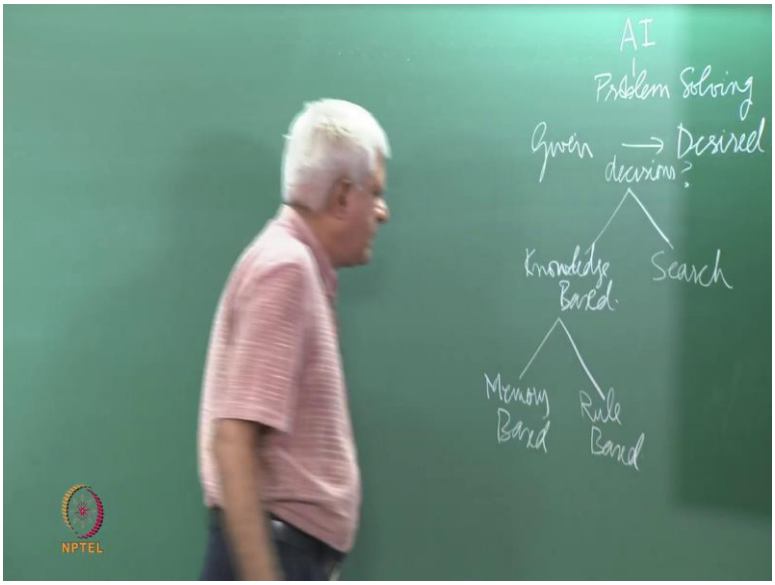
Artificial Intelligence
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 06
State Space Search Intro

(Refer Slide Time: 00:26)

So, today we start with first technical topic and that is state space search. So, let me begin by first putting this whole thing into perspective.

(Refer Slide Time: 00:53)



I will just write this ((Refer Time: 00:49)) these. So, we are looking at one aspect of A I, which we will call as problem solving. As we said, there are many other aspects that we will not cover in this course for example, learning or using experience or reasoning with knowledge. We will focus on problem solving and by problem solving we mean that, the agent is in a desired, is in some situation and wants to be in some desired situation. So, given to desired situation and the task of the agent is to make a series of decisions or a series of moves, which will transform the given situation to the desired situation.

So, the task is to find these decisions, and that is the task we will be addressing. Now, in general, there are two approaches to this. One is the one that we are following in this course, which is search. Our first principles base approach, which means that, you have some kind of representation of the domain. And you simulate, what would happen in the domain as the consequence of your decisions. And then try to find those decisions, which will achieve the goal that you are aiming at. The other is knowledge based, in which you are trying to exploit knowledge, which has been accrued by some means essentially.

So, in general of course, knowledge says derived from experience, either your own experience or somebody else with experience and convey through in a form of books and lectures and things like that or stories. But, given the knowledge, we want to up exploit

those techniques. The knowledge base techniques themselves could be classified into two kinds, which is memory based and rule based. Memory based techniques try to exploit stored experience directly essentially. So, this area is also known as case base listening.

And in fact, we have a separate course, which will be offered next semester called memory based listening in A I, which will focus entirely on this approaches to problem solving. And what the memory based approach does is that it, stores experience and what do we mean by experiences. Each case as we call it is the pair, made up of a problem description and a solution that work for the problem description. So, every time is solve a problem, he could be a human being solving a problem and put in the case into the memory and so on.

So, every time is solve a problem, you store the problem and the description and the solution description into this case base. And when a new problem occurs, you retrieve the best matching problem description and use or reuse the solution that was told along with that essentially. So, that is the way, ((Refer Time: 04:36)) of using experience, that is how, we learn many things for example, how to make ((Refer Time: 04:41)) or something like that or how to solve some kind of integral equations, whatever the case may be. The rule base approach is on the other hand, does not use experiences directly, but realize on rules or you might say negates of knowledge extracted from experiences.

And very often a rule base approach needs a human inter mediatory to convert experience into rules. So, the standard approach that was forward an expert system, which we is we talked about dendral and mycin and prospector. These were rule base systems and the way there they were built was that so called, knowledge engineers went and spoke to domain experts and try to elicit from them, the knowledge that the use for solving their problem. It could be diagnosis of whatever the problem was and try to put that knowledge in the form of rules essentially. So, we have knowledge base method as suppose to search base method.

And we will be largely exploring search base methods in this course essentially. Of course, it is not as a search base methods or divide of knowledge, because we have to

still model the domain. We still have to create the platform on which this search will happen. So, that extent we do need knowledge for search base methods essentially. As we move along in the course, we will see that search by itself is not a very efficient means of solving problem. Because, we will see that, we run into something call ((Refer Time: 06:29)) explosion, that as number of possibility that we have to explore with be too high essentially.

So, very soon in fact, in the next week itself, we will introduce some element of knowledge into search. We will try to see, how search can be guided by some kind of domain knowledge and we call this heuristic search, and the knowledge that we will use is called heuristic knowledge. After that, we will move towards standardizing representation of the domain. So, I to when we begin with, we will just make some ad hoc assumptions has to how the domain is going to be represented, because we will be focusing more on search. But, towards a letter half of the course, we will also try to focus on standardize behave to representing domains and you will see logic base approach for doing that.

So, we will do little bit about logic representation and listing as we go along. Now, search can be of course in many difference spaces. We begin with, what we call is state space search. So, let us a topic today, state space search essentially. So, let me show you this puzzle, which I might have spoken about some time ago. So, you know this is a rubrics cube. And it is for this six faces and you can move this face or you can move this face or you can move any of this six faces by multiples of ninety degrees.

And so that is a problem that you have solve, what is the problem you have to solve that, unlike these faces that you can see here, which have mixed up colors, you want all the spaces to have only one color. So, every space should looks something like this. So, this space is green color and if I can somehow make rest of the spaces of one color then I can say I have solved the problem. So, the situation is that I am given this rubrics cube as it is and the goal is to solve it, meaning get all faces to have the same colors essentially.

Now, I am sure you must be familiar with this puzzle, it was a device in the mid seventies, late seventies by architect called rubric. And it became quite a craze in those

days; people would spend hours and days together trying to solve it essentially. Now, you I want to, first thing I want to do with this is to highlight the case that, between these two approaches the problem solving, search base and knowledge base essentially. If I were to give someone this rubrics cube, who has lots seen at before and ask that person to solve the rubrics cube. She or he would essentially do it trial in error, he would try to see, what to do with this and what to do this and try to explore a sequence of moves, which will solve the puzzle.

Now, this puzzle is interesting it looks simple, because as you can see, if you look at the top face, it has got three faces, the top faces is solved, everything green here and these sides, cube let us are of one color essentially, I need to solve the rest. The trouble is once I have solved this top face, I can only move this bottom face and there is nothing yet, no other movement I can make essentially. Remember there are six spaces, I could have moved any of those six spaces, if I am not going to disturb this, I can move only this. But, if I move anything else for example, if I make this change then you can see that I am disturbing this faces essentially.

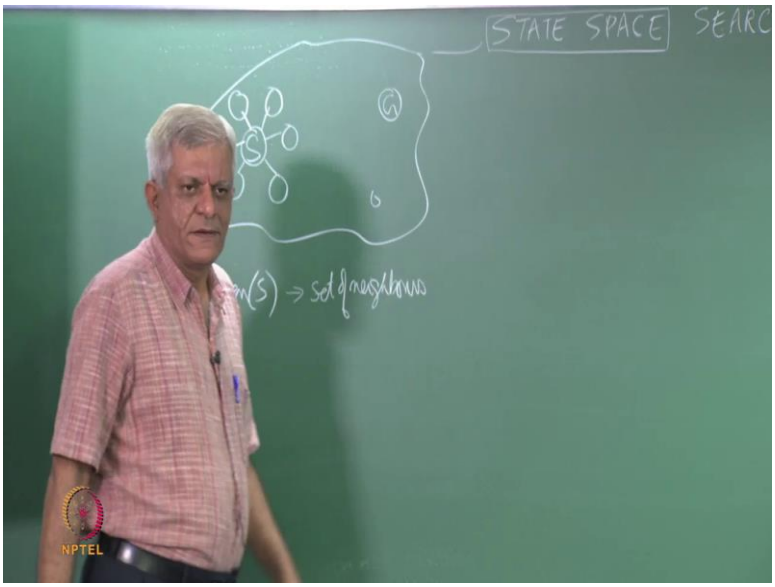
And in the process, if I am trying to solve the rest of the cube, I end up disturbing this cubes that is why, this problem is an interesting problem and has been given quite a bit of a study. Now, how many of you know how to solve the rubrics cube, only one, two, three, it is, so some people at least. So, noticed that the question was how many of you know how to solve the rubrics cube, which means that, if I want to give you this cube, you with virtually without thinking do a sequence of moves within then the making this solve.

So, you are exploiting these techniques, which we call as knowledge base techniques essentially. So, it could be there this knowledge has been given to you in a form of set of macro moves that to achieve this into this corner, this is the sequence of moves and that kind of thing essentially. So, that is the knowledge base approach, this very once you have the knowledge, it is the very simple way of solving a problem, whether trouble is knowledge has to come from some were. And what happens, when you have a new problem to solve.

Then there is no knowledge to fall back on, it is then that we need to take records to search base method. So, we call search base method, just first principle methods that you want, you do not have any carry over from the past. You are a given a problem and you have to model the problem and you have to find the solution, we have solving the problem essentially. So, we does not matter what the problem is, you can take any problem and we will try to solve the problem.

So, one of the thing that we want to do in A I is not to solve the rubrics you by itself, but to find is problem solving strategy on mechanism, which will solve virtually any problem, which can be posed as a state space search problem. So, what do you mean by state space search, that I am given the cube, this particular configuration of this cubic cube is a state. The desire configuration of the cube is another state and in between there are thousands of other states essentially.

(Refer Slide Time: 12:40)



And I need to find a mechanism. So, this state space is the set of states. So, let us say we draw it as the set like this and I will use S for a start state or the given state and G for a goal state. So, each of the anything state and there are other states as well, some well you know, which we. Now, how all these, how do we transform the start stage to the goal stage, we have to have a sequence of moves essentially. So, what are the sequences of

moves in the rubrics cube? So, if I call this is a top face, I can say rotate the top by 90 degrees or rotate the top by 180 degrees or by 270 degrees.

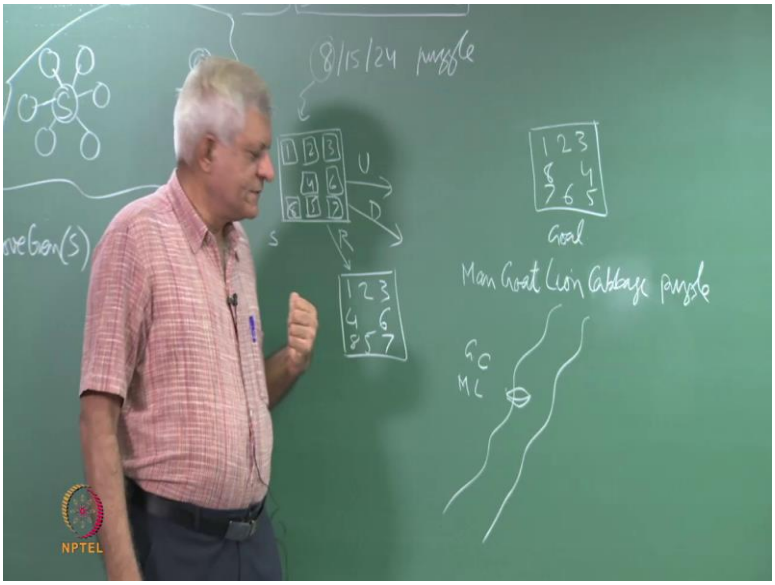
So, these three moves are possible on this space, more than that you will come back to the same stage likewise, three moves for every faces essentially. So, 6 into 3, 18 moves I have, which I can transform this state into a different state essentially. So, we will model this process as moves, which will transform you take you from one state to another state. So, in this case I have drawn 6 states. So, if some problem has some 6 moves, you can move in moves you can make in given state then the graph would look like that.

Now, you can see, that the once you incorporate this motion of a move, what is the move? The move is something an action, which takes you from one state to neighboring state essentially. And even any state, there is the set of neighboring state that you can move to, that is define by the domain essentially. So, we will assume that we have a function called Move Gen function. It will take a state as an input and it term set of neighbors.

So, what we are trying to do is to separate the domain from the problem solving strategy. So, what I am telling you at this moment is that, choose a domain and take this as a small exercise that you should try out yourself, choose some domain of interest that you have define, how to represent the state and define the move gen function. The move gen function is the function in the programming sense, that it takes as input a state and returns the set of neighbors for that state essentially.

So, that is one thing that we want, so let us take another small puzzle. So, if we look at the A I text books, you will find that they are sort of sprinkle with puzzles, because puzzles are easy to describe, easy to represent, well defined and you know, you can do work with them. So, there is you might say a younger cousin of this rubrics cubes, which is on a flat, two dimensional puzzle, which is called things like 8 puzzle.

(Refer Slide Time: 16:06)



So, we are 8, 15, 24, in fact, any n square minus 1 puzzle. So, 8 is 3 square minus 1, 15 is 4 square minus 4 and so on. So, the 8 puzzle in particular is a puzzle any must have seen this, which is got 8 tiles that is why, it is called 8 puzzles and these tiles have labels on them. So, let us said number 1, 2, 3, 4, 5, 6, 7, 8 and this is let us say the start state and the goal state could be something like 1, 2, 3, 4, 5, 6, 7, 8, so I am not going the tiles state, goal state.

Now, in this puzzle I can have, as we can see basically, four different kinds of moves and some people can to think of it as a movement of this blank, remember this, so what is the puzzle you can slide. So, there is a blank square here, blank location here, you can slide this tile here or you can slide this tile here or you slide this tile here, essentially. And then in the process you will create the new blanks. So, for example, you could do something like this, 1, 2, 3, 4, 6, I could do this. So, this could be a move that I could make and you can give a label to this move.

So, you can either say that you are moved four to the left or you can say that, you have moved the blank to the right; they are just equivalent face of saying the same thing. So, let us say that we call this right, which means I have move the blank to the right. And then you can see that, I can have an up move and I can also have a down move. So, I can

generate three successes states for this particular stage essentially. The whole question is how do you represent this, so I will not go too much into that detail and I will ask you to choose an interesting problem, describe the state representation and construct the move gen function.

So, in this case the move gen function will return this state and this state, so it will return a set of states. We will not give too much of importance, so the names of this moves at this moment, but later when we talk of planning, we will be more interest than that. So, we are interested in the set of states. Let me take one more example, which is well known river crossing puzzles. So, you must have heard about this river crossing puzzles, which basically says that, in most puzzles there is a boat, in which only two things can go it one time, and there is a whole set of entities on one side of the river, and they have to be transport to the other side of the river, and there are some constrains about what can to go together and what cannot go together.

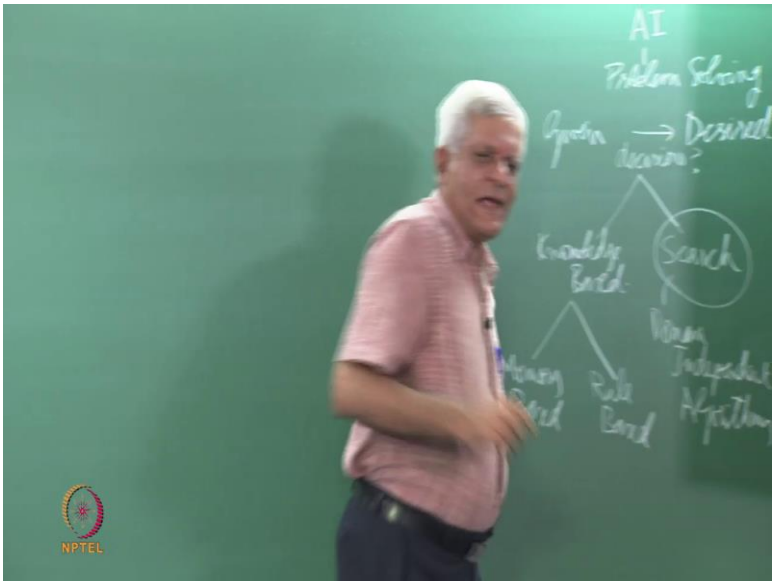
So, you must have heard about the machineries and the ((Refer Time: 19:37)) puzzle there, three machineries and three ((Refer Time: 19:39)) and they have to cross the river and the boat can take only two people, how do they cross essentially. So, as a simply version of that is the man, goat, lion problem. So, what is this saying, this is saying that, I have some river and on one side of the river, there is the man, I forgot one more thing that is the cabbage. So, there is a man, there is a goat, there is a lion and there is the cabbage and there is a boat of size two and only the man can rode the boat.

And the man needs to take everything on the other side of the river, but he can take only one thing at the time. And the difficulty is that, if he leaves the goat alone with the cabbage, the goat will eat the cabbage, and if we leave the lion alone with the goat, lion will eat the goat. So, he does not want that situation, how can he get these three positions across on the other side. You must have seen this recent movie, which these some boy and lion in a boat together stuck for across the ocean.

So, obviously this is the very simple puzzle, it is not hard to solve, though the question is not that the addressing here. The question is that, how can we pose this problem or this problem or the rubrics cube as something that we can apply some general purpose

methods to solve essentially.

(Refer Slide Time: 21:46)



So, our goal would be to do something called our search should be domain independent. The algorithm that we are going to look at should not depend upon the domain that we are trying to solve the problem in. So, it could be the rubrics cube or it could be one of these puzzles or it could be a scheduling problem or it could be something different essentially. If we can abstract away from the domain, then we do not leave need to look at the domain and what do a mean by obstructing away from the domain.

In our case, the first thing is to design a move gen function for the domain, which says that given a state, this function should tell me, what the neighboring states are or should a turn the set of neighboring states to be essentially. And obviously, when I say given a state, I mean there is some representation of the state that you have to be concerned with essentially. So, how can we represent? So, these of course can possibly be represented as the 3 by 3 array or something in that, and it might be useful to think about it like that, but I will leave it for you to decide.

Let us talk about this problem, how do we represent this man, goats, lion, cabbage problem? So, the solution is simple I am sure you know it, we should first take the goat

then come back with the boat, take the lion, bring the goat back, take the cabbage, come back and take the goat. So, you know likewise the mistake is in cabbage, all of these puzzles have solutions, which are not so hard to find, can we write a program to do that is the question, without having to write a programs specifically for this puzzle, can we write general purpose program still solve puzzles like these essentially.

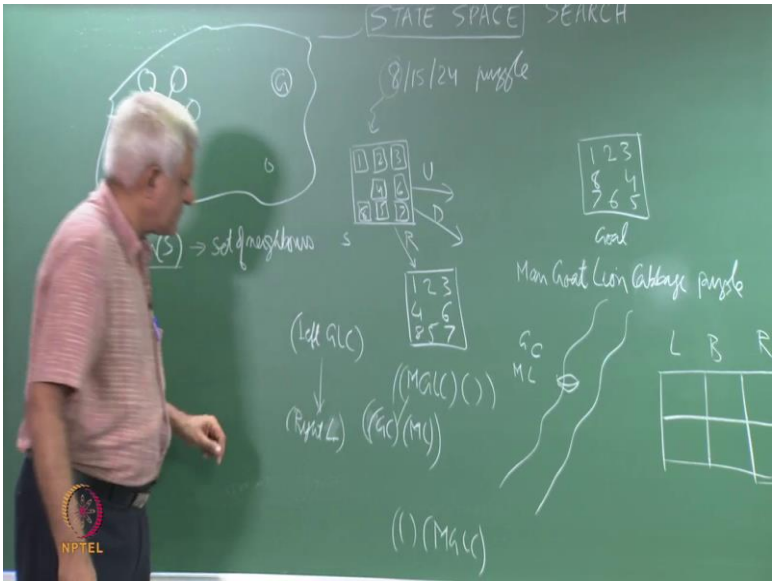
So, there are many other puzzles, the water jug problem you are given a jug 5 litre jug and a 4 litre jug or you have given a 4 litre jug and a 9 litre jug, can you measure out 6 liters, these kind of problems essentially. So, how do you represent this? So, I want some input from the class, some suggestion how can I represent this problem? How can I, what is the state representation and secondly, what is going to be the move gen function. So, the move gen function will be obviously something, which is operate upon the representation that we choose. A three cross two arrays.

Student: ((Refer Time: 24:24))

Why three, why not four.

Student: ((Refer Time: 24:31))

(Refer Slide Time: 24:40)



So, you are saying that, I have an array like this, and this stands for left show, this stands for the boat and this stands for the right show is that what you are saying. So, it is an array of what, because there is more than one thing which is possible on each location. Let us say, array of list show, so it is an array of list essentially, by how do you write a move gen function. So, what is the initial stage, the initial state is everything is in this square, why do we have two, why do we have second row.

Student: ((Refer Time: 25:21))

Then you should have four I think. So, that is when possibility I think, but when you create a representation, you should also worry about the algorithm that you write on the representation, because very often represent the choose, the representation that you choose is helpful in sort of writing algorithm for in our case, we are interest in this move gen functional algorithms. So, let me make another suggestion, which is that I have a list of two lists. So, a list of two lists, the first list contains things which are on the left bank and the second list contains things which are on the right bank.

So, I could choose that essentially. If I choose that list then my list were look like this G, this is my given start state and my goal state is. But, the question is how do I write the move gen function? So, that is the thing that I am trying to derive at, what representation is goat and what is known. Now, how would you write a move gen function for this representation, you would have to first search the state. So, this is the state representation, you have to first. So, will we have a make, made an implicit assumption here, that the boat, we are not talked where the boat is.

And we can effort to do that, because we make an implicit assumption that, wherever the man is the boat is there essentially, because the boat cannot go by itself some were. So, having made that assumption, which is hidden into some were here, how do I write a move gen function. I will first need to inspect this representation, this list of list find out where M is and then copy something from there, out cuts something from there other along with M and take it to the other side, something could be nothing also essentially.

So, which means I can go from this state to this could be even successes state

essentially. The question is how do I write this successes state. So, I will not spend too much time here, but may be will, we will look at one more possible representation. Now, one thing that you would observe is that, this second list is ((Refer Time: 28:14)) we do not really need the second list essentially. If we know that they are these only four characters around, the first list is enough to tell us what the state is, so that is one thing essentially.

And what was the first list represent it to represent the entities, which are on the left bank or something like this. So, let me suggest another thing, which is that you do not represent things which are on the left bank, but you represent things which are on the side of the board and you will also represent, which side the boat is essentially. So, which means inside of this, I would have something like this. So, let us say I do not represent the man; I am just trying to explore different possibilities, I just say left G L C and this is my representation.

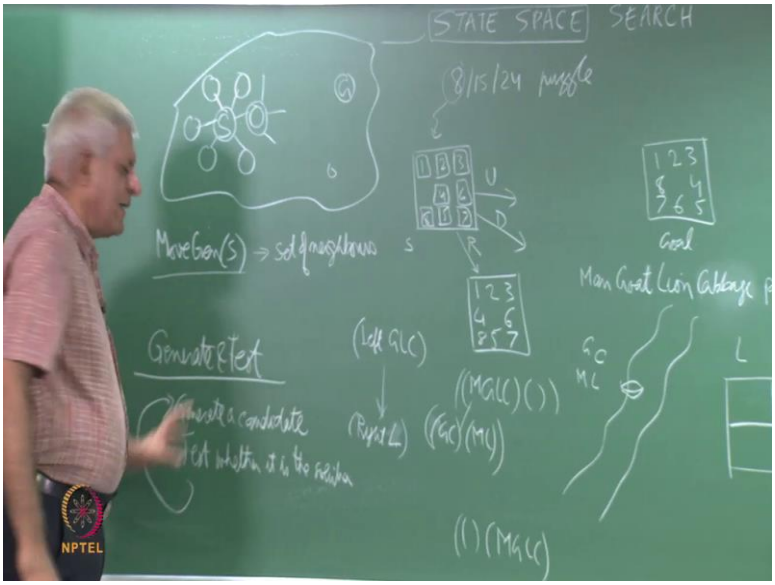
So, what I mean by this is, their boat is on left hand side and which means the man is also on the left hand side and along with the man, there is the goat and the lion and the cabbage all of them. So, the advantage I see of this representation is at my move gen function would be simpler, at least that is what I feel. Because, it is going to be symmetric, whether I am going from left hand side to right hand side or right hand side to left hand side, the operations that I will have to do would have to be identical, which means that I will delete some elements from my list and create a compliment set, which will be the new list essentially.

So, for example from this, I will delete G and take it to the right hand side. So, this could be a move essentially or if I want represent the same move, I will delete L. So, I have taken the lion to the other side essentially, it not a very good moves of course, because the goat will eat the cabbage, but the state space algorithm has to search through all possibilities. So, we are trying to look at that essentially.

I would leave it as an exercise for you to try out different representations for this and see, and actually write a program, which will take a state as an input and return the set of neighboring states as an output. So, there are three moves possible in this case. In fact,

there are four moves possible, the man can go alone or the man can take the lion or the man can take the goat or the man can take the cabbage. So, it is the written this four sets as my output to the function and this move gen function is a function which will allow it and navigate the state space.

(Refer Slide Time: 31:25)



So, I will apply the move gen function to this state, I will get a set of neighboring states. I will go to the neighboring state and see if that is the state I was interested in. Otherwise, I will alive apply the move gen function to that state. And in this process, I will navigate the state space, trying to reach the goal state that I will interest than essentially. So, this general strategy that we are going to follow is has a generic name, which is called generate and test. It says generate a candidate in when we says, candidate we mean the candidate state and test whether it is a solution.

And we put this into a loop, and this is the high level strategy that we are going to refine over the next few weeks generated. Generate a candidate and see if we that is the goal say that we interested in our task is to explore this state space in search of the goal state essentially. So, will keep trying out new states and seeing whether that is the goal state or not, how do we see whether there is we are in the goal state or not, we need another domain function and that domain function, we will call as a goal test function.

So, goal test will take a state as an input and output either yes or no. It will tell me whether the state that I am looking at is a goal state or not a goal state. These are the only two functions I need, which know need to know anything about the domain at all. So, once I have return this two functions for a domain, I can practically forget about the domain, I can I do not have to worry about whether I am solving the rubrics cube or whether I am solving this eight puzzle or whether I am solving the this river crossing puzzle, does not matter.

I have a move gen function, I have a goal test function and I would not find the sequence of moves which will taking to the goal test and so, this is a very abstract problem essentially. Incidentally, while we are at the rubrics cube, only very recently and by recently I mean in the last five to ten years have search algorithms been able to search for optimal solutions of a rubrics of a problem like this essentially. And by optimal solutions, I mean the shortest, the smallest number of moves that you need to make essentially.

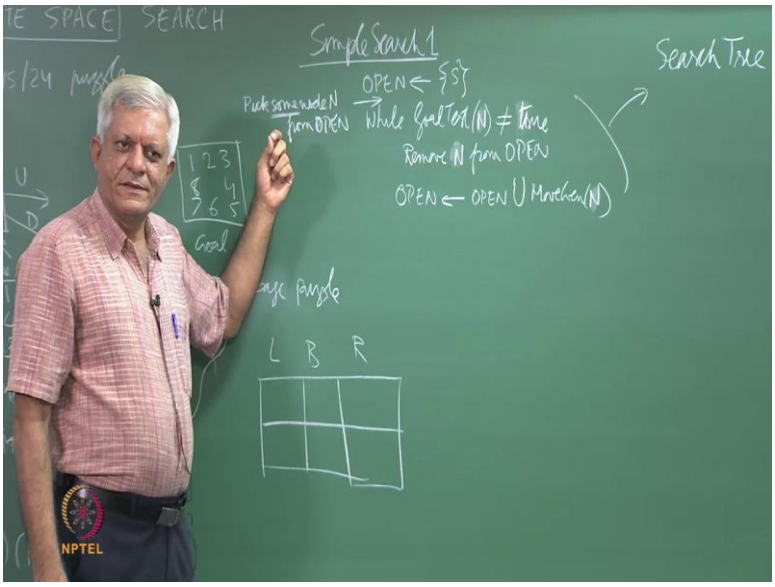
If you follow the message that somebody has thought you would says, take the top layer first, then make the second layer, then get this bottom corners, get the bottom squares and then get the orientation. So, these kinds of macro moves, they will not necessarily give you the shortest solution essentially. So, they will solve the problem, but they will not give you the optimal solution. So, this is something that we had also mentioned about human problem solving, that we do not necessarily always try to optimize things, as long as we can solve the problem reasonably well, then we are happy with the solution essentially, which is where the knowledge base techniques are so important essentially.

So, the move gen function was this would be an interesting exercise. As you can see there would be 18 possible moves for every given state, corresponding to the 3 moves for each space, so there are 6 spaces. So, any of those 3 would give a move gen functions. So, we can see the state space for this is quiet large essentially and whether I can leave it as the small exercise for you to see, how many possible different configuration there are to these puzzle.

So, whatever done we have abstracted away from the domain and our, we are only going

to work with these two functions. So, the algorithm that we will write now, you will just use this functions and do not worry about what the domain is essentially. And what is that idea that once we have design these algorithm, then we can plug in any domain and it will solve problems in the domain for us. So, there is this generality about our approach essentially.

(Refer Slide Time: 36:07)



So, let us if find this algorithm a little bit and we will call it simple search one. So, let us work with sets for this moment, we have to see when we generate this successes or neighbors, we have to store there somewhere. And we will store them in a set, which is traditionally has been called as the opened, open list or open set essentially. So, we begin by saying open gets start state. So, we put the start state in the open and though the following, while or instead of y it is say true, remove, I am using the set union operation, because I forgot one step and the step that is going to be here is pick some node N from open.

So, let us call this N now, because we are not only talking about the start state, but of any note that we pick from this list, this set open. So, what is this, this refinement of this algorithm is s, I create this open set, set called open, in which I put the start state to begin with. And in general, my algorithm says pick some note, so we are calling it on note,

because we are set of already started thinking about these as a graph over which we are searching.

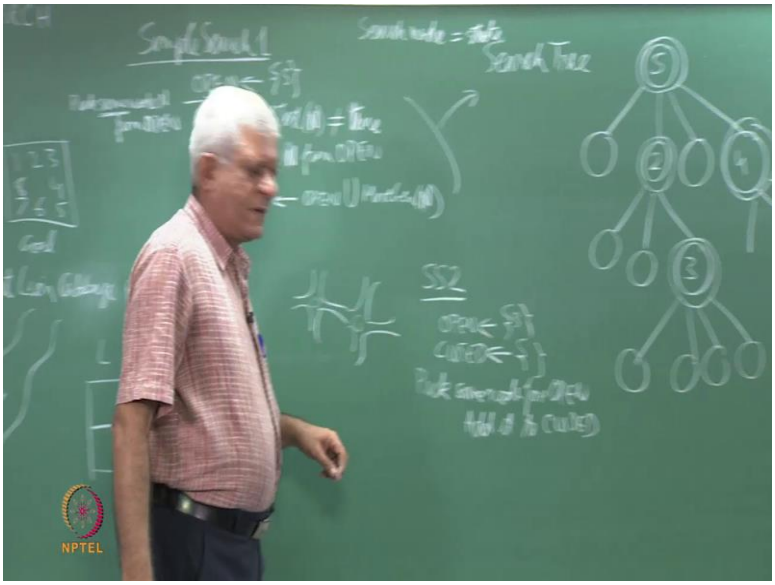
So, each state is a note, take some note N from open, test whether it is a goal state, so the generate and test idea. If it is true, then we should return, we are found the solution; if it is not true, we will remove this N from open, and add instead the successors of open or the neighbors of the open to that. So, I will keep adding things to my open list and keep picking some elements from that test ((Refer Time: 39:24)). So, that is the simple basic idea we start of it essentially, what this is ((Refer Time: 39:37)) is a search tree will start by criticizing this algorithm very shortly, because this is by no means an interesting algorithm.

But, it gives as a basic idea for what we are trying to do, what this algorithm does it generate the search tree. So, one thing that I should sort of clarify is that, even though we see the state spaces as a graph, because of every move we can go every state you can go to the neighboring states. So, every edge is the move from one state to a neighboring state and the whole state is a graph. And essentially, what we are trying to do is to find a path from a start state to a goal state in this graph essentially.

So, it is basic into the graph search algorithm that you must have encounter then some other in this thing. The only difference here is that the graph is not given to you, that nobody says that, this is the entire graph for the rubrics cube for example and then finds the path essentially. You have to generate the graph on the fly and that is done, that is what generates the search tree essentially. So, we can visualize what is happening with this algorithm.

Now, notice that, there is the very, I do not know whether you can read this, pick some node N from open, so this some node is there N. A considerable amount of time would be spent on refining this notion of some, because it really critically depends upon this some has together your heading towards a goal or not heading towards a goal essentially. So, will spent some time on that, but as of now, we will has say that, we are not specified it, so you some criteria to pick a node from open.

(Refer Slide Time: 41:28)



And so, what is the search tree we are generating. We start with some search, the start node S. So, we will use state versus node interchangeably at least for now. So, search node is equal to state, the same representation. We will move away from this in a movement. So, the first, the route node is a search start state or start note and we inspect that node, route note what do you mean by inspect, we apply the goal test function and we see, whether that is a goal.

Because, sometime the problem may not need anything to be done essentially you know, it is somebody gives you solved cube and says solve it, you will say here, it is solved already essentially, you do not have to do anything, but so you test whether it is a essentially. So, we inspect the start state of the route note. And let us say that, this double circle stand for deleting it from open. So, what I am drawing here is open list, I had S in open and then I delete it, but I replace it with some successors of S. So, my open has go now got four, four successors.

So, for example, in this situation I would have added three successors, because of the 3 moves I can make or in this situation, I would have added four successors. So, depending on the state, some numbers of the neighbors would be generated and they would be added. Then my algorithm says, pick some node from open, we have not specified which

nodes. So, let us say we pick some node and we generate add it successors two. So, this is the general process that our search algorithm is doing to follow.

We have this set of open nodes, we will pick one from there and see whether that is a goal node or not essentially. So, this actually reminds me of this in a moment, green this interesting story about Hercules and hydra in Greek methodology. So, as you know, how many of you heard about this story. So, Hercules fighting this many headed monster called hydra. And the problem is every time he cuts one head of the hydra many more appearance essentially. So, we can see, that is what our search algorithm is facing.

Every time, you cut one head or one node from the search tree, you get many more. So, as you can imagine this, the open is set is going and going and we have to find a way of solving this. So, he does not have to be this, the next one could be, so this is let us say, number 2, this is number 3, this could be number 4 you are not specified, which node to take, we will do that in a moment.

Student: ((Refer Time: 44:41))

So, we will start criticizing ((Refer Time: 44:51)) in a moment. So, his question is what stops you from going in a cycle. So, that is in fact, the first problem with our search algorithm, I would have about to ask you, what is wrong with this algorithm and that is the first thing, which is wrong. And so, if I take this for example, rubrics cube again, I can say make this move rotate right, this right face by 90 degrees. Then what are the moves available to you, out of the 18 moves that are available to me, one of them is this one, what if I choose this, then what if I choose this, then what if I choose this.

Then I have got in the cycle essentially and of course, you can imagine that there are hundreds of cycles hidden there, I can just keep rotating this that is another cycle or many other things are possible essentially, that is one problem with this. So, let us first address that problem. So, one way of thinking about this search algorithms is to imagine that, you are in a maze. So, you know ((Refer Time: 45:56)) have this mazes in olden times were there is some. So, you are at some corner and you can see four roads, you are inside some building, you can see some four paths going, you go to this path, then you

see another four paths and so on.

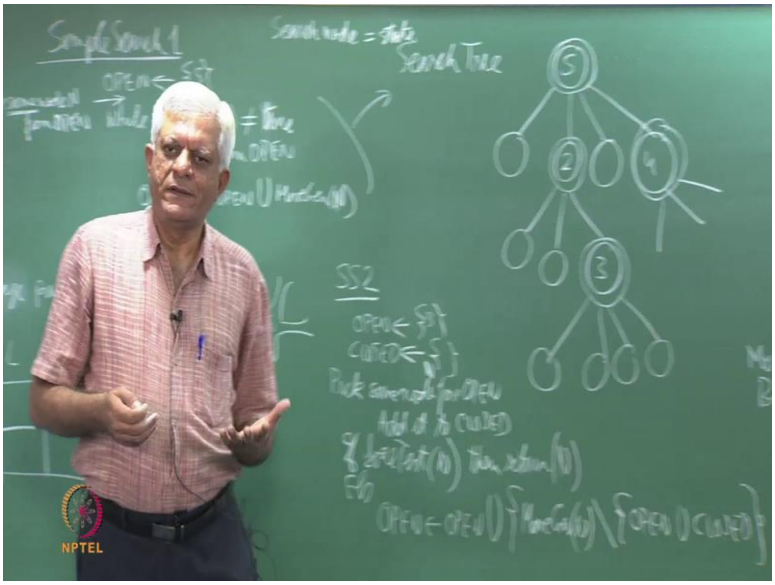
So, every, so this is like a node and this is like an edge. So, you move from here to here, you see more nodes, how do you get ((Refer Time: 46:24)) of a maze without getting lost without getting into cycles. There is again this are another interesting story from Greek times, I do not remember who was this person who solved it by taking the thread along with ((Refer Time: 46:41)) said that he will mark every path, if you see the thread again. Then he has he knows at his back to that, something some state that he has visited before. So, in terms of looping, what do we want, we want our search algorithm.

So, if I am going to start this particular state, I do not want to come back to this state by going through a sequence of moves. I have to somehow prevent that, which means one side waits this move from here, I should never visit this state again how do I do that.

Student: ((Refer Time: 47:17))

Mark that state has visited, that is the simplest way of doing it. And traditionally, what we do is we introduce another list and well I am, I already started calling with the list ((Refer Time: 47:40)) I will be calling with the list.

(Refer Slide Time: 47:47)



Another set let us see and we call it closed essentially. So, the algorithm is modified as this. So, this is simple search two, open as before gets the start state closed as is the new this thing, which start with the empty set and pick some node from open. So, the same and the new step, which is add it to closed, I can right it as a set notation, but anyway I am writing it in English. Then, so let us say that note is N as before, if N, I do not know whether you can read this I should write little bit more carefully, if goal state N then return N that is what I am saying that returns the state, else before add.

So, what should I add to my open know, union what move gen N minus, anything in closed. And let me prove upon that the little bit and say even anything in open essentially, because it is possible that two states may be generated by the same successors may be generated by two states essentially, two different states essentially. So, for example, I can start with this state, then go this state, then go this state and this state would have generated. This state would have generated the 180 degree move, where I could have gone here.

So, it would have put it in open or not that move let us say this one. So, let me start again, at this point on my open list is the 270 degree move, which will bring the green thing here. So, I have added it to open, but I have made this move, from this move I can make a 180 degree move, which will again bring the green thing here. But, I do not want to do that, because I have already added this in the open list essentially. So, you just think a little bit about that, from this I will remove the set open union closed. So, I will not add any note that I have generated before or seen before to open, my open will get new notes essentially.

So, this will stop me from going into loop essentially. So, there is this, if right what if move gen this, this thing does not add anything new. Then I am also getting into a loop or some kind essentially. So, I should have a test here, pick some node from open, if open has become empty then I should say, there is no possible solution. So, I am not adding it right now, but we will add it this thing, how can open become empty, if we have seen all possible states and the solution state is not available to you. So, for example, this puzzle that 8 puzzle that you know, all the permutations of these 8 tiles are actually into two disjoint spaces.

So, there is one graph, where you can navigate from one set of state to any other state, but there is a disjoint set, where you can also move from one set to another set, but you cannot go from this set to that set essentially. And that can be obtain with simplify you know flipping to two tiles essentially, they are in ((Refer Time: 52:30)) stage you can never get them back, likewise the rubrics cubes. So, this is the tricks with some people know you can take it out, take out this thing and reassembled the whole cube essentially. But, the thing is you can reassemble the cube back in twelve different ways.

So, it is possible that, this particular set of states contains my goal state, but another set of state does not contain goal state. There is one more problem with this algorithm or this algorithm, which we will. So, I will stop here now, we will when we come back, we will look at what this other problem is. So, I wanted to think about this in the mean while and when we come back, we want to look at this, what is the second problem in this state and try to address that. And after that, we will look at the behavior of difference strategy is when he says some, what are the options available to us and how do the change the behavior of our search essentially. So, we will do that in some sense in the next class, which is after five minutes.