactly what I am explaining here in this. Slides, please? So, we update hm to be the maximum of hm and hn minus cnm, right? Okay. Let us go down further. Inadmissible heuristics. Now, see: we have seen that the more accurate that your heuristic is, the more number of states are pushed beyond the C* boundary. Right? In A*, if C* is the optimal cost, then all states having a cost less than C*, it must be expanded. So, the ones that are beyond C* will be not be expanded. If your heuristic is weak, then lot of states will be within C*, because the heuristic function weak, right?

If the heuristic function is zero at all points, then you have uniform cost search, in which case, all those states which have the actual cost less than C* will have to be expanded. If you have a good heuristics, then some of these states will now have cost greater than C*, and that is why we have the advantage of heuristic search. Those states are not visited because of the heuristic function. Now, if you have an overestimating heuristic, then many more of those states can again go outside the C* boundary, right? As long as those states are not on the paths to your optimal solutions, you stand to gain. Could I make myself clear vaguely clear? Right. So, here, if you use an underestimating heuristics, then I have, let us say, this set of states which with fn less than C*, okay? And the goal is one of these states, right, or just beyond this. So, this is on the optimal solution, right?

Now, if we use an overestimating heuristics, then the cost of these; sum of these states these states the cost of these states can be more, because the heuristic is overestimating. So, these costs can be larger than what they are now. Same states, but because the h value is more, so, h plus g is also more, right? So, it is quite possible that some of these states, let us say some of these states now, will have a cost greater than C*, when we use the new heuristic function and compute f dash n based on the heuristic function h dash n. For using these new heuristic function, which is an overestimating heuristic function, some of these states can jump over to the greater than C* region, right? Then, the set of states your A* will visit, has actually decreased, because it is now going to visit only this set of states.

(Refer Slide Time: 42:27)

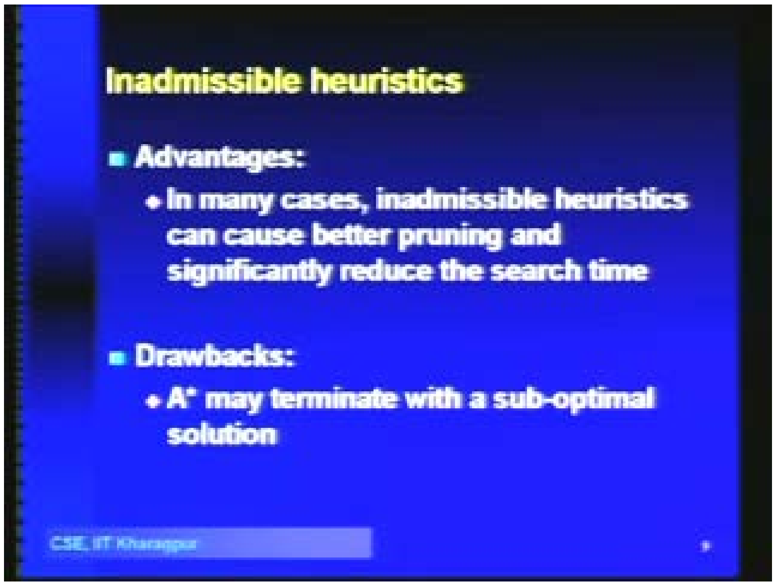


These are not going to be visited anymore. These are not going to be visited anymore, right? So, we do stand to gain? But, if it pushes these states also beyond C*, then again, the gain is not that much, right? There is a tradeoff that we want to do. Suppose we are not interested in getting the exact optimal solution; we are satisfied whether if it is closed optimal. So, what we will do is, we will make the heuristic overestimate. That is going to push several states, other states, beyond C*. So, the set of states that I will be expanding using A* will be, now, much fewer, right? If you have a sub-optimal goal within that set of states, then you are done, right? So, this balance is again used in many cases.

For example, in several in bioinformatics, there are now several problems, where we are using heuristic search technique, and it has been found, that in some of the more tougher kinds of problems, like multiple sequence alignment, we will discuss these problems in the tutorial. What happens is that, if you use overestimating heuristics, the performance improves by an order of magnitude, almost, so, by 10 times, 50 times, it improves in terms of time, and because there are lots of solutions and very close together, getting a sub-optimal solution quickly is easy, but if you want to improve that to get the optimal solution, you have to do a lot of work. But, in many cases, we do not require that amount of accuracy. (Student speaking). Okay. So, if you want to say that I want optimal- some k times optimal, where k is say 1.5, I am satisfied if it is 1.5 times optimal, within 1.5 times optimal.

Then, you can tune your inadmissible heuristics, so that it overestimates, by not more than 1.5 times, and then you use that heuristic. That guarantees that the solution is going to be “ “, right, and the drawback that we have for inadmissible heuristics is that A* may terminate with a sub-optimal solution. But if that is acceptable, then using inadmissible heuristics is a smart way of cutting down the search time.

(Refer Slide Time: 45:35)



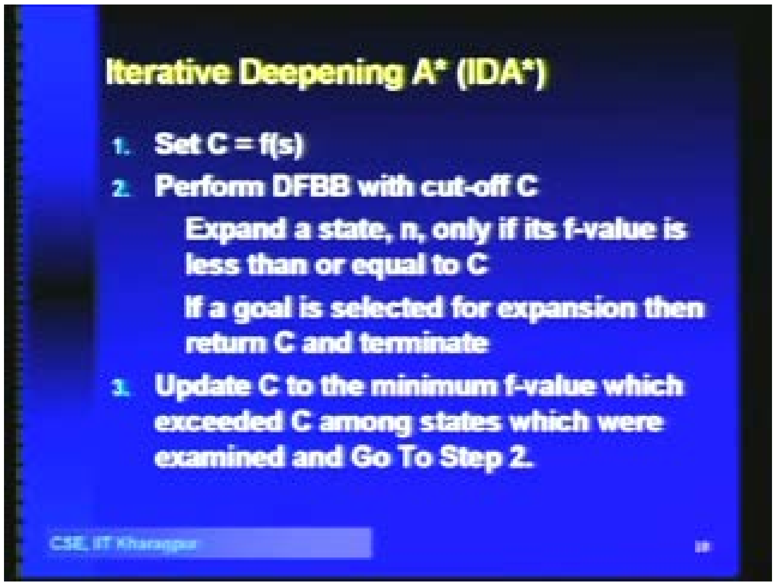
Inadmissible heuristics

- **Advantages:**
  - In many cases, inadmissible heuristics can cause better pruning and significantly reduce the search time
- **Drawbacks:**
  - A* may terminate with a sub-optimal solution

CSE, IIT Kharagpur 9

Just as we had iterative deepening, we do have iterative deepening A*, and the idea is similar- that we will use depth first search, right, but unlike depth first search without heuristics, what we are going to do here is: we are going to use depth first search using the heuristics. Initially, we will set c to be the cost of the start state. Then, we will perform depth first branch and bound with the cut off c. So, we are going to expand all states that have the f value less than or equal to c. If during this, you select a goal for expansion, then we return that and terminate. Otherwise, otherwise- we will update c to the minimum f value, which exceeded c among the states which were examined, and repeat the same.

(Refer Slide Time: 46:58)



Iterative Deepening A* (IDA*)

1. Set C = f(s)
2. Perform DFBB with cut-off C
   - Expand a state, n, only if its f-value is less than or equal to C
   - If a goal is selected for expansion then return C and terminate
3. Update C to the minimum f-value which exceeded C among states which were examined and Go To Step 2.

CSE, IIT Kharagpur 10

So, this picture will explain the thing nicely. We start with the state s, which has cost fs. We perform depth first branch and bound with this, so it could be the case, that s is the only state which has this cost, or it could be that s and several of its successors have the cost fs. So, we expand all the states, right? If we have found a goal anywhere here, then we terminate and return that goal. Otherwise, we look at this frontier. What is this frontier? This frontier is the set of states whose cost exceeded the current bound, so initially, the current bound was fs. So, this frontier is the set of states this is the set of states, where fn has exceeded c, right?

All the parents of these states had cost of c or less, right? That is why they were expanded. We did a depth first branch and bound, and we backtracked only when we found that we had reached a state whose cost is greater than c. We found that cost greater than c, backtrack, tried other path, again reached a state greater than c, backtrack, found other path, again reached a state cost greater than c. So, this frontier, from where we all backtrack during depth first branch and bound, consists of those states, whose cost exceeds c and whose parents' cost is less than or equal to c, right?

From this frontier, I pick up the minimum cost state, right? So, let us say that z is the minimum cost state in this frontier, right, and then update c to fz, okay? What is fz? f z is the minimum cost state, which exceeds a cost of c; the minimum cost state, which exceeds the cost of c, right? That is fz, right? If you think of A*, then once it had finished off all the predecessors of this frontier states, it would have picked up this state z from open, because open would have had this frontier states, right? And it would have picked up the minimum cost state from there. So, A* would have picked up this state only. Now, what we do is, we do a depth first branch and bound with z- with fz -as the cutoff. So, with this new c as the cutoff, what is going to happen?

In this frontier, this state is going to get expanded, so that frontier is going to get pushed a little bit forward. Now, the new frontier is these states and these states, right? Again, if you look at this frontier, pick up the one which has the minimum cost; let us say, that some state q here now, has the minimum cost. So, set c to fq, right? So, what are we doing? We are always expanding a new state, as long as we have already exhausted all states having less cost, right, and that is exactly what would have also have done. It will always expand a new state, provided all states with less cost in open has been expanded, right, and that is where the cost increases. When I have finished all states having a given cost in open, it is only then that I pick up the next largest cost- next larger cost- right? The advantage of doing this is that we do not store open or closed. We do not require open or closed.

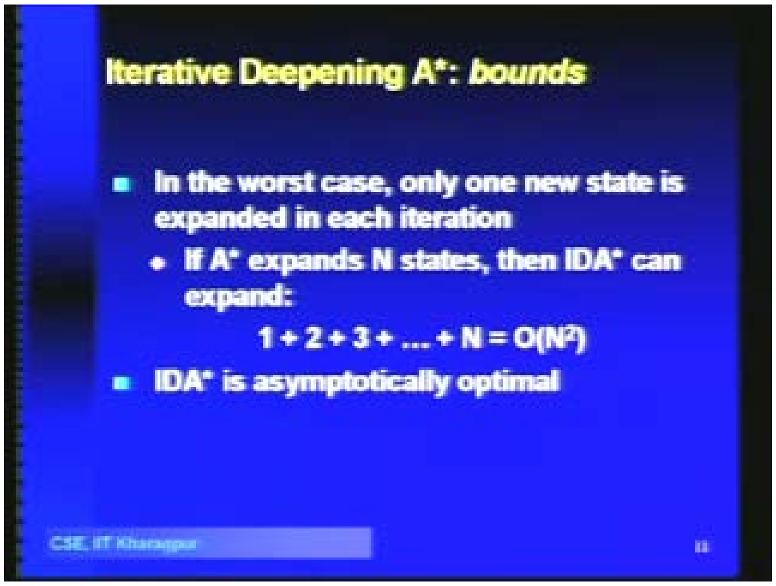
We are just doing depth first. So, space requirement is linear in the size of the paths that you are exploring, so we do not have the space blow up of A*, but at the same time, because we are doing iterative deepening, so we are not going purely depth first, in which case, we would have a missed a goal which was pretty close to the start state. So, we are going in a in A* style, but without saving open or closed, by progressively extending the frontier. And in practice, this works quite well. To see the kind of bounds that iterative deepening gives us, consider A*. It expands n states.

Then I d- A* can expand, 1 plus 2 plus 3 plus up to n or order n square. Why so? Because, if you look at the iterations, in every iteration, what may happen is, when you pick up a new state, it may happen, that in the next iteration only, that state is expanded. If all the state costs are- text please, text. Yes, in every iteration of, what may happen is, when you pick up the state from the frontier, this maybe the only state having this cost, in which case, it is the only state that is going to get expanded, and in the next iteration, you will visit all these; again, you will pick up one state and just expand that. And in the next iteration, you will visit all these, right?

So, in this case, what happens is, if you look at the A* sequence of states here, in the first iteration, you may expand one; second iteration, you are expanding 2 states; every iteration, only one new state is getting expanded; third iteration, 3 states are getting expanded, and so on, until you expand all n states. And that is the set of states which A* will expand. So, (Student speaking), because you are not saving them. Because you are not saving them, so, we are again doing a DSBB on the state space, with a new cost cutoff, right? So, it is all in the interest of saving space, because space is the thing which will kill you in this kind of state spaces. So, this gives you, in the worst case, as you can see, order of n square, where n is the set of states which A* expands. n is the set of states that I was mentioning; all states with cost less than C*.

This is going to it is going to be, in the worst case, quadratic in time, as compared to A*. So, the Time: increase is only quadratic, but the space is exponentially saved, because it can grow in order of b to the power of m, where b is the branching factor. But here. you are doing in linear space, right? So, it is asymptotically optimal.

(Refer Slide Time: 55:30)



Iterative Deepening A*: *bounds*

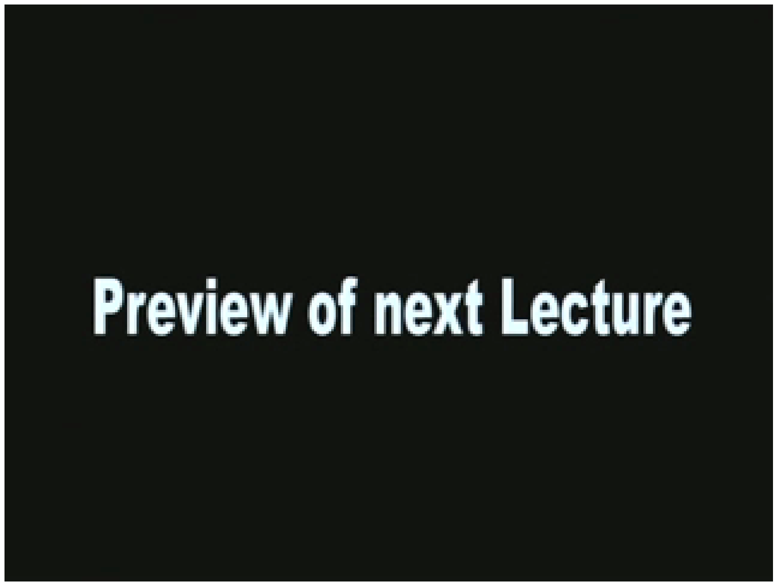
- In the worst case, only one new state is expanded in each iteration
  - If A* expands N states, then IDA* can expand:

    $1 + 2 + 3 + \ldots + N = O(N^2)$
- IDA* is asymptotically optimal

CSE, IIT Kharagpur

Okay. So, there are several extensions of these basic memory bounded search strategies. Maybe someTime: later, in some later lectures, I will just touch upon the other kinds of strategies that we have. But from the next lecture onwards, we are going to move into problem reduction search and game trees.

(Refer Slide Time: 56:01)

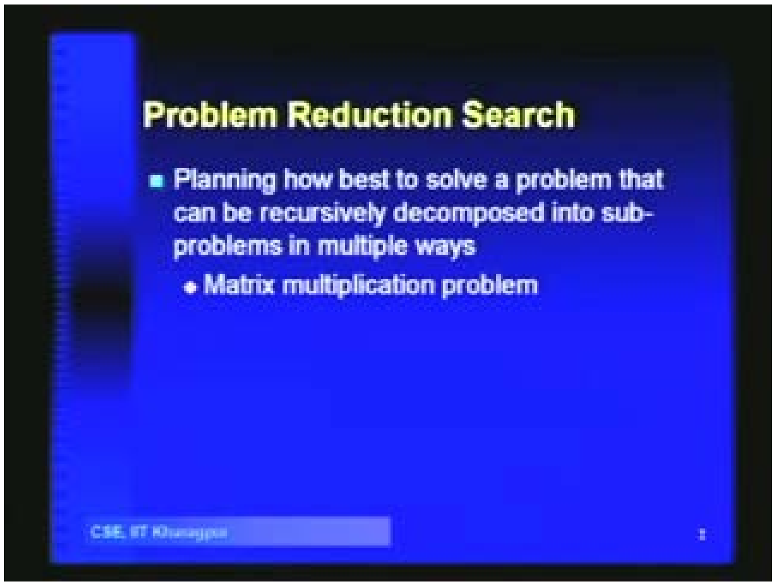


Preview of next Lecture

In the last couple of lectures, we were discussing state space search. And if you remember, in the initial introduction that I gave on search, we said that there are 3 different paradigms for problem solving using search, broadly. One was state space search, of which there are a few topics which I left. We will cover that up later. The second topic is problem reduction search, and under problem reduction search, we will look at 2 kinds of graph search, namely and/or graphs and game trees. And/or graphs are a kind of structure which we will study, and it has several different applications, though people do not always refer them as and/or graph search, but the underlying philosophy is the same. And then, we will look at game trees, which is the back bone of game playing programs and for programs which optimize in the presence of an adversary.

So, game search situations where you have adversaries, and there is some criterion, that you have to optimize. And in the presence of the adversary, you have to do that optimization. So, problem reduction search can be broadly defined as planning how best to solve a problem that can be recursively decomposed into sub-problems, in multiple ways. We can solve the same problem by decomposition. There are more than one decompositions of the same problem, and we have to decide which is the best way to decompose the problem, so that the total solution cost, quality of solution, or the effort of searching is minimized. So, to start with, let us consider the matrix multiplication problem, where you are given a set of matrices, A1to A_n, and we have to find out the product of them.

(Refer Slide Time: 58:01)



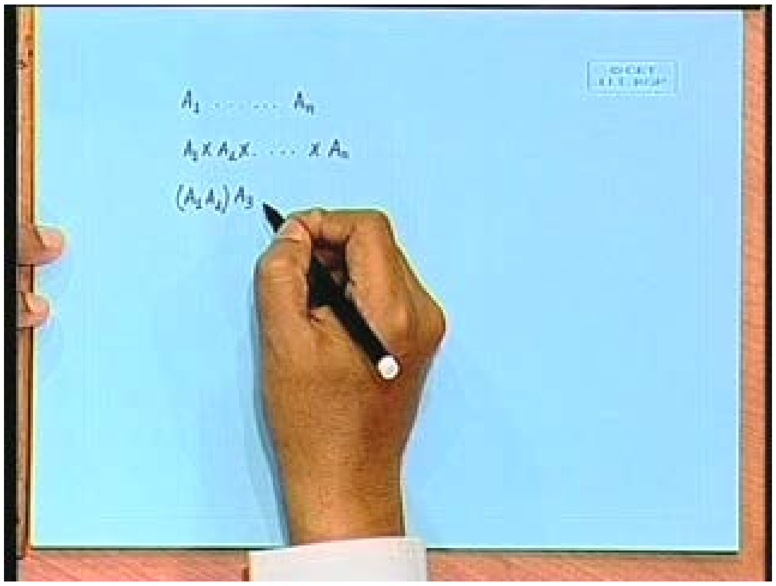
Problem Reduction Search

- Planning how best to solve a problem that can be recursively decomposed into sub-problems in multiple ways
  - Matrix multiplication problem

CSE, IIT Kharagpur

We have to find out A1, A2 to A_n, right? Now, we can do it in several ways. For example, we can first multiply: A1A2. With that product, we can multiply A3, right?

(Refer Slide Time: 58:37)



$A_1 \cdots\cdots A_n$

$A_1 \times A_2 \times \cdots \times A_n$

$(A_1 A_2) A_3$