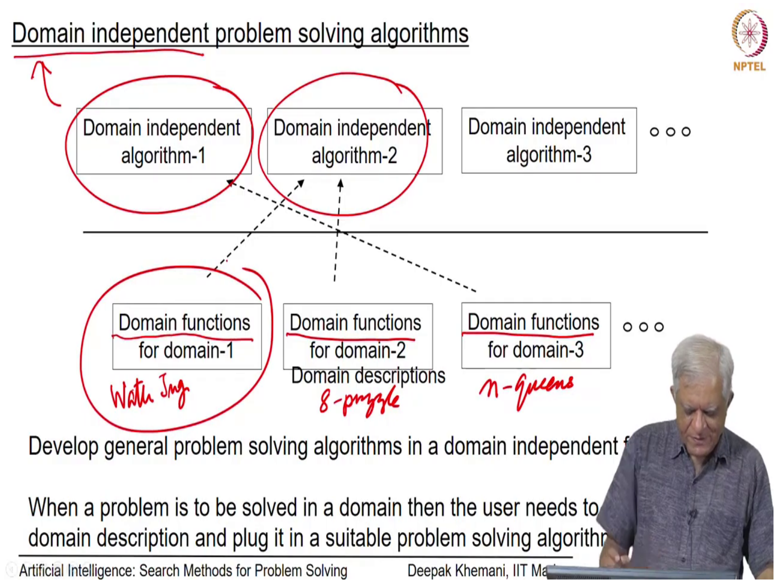
Artificial Intelligence: Search Methods for Problem Solving

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

Lecture – 18
Domain Independent Algorithms

(Refer Slide Time: 00:14)


Domain independent problem solving algorithms

Domain independent algorithm-1
Domain independent algorithm-2
Domain independent algorithm-3
Domain functions for domain-1
Domain functions for domain-2
Domain functions for domain-3
Domain descriptions
Water Jug
8-puzzle
n-Queens

Develop general problem solving algorithms in a domain independent f

When a problem is to be solved in a domain then the user needs to
domain description and plug it in a suitable problem solving algorithr

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

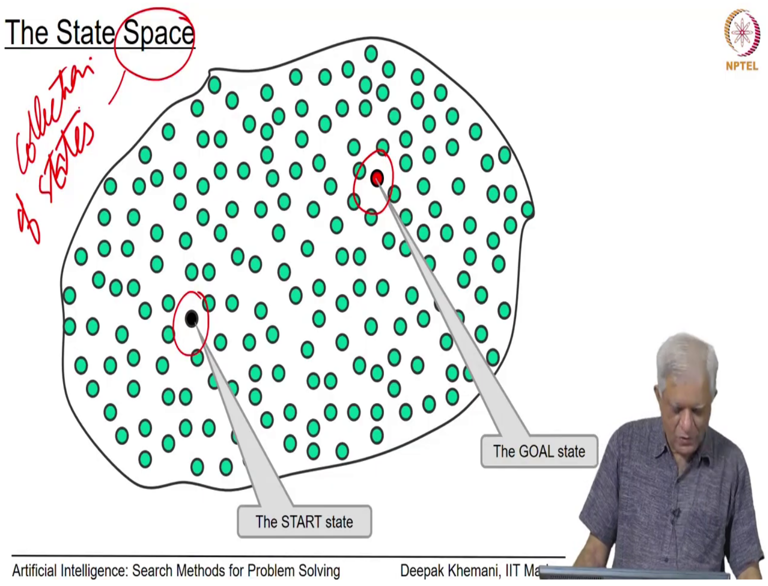
So let us begin in the last class we kind of looked at what do we mean by problem solving and what are the kind of problems that we can model which can be solved by first principle methods. So our goal of course is not to solve those individual problems that we discussed in the last class.

So for example, we discussed the water jug problem or we discussed the 8 puzzle or we discussed the n queens; we did not discuss n queens in general we discussed 6 queens, but you can generalize that to n queen problem. So, we have all these different problems which are that we want to solve which some user wants to solve and we would expect that the user will give us the domain functions in those problems.

What are the representations and what are the moves that we can make is left to the user to define. Our focus would be on domain independent methods and we will look at different kind of search algorithms. So, you may have an algorithm 1 or algorithm 2 and so on, these are all domain independent.

Which means that we are not writing a program to solve the water jug problem or the 8 puzzle or the n queens or the traveling salesman problem or whatever the case may be. We want to model this whole problem solving process as a process of searching in some space and that space is going to be the state space to begin with for us and we will see how algorithms can solve problems using in the state space essentially ok.

(Refer Slide Time: 02:14)

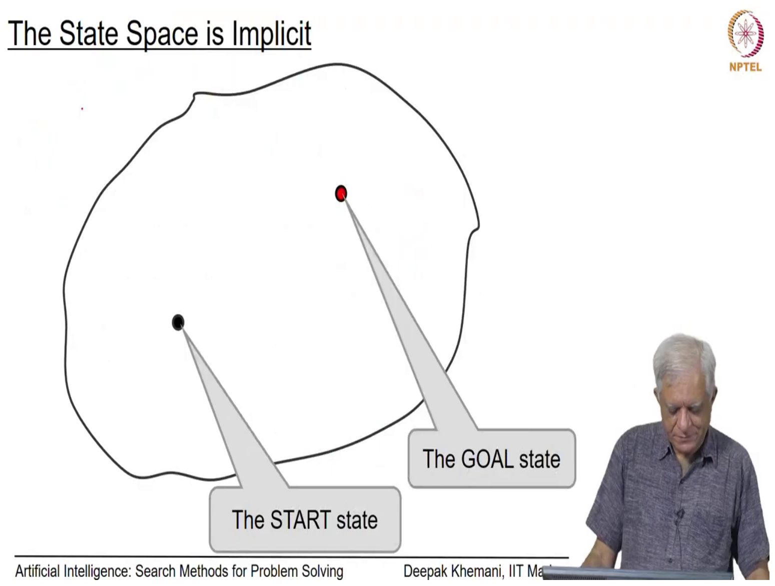


So what we imagined is that there is a space of possible states, in which one of them is identified as a start state and at least one if not more than one. As we saw in the water jug problem if you want to measure out 4 liters, then there may be more than one state which is a Goal state.

But let us for simplicity assume that we have one goal state and it can easily be without loss of generality extended to multiple goal states and our task is to somehow solve this problem of transforming the START state into a GOAL state.

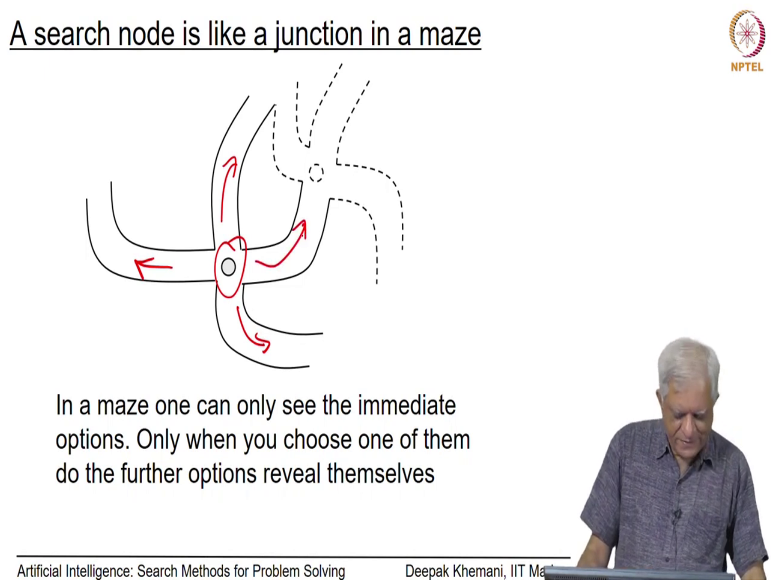
So, the space in which the problem is being looked at is will be call the state space. In the sense that the space is a collection of states, as AI people we will not worry about what the states are we will assume that there is space in which we can identify the start state and the goal state and we want to somehow solve that problem.

(Refer Slide Time: 03:34)


The State Space is Implicit
The GOAL state
The START state
Artificial Intelligence: Search Methods for Problem Solving
Deepak Khemani, IIT Ma

The state space as I said in the end of the last class is implicit, the graph is not given to us what is really given to us is a start state and the goal state essentially. And what we expect from the user is to give us some domain functions which will help us navigate the state navigate the space and also help us to dominate our process of search essentially.

(Refer Slide Time: 04:08)



A search node is like a junction in a maze

In a maze one can only see the immediate options. Only when you choose one of them do the further options reveal themselves

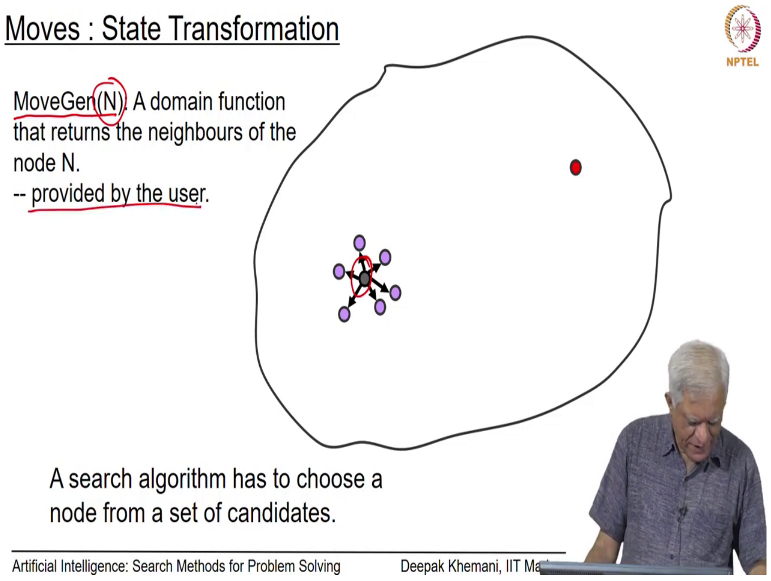
Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Ma

Now a search node in some space is like a junction in a maze. So, if you imagine that you are in this maze and you are at this node and you want to you know somehow get out of the maze or something like that; you cannot see the entire maze.

As human beings when we draw graphs and we draw things like that we can see a nice picture, if you want to draw for example finding a route from let us say IIT Madras to Chennai central station and you are looking at this entire map. Then you have a global picture which says that it looks like I have to go in that direction and these are the roads and so on and so forth.

The graph as algorithm does not have this benefit of this global view, it has only a local view that it is in a given state and it can go to some place which is directly connected to that state essentially.

(Refer Slide Time: 05:11)



Moves : State Transformation

MoveGen(N): A domain function that returns the neighbours of the node N.
-- provided by the user.

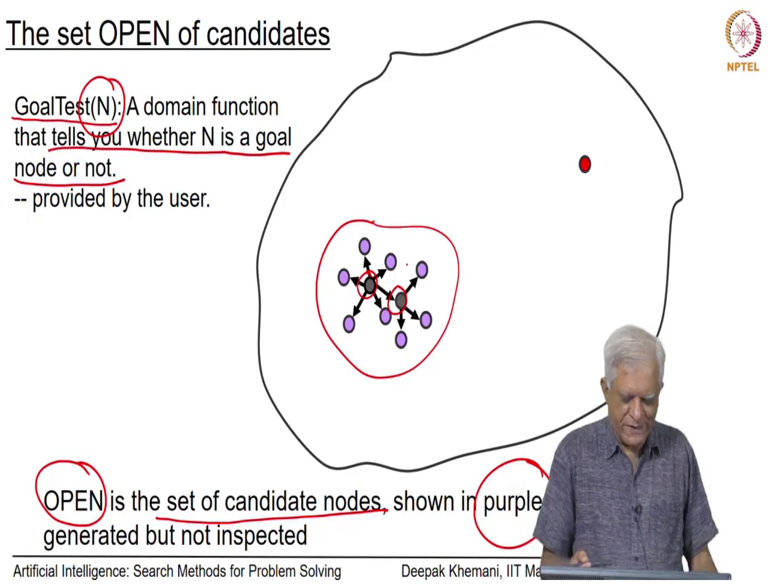
A search algorithm has to choose a node from a set of candidates.

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

And that is depicted in our formulation, as a state transition function or MoveGen function or a neighbourhood function; which takes any node N. In this case this node N as input and returns to us as to what are the neighbouring states for that node. In other words what are the moves that you can make from this node.

So, when we started with this 8 what are the problem? We saw that when you have 8 liters jug full and the other two jugs empty you could make 2 moves; you could either pour out 5 liters into the 5 liter jug or you could pour out 3 liters into the 3 liter jug. So, the MoveGen function basically tells you that in a given state where all you can move, we also expect that this would be provided by the user.

(Refer Slide Time: 06:13)



The set OPEN of candidates

GoalTest(N): A domain function that tells you whether N is a goal node or not.
-- provided by the user.

OPEN is the set of candidate nodes, shown in purple
generated but not inspected

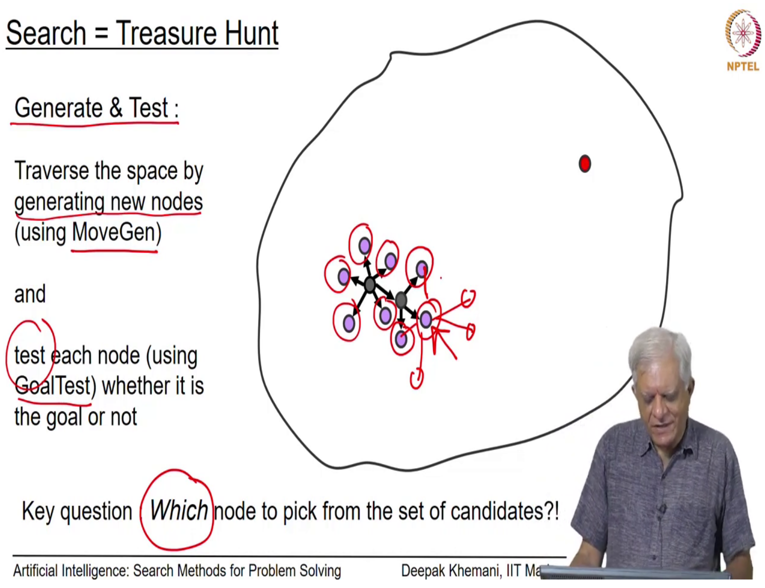
Artificial Intelligence: Search Methods for Problem Solving
Deepak Khemani, IIT Ma

So, as we said earlier the domain functions should be given to us by the user, we will focus only on solving the problems. So, once a user gives us this domain function it has given us the ability to navigate this space, not only do we navigate this space we keep generating the graph on the fly as we go along.

The other function that we expect from the user is we will call it as a GoalTest function. So, what this function will do is take any node as input and tells you whether N is a goal node or not essentially. So, we will call this function as a GoalTest function. So, we will essentially do search over this space and keep searching over all the nodes keep applying the GoalTest function and eventually when we find that we are at the goal then we can terminate it.

We will use a terminology that there is a set of candidate nodes which we will call as OPEN and which have been shown in purple color here. So here for example, you have 2 nodes that you have already visited and they did not turn out to be the goal state and in the process you generated all these other candidates and essentially you have to pick one of them and see if it is a goal state. If it is not a goal state you want to move on to the next state and that kind of a thing essentially.

(Refer Slide Time: 07:40)


Search = Treasure Hunt

Generate & Test :

Traverse the space by generating new nodes (using MoveGen)

and

test each node (using GoalTest) whether it is the goal or not

Key question: Which node to pick from the set of candidates?!

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

So, our search is a little bit like Treasure Hunt essentially, that we have been told that one state let us say hence a part of goal in it or something like that. But we do not know which state and we have to you know keep searching. So, it is like if you are let us say a friend of yours gives you the keys to his hostel room and says you can go and rest in my room, but he forgets to tell you which room is his room.

Then the only option for you would be to go to all the rooms and keep trying the key and see which one is the thing. The search space that we are talking about is very similar to that ah, but we are trying to model it in a very general way essentially. So, that technique that we use here has traditionally we call as Generate and Test. Generate a candidate state and test whether it is a goal state or not.

So, we traverse the space by generating new nodes and we can do that by the MoveGen function which the user will give us and we test each node using the goal test function, which also we will assume that the user has given us. So, our task is not to worry about the domain henceforth; our task is to simply navigate this space and arrive at the goal state in as efficiently as possible.

And the key question here is going to be which node to pick from the set of candidates states. So, in this diagram you see that we had all these nodes which we said were on the open list and which one should we inspect next.

Now obviously, if we somehow knew magically as to which is the one that takes you to the goal state, we would do that would amount to having what the people in theory community say having an oracle, which tells you which is the right choice to make and that is in some sense the people in non deterministic programming they use this terminology; that you non deterministically choose the right node.

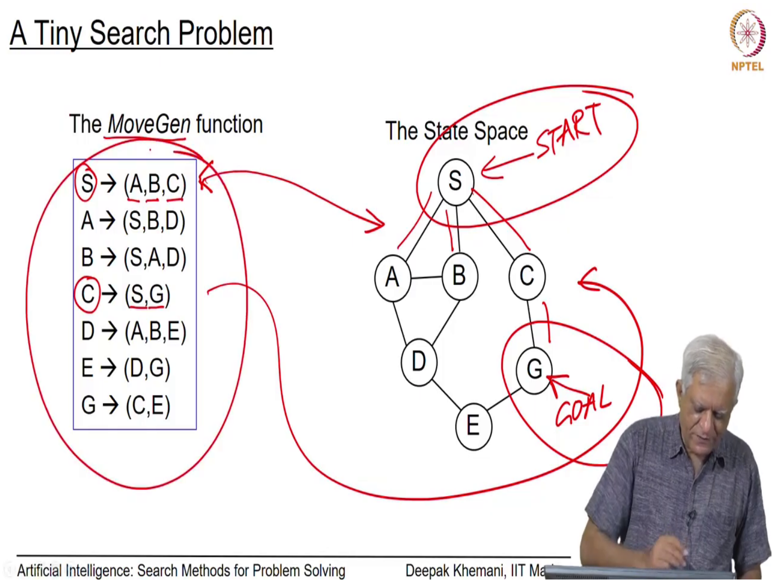
But we do not have the benefit of an oracle and the way to convert such non deterministic choices to deterministic choices is to do search essentially. So, you will essentially end up in the worst case end up looking at all these nodes and all the new nodes that you will generate and so on and so forth. And eventually hopefully find the path to the goal essentially.

Initially we will look at simple algorithms to do this so we will pick one node and we will test whether it is a goal center. Let us say we pick this node we will test whether it is a goal node or not; if it is not the goal node we will generate it is children again so we will add more nodes to the graph and so on and so forth.

And we will upload some simple approaches which we will do the search, a little bit later we will look at algorithms which try to get a sense of direction. Trying to approximate this idea of this oracle which tells you which is the right choice to make, simply by looking at the choices we cannot make the right choice.

Because as we mentioned earlier we have only local information, we only know which are the next states that you can go to. We do not know which state will lead to the goal state. But we will try and look at see how we can incorporate certain amount of domain knowledge to guide us towards the goal node. But that will come into something that we will call do later and it is called heuristic search. First let us look at the mechanism for doing search and how that can be done.

(Refer Slide Time: 11:23)



A Tiny Search Problem

The *MoveGen* function

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

The State Space

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

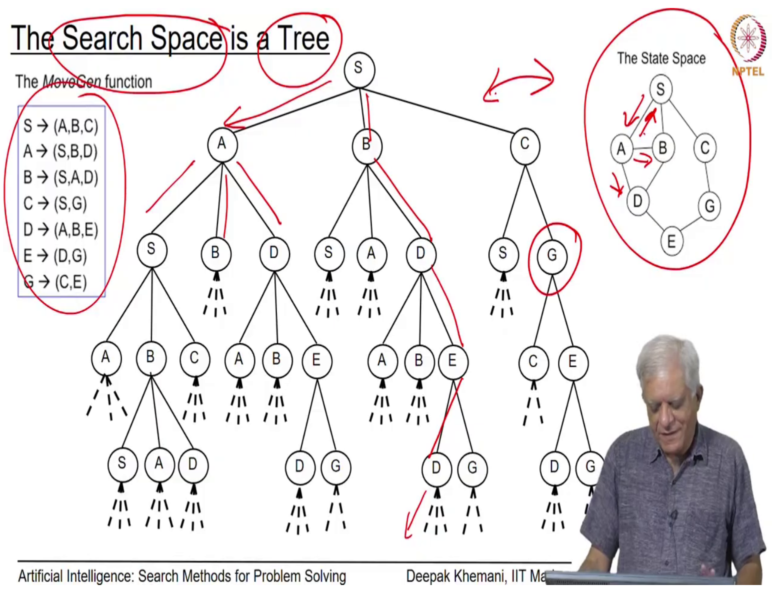
So, to do that we will work with a small search problem and the problem is given to us in the form of a MoveGen function. The MoveGen function is given in this table here and the way to read this table is that from the State S you can move to either A or to B or to C.

So, the MoveGen function take S as input and returns the list or set A, B, C. From the set C for example you could move either to S or to G essentially. So, the MoveGen function tells you how to traverse the space and these two examples you can see is that from S you can go to A or S you can go to B or S you go to C and this is what is depicted by this particular move here essentially.

Likewise, from C you can go to S or G and that is depicted by here essentially. So, essentially the MoveGen function is what is given to us, the graph that you see on the right hand side is not given to us. The graph is implicit it is a state space that we are talking about that we have to traverse and we will generate the graph as we will see on the fly as we as we proceed.

So, what is really given to us is that this is a start state and let us assume that what is given to us is that this is the goal state. So, I will generally use S for the start state and G for the goal state, but that has of course no special meaning at all just for the sake of convenience. So, you are given so what is given to you are given the start state you are given the goal state and you are given the MoveGen function and the MoveGen function implicitly will generate the graph and our task is to find a path from the start state to the goal state.

(Refer Slide Time: 13:25)


The Search Space is a Tree

The MoveGen function

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

So, the state space is a graph as we have seen, but we are not; but we are going to look at algorithms which use the MoveGen function to explore that space and in the process what they will do is to generate what is call the Search Space. And the search space is generally a tree and it basically a tree of choices that you can make essentially.

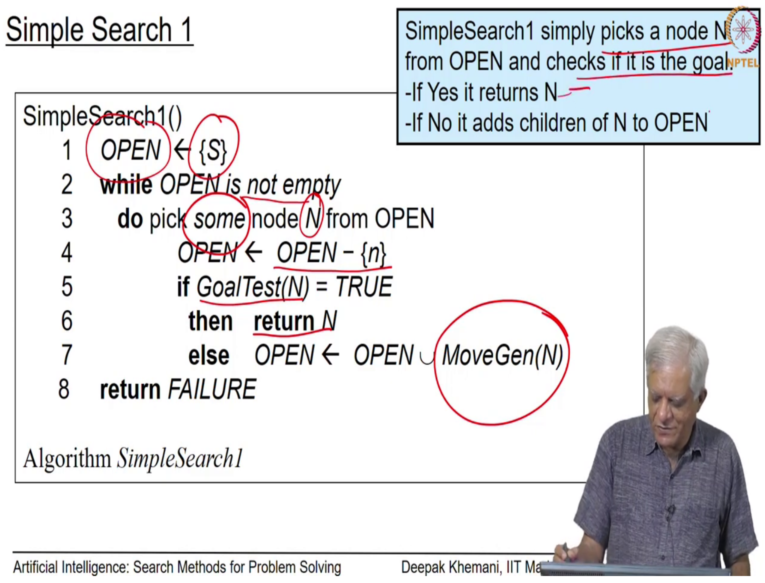
So for example, this tree depicts the fact that from S you can go to A which is amounts of saying that you can go from the graph S to A here and then from A you can go back to S. If you want or you can go to B or you can go to D. So, again you can go back here or you can go here or you can go here. So, those arrows that you see are kind of explicit in the search space and the search space is essentially a tree.

This tree is essentially as we have depicted here depicts all possible traversals you can do without termination. Of course, in practice we would want to terminate when we reach the goal state. So for example, if you were to reach this state we do not want to really move away from that, but that is the what the algorithm will do.

But what the algorithm is exploring is a search space which is a tree, which basically depicts what all you can do. So, you can start from S go to B from B go to D D go to E D and maybe go to D back again and then go somewhere else. So, you can keep traversing the space and the space that we will search in will typically be a tree and it will correspond to the state space in which we are trying to solve the problems.

Algorithms will generate these trees on the fly, the trees not given to us that is why it is only depicted partially here and it will essentially work towards a solution.

(Refer Slide Time: 15:21)



Simple Search 1

```
SimpleSearch1()
1   OPEN ← {S}
2   while OPEN is not empty
3     do pick some node N from OPEN
4           OPEN ← OPEN – {n}
5           if GoalTest(N) = TRUE
6             then  return N
7             else  OPEN ← OPEN ∪ MoveGen(N)
8   return FAILURE
```

Algorithm *SimpleSearch1*

SimpleSearch1 simply picks a node N from OPEN and checks if it is the goal
-If Yes it returns N
-If No it adds children of N to OPEN

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

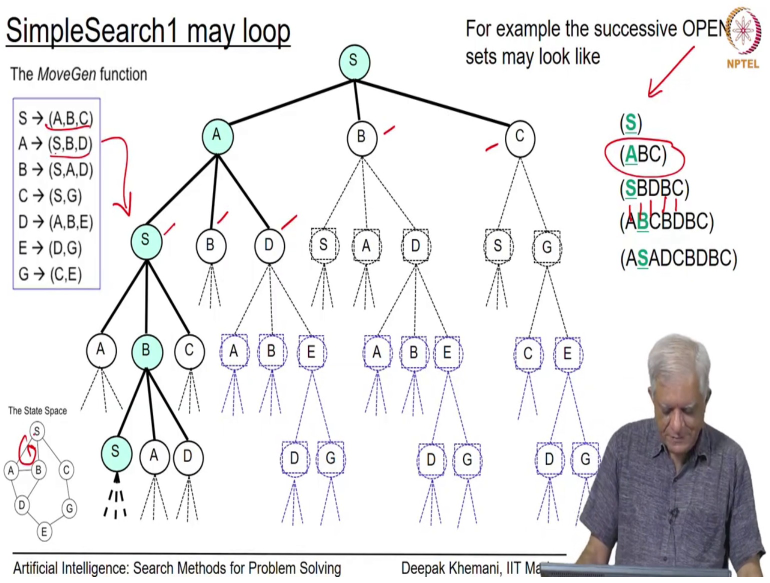
So, the simplest algorithm that we can think of is as follows, we will call it simple Search 1. We had mentioned that our list of candidates will be called OPEN and what we are given is only the start state. So, we put that into we will call that as OPEN and our algorithm simply says that as long as our set of candidates is not empty pick some node ok.

So this some is going to be the keyword, but for the moment we will assume that there is certain amount of non determinism all that you are going to you know do trial and error. So, it does not matter we will come we will try to refine this notion of some as we go along, because how efficient your problem solver is will depend upon which node you pick as we said earlier.

If you had an oracle we would know exactly which node to pick, but we do not have an oracle, so we have to do some amount of trial and error. So, we pick some node we will call that node N and we will remove it from OPEN and then we will apply the GoalTest function to N. If the GoalTest and function is returns true, then we just exit and say we have found the goal node else what do we do we generate the neighbors of N using the MoveGen function and we add them to open essentially.

So, this is the union operator and so we add them to OPEN and so we keep maintaining this OPEN list and this thing. So, as you can see this basically illustrates a generate and test procedure, pick some node N from OPEN check, if it is goal if yes return N if no add more nodes to OPEN.

(Refer Slide Time: 16:58)



SimpleSearch1 may loop

The *MoveGen* function

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

For example the successive OPEN sets may look like

(S)
(ABC)
(SBDBC)
(ABCBDBC)
(ASADCBDBC)

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

So, let us see how this may happen and the problem with our very simple formulation is that this algorithm may end up going into a loop and let us follow the algorithm as it searches. So, again on the left hand side you can see the MoveGen function given to us, below that we can see the state space that we are trying to explore. But not what now what we will draw is the search tree as explored by our algorithms essentially.

So, we start with this node S our open contains only S to start with. So, we test whether S is the goal or not S is not the goal we have already agreed upon the G is a goal node ah. So, S is not the goal so we will remove it from OPEN and we will add it is neighbours. What are the neighbours the neighbors are A, B, C so we will add those to the neighbour. And in this process as you can see the search tree is generated on the fly essentially and the colored nodes the sign nodes represent the fact that which is a choice you are making.

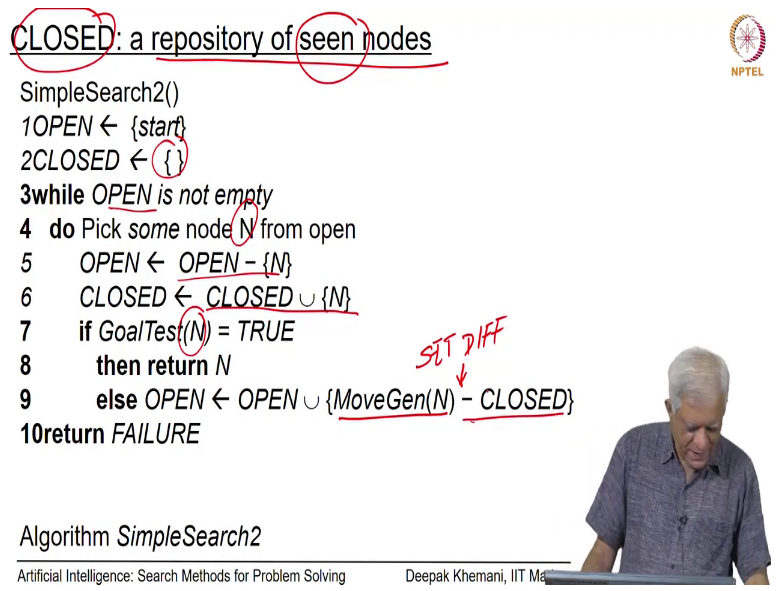
So at this moment as you can see there are three nodes on OPEN A B and C and let us assume since we have said pick some node that our algorithm will has picked the node A from this. So, we take the node A test whether it is a goal or not it is not the goal. So, we add it is neighbours which are S B and D as you can see here. So, we add this S B and D to the search tree and we have all this nodes sitting on OPEN.

What is an OPEN? We have S here we have B here D here B and C and that is depicted here, that you could have chosen S B D we could have chosen this S or B or D or B or C. But let us assume that we have chosen S here essentially; which is of course as you can see it is not a very good choice, but still that is a possibility. And then we generate add the children of S again which as you can see is the same set A, B, C.

And then we add those children of B this time and that is S A D which is actually quite sad, because we are going into an infinite loop and this path is never going to terminate. So, our algorithm the simple search algorithm could just go down this path and keep repeating going into the cycles.

So it could for example, you know simply keep going into cycle of S A B or something like that and never you know move beyond that we do not want really that and so that is the first thing that we would want to do. And this algorithm has not explored the remaining search tree at all essentially and it is gone down a loop essentially.

(Refer Slide Time: 19:51)



CLOSED: a repository of seen nodes

```
SimpleSearch2()
1OPEN ← {start}
2CLOSED ← { }
3while OPEN is not empty
4   do Pick some node N from open
5      OPEN ← OPEN – {N}
6      CLOSED ← CLOSED ∪ {N}
7      if GoalTest(N) = TRUE
8         then return N
9         else OPEN ← OPEN ∪ {MoveGen(N) – CLOSED}
10return FAILURE
```

Algorithm *SimpleSearch2*

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

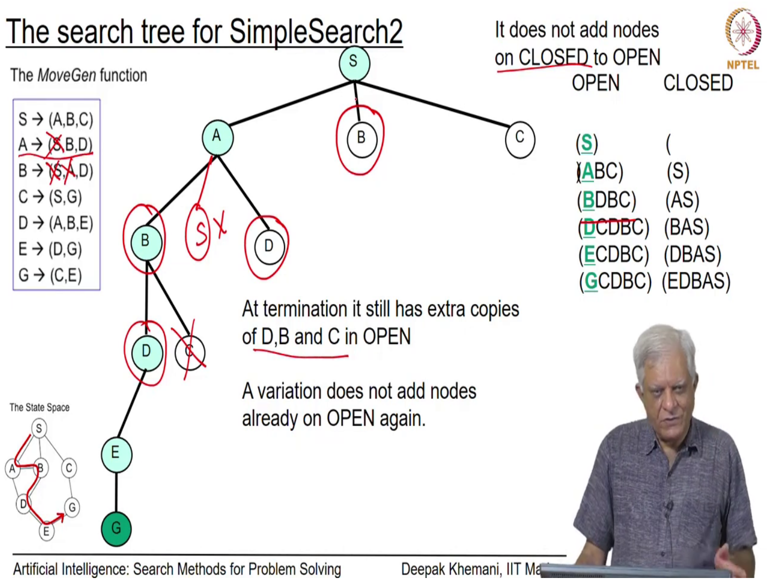
So, what do we do? We decide that if we have visited a node once in the state space, we will not visit it again essentially. Which means that if you have inspected a node and found that it is not the goal node then never keep that as a candidate.

How do we do that? We do that by maintaining another list and this is a repository of seen nodes essentially. So once you have seen nodes, then you do not put them again we will call this algorithm as simple search 2. As before OPEN will get the start node CLOSE will be empty to start with and while OPEN is not empty as before ah; pick some node from OPEN remove it from OPEN add that node to close this essentially. This is critical again test whether this N is a goal node or not.

If it is not the goal node then return, if it is a goal node return N, if it is not then call move gen function with N and from the neighbours that this gives remove whatever is on closed. So this is set difference, sometimes people use a different notation the backslash was a difference, but as long as there is no confusion.

So, the idea here is that from this set of neighbours which is returned by the MoveGen function, we remove any node which is present in closed and this is depicted by this set difference operator which removes those things. And in this process you can see that we will never visit the same node again and again.

(Refer Slide Time: 21:34)



The search tree for SimpleSearch2

The *MoveGen* function

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

It does not add nodes on CLOSED to OPEN

OPEN	CLOSED
(S)	(
(ABC)	(S)
(BDBC)	(AS)
~~(DCDBC)~~	(BAS)
(ECDBC)	(DBAS)
(GCDBC)	(EDBAS)

At termination it still has extra copies of D,B and C in OPEN

A variation does not add nodes already on OPEN again.

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

So, hopefully we will not get into an infinite loop and let us see how this algorithm performs. Now we have called it SimpleSearch 2 and we have already said it does not add nodes which are on CLOSED again to OPEN. So now it is maintaining both OPEN and CLOSED list. And as before we start with the start node S and S is in OPEN and close is empty, this bracket is kind of flown to the next line. But that should not bother us.

Then we generate as before the children of S which is A B and C and as before let us assume that we picked the node A and we generate a children of B essentially. Now this is where the difference comes into play. So, this time when we see that the children of A are S B and D, but S is already in CLOSED. So, we are not going to now add it as children of S, so there is no S here.

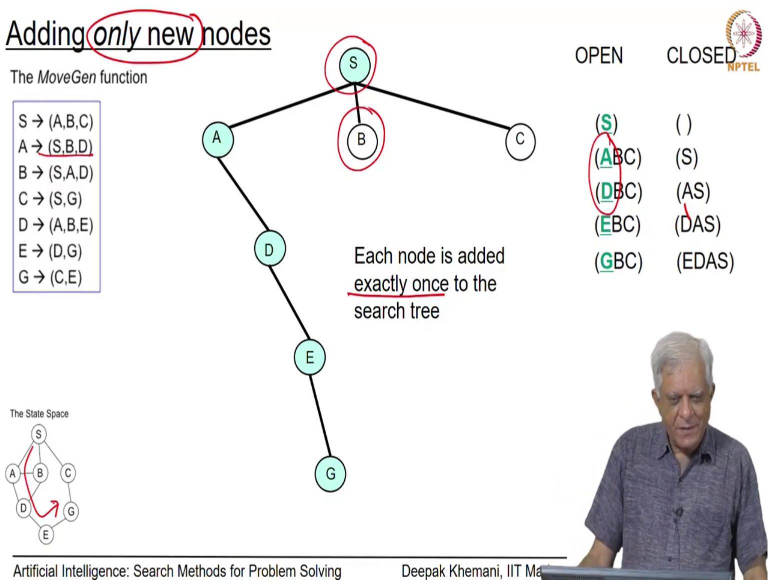
We could have added in the previous case we generated S and we made the wrong choice of going back to S, but because we are maintaining this close list S will not be generated. So, the choice is only between B and D. So, let us assume that we actually the choice is between B D and B and C as we can see from the OPEN list here. So, let us assume that somehow we have chosen this node B and then we generate the children of B.

Again you can see that from the children of B we have a actually this is an error this should not have been there that we have not added S we have not added A. So, ideally would have added only D here this is a error on my part so we can leave that out. And then we go to e and then we find that there is a path that we have found to E and the path that we have found is depicted here in this node essentially.

So the nice thing about this approach of using this closed is that you do not get into these loops and eventually the graph search algorithm will traverse this entire tree. And either find the path to the goal if there exist one and we will talk about this a little bit later or it will terminate by saying I have seen all the nodes none of them is the goal node and that is possible because we are going to visit every node exactly once essentially.

As you can see that at termination it is still has extra copies of D B and C well not C because this C was an error here, but at least of D and B. And D we had visited but there is a copy lurking around somewhere B we had visited there is another copy lurking around somewhere. We could avoid that also by saying that not only do not add those nodes which are already on CLOSED, do not add those nodes which are also on OPEN essentially.

(Refer Slide Time: 24:40)



Adding *only* new nodes

The *MoveGen* function

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

Each node is added exactly once to the search tree

OPEN	CLOSED
(S)	( )
(ABC)	(S)
(DBC)	(AS)
(EBC)	(DAS)
(GBC)	(EDAS)

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

So which means this is the next version of the algorithm, we add only new nodes. Which means nodes which have not been added to the graph at all. So, which means now every node is added to the graph that we are or the search tree that we are generating exactly once.

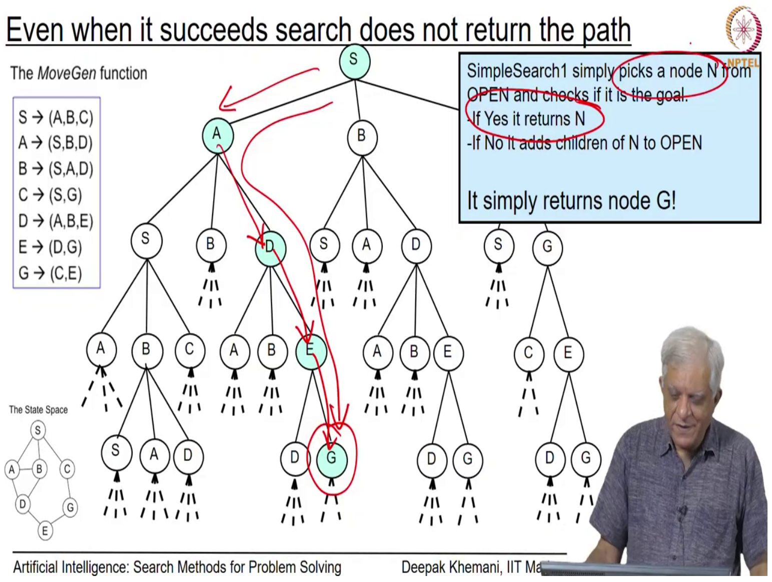
So, let us follow the process of this algorithm, you start with the start node as before and as before we generated children A, B, C. And as before we now I have A, B, C in OPEN and the start node has gotten gone into CLOSED and then we generate the children of A, B, C children of A. Now again notice that the children of A was S and B and D S is already an CLOSED and B is already on OPEN.

So, our this version of the algorithm only adds the new node D to the OPEN list and that is depicted here. That when you remove A and we put A into the CLOSE list, but we add only D to open list. And the reason for this is that we have not added B, because B was already on open and at some point we are going to look at it.

So, where I add keep two copies and then this goes further from D it goes to E and from E it goes to G and then it terminates. Now you can see that each node in the state space is added exactly once to the search tree.

Also notice that this has generated the different paths the path this has found is S to A to D to E to G, but the search tree that it has generated is smaller than the search tree that the previous algorithm is generated. And that will obviously will affect the time that the algorithm will spend searching this search tree.

(Refer Slide Time: 26:38)



Even when it succeeds search does not return the path

The *MoveGen* function

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

SimpleSearch1 simply picks a node N from OPEN and checks if it is the goal.
-If Yes it returns N
-If No it adds children of N to OPEN

It simply returns node G!

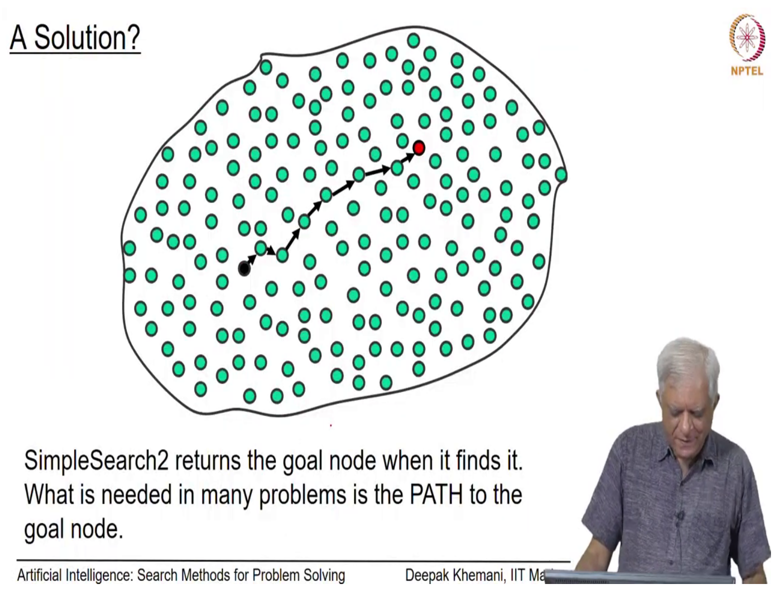
Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Ma

Now there is another problem with our algorithm our problem with the algorithm is that even if it finds a path to the goal, it does not tell us what the path is. From the way that the algorithm that we have written we have said if it is a goal then return the node N.

So, in this another instance of the simple search algorithm without any constraints is that you could possibly have found this path to the goal and then you would terminate, because this is a goal node. And what would it say it would simply say yes I found the goal node.

What we are really interested in is finding a path, we want to know that from S you should go to A from A you should go to D from D you should go to E and from E you go to it. So, that is the path we are interested in, so that is the next improvement that we will make which is to return a path essentially ok.

(Refer Slide Time: 27:31)



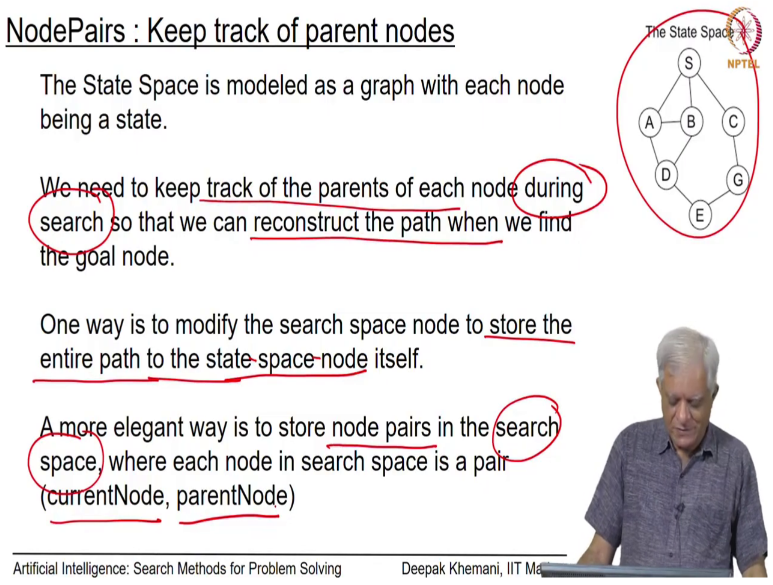
A Solution?

SimpleSearch2 returns the goal node when it finds it. What is needed in many problems is the PATH to the goal node.

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

So, abstractly speaking we just do not want to know that we have found the goal node, we want to know what is a paths to the goal node and that is what we expect our algorithm to return essentially.

(Refer Slide Time: 27:45)



NodePairs : Keep track of parent nodes

The State Space is modeled as a graph with each node being a state.

We need to keep track of the parents of each node during search so that we can reconstruct the path when we find the goal node.

One way is to modify the search space node to store the entire path to the state-space-node itself.

A more elegant way is to store node pairs in the search space, where each node in search space is a pair (currentNode, parentNode)

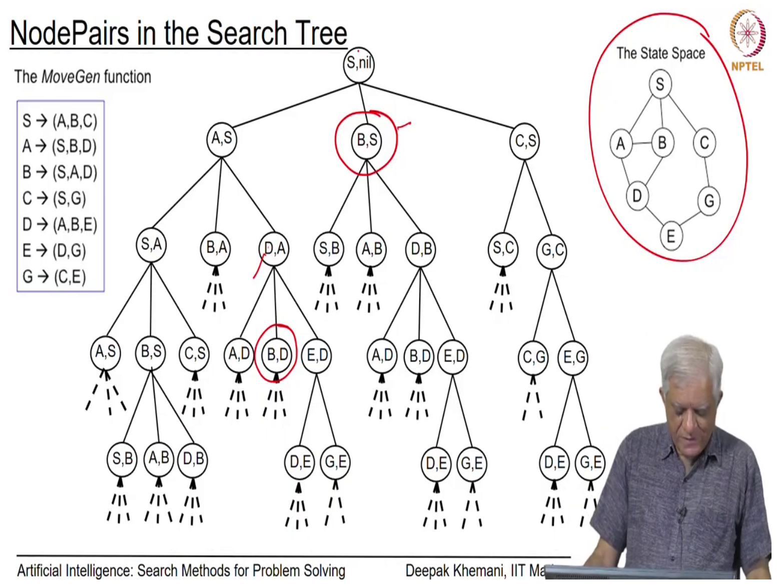
Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Ma

So, how do we get this path we use this nice mechanism of keeping track of where we came from ok. So, remember you must have heard about this old story this Greek story, where you know some as a punishment one young man was sent to the maze and it was expected that he will not be able to get out of the maze. But he was smart enough and he took a string or thread with him and he kept laying the thread where he went along. So, that he knew exactly where he came from.

We want to do something similar and the state space as we have seen is modeled as a graph and that is depicted here, that is the space in which we are trying to solve the problem. But we want to also keep track of the parents of each node, where did that node get generated from do you and we want to do it during search. So, in the search tree we want to keep track of parents, so that we can reconstruct the path when we find the goal node essentially.

One way to modify the algorithm is to store the entire path to the state space node. So, to the state space node store the entire path in the search tree, but that is a little bit inefficient. What we would do and what is more elegant is to store node pairs in the search space. So, not in the state space this we are talking about a search tree, where each node in the search space is a pair and the pair represents a currentNode and the parentNode.

(Refer Slide Time: 29:41)


NodePairs in the Search Tree

The *MoveGen* function

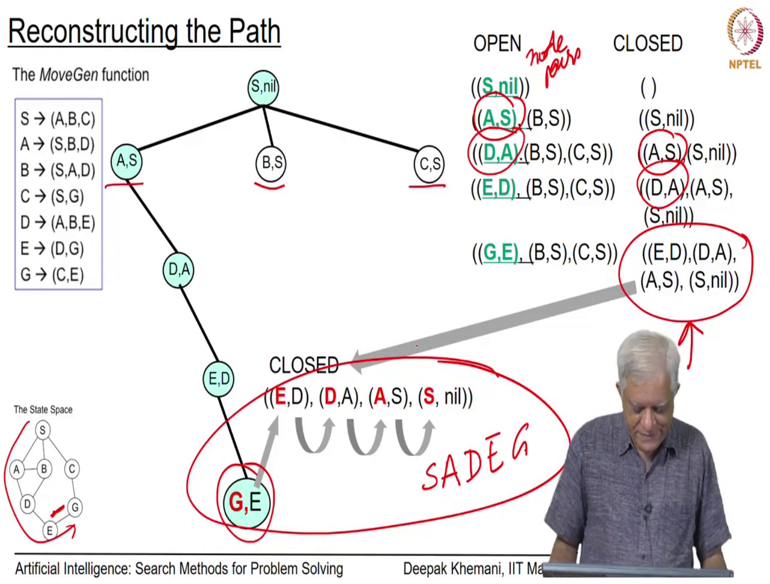
S → (A,B,C)
A → (S,B,D)
B → (S,A,D)
C → (S,G)
D → (A,B,E)
E → (D,G)
G → (C,E)

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

So, we know who was the parent of the currentNode and we can trace the path back to the source node essentially. So, our modified search space the state space is still the same as you can see here, but the search space has got modified. Now instead of one node one state being one node in the search space, now we have a pair of states as a node in the search space.

So for example, this pair which is B comma S says that this node is B, but it is parent is S. Likewise, if I look at this node this says that this node is B and it is parent is D here. So, we know that if we are at B where did we come from D if we are at this B, then we came from B. If we if we are at this B then he came from S and then we can you know once we have this information we can reconstruct the path.

(Refer Slide Time: 30:42)



So, let us see how that happens. So, let us say in our last version of the algorithm this is the algorithm that we had found, this is a path that our algorithm found except that in the search space now we have these pairs of nodes. The graph is the same as we saw some time ago. The notation has changed both OPEN and CLOSED contained NodePairs

So, we start with the node pair S nil and then when we add children we as you can see we can add A, S or B, S or C, S and so on and so forth. The same node pairs go into the closed list as well. So, when we remove A, S for example, we put it into CLOSED then when we remove D, A for example we put it into CLOSE. So we just so this is a nice elegant way of doing this whole thing.

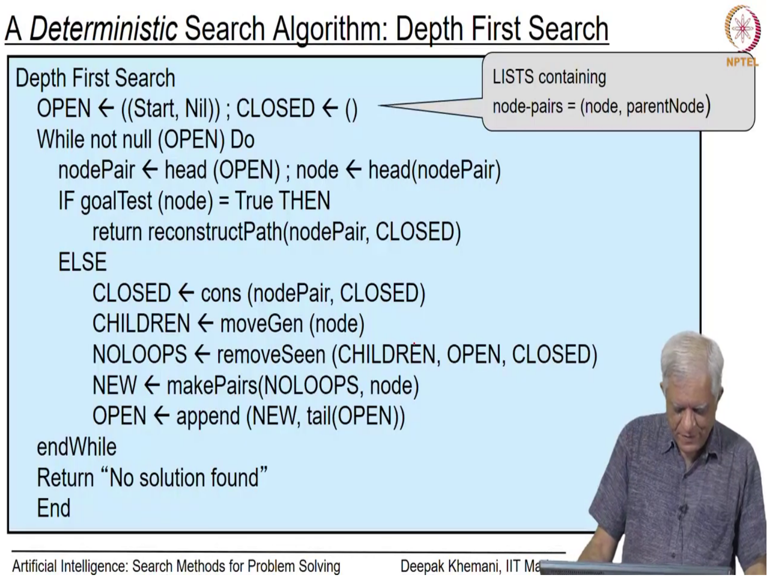
At some point of course, we have found the goal node we have to modify our algorithm a little bit to extract the goal node from the node pairs every node from the node pair and test whether it is a goal or not. But that we will specify in a little bit more detail in the next class. But the general idea is that you have found the goal node and now you want to reconstruct the path.

So, what do you know that you reach the node G and you came to G from E essentially. So, this is what we know essentially that we came from here. But now thanks to the fact that we have stored these node pairs, we can reconstruct the path. So, we take this close list which is at the point when we find the goal node G at that point close contains these four pairs E, D D, A A, S and S nil.

And now we can simply reconstruct the path, we know that parent of G is E. So, in the close we look for the node where E is the current node and from there we know that the parent of E was D then we looked for the parent of D then we look for the which is A and then we look for the parent of A which is S and then we see that S does not have a parent. So, we have path here. So, as you can see this is the reverse of the path and then we can just simply copy it from here S A D E G and that is the path that our algorithm has found.

So, by adding this node pairs to our representation, we have facilitate the process of reconstructing the path. Then algorithm for reconstruction we will see in the next class in a little bit more detail. But the basic idea is that because we have stored everything in this closed list every pair of parents, we can use that information as we have seen here to reconstruct the path. Later on we will see that instead of storing these pairs another way would be to store back pointers and but that we will see later essentially ok.

(Refer Slide Time: 33:58)



A *Deterministic* Search Algorithm: Depth First Search

```
Depth First Search
   OPEN ← ((Start, Nil)) ; CLOSED ← ()
   While not null (OPEN) Do
      nodePair ← head (OPEN) ; node ← head(nodePair)
      IF goalTest (node) = True THEN
           return reconstructPath(nodePair, CLOSED)
      ELSE
           CLOSED ← cons (nodePair, CLOSED)
           CHILDREN ← moveGen (node)
           NOLOOPS ← removeSeen (CHILDREN, OPEN, CLOSED)
           NEW ← makePairs(NOLOOPS, node)
           OPEN ← append (NEW, tail(OPEN))
   endWhile
   Return "No solution found"
   End
```

LISTS containing
node-pairs = (node, parentNode)

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

So, I think we will stop here, in the next class we will look at this algorithm in a little bit more detail this particular algorithm is called Depth First Search and it also removes the non determinism that we had earlier.

Because we had said that our algorithm picks some node we did not say which node. Now we will make our algorithms more deterministic and then try to explore what are the different possible ways you can search the space. So, we will stop here and we will take this up in the next class which will be on a Deterministic Algorithm to do whatever we have discussed. And then we will move on to variations of those algorithms so see you next time.