Artificial Intelligence

Prof. P. Dasgupta

Department of Computer Science & Engineering

Indian Institute of Technology, Kharagpur

Lecture – 1

Introduction

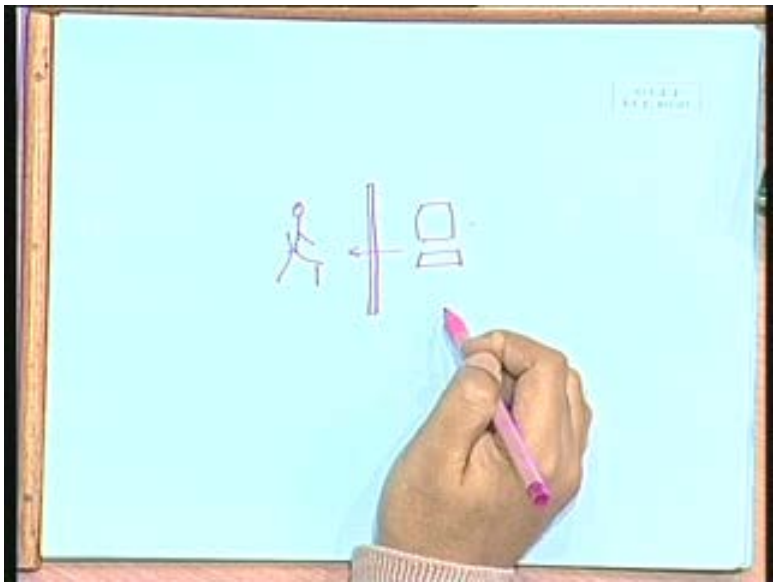
Welcome to this course. So, we are going to do this course in a slightly different way this time, because of the audio-visual way in which we are going to do this. Possibly, for many of you, this might be a new kind of experience. I am doing it for the second time, so, it is quite new for me as well. I like to teach more on black board, but let us try this out for one semester and see how it goes. So, the subject that we are going to cover in this course is artificial intelligence, and this is one subject which has which is really old. It has been there for a long time. Every subject has an up and down. There is some point of time, where there is a lot of hype about the subject. So, in AI also, there was a time when people claimed a lot and yet at that point of time, computation was not up to that mark to really make all that happen. That hype has gone down now and we now know better that what we can do and what we cannot do.

So, this course is to show you some of the things that we can do and also to show you what are the things that AI promises for the future. And because the hype is not there so much now, so, we can really go down to what actually are the feasible things, because we understand computation a lot better now, than maybe seventy years before. So, in this course, the course number is CS4002. And so, I will start with describing what is called Turing Test.

This Turing Test was proposed by Alan Turing in 1950 and the idea is like this- that you have a computer and it is being interrogated by a human via tele-type. So, what happens is that you have a partition and you have computer on one side, and you have a human being sitting on the other. And the human being interrogates the computer. It does not

know whether on the other side there is a computer or there is a human being. And if the computer can give the answers in such a way, that the human being is not able to determine whether there is a computer on the other side or a human being, then this computer passes the Turing Test. Now, this test was proposed in 1950.

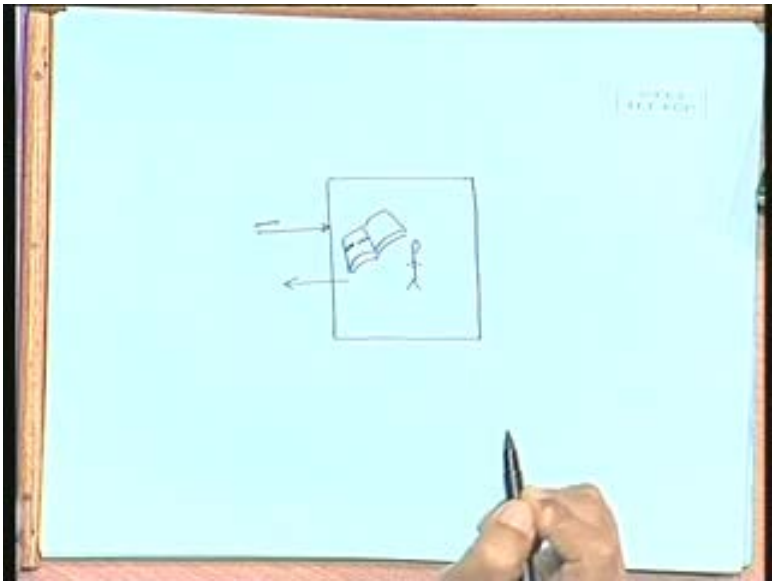
(Refer Slide Time 03:28)



At that time we had hardly any of natural language processing. We did not have much of logical deduction. So, it was quite a tall task to really expect a computer to be able to do this, but even at that time there were people who challenged this very idea. They said that just because a computer is able to answer in a human-like fashion, to deceive a human being, does not really mean that artificial intelligence is born. And so, they came up with what is called the Chinese Room Argument. And the Chinese Room Argument was something like this- that you have a room. Inside the room, there is an operator who does not know Chinese, but that operator does have a huge volume of Chinese literature. So, what does it have? So here, there is some Chinese people who are sending in some Chinese texts and what this person does is, it just looks up this table this document, and whatever is written on the other side corresponding to this text, it simply outputs that. And the person on this side believes that the person who is inside knows Chinese,

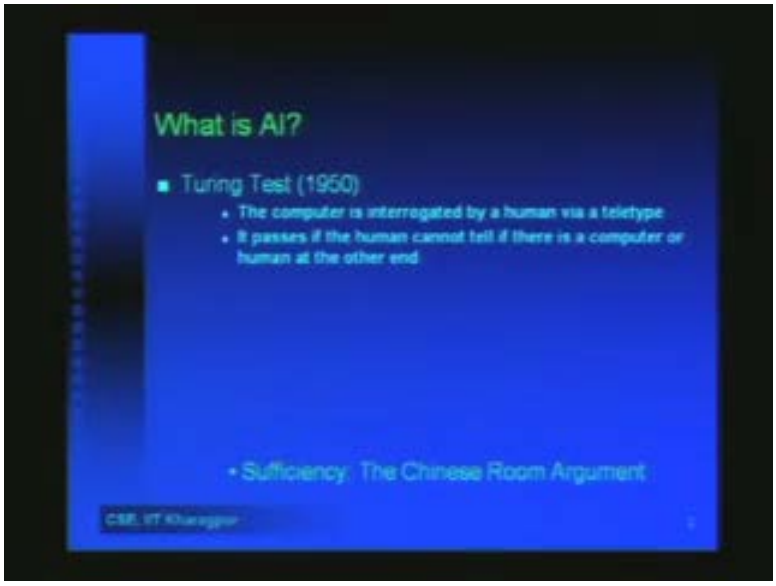
because it is being able to read in Chinese, and again produce something again in Chinese. But actually his person here does not know Chinese.

(Refer Slide Time 04:54)



So, it is just a matter of translating. What was conjectured at that time is, there is a lot of translations, automated translations, that computer should be able to do. And if you are able to do that smartly enough, then it is not impossible to be able to deceive a person. that is not That does not mean that the semantic understanding of the language is there with the individual.

(Refer Slide Time 05:36)



Even today, when you have a computer translating from one language to other, it does not necessarily mean that the computer is able to semantically understand the language. So, cognition and understanding is a different thing, as compared to simple translation from one language to another.

So, let us try to see what are the basic things around which AI has been built. The first thing that people tried to do, is was to address the problem of automated problem solving. So, people tried to see that how much can the computer aid us in our ability to solve problems. The first kinds of things that people came up, in automated problem solving, was search. And search is nothing but an efficient trial-and-error kind of algorithm. So, what happens is, I do not know actually how to solve the problem. The problem is there. I do not know the exact steps in solving the problem. Take for example sorting. Today, I know exactly- I know five different algorithms for sorting, right? If suppose you are the first person who has come up with sorting problem, then at that time, you do not know how to best write a sorting algorithm. So, what we do is, in that case, we adopt a trial-and-error method. We try something, see if we reach the solution. Otherwise, we try something else and reach the solution. So, very naïve way of sorting would be that you just randomly re-order the elements, and see whether they are sorted or not, right?

So, search starts from there, and then, when you have problems which are combinatorial complex, and it is no deterministic polynomial time algorithm, for solving it, then we adopt trial-and-error search techniques. So, what happens is, the kind of problems that we will address in this course will have enormous computational complexity. If you remember from your algorithms course, that there are problems which are that can be solved in polynomial time, others which are np complete or np hard, and still harder problems. And then, in search, what happens is, we come across what is called as space time trade-off. So, if you have more space, then you are able to solve a problem better. So, if you have less space, then the time complexity goes up. We will study in our search algorithms, that how we can balance out space and time for really big problems. And then, we will study what is called heuristics. That is, the use of domain specific knowledge, to accelerate the search process.

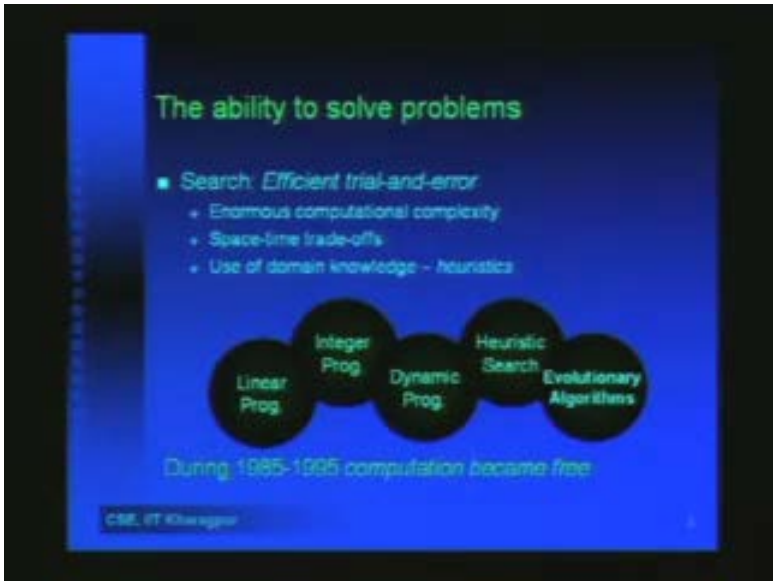
There has been several paradigms of automated problem solvings, which have evolved over the past. Some of them are as follows: We have linear programming; linear programming for a long time was not known to be in peak. Then, do you know who finally showed that it is in peak? There is a Karmarkar's algorithm. Karmarkar, who showed that linear programming is in peak. So, that was the one of the major breakthroughs in computation. Then integer programming. See, these are some of the topics which we do not study as part of this CS course, but we should study as part of this CS course, because these are also very well-defined and very efficient automated problem solving mechanisms. Integer programming basically addresses problems which are harder than linear programming, and if time permits, we may have a peep into this. We have already studied dynamic programming as part of your algorithms course. This is also useful for solving large combinatorial problems, which satisfy a principle of optimality. We will discuss that later.

Not all problems are amenable to be solved with dynamic programming, but some of them have a specific property, which makes them useful to be solved by dynamic programming techniques. Then, we will study a subject called heuristic search, which is

the first topic that we will study after this lecture, where we will see how we can use domain specific knowledge to aid this. And we will also study a little bit of evolutionary algorithms, which includes things like neural networks and genetic algorithms, which have come up more recently and are slightly different kind of search algorithms.

Now, the enormous problem solving complexity rendered most of these paradigms to be applicable to very limited domains when they were born and then what happened was, during 1985 to 1995, computation became free. So, if you go back to some time before 1985, that is, about the time when we were in high school and then we joined IIT for doing our B.Tech. So, at those times, you know, people used to queue up in front of the computer center from five o'clock in the morning, with a pack of punch cards, right, and those were punch cards which contained FORTRAN programs. So, people used to go and run them and then if you had printed 'd zero' instead of 'do' in fortran, then it means you queue up again next morning at five o'clock, right, and then, when we joined, the systems were not booting by themselves. So, you have to actually, there was a boot sequence. You have to There were no microprocessors at that time. So, there were the CPU was PCB, which had a separate ALU and separate components and you actually had to initialize all the registers, and set the program counter, and then set the computer running, and we used to crave for learning how to boot a computer, right. So, from that, to sometime, if you look at it, in 1995, there is an enormous change in the amount of computation power that you have on the desktop.

(Refer Slide Time 12:12)



So, today, on your desktop, you have more power than you had on the best computers on that time. This has helped us in really being able to solve larger problems and yeah, it is becoming a reality through this process, but there are other bottlenecks which will come. The second thing that we want to do in AI is to be able to use knowledge and to have the ability of deduction. So, here, the fundamental question is how to store and retrieve knowledge and then, how to interpret facts and rules and be able to deduce. Now, there is a gap between what we call knowledge and what we mean by realization on understanding.

So, I give an example for this- imagine you are asking a baby to learn colors. See, 'color' is a concept. When you tell a baby, that look this ball is red, it does not know whether it is red because of the shape, because of the weight or because of the dimensions. It does not know what 'color' is. So, when you say that this is red, it does not know why this is red. Maybe it will pick up another round thing and say that is red too, right? So, there is a point of time, when parents start getting frustrated, and they say that my child is never going to learn colors and then one fine day, the child realizes that what is the concept behind colors, and on that day it will start citing out all kinds of colors and say, yes this is red, this is green, this is black, right? So, what I am saying is that you can impart

knowledge by saying that this is red, this is blue, but then, as long as you have that knowledge, it does not mean that you have understood the concept called colors.

Similarly, when we are talking about deduction, you can have lots of data in your hard disk, you can have lots of data in your data base, but that does not mean that you have actually learnt the rules which govern the domain in which you are using it, right? So, there is a gap between this- knowing all that is there to know, and actually understanding the rules and being able to interpret and use those rules to deduce new data. So, we will see what kinds of logic that enable us to do this. We will study first-order predicate logic. We will study proposition, and logic, and a little bit of other kinds of logic, to see how we can do reasoning with those kinds of logics. And there is another interesting subject which has come up in the past decade. It is on the logics of knowledge.

Now, this is a very interesting concept. This concept is- I am just giving you a glimpse of certain things that are interesting and that may be covered in this course. So, a logic of knowledge is like this: suppose we take two young children who have come home after playing. So, they have mud on their forehead. Now, their mother is tells the children, that whoever has mud on his forehead, just go and wash. Initially, both wait, they do not go and then they go together. Let us understand why. See, when they wait, actually, one is seeing whether the other is going, because they do not know whether they have mud on their own forehead. So, they the person A assumes, that okay, person B has mud on his forehead. So, he must be going for washing. But then, he sees that B is not going, and why is B not going? Because B has seen someone else, namely me, that I have mud on my forehead and just because I was waiting for a very similar reason, he is also waiting for me to go. So, in the second cycle, they both realize that they have mud on their respective foreheads and they then they go for washing. So, there is, again, a difference between me knowing, you knowing that I know, and I knowing that you know that I know, right? So, this is the kind of knowledge hierarchy that will build up and that is something which is captured by logics of knowledge.

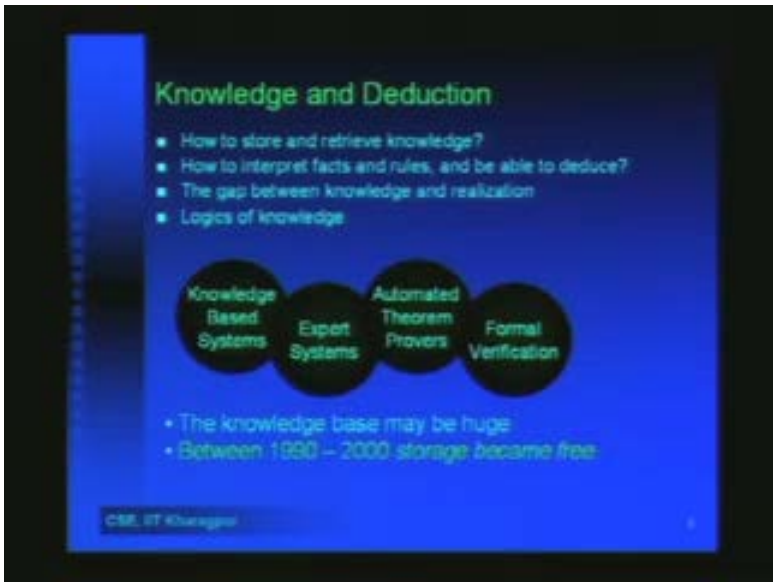
So, again, these are some very interesting things which have come: the paradigms that have come up from knowledge, and deduction, and knowledge based systems. Then, there were a lot of expert systems which had come up at some point of time. We still have a lot of systems which are able to use facts and rules and give you either expert opinions or decision support systems. Then, there are automated theorem-provers. These have There has been a phenomenal growth in this area- on automated theorem-provers. And there are some really good theorem-provers which are there on the net; I will give you the links later on and you can download and try out some of them. Now, in automated theorem proving, what happens is that you encode a problem in logic and then you try to solve it using the theorem-prover. And in recent times, there has been some theorems, which people could not solve mechanically, because of the size of the proofs, and we actually had automated theorem-provers doing the jobs for us. And notable among these is: well, it was not exactly done with an automated theorem-prover, but one of the automated proofs that we got in recent times was the proof of the four-colored theorem.

The four-colored theorem is: you are given a graph- you want to, you are given a planar graph- and you want to see whether you can color it four colors. And it was an open question for a long time- that whether all planar graphs can be colored with four colors or not. And people could not actually prove that four colors are sufficient. And then, people devised a way of generating the kinds of configurations through which you should be able to prove whether four colors suffice or not and then this set of configurations start growing. So, you need an automated mechanism of generating them and establishing the proof. So, the initial proof- the computer generated proof- was huge, and obviously mathematicians did not accept it, because they said that if you did not have a way of checking whether the proof is correct, then what good is your theorem proof. So, then, actually we have, again, a class of theorem-provers, which takes a proof and tries to deduce another proof which is humanly legible. So, they try to club together different steps of the proof and try to give you a proof with shorter number of steps. So, there are theorem-provers, which attempt to do that kind of thing and actually, even for the four colored theorem, it was brought down to a shorter proof, which was then verified by

human beings. Another area in which knowledge and deduction is being used to a large extent is formal verification.

This is an area in which I work. This is has found a lot of application in verification of circuits, where you have huge circuits of millions of gates and you have to be able to verify whether the circuit is going to do what you expect it to do. And there are techniques in formal verification which will help you in doing this.

(Refer Slide Time 20:27)



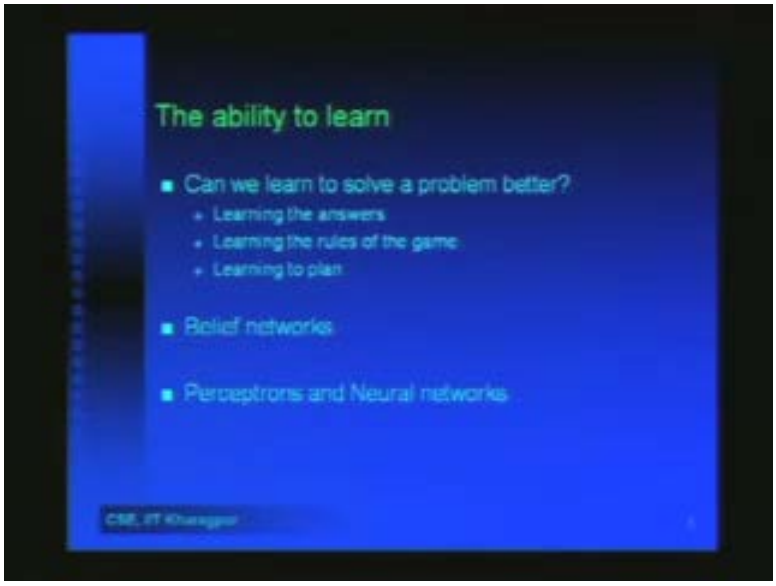
So, the main bottleneck that we had in knowledge and deduction was that the knowledge base or the fact base can be potentially very large. If you look at the earliest chess playing computers, they did not have much of a knowledge base. They just did a kind of brute force search. We will study that those kinds of searches that game playing programs do, but later on, the more advanced chess playing computers rely on the huge databases, huge knowledge bases. They try to use as much of prior knowledge as possible and this has become possible today to a large extent, because storage has become free. So, in the '90s to the 2000's, storage became free. We had huge disks, expensive storage mediums. Now, you have huge disks in the size of a matchbox and they are also pretty fast. So, what is

going to happen in the next decade is, in the next decade communication will become free.

Now, our main bottleneck is communication. Here the bottleneck: on your desktop, you have lots of space, you have lots of computation, but communication is becoming important, because everything you do not have locally. You have the web. You have other kinds of resources all around the world and communication is going to become free in the next decade, so that next time when your washing machine goes out of order, you do not have to bother. It is directly going to communicate with the dealer and put the complaint, right? So, those kinds of things are going to come and that is also going to help the AI paradigm to a large extent.

The third thing that we want to look at is the ability to learn. This is another important aspect of artificial intelligence. So, there, we try to find out, that can we learn to solve a problem better. So, we know how to solve a problem. We have devised search algorithms. We have devised deductive algorithms for solving a problem. And then, can we slowly start learning the rules of the game and can we learn to plan, right? So, we are going to study a little bit of learning in this course. Specifically, we will study a little bit of what is called belief networks and then, we will also study some parts of preceptors and neural networks, and how we learn with these kinds of technology.

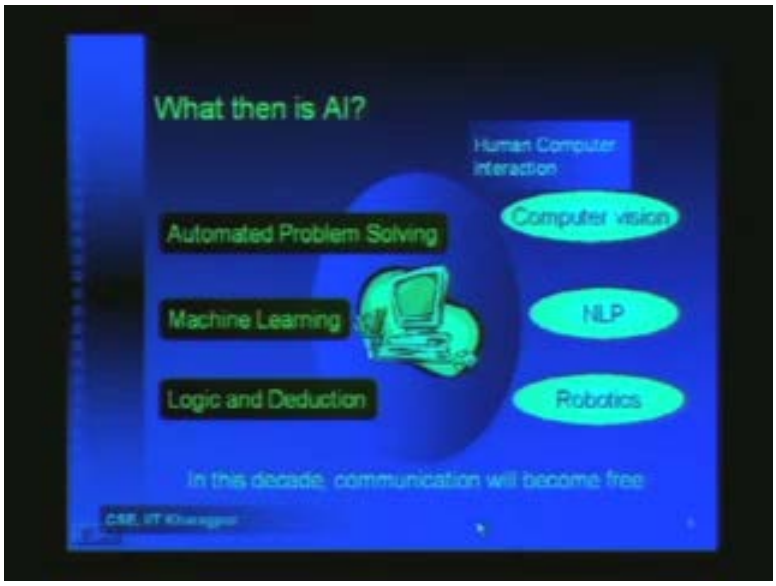
(Refer Slide Time 23:00)



So then, what is AI? What we have seen so far, is that we have now, at the current point of time, a good amount of power in terms of automated problem solving, a good amount of understanding of machine learning and also a fair amount of technology in logic and deduction. So, this is the basic computational background, which is necessary for AI to become, here, to make a computer act as if it has intelligence, but that is not all. What is coming up and in a very large way, are these things: Computer vision, so we can equip the computer to actually get a lot of sensory inputs from the environment; so that we can get rid of keyboards and mouses. See, these are very old fashioned things. We are used to keyboards and mouses, but then there is no reason why we should use something like a key board. If we can If we are able to talk to the computer and make it read text from whatever we write, then things will change. Then, the ability to interpret natural languages- NLP. This is another area which is going to make lots of advancements in the forthcoming years. And then we have robotics, where we actually enable the computer to perform realistic actions. Not just computational actions, but actions with objects and with actual ___. So, what is actually remaining, as part of this course, is human-computer interaction. But, since this course is predominantly a computer science course, so, we are going to focus more on this side.

We are going to focus more on automated problem solving, on machine learning, logic and deduction, and we are not going to really go into the details of this. But there are other courses, in which you can pick up inputs about computer vision, natural language processing and robotics. In this course we will be sticking to this side.

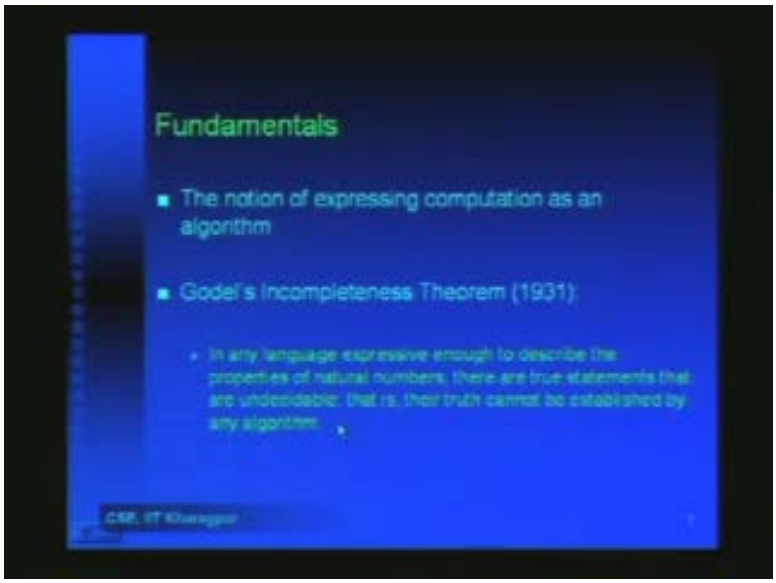
(Refer Slide Time 25:30)



So, the fundamental things that we need to have a background on, is algorithms, but specifically, what we are going to learn from this course is how we can actually express different kinds of problems into our known algorithmic frameworks. And we will learn new algorithms as well. Are you familiar with Godel's Incompleteness Theorem? So, this is the first thing that you should read up, okay? What this says is, in any language expressive enough to describe the properties of natural numbers, there are two statements that are un-decidable. That is, the truth cannot be established by any algorithm. So, you are familiar with un-decidable problems? Have you done your course on FLAT- Formal Language and Automated Theory? So, Turing machines and stuff like that, you know right? Holding problem of Turing machine, right? So, we are going to look at some other un-decidable problems, like, we will see proving first order logic formulas, etc., where you have un-decidability creeping up again. And we will see how this theorem- Gödel's Incompleteness Theorem- is actually an encompassing theorem, which says that if you

are able to express the properties of natural numbers, then this logic, which is able to do this, will contain statements that are un-decided, right? So, you cannot have a proof which is going which will be able to do everything including reasoning about natural elements, right? There will be some unsolvable things, but the world does not stop there. We have to still solve those problems.

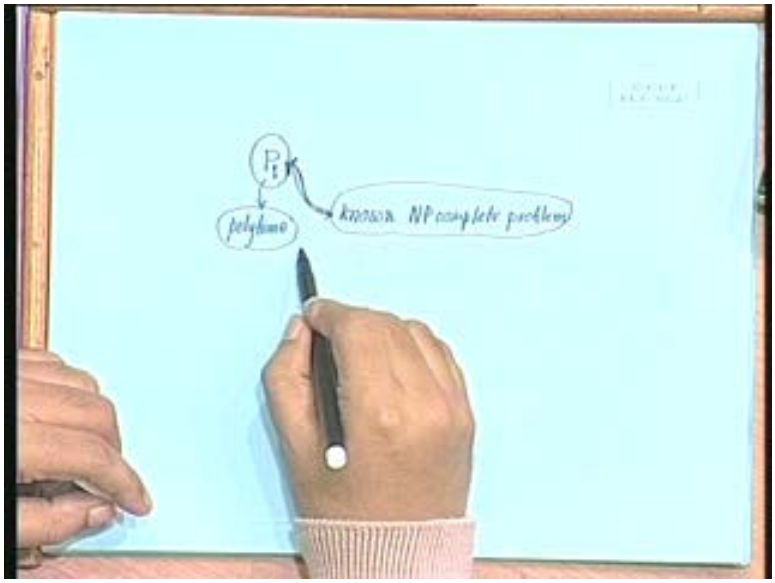
(Refer Slide Time 27:32)



So, we will see that how we can get around some of these problems and still be able to do things from an engineering point of view. Then, we have the Church Turing Thesis, which was proposed in 1936, which says that the Turing machine is capable of computing any computable function. Now, this is the accepted definition of computability. There is no real reason why computability is defined in this way, but that is what we have done so far- that we have actually defined computing in terms of the computability of Turing machines and that is the accepted definition with which we will go. And then another important thing which affected AI in a large way was the notion of intractability and np completeness, and the and the concept of reduction. Now, if I have a problem p and I want to show that this is np complete- I have a problem, say p1 and I want to show that this is np complete, what do I have to do? Is it the case that I translate this p1 to some known? Is this what I will do? Deduce, translate, p1 to some known np

complete problem? Yes or no? How many say yes? (Student speaking) Right. It is exactly the other way round. What we need to do is, we need to take an np complete problem and then reduce that to an instance of p1. Is that the case? And why? Because then, you can use the following argument- that if I have a polynomial time algorithm if I have a poly time algorithm for solving p1, then I can solve this np complete problem also in polynomial time, provided that I can do this translation in polynomial time. So, if I can translate this problem in polynomial time to p1 and I have a polynomial time algorithm for solving p one, then I have solved the np complete problem also in polynomial time.

(Refer Slide Time 30:25)

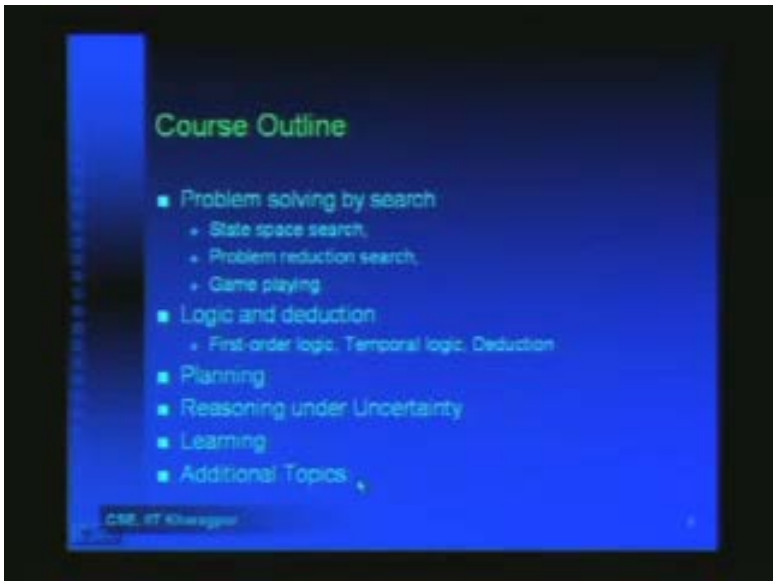


So, that is why, if we do if we are able to make this translation, then we can use this argument and say that okay, most likely we will not have a polynomial time algorithm for p1, unless p is equal to np, right?

So, the course outline is like this: that we start with problem solving by search, then under which we will look at state space search, problem reduction search, game playing search, etc. Then we will have logic and reduction, in which we will study first order logic, propositional logic, a little bit of temporal logic and deduction. Then we will have a chapter on planning. Then we will study a very interesting area, of reasoning under

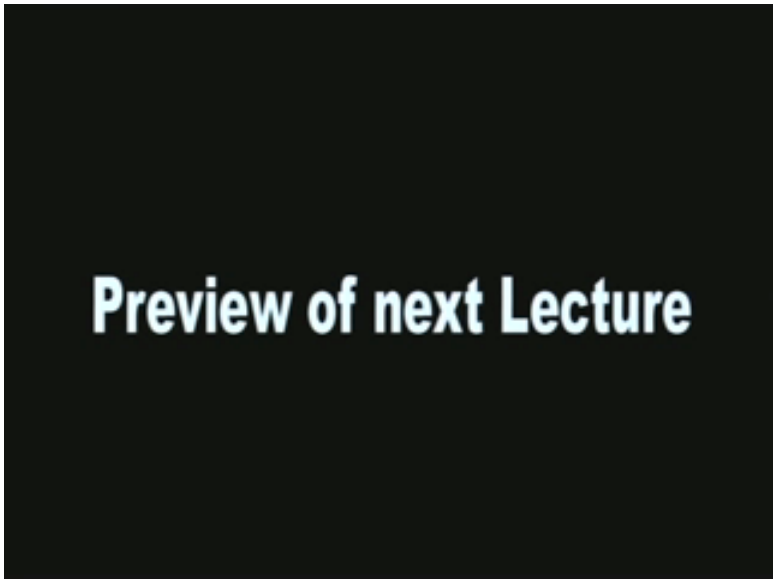
uncertainty, where we will have to deduce, but then we are not hundred percent sure about the facts, we are not hundred percent sure about the rules, so, we have to perform the reasoning knowing that the knowledge and the facts that we have, have some associated probability victim. Then we will also study a chapter on learning and a few additional topics which during the course we will see.

(Refer Slide Time 31:48)

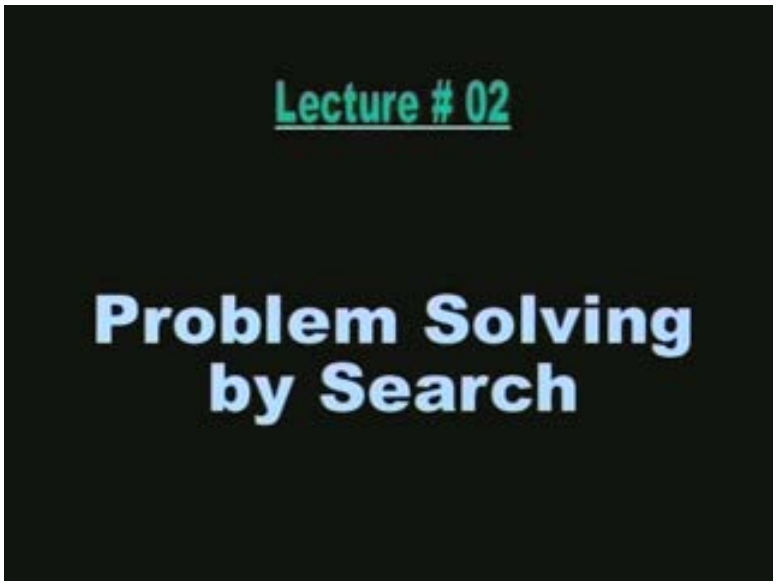


These are the references: I will be mostly following the book ‘Artificial Intelligence- A Modern Approach’ by Stuart Russell and Peter Norvig. There is Pearson L.P. low price edition for this. And then, I will also look up a little bit of ‘Principles of Artificial Intelligence’ of Nilson. Now, both of these books are there in sufficient numbers in your departmental library.

(Refer Slide Time 32:28)



(Refer Slide Time 32:30)



So, today's class is the first chapter that we are going to take up in this course. The topic is problem solving by search. Slides please? Slides. So, under problem solving by search, we will look at several different search frameworks.

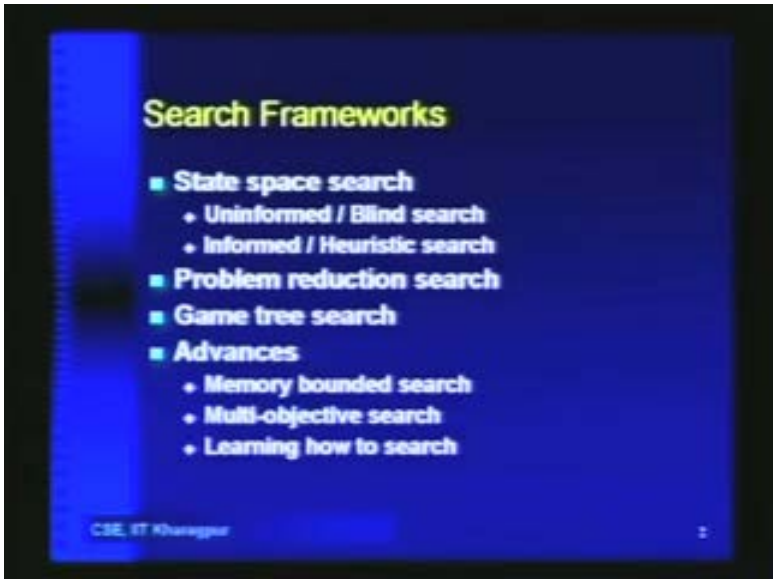
The first is what we will call state space search. So, here, we will see how a real world problem can be mapped into a state space and then we can have search algorithms, for searching out the goals in the state space. So, under these, we are going to look at two

different types of search- namely un-informed search or blind search: these are search problems where we do not know any domain specific information, so, we have to search without any additional knowledge and the second thing will be informed or heuristic search, where we will have functions to guide the search procedure. And we will also study some algorithms for informed heuristic search.

The next topic that we will study is problem reduction search. This will be done in the next lecture and this will talk about problems which can be decomposed into sub-problems in a " " similar to dynamic programming, except that we will try to understand through search, how best to solve the problem.

So, take for example integration by parts. You remember integration by parts? So, we can solve the same problem, in different ways. We will associate a cost function with every kind of solution and then try to see what is the best way to decompose the problem into parts, and solve it the best way, okay? So, that is what we will learn in problem reduction search. And finally, we will have a look at game tree search, to give you an idea about how how chess playing programs work. We will have a few lectures on searching game trees and see how you can do optimization in the presence of an adversary. And I will briefly touch upon some advances in the search area, which is mostly a research done in our department by our AI group. And this is a memory bounded search, multi-objective search and learning how to search. So, these will be- I will just briefly touch upon what these topics are, right?

(Refer Slide Time 35:05)



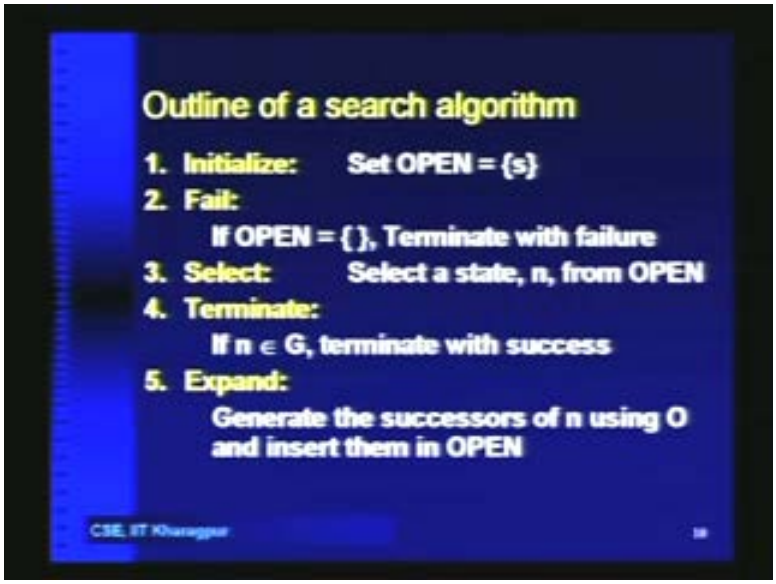
Coming to state space search, the basic search problem is as follows: we are given four tuples; the four tuples in the four tuples, the first is s, which is the set of states which is implicitly specified. Now, this is very important. If you have a graph and you are asked to find out the shortest path from one node to another node in a graph, well, we know that we can do it by Dijkstra's, or any other shortest path algorithm. And the problem is not hard. We have polynomial time algorithms for doing that. Other than that, if you apply something like depth first search and in DFS if you cannot determine whether you have visited some state, then you can potentially go into an infinite loop. So, we will have to devise mechanisms by which we are able to check whether we are visiting some state, or if we are arriving at the same state along some other path, right, and so on.

So, now we will look at the outline of our first basic search algorithm and we will develop this search algorithm to encompass all the different kinds of AI search algorithms that we will study. We start by maintaining a list called open. This is the this terminology of a list called open was devised many decades back and so, we will continue with the same names and jargons. So, initially, our list open will contain the start state. This is the first step- initialization. Then, we have one termination criterion. If we find that the list open is empty, then we will terminate with failure, right? Let us look

at it little bit more. Then it will become clear. Third step is: we select a node n from open- we select any state n from open- and then we check whether n is a goal. If n is a goal, then we have a found a goal, so, we terminate with success.

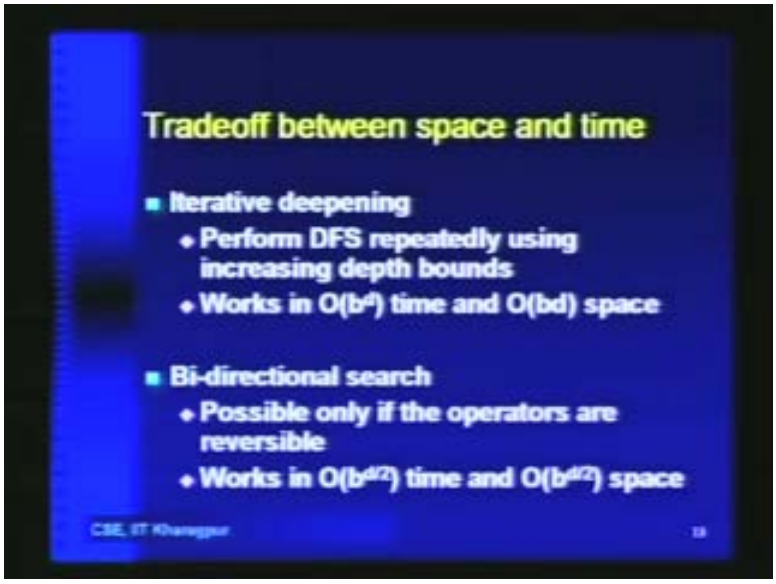
Remember that our objective is to start from the start state and find a path to a goal state. So, if n belongs to g, then we terminate with success. Otherwise, we generate the successors of n using the transition operators o and insert these new states in open, right? So, what we are doing is, we are starting from one state and then expanding that state to generate its successor states, by using these state transition operators and inserting these states into open.

(Refer Slide Time 37:43)



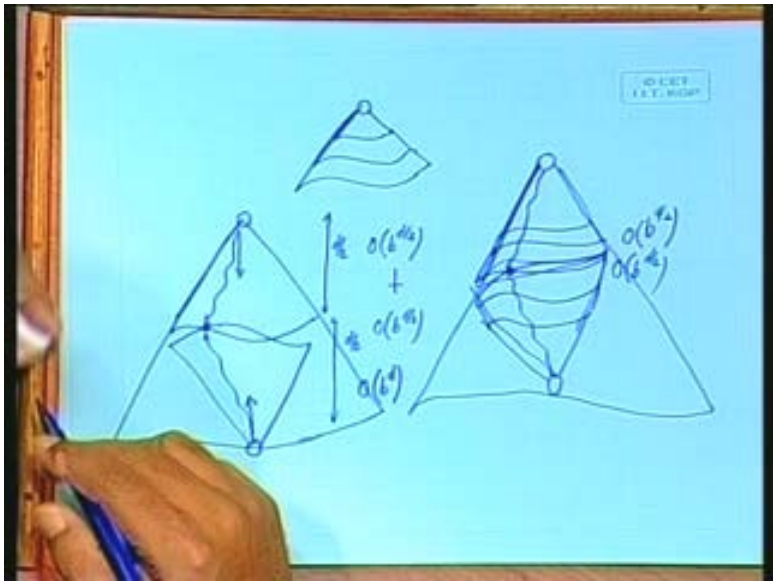
So, at this point of time, I have not mentioned whether open is a FIFO or a LIFO, or whether it is a priority queue, nothing. And these algorithms will vary, depending on the kind of data structure that we use for open. But this can only work if the state transition operators are reversible, so that you can start from the goal and work backwards. And in puzzles like fifteen puzzle, etc., it is indeed reversible. So, you can carry out from that side.

(Refer Slide Time 38:44)



(Student speaking). Of the bidirectional search? Well we can start- if you did normal breadth first search, then you will go straight from the start state and expand all nodes up to d, right? So, your complexity will be order of b to the power of d. Here, what we are doing is, we are starting from the start state and moving in this direction. And we are also starting from the goal state and moving in the other direction, using the rewards of the state transition operators, right? And we continue until the frontiers of the searches from these two sides meet at some point of time, at some place. When you know the common node in these two, then you can find out a path from the start state to the goal state. Now, you see that because you are doing one iteration from this side, one iteration from that side, so, you will be doing d to the power of two iterations from this side and d to the power of two iterations from the other side, right? Order of d to the power two from this side, order of d to the power two from the other side- if you do d to the power of two iterations from this side, your complexity for BFS is order of b to the power b by two, and also from the other side, it is order of b to the power of d by two. And if you add up this plus this, we still know if order of b to the power of d by two, right?

(Refer Slide Time 40:14)



So, you are actually making a complexity- "__" the complexity by half, right? If you have order of b to the power d, you have order of b to the power of d by two. It is significant, actually, right? So, I complete this lecture here, and from tomorrow onwards, we will start looking at the search problem with the introduction of costs. We will see what happens when the state transition operator have associated costs, when the goals have associated costs.