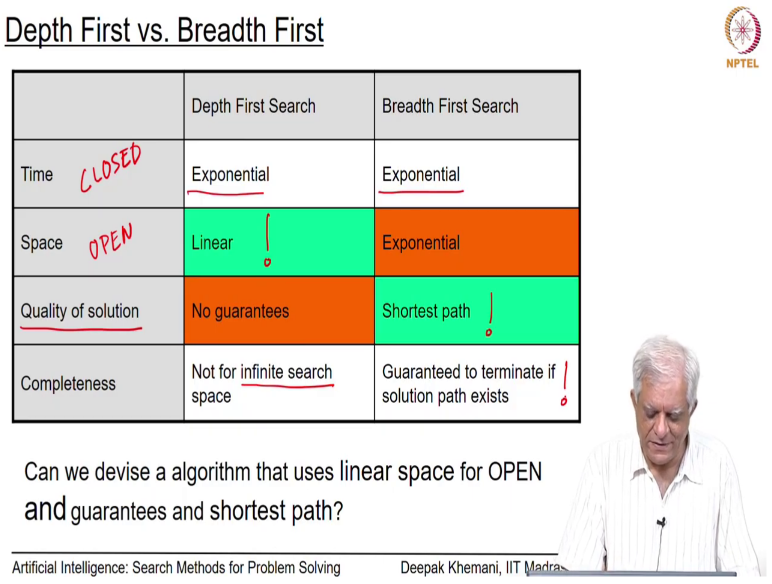
Artificial Intelligence: Search Methods for Problem Solving

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

Lecture – 22
Depth First Iterative Deepening

(Refer Slide Time: 00:16)



Depth First vs. Breadth First

	Depth First Search	Breadth First Search
Time CLOSED	Exponential	Exponential
Space OPEN	Linear !	Exponential
Quality of solution	No guarantees	Shortest path !
Completeness	Not for infinite search space	Guaranteed to terminate if solution path exists !

Can we devise a algorithm that uses linear space for OPEN and guarantees and shortest path?

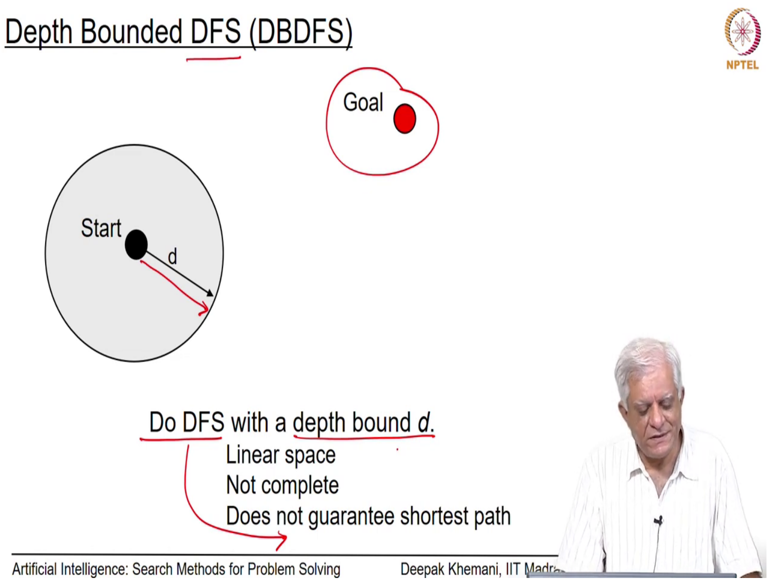
Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

So, if you look at these two properties of the two algorithms, then you can see that for both of them time complexity is exponential in nature. If you look at space complexity which is a size of open, time complexity can be seen thing sort of is a size of closed because those are the nodes that you have seen and space complexity is the set of candidates that you are keeping which is a size of open.

And you can see here that depth first search means own space complexity. But breadth first search means on the quality of the solution because we have already observed that it will give you the shortest path.

In terms of completeness for finite search space is both are complete, in the sense both will end up exploring the entire space; but if you have infinite search space, we have seen that depth first search can go into an infinite loop essentially. Whereas, again this is a plus point for a breadth first search it is guaranteed to terminate, if there is a paths to the goal node. Because the path will always be of a finite length and depth first search breadth first search goes down level by layer exploring paths of detail length.

(Refer Slide Time: 01:32)



Depth Bounded DFS (DBDFS)

Do DFS with a depth bound *d*.

Linear space
Not complete
Does not guarantee shortest path

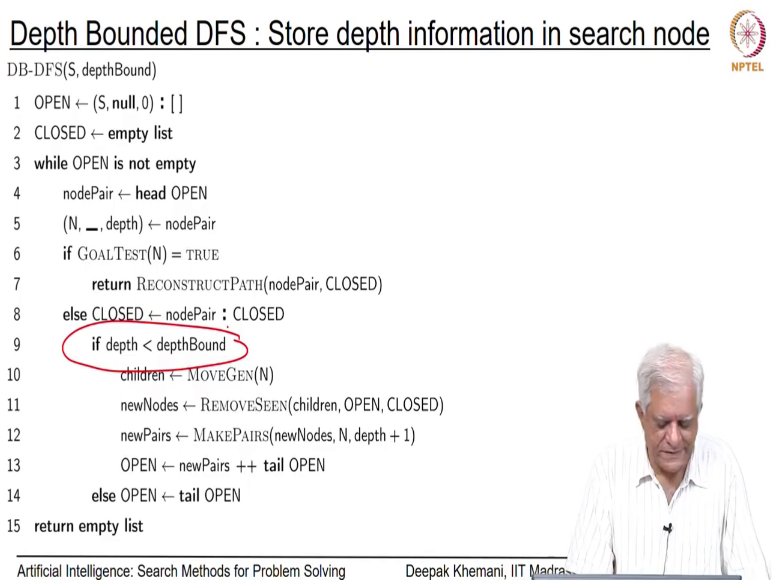
Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Madras

So, can we have an algorithm that uses linear space for open and which guarantees the shortest path ok? So, luckily we have such an algorithm, but before we move to that algorithm, let us look at a simpler algorithm and this is called Depth Bounded Depth First Search and essentially, what you are doing is depth first search, but you have a depth bound as shown here. So, in some sense, you have to stay within this circle which is like a Lakshman-Rekha that algorithm cannot go outside and, but otherwise the algorithm is depth first search.

So, essentially what we say do depth first with a depth bound of d, where d is given to us and as you can see this will be linear space because this is really depth first search and as you can see this is not complete because as you can see the goal is outside this boundary.

So, it will never find that goal and also, because it is depth first search, it will not guarantee your shortest path. Because even if there were two goals within its boundary, it might have found the one with the longer distance ok. So, of course, this is not something very exciting, but we can use this algorithm to construct a new algorithm which is as follows ok.

(Refer Slide Time: 02:56)



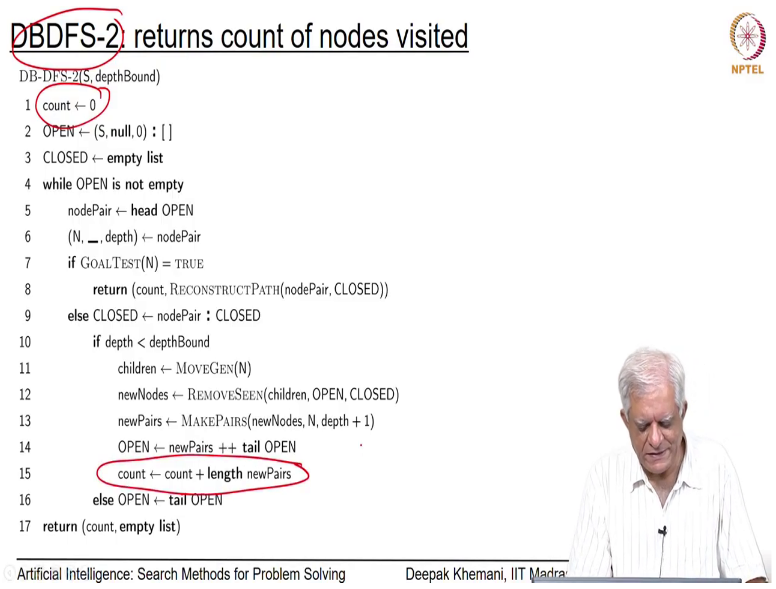
Depth Bounded DFS : Store depth information in search node

```
DB-DFS(S, depthBound)
OPEN ← (S, null, 0) : [ ]
CLOSED ← empty list
while OPEN is not empty
    nodePair ← head OPEN
    (N, _, depth) ← nodePair
    if GoalTest(N) = TRUE
        return ReconstructPath(nodePair, CLOSED)
    else CLOSED ← nodePair : CLOSED
        if depth < depthBound
            children ← MoveGen(N)
            newNodes ← RemoveSeen(children, OPEN, CLOSED)
            newPairs ← MakePairs(newNodes, N, depth + 1)
            OPEN ← newPairs ++ tail OPEN
        else OPEN ← tail OPEN
return empty list
```

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

So, this is just a code for depth bounded depth first search, if you were writing the program and the key thing to note here is that you move forward, only if you have within the depths bound and otherwise, you do not move forward essentially.

(Refer Slide Time: 03:16)



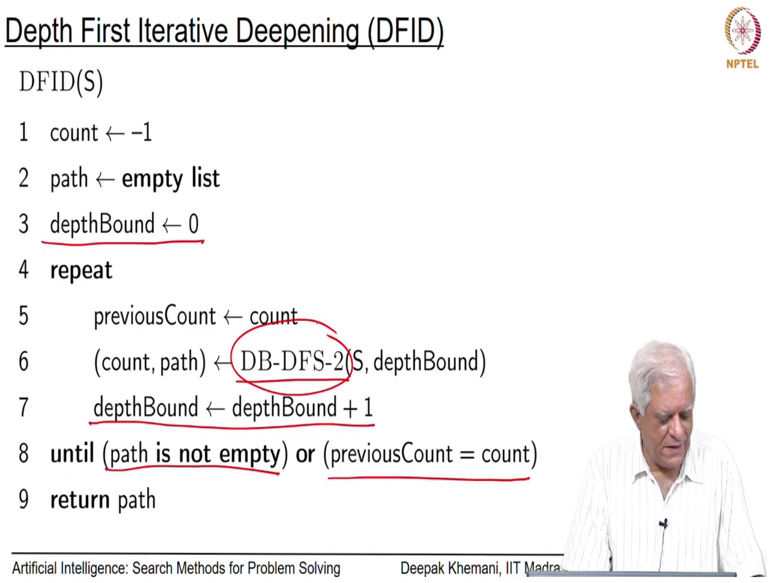
DBDFS-2 returns count of nodes visited

```
DB-DFS-2(S, depthBound)
count ← 0
OPEN ← (S, null, 0) : [ ]
CLOSED ← empty list
while OPEN is not empty
    nodePair ← head OPEN
    (N, _, depth) ← nodePair
    if GoalTest(N) = TRUE
        return (count, ReconstructPath(nodePair, CLOSED))
    else CLOSED ← nodePair : CLOSED
        if depth < depthBound
            children ← MoveGen(N)
            newNodes ← RemoveSeen(children, OPEN, CLOSED)
            newPairs ← MakePairs(newNodes, N, depth + 1)
            OPEN ← newPairs ++ tail OPEN
            count ← count + length newPairs
        else OPEN ← tail OPEN
return (count, empty list)
```

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

There is a variation on this algorithm which we call DBDFS-2 and what this does is that it maintains a count of the number of nodes. So, initially we say count is equal to 0 and that is a number of nodes it is seen and as we keep increasing, we keep adding count to the new nodes that we have seen essentially ok.

(Refer Slide Time: 03:46)



Depth First Iterative Deepening (DFID)

```
DFID(S)
1  count ← −1
2  path ← empty list
3  depthBound ← 0
4  repeat
5      previousCount ← count
6      (count, path) ← DB-DFS-2(S, depthBound)
7      depthBound ← depthBound + 1
8  until (path is not empty) or (previousCount = count)
9  return path
```

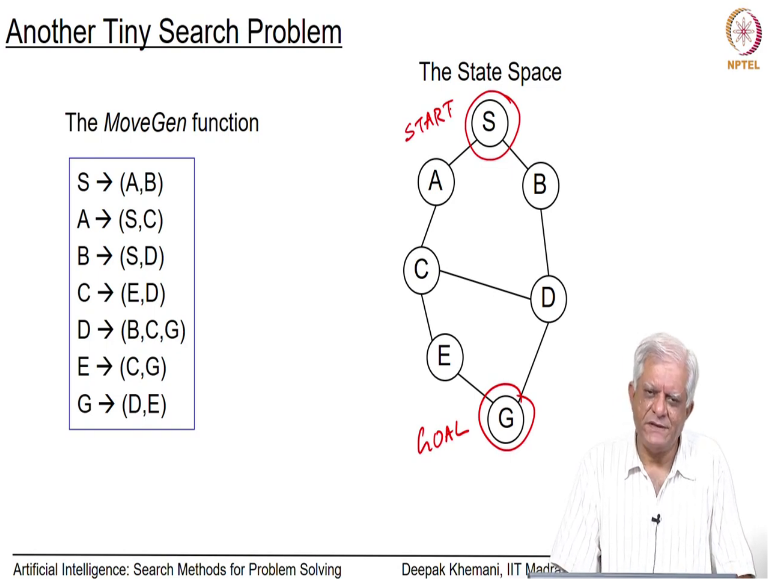
Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Madras

So, this gives us a clue for the algorithm that we are interested in which is depth first iterative deepening. It is essentially a series of depth bounded depth first search is done by increasing the depths bound gradually. So, the key thing is that we initially say depth bound is equal to 0 and then, as we do a sequence of DB-DFS calls, we keep increasing the depth bound by 1 essentially. So, essentially, we are doing iterative deepening, in the sense that iteratively we do DFS which is longer and longer and longer.

And we continue doing this to till either of the two criterias met; one is that if the previous count in the previous cycle is the same as in this cycle which means we have not visiting any new nodes and therefore, we can return with failure and otherwise, if we have found the paths, then we return the path essentially.

Now, you can see that that because this algorithm is doing a sequence of depth first searches or depth bounded depth first searches, its space complexity will be linear. But since it is going iteratively deeper level by level, it will end up finding the shortest path essentially.

(Refer Slide Time: 05:11)



Another Tiny Search Problem

The *MoveGen* function

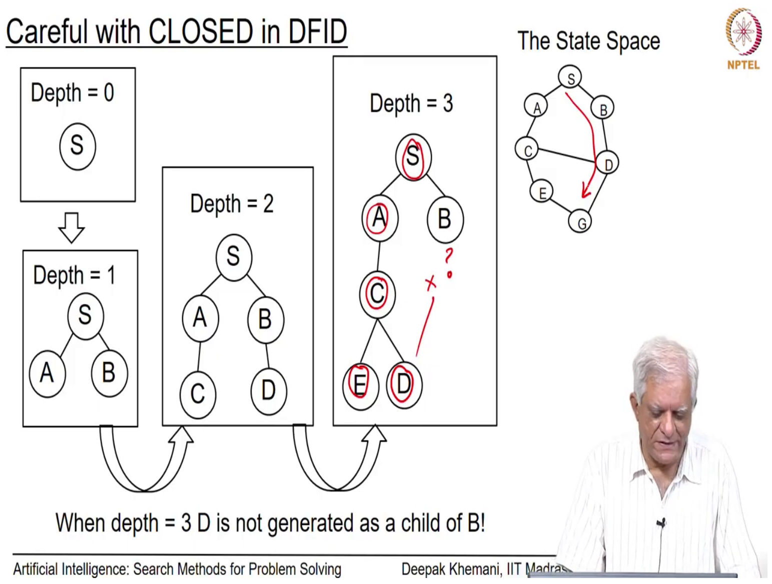
S → (A,B)
A → (S,C)
B → (S,D)
C → (E,D)
D → (B,C,G)
E → (C,G)
G → (D,E)

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

So, let us look at a small example and I want to just illustrate a point that one has to be careful about. This point was raised when I was teaching this course in IIT, Dharwad and a student called Siddharth Sagar pointed this out to me that if you are maintaining the closed list, then DFID will not find the shortest path.

So, let us see what is happening here. So, this is a small search space here and as before we will say this is the start node and this is the goal node and you will quickly see how this behaves algorithm. So, what is DFID do? It does a sequence of searches essentially.

(Refer Slide Time: 06:01)

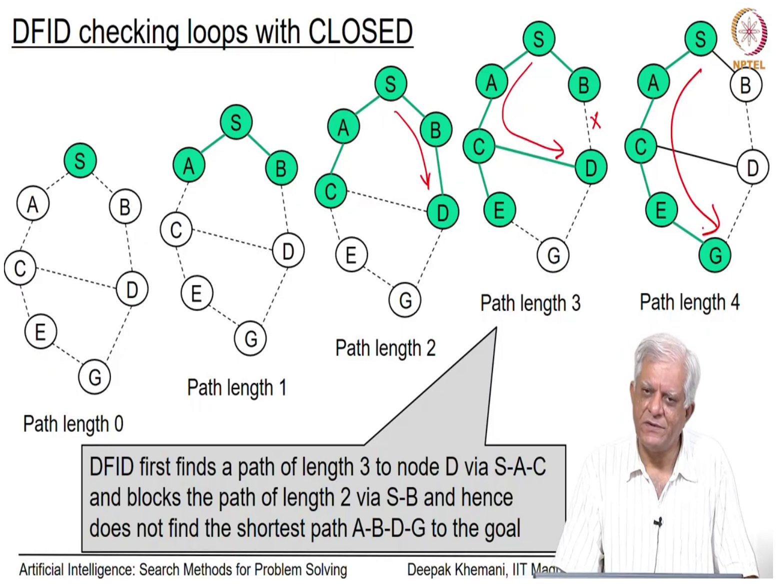

Careful with CLOSED in DFID
Depth = 0
Depth = 1
Depth = 2
Depth = 3
The State Space
When depth = 3 D is not generated as a child of B!
Artificial Intelligence: Search Methods for Problem Solving
Deepak Khemani, IIT Madras

So, this is the state space in the first cycle, where depth equal to 0, it only looks at the start node. In the second cycle, where depth is equal to 1, it goes on one level or it looks at paths of length 1. Then, in the third cycle when depth becomes 2, it does DFS and looks at paths of length 3. In the fourth cycle, it looks at depth 3 and it looks at path of length 3.

Now, if you have; if you have pruning nodes which are already on closed. So, remember that this is doing depth first search. So, this will go into close, this will go into close, this will go into close, this will go into close, this will go into close and then, when we have to generate children of B, we cannot generate D. Because D is already enclosed.

So, if you observe this graph, you will see that the shortest path to the goal is this one which is goes through B and D; but our search algorithm will not able will not be able to find this and this is because we are maintaining this closed list ok.

(Refer Slide Time: 07:10)



DFID checking loops with CLOSED

DFID first finds a path of length 3 to node D via S-A-C and blocks the path of length 2 via S-B and hence does not find the shortest path A-B-D-G to the goal

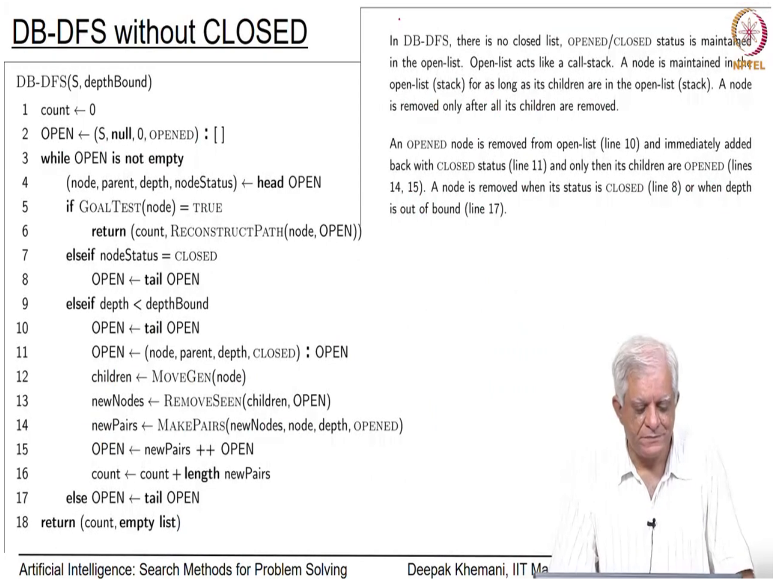
Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madr

So, let us look at this phenomenon again and that how checking with this closed list can create a problem. Now, we are looking at the graph and the same problem the same search that is happening. When it explores paths of length 0, it has seen only the start node; when it explores path of length 1, it has generated A and B.

The dotted edges are the ones, it has not generated on the way. When it looks at paths of length 2, it has looked at the nodes S-A-C and S-B-D at this point it has found the shortest paths to D.

But when it looks at paths of length 3, it finds the path to D which is the longer path and more importantly, it does not allow D to be generated as a child of B because its already been inspected and consequently, it does not find the shortest path and eventually, when we do the next cycle paths of length 4, it finds a longer paths to the goal node and so, therefore, maintaining closed is a no if you are doing DFID.

(Refer Slide Time: 08:26)



DB-DFS without CLOSED

```
DB-DFS(S, depthBound)
1   count ← 0
2   OPEN ← (S, null, 0, OPENED) : [ ]
3   while OPEN is not empty
4       (node, parent, depth, nodeStatus) ← head OPEN
5       if GoalTest(node) = TRUE
6           return (count, ReconstructPath(node, OPEN))
7       elseif nodeStatus = CLOSED
8           OPEN ← tail OPEN
9       elseif depth < depthBound
10          OPEN ← tail OPEN
11          OPEN ← (node, parent, depth, CLOSED) : OPEN
12          children ← MoveGen(node)
13          newNodes ← RemoveSeen(children, OPEN)
14          newPairs ← MakePairs(newNodes, node, depth, OPENED)
15          OPEN ← newPairs ++ OPEN
16          count ← count + length newPairs
17      else OPEN ← tail OPEN
18  return (count, empty list)
```

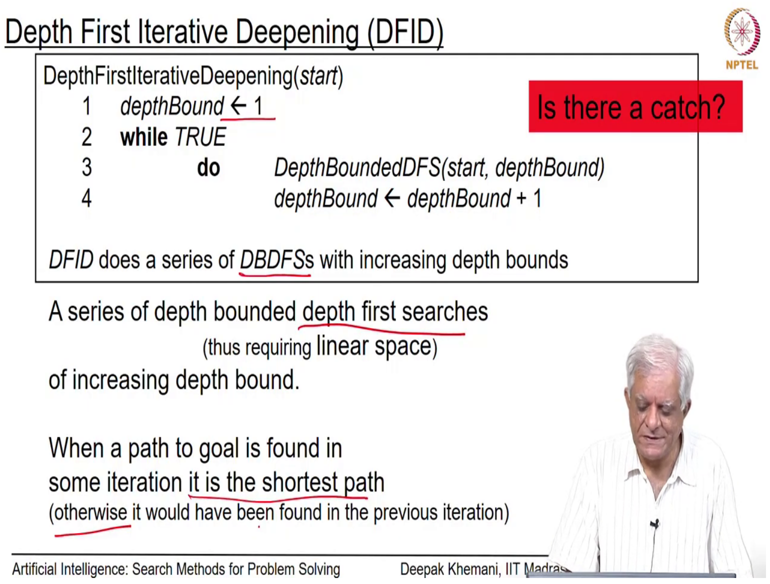
In DB-DFS, there is no closed list, OPENED/CLOSED status is maintained in the open-list. Open-list acts like a call-stack. A node is maintained in the open-list (stack) for as long as its children are in the open-list (stack). A node is removed only after all its children are removed.

An OPENED node is removed from open-list (line 10) and immediately added back with CLOSED status (line 11) and only then its children are OPENED (lines 14, 15). A node is removed when its status is CLOSED (line 8) or when depth is out of bound (line 17).

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

So, this is just an algorithm, where we maintain, the we run DFID without maintaining the closed node. For those of you, who are interested in programming, you can go back to this slide and have a look essentially.

(Refer Slide Time: 08:42)



Depth First Iterative Deepening (DFID)

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

DFID does a series of *DBDFSs* with increasing depth bounds

Is there a catch?

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 Madras

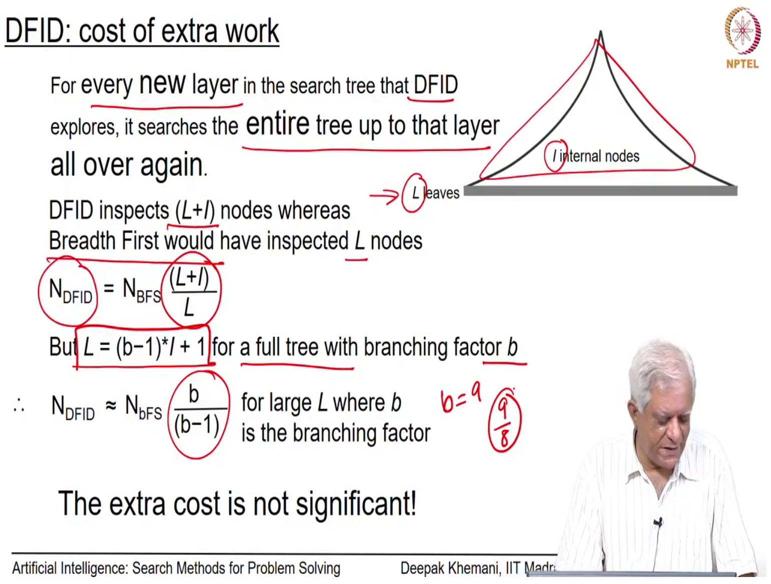
But essentially, if you look at the depth first iterative deepening algorithm what it does is that it initially sends sets the depth bound and then, it keeps increasing the depth bound to 1 essentially.

That is the series of depth first depth bounded depth first searches with increasing depth essentially, till it finds a goal node or till it exhausts looking at the entire graph, if it is a finite graph. If it is an infinite graph, it may never terminate. So, it is a series of depth bounded first searches requiring linear space because it is depth first search with increasing depths bound.

When the path to the goal is found some iteration in found in some iteration, it is the shortest path because otherwise it would have found it in the previous path essentially. So, DFID and mind you DFID when you are not maintaining the closed list will give you the shortest path and it requires linear space for the open list.

But is there a catch? I mean are we getting something for free, we are getting the advantages of depth first search and breadth first search, let us just quickly look at this.

(Refer Slide Time: 09:54)



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

$$\therefore \quad N_{DFID} \approx N_{bFS} \frac{b}{(b-1)}$$ 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 Madr

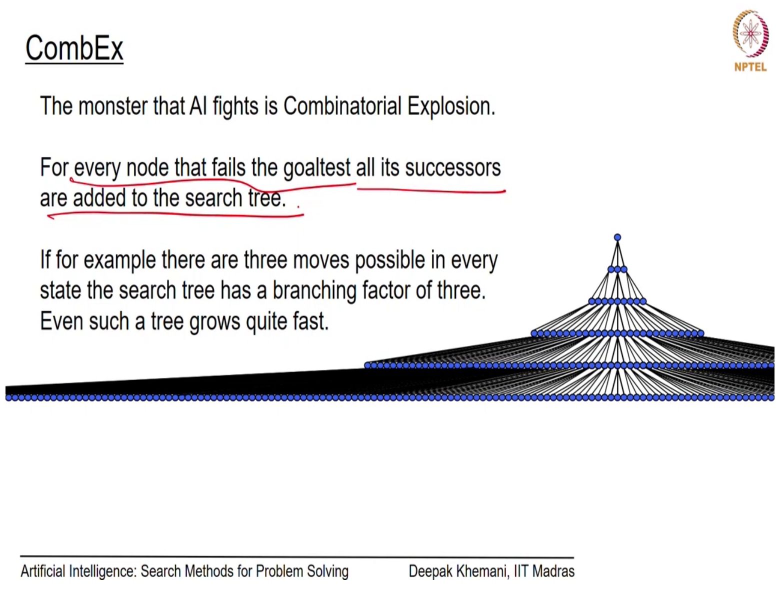
Obviously, it does extra work to do. For every new layer that DFID explores which is this last layer in this figure, it searches the entire tree up to that layer all over again. So, all these tree it searches again. So, it is in some sense, its doing breadth first search; but it is doing those additional internal nodes as well as.

So, if there are I internal nodes and L leaves, then DFID inspects L plus I nodes; whereas, breadth first would have inspected only L nodes. So, how much is this extra work? Now, that is a interesting observation about exponential growth is that the number of leaves generally significantly outnumber the number of internal nodes, which means the set of candidates, the number of candidates generated at every level is always larger than the all the candidates that you have seen in the path essentially ok.

So, the numbers in my DFID would be a ratio of L plus I divided by L, where L plus I is what DFID sees and L is what breadth first search sees. But you know that for a full tree with branching factor b, the number of leaves is given by this expression b minus 1 into number of internal nodes plus 1. And therefore, if we do this ratio and we ignore small constants, we say the ratio is only b over b minus 1 and again, which is not very significant, which means that for example, if b where to be 9 as before, then this would be 9 by 8.

So, it is not significantly extra works that DFID is doing, but it is giving us the shortest path with linear space ok. So, the cost is not significant, but the thing you must keep in mind is that it is still exponential in nature. All these three algorithms depth first search, breadth first search and these things are exponential in nature.

(Refer Slide Time: 12:02)



CombEx

The monster that AI fights is Combinatorial Explosion.

For every node that fails the goaltest all its successors are added to the search tree.

If for example there are three moves possible in every state the search tree has a branching factor of three. Even such a tree grows quite fast.

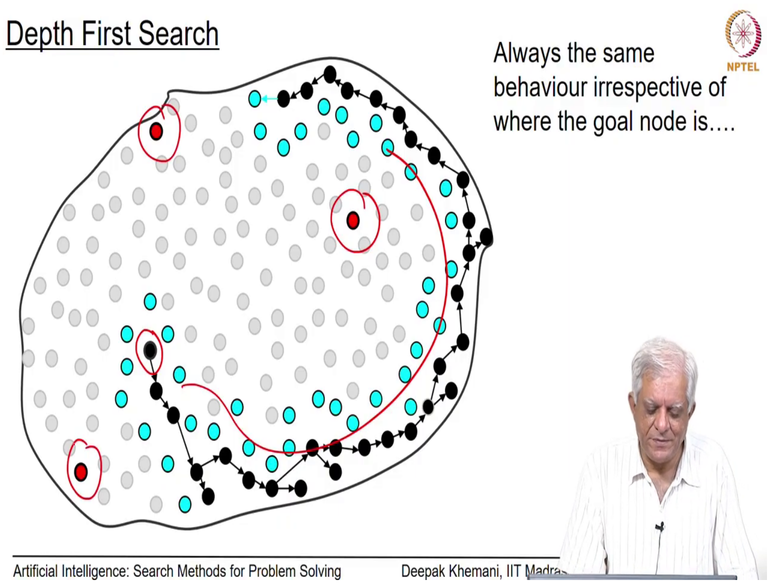
Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

And the monster that we are fighting in AI or combating in AI is the combinatorial explosion or exponentially growing search space, which is depicted in this tree which has you can see growing on your right hand side of the screen. For every node that fails the goal test all its successors are added to the search tree.

So, this is a little bit like if you are familiar with Greek mythology, the monster hydra that Hercules was fighting that for every head of the monster that Hercules would chop off, a whole host of heads will appear and a exponentially growing search tree essentially signifies that and as you can see that even when branching factor is three, then the search tree is growing large enough that we cannot draw the next level on this board essentially.

So, essentially, we need some other approaches to fight this combinatorial explosion.

(Refer Slide Time: 13:07)



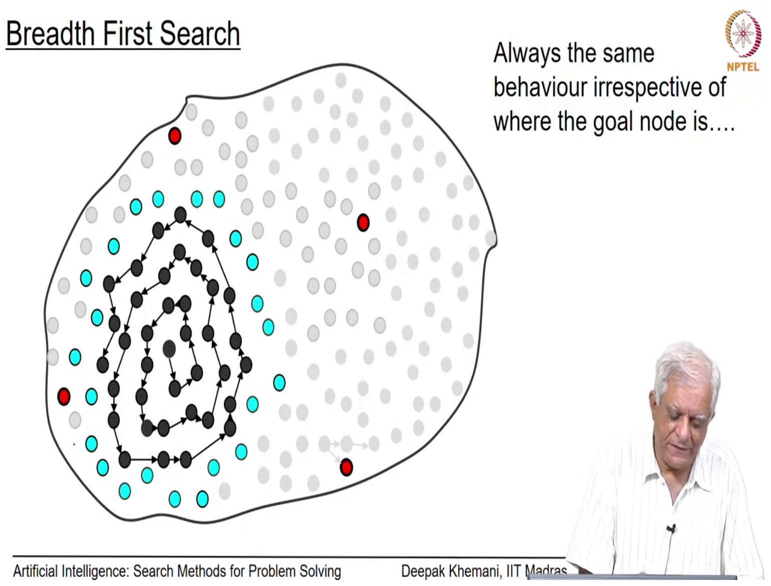
Depth First Search

Always the same behaviour irrespective of where the goal node is....

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

And we will do that in the next class. We will leave with the observation that the depth first search algorithm always behaves in the same manner irrespective of whether the goal was here or whether it was here or whether it was here and in that sense it is a blind search algorithm.

(Refer Slide Time: 13:29)

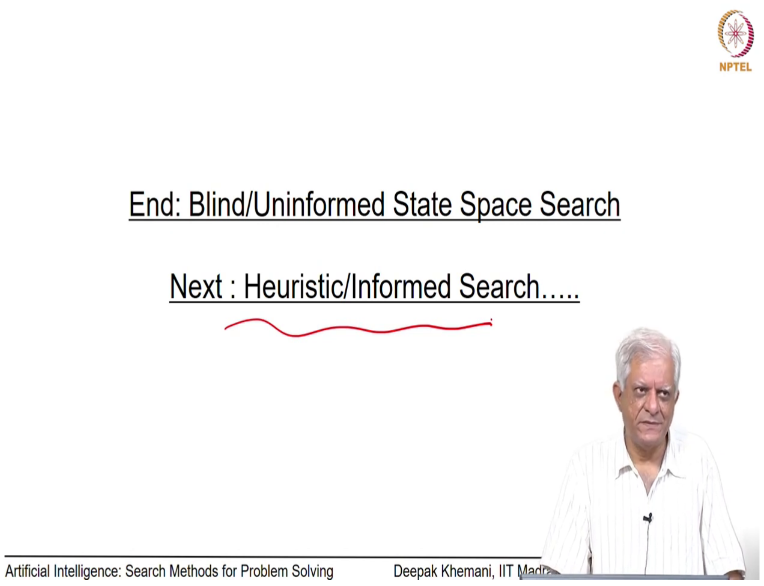

Breadth First Search

Always the same behaviour irrespective of where the goal node is....

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

The same is the case with breadth first search irrespective of whether the goal is here or here or here it just (Refer Time: 13:36) goes around doing, what he does all the time without even being aware of where the goal node is.

(Refer Slide Time: 13:46)



End: Blind/Uninformed State Space Search

Next : Heuristic/Informed Search…..

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

So, the next stage that we will look at is called Heuristic search and that is what we will do in the next class ok. So, see you for the next class.