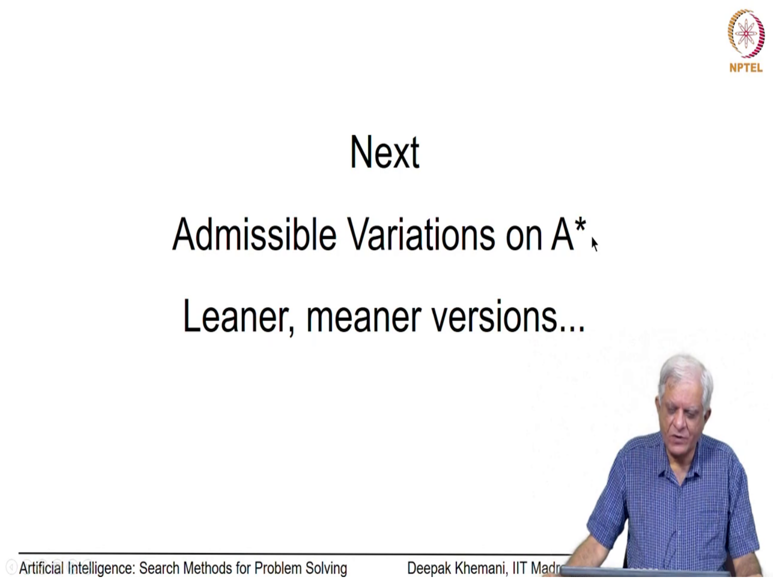
Artificial Intelligence: Search Methods for Problem Solving

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

Chapter – 05
A First Course in Artificial Intelligence
Lecture – 42
A*: Leaner Admissible Variations

(Refer Slide Time: 00:14)


Next

Admissible Variations on A*

Leaner, meaner versions...

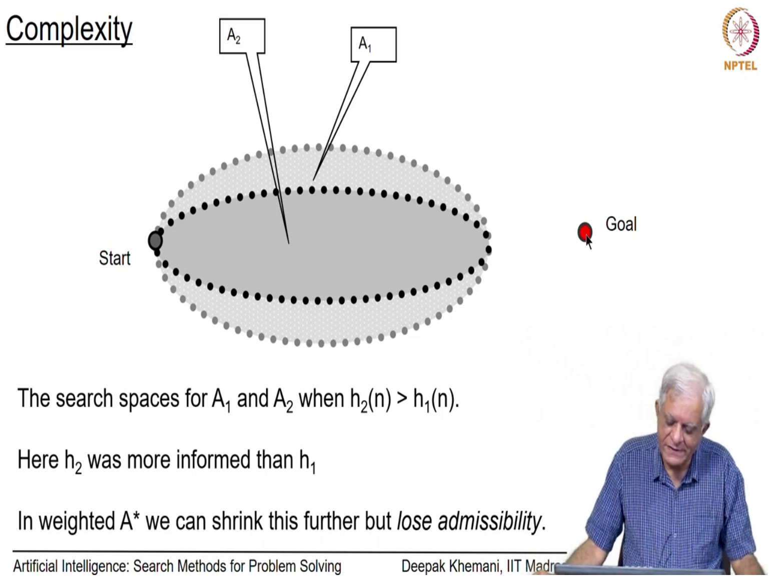
Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Madr

So, welcome back, we continue with our study of is A star and its variations. We have seen that we can view the whole thing in a generalized framework, where f of n is equal to g of n plus w into h of n and we saw some impact of the weight in the sense.

Now, let us focus on Admissible versions Variations of A star which are leaner in terms of space requirements. And, we will see later on that in this century and perhaps a little bit earlier; the need to explore very large graphs was presented by problems that came up from different areas of research essentially.

In particular, the task of sequence alignment from bioinformatics and we will look at that very briefly before we in the next session essentially, before we move on to other variations. So, but we first look at admissible variations which came in the last century and the goal you must remember is to cut down on the space complexity of the algorithm.

(Refer Slide Time: 01:28)



Complexity

The search spaces for A_1 and A_2 when $h_2(n) > h_1(n)$.

Here h_2 was more informed than h_1

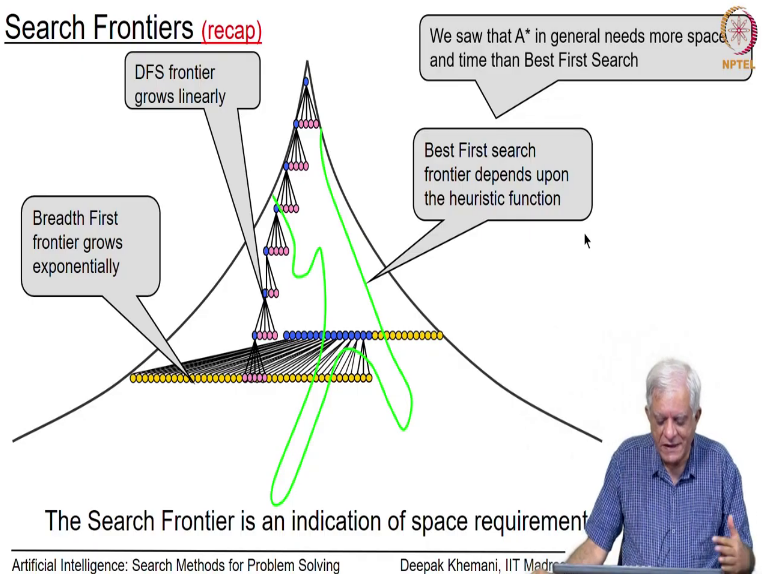
In weighted A* we can shrink this further but *lose admissibility*.

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

Now, this is something that we have already seen. We saw the two versions of A star: A 1 and A 2, where one heuristic function h 2 of n was more informed than h 1 of n. And, we saw that the space explored by A 2 is contained in the space explored by A 1, which means that it will not only finish faster, it will also use up less space.

Then in the last session, we saw that we can use weighted A star which can further shrink the space that is explored by the algorithm. But, we have the danger of losing admissibility as was illustrated in the example that we saw in the last session.

(Refer Slide Time: 02:22)



Search Frontiers (recap)

DFS frontier grows linearly

We saw that A* in general needs more space and time than Best First Search

Best First search frontier depends upon the heuristic function

Breadth First frontier grows exponentially

The Search Frontier is an indication of space requirement

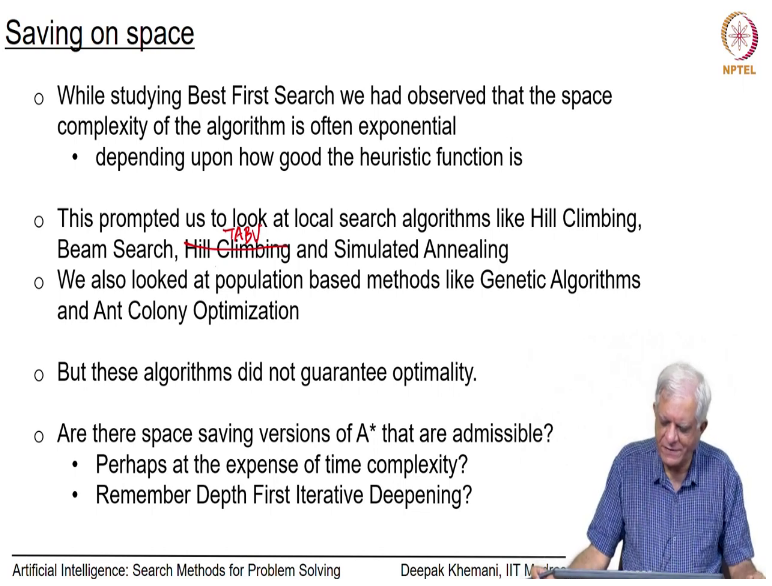
Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

Now, let us look at variations which are still admissible essentially. And we want to cut down on space, because empirically and this is something that we this is a graph that we saw when we were looking at best first search. We said that thus the open list for best first search also tends to be exponential in nature; even though it may not be as bad as that of breadth first search.

Breadth first is definitely exponential, breadth first was linear and best first we said that by and large tended to be exponentials. But, we also made the observation that its performance depends upon how good the heuristic function is. Then, we saw that A star in general needs more space than best first research.

And, we saw that in the last session when we compared A star with best first search and not only with best first search with branch and bound and nw A star. So, A star in general needs more space than best first search and best first search itself can tend to be exponential, if the heuristic function is not particularly good. So, let us look at variations of A star which require less space.

(Refer Slide Time: 03:37)



Saving on space

- While studying Best First Search we had observed that the space complexity of the algorithm is often exponential
  - depending upon how good the heuristic function is
- This prompted us to look at local search algorithms like Hill Climbing, Beam Search, ~~Hill Climbing~~ TABU and Simulated Annealing
- We also looked at population based methods like Genetic Algorithms and Ant Colony Optimization
- But these algorithms did not guarantee optimality.
- Are there space saving versions of A* that are admissible?
  - Perhaps at the expense of time complexity?
  - Remember Depth First Iterative Deepening?

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

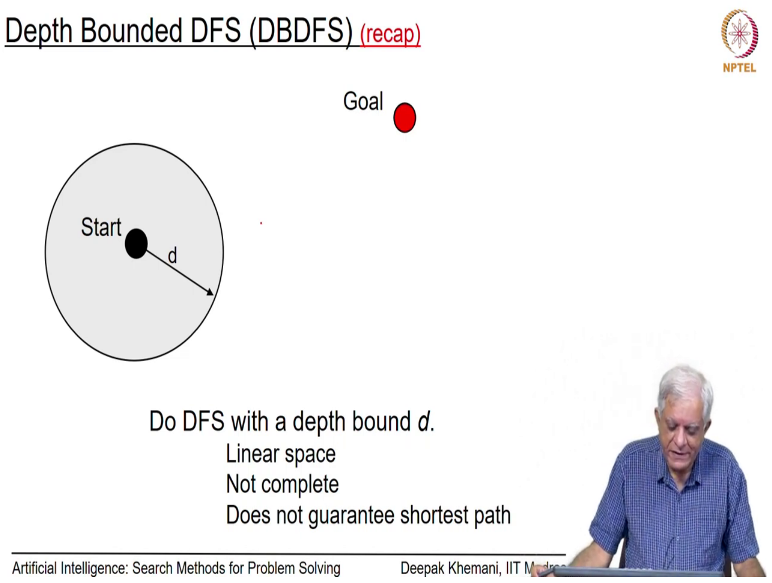
So, we want to save one space and this will be our focus for the next few sessions. So, this is what I just said that while studying best first search we have observed that the space complexity of the algorithm is often exponential. And, it really depends upon how good the heuristic function is essentially.

This led us when we were studying best first search to look at local search algorithms. We started with hill climbing then we went on to beam search and Tabu search and simulated annealing. So, this should be Tabu search and simulated annealing. And, later on we also moved on to population based methods like genetic algorithms and ant colony optimization.

But, we also saw that these algorithms do not guarantee optimality, though algorithms like genetic algorithm, simulated annealing ant colonial optimization often give you very good solutions. But, at the moment we are focusing on guaranteeing optimality. So, the question we want to ask is are there space saving versions of A star that are also admissible?

Perhaps at the expense of time complexity and if you remember, what we did when we studied depth first iterative deepening? We said that we can trade off space versus with time and get a algorithm which requires much less space. And, the first algorithm that we will see is in fact, an extension of DFID or Depth First Iterative Deepening.

(Refer Slide Time: 05:10)



Depth Bounded DFS (DBDFS) (recap)

Goal
Start
d

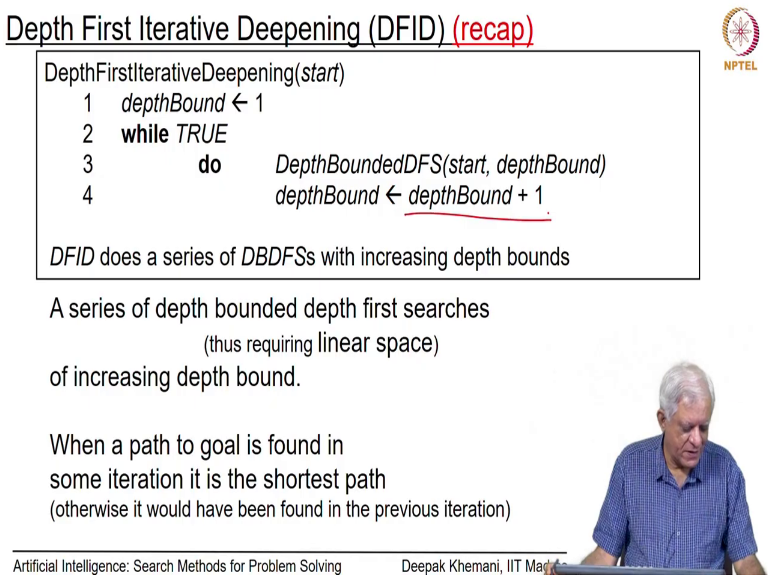
Do DFS with a depth bound *d*.
- Linear space
- Not complete
- Does not guarantee shortest path

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

So, just to recap, we started with doing depth first iteratively deepening by first writing an algorithm called Depth Bounded DFS. Depth bounded DFS simply says that there is a bound beyond which depth first search cannot go and you simply do depth first search within that bound. The boundary showed in this example in a circle which is shaded. So, you do depth first search only up to that bound, whatever the bound is d in this case and it requires linear space.

Why? Because, it is depth first search and we have seen that the open list for depth first search grows only linearly. But, it is not complete; now that as you can see if it is going to stay within this bound, it is not even going to find a path to the goal, forget about carrying taking a shortest path essentially. So, it is neither complete nor admissible which is a property of depth first search.

(Refer Slide Time: 06:12)



Depth First Iterative Deepening (DFID) (recap)

```
DepthFirstIterativeDeepening(start)
  1  depthBound ← 1
  2  while TRUE
  3      do   DepthBoundedDFS(start, depthBound)
  4           depthBound ← depthBound + 1
```

DFID does a series of *DBDFS*s with increasing depth bounds

A series of depth bounded depth first searches (thus requiring linear space) of increasing depth bound.

When a path to goal is found in some iteration it is the shortest path (otherwise it would have been found in the previous iteration)

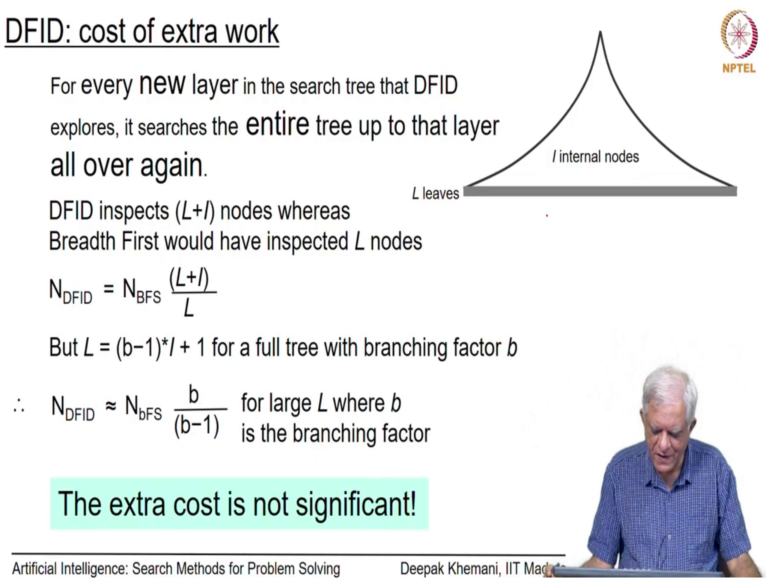
Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Mad

But, depth first iterative deepening essentially what it does is that, it does a sequence of depth first searches and in every iteration it increases the depth bound by 1 essentially. So, this is was this was depicted in this here that after you have done the depth DepthBoundedDFS up to a certain depth if you have not found the solution, increase the bound by 1 and then again do DFS.

So, it was a sequence of DFS searches of increasing bound. A sequence of depths bounded depth first search because each of these therefore, searches was bounded by depth. And therefore, they required linear space because they were essentially depth first search.

But when the path to the goal is found in some iteration, it is the shortest path because otherwise it would have found the path in the previous section; if there was a shorter path to a goal. So, in that sense the behaviour of DFID was like breadth first search, but internally it was like depth first search; of course, we had to do a sequence of searches.

(Refer Slide Time: 07:22)



DFID: cost of extra work

For every new layer in the search tree that DFID explores, it searches the entire tree up to that layer all over again.

DFID inspects (L+I) nodes whereas Breadth First would have inspected L nodes

$$N_{DFID} = N_{BFS} \frac{(L+I)}{L}$$

But $L = (b-1)*I + 1$ for a full tree with branching factor b

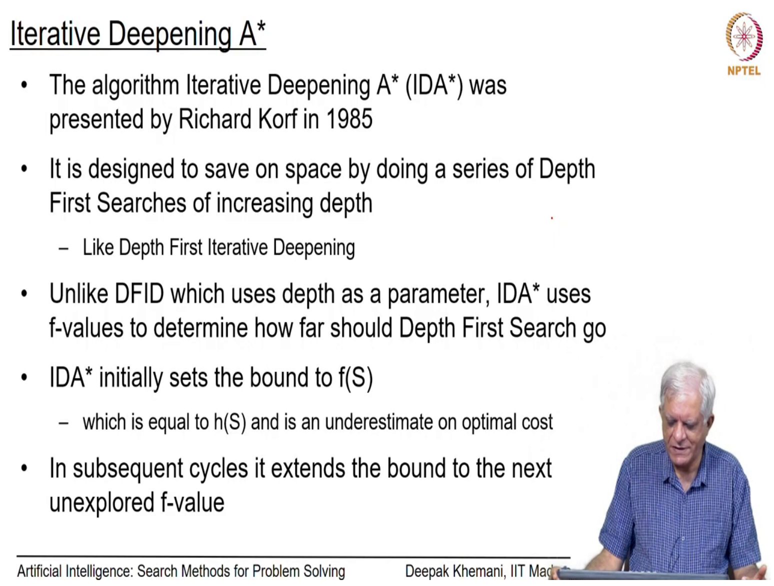
$$\therefore\ N_{DFID} \approx N_{bFS} \frac{b}{(b-1)}$$ for large L where b is the branching factor

The extra cost is not significant!

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

And, we had done a little bit of analysis about how much is this extra work that we do in DFID. And, we here come to the conclusion that for a space search space which grows exponentially, the extra cost was insignificant essentially. It was only b over b minus 1 times the value of what breadth first search would have done essentially.

(Refer Slide Time: 07:49)



Iterative Deepening A*

- The algorithm Iterative Deepening A* (IDA*) was presented by Richard Korf in 1985
- It is designed to save on space by doing a series of Depth First Searches of increasing depth
  - Like Depth First Iterative Deepening
- Unlike DFID which uses depth as a parameter, IDA* uses f-values to determine how far should Depth First Search go
- IDA* initially sets the bound to f(S)
  - which is equal to h(S) and is an underestimate on optimal cost
- In subsequent cycles it extends the bound to the next unexplored f-value

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

So, it is clearly a motivation for devising a new algorithm and this new algorithm is called iterative deepening A star. The algorithm was given to us by Richard Korf in 1985.

And, it is designed to save on space by doing a series of depth first searches of increasing depth exactly like DFID. Unlike DFID which uses depth of the load as a parameter. Remember that when we were studying the DFID and depth first search and best for search; we did not have edge cost.

So, our notion of optimality was the length of the path, but now edges have cost associated with them. So, length of the path may not be the right IDA as we have observed, that we have to have a different way of measuring the cost of the solution found. So, DFID was an extension of breadth first search or rather it was depth first search trying to behave like breadth first search.

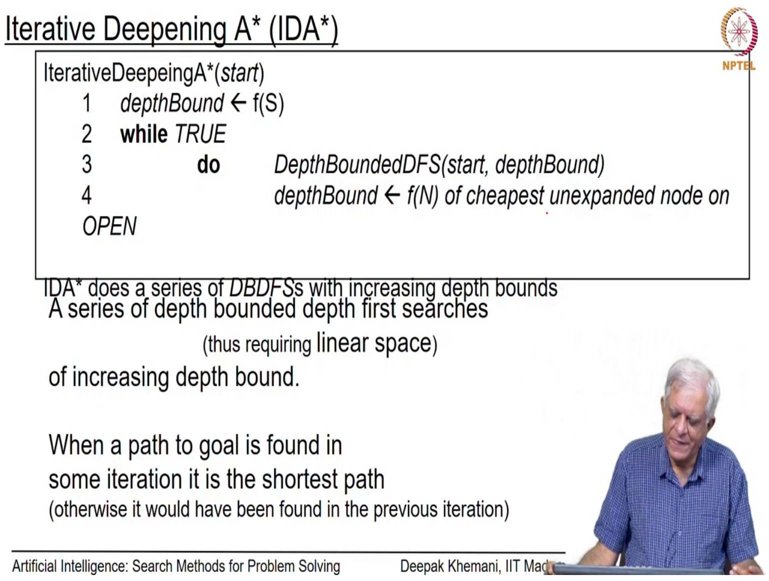
In a similar manner IDA A star is going to be depth first search trying to behave like A star essentially. And, what it will do is that it will use f-values to determine how far the depth first search should go. So, in all these variations DFID, IDA star; it is essentially depth first searches of increasing bound. It is just that how the bound is determined is different in the two cases.

IDA A star initially sets the bound to f of S which is equal to h of S, if you remember because g of S is 0. And, we know that h of S is an underestimating functions which underestimates the optimal cost and so, it will never find solution. Depth first search will never venture into a solution which is more expensive than the optimal solution.

Because, it starts off by looking at depth only where at best a solution can be found, at best h of S can be equal to h star of S. And if that happens will be the case, then it will find the solution. In subsequent cycles, it extends the bound to the next unexplored f-value.

Now, this is different from DFID, in DFID we just increase the depth by 1, but here we are extending the path length to the next node, that you have not explored (Refer Time: 10:11).

(Refer Slide Time: 10:15)



Iterative Deepening A* (IDA*)

```
IterativeDeepeingA*(start)
    1   depthBound ← f(S)
    2   while TRUE
    3           do      DepthBoundedDFS(start, depthBound)
    4                   depthBound ← f(N) of cheapest unexpanded node on
    OPEN
```

IDA* does a series of *DBDFS*s with increasing depth bounds

A series of depth bounded depth first searches
(thus requiring linear space)
of increasing depth bound.

When a path to goal is found in
some iteration it is the shortest path
(otherwise it would have been found in the previous iteration)

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

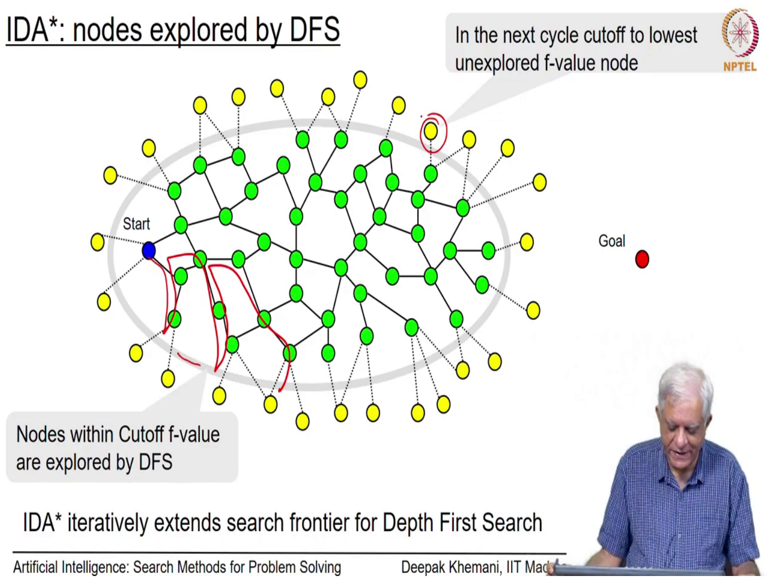
We will see that this may cause problems of its own, but let us first look at the algorithm itself. So, iterative deepening is A star has stars with a depth bound of f of S. And, exactly like in the previous case, it does depth first depth bounded depth first search. If the search fails to find a path to the goal, it increases depth bound. But, this time it increases it to the f value of the next cheapest node that was not expanded in the last round.

So, it does a sequence of with increasing depth bond, thus requiring linear space as before; everything else does not change. The argument which says that it will find an optimal path is the same as the argument that we gave in DFID. Because, when a path is found in some iterations it must be the shortest path; because otherwise it have been found in the previous iteration.

So, this is how DFID is a variation, IDA star is a variation of DFID. DF DFID was depth first trying to behave like breadth first because breadth first gave you the shortest path in terms of number of ops or the length of the path. In IDA star the same thing is happening increasing number of depth first searches.

But, now it is trying to behave like A star because, it will find us the shortest cost path; where the cost is the sum of the edges and some of the cost of the edges in the path that is found ok.

(Refer Slide Time: 12:00)


IDA*: nodes explored by DFS

In the next cycle cutoff to lowest unexplored f-value node

Nodes within Cutoff f-value are explored by DFS

IDA* iteratively extends search frontier for Depth First Search

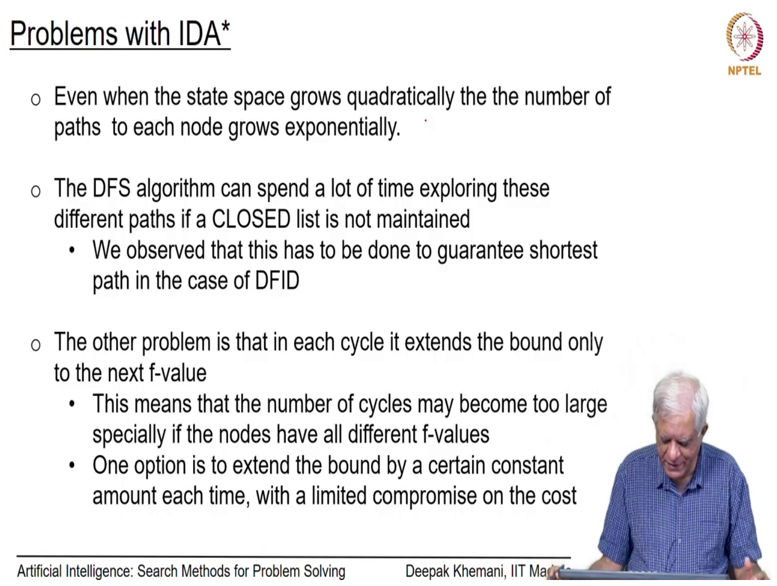
So, if you were to look at the space explored by IDA star, this diagram shows you that at any given cut off or depth bomb which is shown by this grey coloured oval halo. The space that IDAs are explored is shown in this, green coloured nodes. And, you can see that there are many ways of reaching different nodes. But, what IDA star will do is that it will do a depth first search which means it will go down this path.

Then it will backtrack go down this path, then it will backtrack go down another path and so on. So, essentially it will do depth first search, but it will stay within this boundary. So, this green node is not the list of closed nodes, it is a list of nodes which have explored by DFS. And, we have said that DFS will require only a linear amount of space.

The nodes in the yellow are the ones which were generated, but which were outside the bound and therefore, they were not inspected. And what IDA star will do that it will take the cheapest of those nodes; for example, in this example it could be this one. And, it will extend the boundary to that f value of that node.

So, in that sense it is a little bit different from DFID. DFID extended the depth by 1, IDAs are increases the bound to the value of the cheapest unexplored node; f value of the cheapest unexplored node.

(Refer Slide Time: 13:47)



Problems with IDA*

- Even when the state space grows quadratically the the number of paths to each node grows exponentially.
- The DFS algorithm can spend a lot of time exploring these different paths if a CLOSED list is not maintained
  - We observed that this has to be done to guarantee shortest path in the case of DFID
- The other problem is that in each cycle it extends the bound only to the next f-value
  - This means that the number of cycles may become too large specially if the nodes have all different f-values
  - One option is to extend the bound by a certain constant amount each time, with a limited compromise on the cost

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

Now, that can lead to problems of its own, as you can imagine. Even when the state space grows quadratically which means that you are looking at a city map or something, where the area increases as your search space increases which is the square of the length; then the number of parts to each node will grow exponentially. Because, there are its a combinatorial problem and we will see that this same combination problem will crop up; when we are talking about sequence alignment.

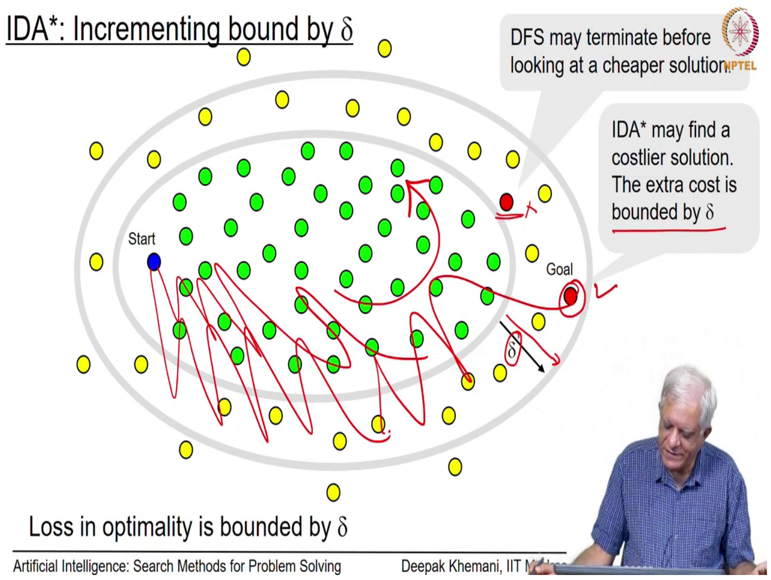
And, which is one reason why we will focus on space saving algorithms. Because the number of combinations goes exponentially, the depth first search algorithm and we saw this in the example of DFID; can explore many different parts to the same node especially when the CLOSED list is not maintained. And, we had argued that if you maintain the closed list, then in DFID we cannot guarantee the optimal path. So, we had said that in the DFID you should not maintain the closed list.

If you are not maintaining the closed list, then repeatedly visiting the same nodes again and again is going to be a problem. But, it is also a problem that IDA star extends the path only to the next unexplored node. Now, in the case of DFID because, it was depth that we were looking and we could think of the tree as a layer tree and you go to level 1, then to level 2, then to level 3 and so on.

At each level you would include many new nodes, but in the case of IDAs star because we are only incrementing the value to the next f value. It is possible that only 1 or 2 or 3 nodes may come into the ambit of the next cycle which means that the number of cycles may be very many easily. So, this is a major problem with IDA star. And, we have had students who have implemented IDA star.

And, they have observed this behaviour that the number of cycles which happens in a randomly generated graph can be quite large essentially. So, one option that we can look at if you are implementing IDA star because, after all IDA star has the advantage of giving you linear space is to not increment it to the next f value, but choose some value constant and say I will increment it by that much.

(Refer Slide Time: 16:12)



IDA*: Incrementing bound by δ

DFS may terminate before looking at a cheaper solution.

IDA* may find a costlier solution. The extra cost is bounded by δ

Loss in optimality is bounded by δ

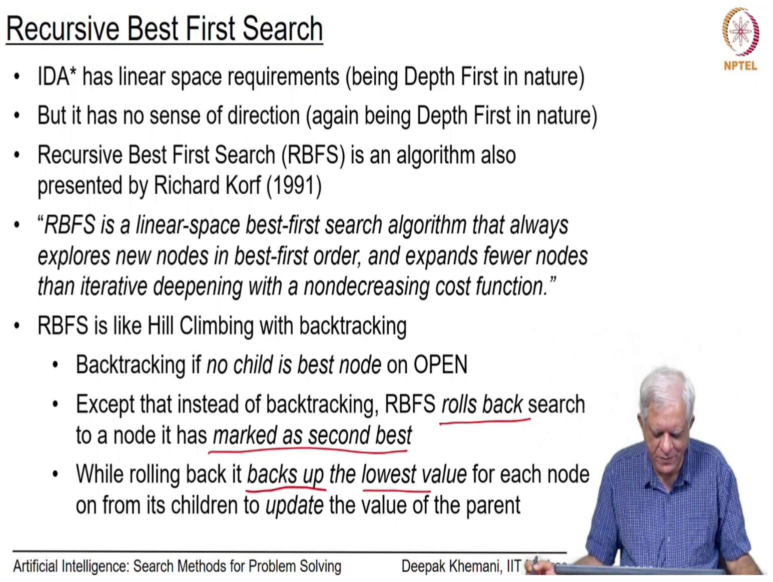
Artificial Intelligence: Search Methods for Problem Solving

Deepak Khemani, IIT Madras

And of course, you may have to pay a price in terms of optimality. So, this diagram depicts this situation, where you are incrementing the bound by certain pre decided value delta. And, then you will do depth first search by this part. So, what does this mean? It has this implication that if it is again searching like this going down this path. In fact, in this case it is going to be like going up to the new bound.

It may first hit this goal node which is actually less which has a greater cost than another node which would have occurred later, because the search is flipping in this direction. And, it may miss out on that this particular node on, it will miss out on this node, but it will find this node which is actually a little bit more expensive. But, the extra cost is bounded by delta and if you are willing to pay that much price then our IDA star is going to speed up much more than the basic IDA star algorithm.

(Refer Slide Time: 17:13)



Recursive Best First Search

- IDA* has linear space requirements (being Depth First in nature)
- But it has no sense of direction (again being Depth First in nature)
- Recursive Best First Search (RBFS) is an algorithm also presented by Richard Korf (1991)
- "*RBFS is a linear-space best-first search algorithm that always explores new nodes in best-first order, and expands fewer nodes than iterative deepening with a nondecreasing cost function.*"
- RBFS is like Hill Climbing with backtracking
    - Backtracking if *no child is best node* on OPEN
    - Except that instead of backtracking, RBFS *rolls back* search to a node it has *marked as second best*
    - While rolling back it *backs up the lowest value* for each node on from its children to *update* the value of the parent

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

But, Richard Korf the person who gave us IDA star also gave us a new algorithm called Recursive Best First Search. And, we have seen that IDA star has linear space requirement because it is the first in nature, but it has no sense of direction; again because it is simply depth first in nature. Recursive best first search is an algorithm given to us by Korf in 1991.

And in his words he says that, "RBFS is a linear-space best and best-first search algorithm that always explores new nodes in the best-first order. So, in some sense it has a sense of direction and he observes that it expands fewer nodes than iterative deepening and by this he means iterative deepening A star with a nondecreasing cost function. "

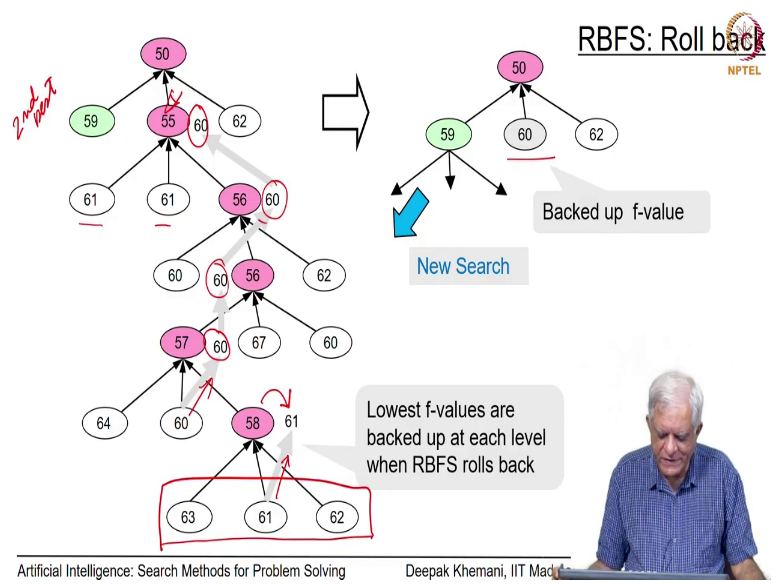
So, RBFS is a little bit like hill climbing with backtracking, that you go down some path requiring linear space; because hill climbing requires linear space. Hill climbing requires constant space, but if you maintain the parent pointers then you would, if you maintain the path backwards it would require linear space with depth. And, you backtrack if no child of the current node is the best node on OPEN essentially.

So, it is a bit like hill climbing where the candidates have been of the parents have also been kept alive, they have been kept in open. And, for the current node if the neighbours does not contain the best node, then you backtrack. But, actually backtracking would mean increasing amount of space, RBFS does not backtrack; it does what they call as rollback.

By rollback you mean you just undo what you did and go back to some ancestor. And, the ancestor is the one which is a sibling of a node which has been marked as the second best node essentially. So, rolling back by itself would not help because then you would come back to the same path because the values h, f values are the same.

So, what RBFS does is that it backs up the lowest value for each node and updates the values of its parents as it rolls back. So, this is a key feature of RBFS that it rolls back and updates the value. So, let us see that with an example; we will not discuss the algorithm in much more detail here.

(Refer Slide Time: 19:43)



RBFS: Roll back

2nd best
New Search
Backed up f-value
Lowest f-values are backed up at each level when RBFS rolls back
NPTEL
Artificial Intelligence: Search Methods for Problem Solving
Deepak Khemani, IIT Madras

And, let us say this is the case where the nodes in pink are they depict the paths, that is generated by A star. So, you see we start with the value of 50, these are f values. And, we will see in the next session that typically f values tend to increase as we go towards the goal. But we will do this a little bit more formally; especially under some conditions that we will study which is called the monotone criteria.

But, in general the f values tend to increase because they tend to become more accurate; because the contribution of h decreases as you are going closer to the goal. So, this is a value, you start with a value of 50, then you have 3 children; 59, 50 and 82, you choose 55 there and you mark 59 as the second best this thing. So, this one is the second best and it is marked and we have it in this colour.

Then you keep progressing, you generate the children of 55. You have 61, 61, 56; you choose the best as long as it is better than the second best. This process goes on, you come down to another 56, then 57, then 58. And then suddenly when you generate the children of this node 58; you find that all of them are worse than the second best node 59.

Now, remember that this observe that the space requirement here is linear, because at every level as you go down you just add a constant number of neighbours which is the what depth first search would have done. Unlike, depth first search what this one is doing is that it has a sense of direction. So, it is not going to the left most part, but it is going to the path which is the which is guided by the heuristic value.

So, in that sense is a little bit like hill climbing, but it is more like depth first search in the sense that it maintains all the parents and their children. And, in this particular example when it finds that hill climbing would not have proceeded. Well, in the sense that it is not like hill climbing because the neighbours are, the next nodes are actually increasing f values and we have seen that.

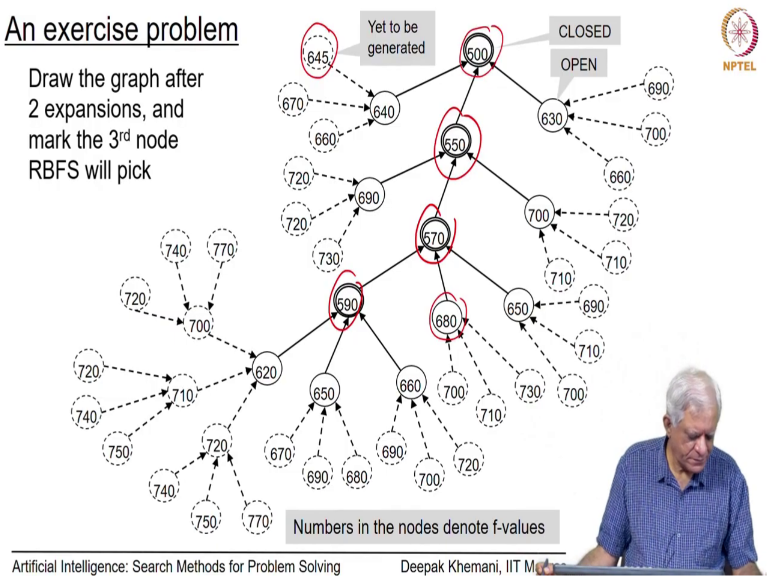
So, that condition that the neighbours have to be better than current node is relaxed here. You move to the best neighbour, but RBFS also keeps the second criteria, which is that you must be better than the second best node that we have seen so far; in this case which is a value of 59. Now, all the 3 nodes generated here at the bottom, they have a value of 63, 61, 62. All 3 are worse than the second best.

So, RBF simply rolls back this entire search tree, but when it rolls back it updates the values of its parents as it does so. And, it backs up the best value for every parent. So, this last node 58 now becomes 61 because, 61 is the best value that it can find amongst this children.

Then between 64, 60 and 61, this 60 gets backed up to its parents. So, 57 gets updated to 60 and so on. So, this process goes, this 60 is again propagated back and propagated back and propagated back all the way up.

So, this node which was 55 has now become 60 and search resumes from this point onwards. So, at this point when it is rollback you can see that the middle node has become a value of 60. Now, the left node has a value of 59 and so, the new source starts from there. So, it goes along these paths and it will roll back has not been necessary maintaining linear space essentially.

(Refer Slide Time: 23:28)



An exercise problem

Draw the graph after 2 expansions, and mark the 3rd node RBFS will pick

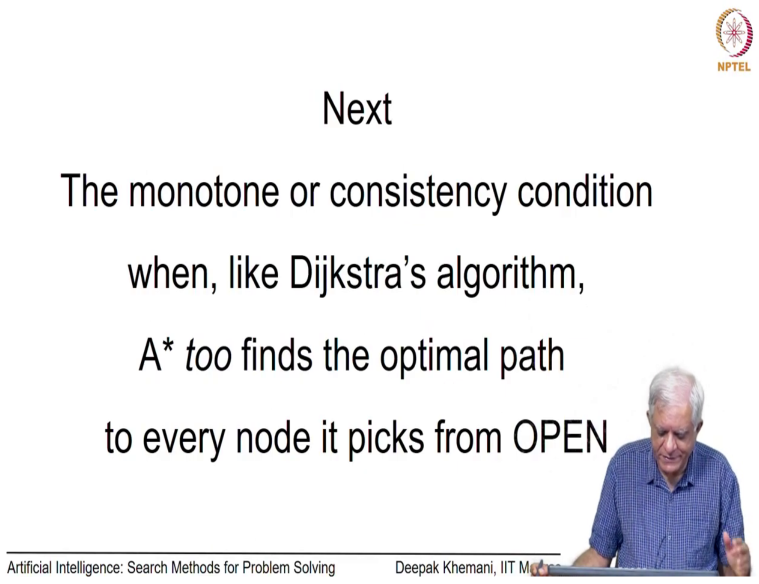
Numbers in the nodes denote f-values

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

So, this is the RBFS algorithm which we have done rather quickly. I will leave you with an example here problem for you to solve. In this example these nodes are the closed nodes, that have been explored. The nodes with single circles are open nodes, that are in open and the nodes, nodes in dash circles are the ones which have yet to be explored. And, in your you should look at which of them you want to explore and the values are f values.

And, what I would like you to do is to draw the graph after 2 expansions have been done from this given state and also mark where the 3rd node that RBFS will pick up. So, we will put up this graph as part of the weekly assignment and you must try to solve this problem.

(Refer Slide Time: 24:11)



Next

The monotone or consistency condition

when, like Dijkstra's algorithm,

A* *too* finds the optimal path

to every node it picks from OPEN

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

Next we will look at what is called as a monotone or consistency condition when, A star will behave like Dijkstra's algorithm; which means that when it puts a node then closed it would have found an optimal path to that node. Or, in other words when it picks a node from OPEN, it would have found an optimal path. And, we will see that this monotone criteria will allow us to write better space saving algorithms which in fact, were devised in this century.

So, we will take that up in the next session. So, see you then.