Artificial Intelligence
Prof. Mausam
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Lecture – 06
Constraint Satisfaction Problems: Exploiting Problem Structure

So, today's job is to face the lecture on constraints satisfaction. And we are we so, we started with defining the constraint satisfaction problem. In each constraint satisfaction problem, we define that there will be a set of variables, Xi, they will they will have a set of domains, Di for each variable. It can be different for different variables also. And we are given some constraints and these constraints are represented in some declarative representation.

The declarative representation basically means that those constraints are explicitly stated in some language and that language is the language which is allowed by our system to represent these constraints. And when we are given these constraints, then what we will do, we will cost the problem of solving constraint satisfaction problem as a search problem. And we talked about the backtracking search algorithm which essentially starts with that empty assignment of variables.

And then at each step picks a variable picks a value and picks a variable and branches on all the values. And in such a scenario we were able to show that you know not sure but I told you that we are able to solve up to 25 queens problem. Notice that it is going to select a variable and branch on the values and that was the major difference between a standard I mean a backtracking search versus a depth first search algorithm.

And then we said that there are lots of norms that we can play with one question is which variable to be take? Another question is which value do we pick? Another question is can we detect failures early? And another question is, can we take advantage of the problem structure, and we talked about the minimum remaining values heuristic for the variable selection degree heuristic for variable selection, least constraint value heuristic for value selection.

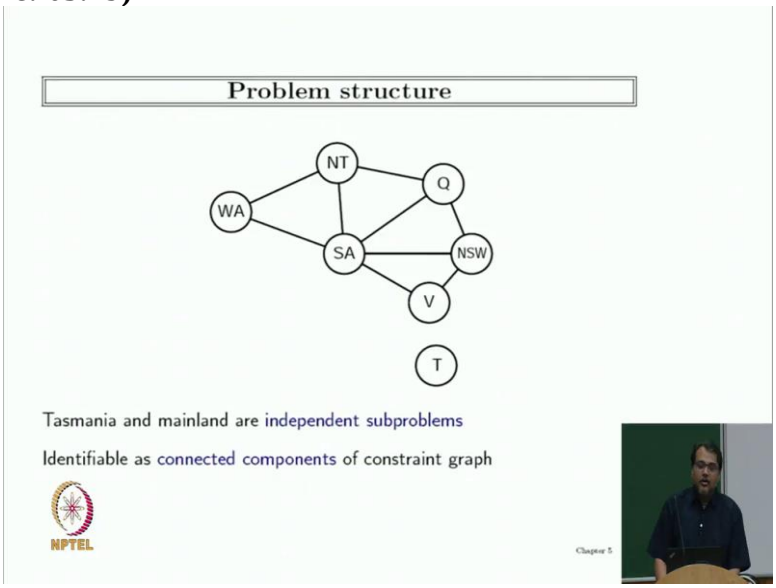
And then we talked about forward checking our consistency and K consistency as our inference algorithms. We talked about the difference between search and inference searches

like trying inferences like proving in inference variables. To prove that these values are no longer possible, or this state is no longer going to lead to a solution that can help us detect a failure early.

We also talked about the fact that search can solve the problem but would not scale very well. Inference can solve the problem but may not scale very well. And a combination of search and inference really, really does well. And we also said that by these ideas you were able to get from 25 queen problems to 1000 queens problem which is a huge deal. Because if you think about it, the search space of 1000 queens problem is quite large, actually.

1000 to the something but we were able to solve that problem using backtracking search is pretty amazing. Now, the last part that we have to study in systematic search is the idea of taking advantage of the problem structure in doing backtracking search and for that we have to talk about 2 ideas, simple ideas. The first idea is the idea of cycle cut sets.

(Refer Slide Time: 03:29)

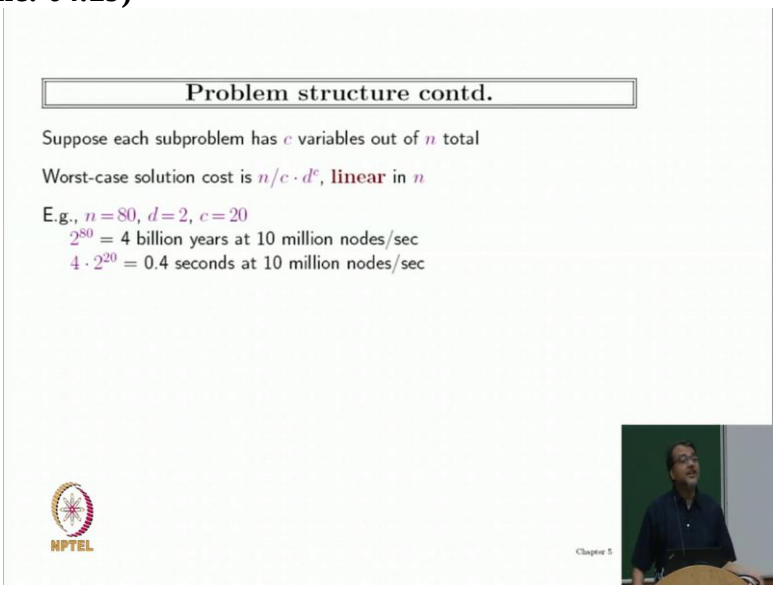


Look at this particular problem. Now you can see that Tasmania and mainland are completely independent subproblems. Therefore should they be solved in a single search algorithm? No reason. We will be wasting time, because when we are backtracking from Western Australia, Northern Territory, the value of Tasmania does not matter. In fact, suppose I give you 2 independent subproblems.

If I give you 2 independent subproblems, then you do not need to solve them together, you can solve them separately. And how would you figure out that a problem has 2 independent

subproblems by just doing a connected component algorithm. So, you do a connected component algorithm and you suddenly realize that there are multiple connected components, which basically means that you can solve all these problems independently.

(Refer Slide Time: 04:29)



So, let us say I have a problem, which has n variables, but each sub problem has c variables, and I have n over c such problems. So, then doing a search of size d to the power n gets reduced to doing n by c searches or size d to the power c. Notice that a c is small, then this is a huge reduction due to the power n is exponential. And if c is small, then d to the power c is kind of constant, assuming c is constant, not a large number.

So that means that my worst case solution cost has become somewhat linear and exponential is a constant. So that is pretty awesome. Linear in n that is, that is pretty amazing. And as a simple example, suppose I had a problem, binary problem, binary CSP with each variable having only 2 values. So these 2, and let us say they were 80 variables. So that means that the search space of the size 2 to the power 80 and if I am doing uninformed search, it will take me 4 billion years at 10 million notes per second to solve this problem.

If I have the space, I think I have the space because we are doing depth first search, for backtracking. On the other hand, if I have a problem in the same problem, I have only 4 connected components not too many. 4 connected components and each connected component of size 20 then I will be solving 4 problems of size 28, which would be 4 into 2 to the power 20.

Which is only 0.4 seconds at 10 million notes per second. Just imagine the amount of speed you can get if your problem has separate components. It is absolutely incredible what you can do. In fact, you do not even need to have completely disconnected components. Let us say I give you 2 components which have one variable common, let us say we call them loosely connected search space.

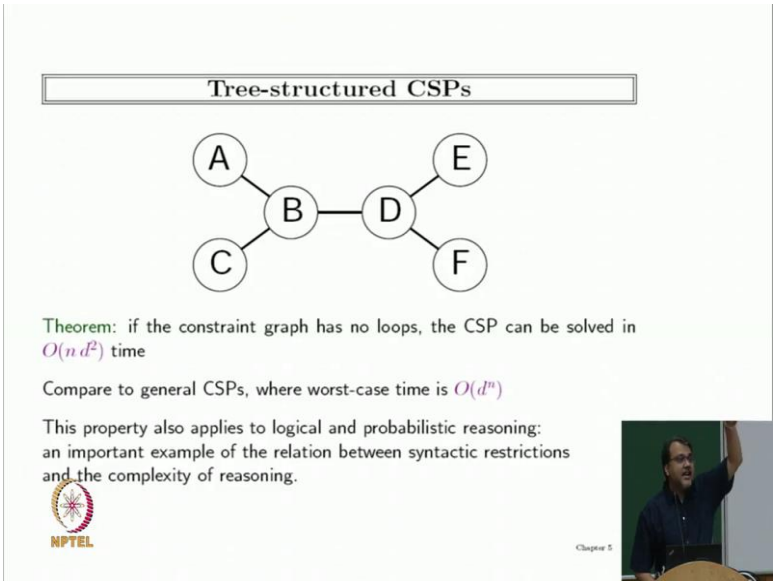
So constraint graph only one variable common, then what can you do? You branch on the first that variable which is common. Yes, that variable can say value 1 value 2 value 3 value 4 whatever. And now, in that sub problem that I have, once I have defined with once I have figured out this particular variable. Now I have 2 independent subproblems. And then I can solve them very fast.

So therefore, if I have problem structure, then and I can figure it out. There is a problem of structure which I can use. Now, in specific problems a human can figure it out in a general problem, you want to find an algorithm which can figure out problem structure, which leads us to more advanced CSP algorithms which will not get into. But there are algorithms which analyze the problem structure, analyze the constraint graph and figure out which kind of CSP algorithm we should try again.

So those kinds of settings also exist. But, of course, in practice, this does not happen very often, because there are not any completely disconnected subcomponents. Usually, they are either loosely connected, in which case we can branch on them first, are there other kinds of problems structures that we can make use of, and one such kind of problem structure is a tree structure.

So, now, theoreticians have been doing this for a living. They say that if you give me a general graph, this is the time complexity but if you give me a planar graph, that will be the time complexity and if you give me a tree that will be the time complexity. Let us be theoretician for a few minutes and let us say our constraint graph is a tree.

(Refer Slide Time: 08:21)



For example, this kind of a tree. And we are going to show by example that for such a constraint graph is does not have any loops that is a tree, the CSP can be solved in order n d square time. Think about d to the power n and d square. It is a huge improvement in time complexity. But this only happens when it is a tree structured CSP, basically you do not need search. That is the beauty of this, you can only do inference and solve it, which I will show you.

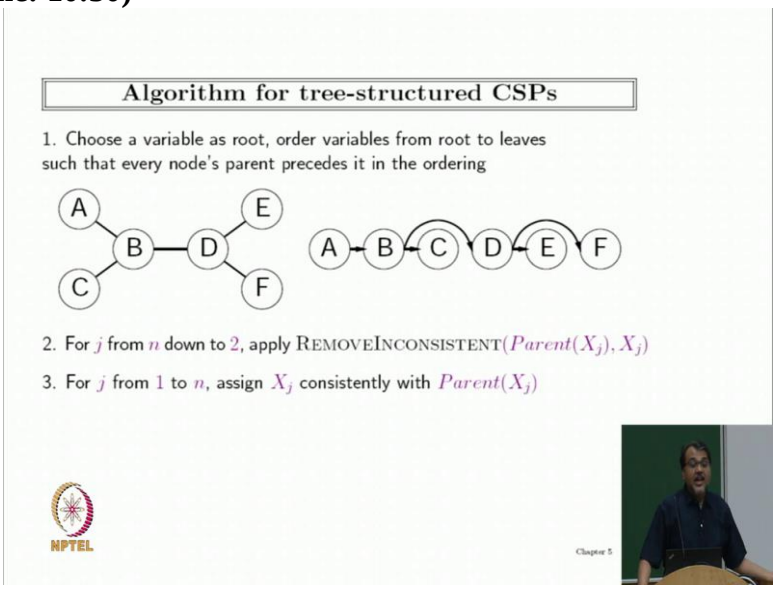
Believe it or not, these kinds of ideas applied to logic applied to probabilistic reasoning also, general purpose graphs are much harder to solve. And if I have a simpler graph, which is like a tree structure then it is much easier to solve. And in fact and we will not talk about it in the course but you know, if you are interested you might check out this whole concept part tree width, so this is the advanced concept.

It says that, you have a graph, you do not have a tree. But is it close to a tree? Or is it too far from a tree. And this notion of close next to a tree is called the tree width. Like, tree widths, if a graph has tree width 1, it is a tree. If a graph has tree width 2, it is no longer a tree, but it is closer to a tree in some semantic notion, then a graph which has tree width 4. And if I make graph had tree width 2, then my running time complexity for that graph can be much better than due to the power n.

And so therefore, people who do advanced research in this often try to analyze their algorithms with respect to the underlying 3 width of the constraint graph. But we are not

going to talk about that too much. Let us talk about what happens when I am given a tree. How can I solve this problem much faster and in n d square time without doing any search whatsoever?

(Refer Slide Time: 10:30)



Now the idea is you hang the tree someone node. So you basically do some kind of a topological sort. Let us say I try to hang this tree from the node A. And then A has child B, and B has child C, and B has child D, and D has child, E and D has child F. Now, you can do any kind of topological sort here, you can hang the whole tree from F, you can hang the whole tree from B, it sort of does not matter because it is a tree.

You can hang it from any root node. It does not matter. But basically we want some linear order. And let us say we got one kind of topological sort. Now let us do the following. Remind yourself of the arc consistency algorithm. The arc consistency algorithm says that I will check an arc and if I will check let us say X to Y arc and if for any possible X that does not exist a possible value of Y, I will remove that value of X from my consideration.

And what we are going to do is we are going to go back from F to A in the reverse topological order. And we will only apply areconsistency. So, let us think about it like this. Suppose I gave you only F then you can pick any value for F, there are no constraints. Suppose, I give you E and F, only E and F then are there any constraints between them, no constraints between them, so, I can give any value for them.

But suppose I give you the sub graph D, E, and F. Then what do I need to know, I need to know which values of D exist such that I have at least one value of E for them and at least one value of F for them, which I can do by an are consistency algorithm. So, I basically apply are consistency with bit between D and E and are consistency between D and F and I checked that for this value of D do I have a value of E or not, E had all the values.

So, if there is no value of D for which he has some value, that means, the problem is unsolvable. Hopefully there are some values of D for which there is some value of E if there is a value of D for which no value of E exists, I removed that value from D. And similarly, I check D to F and if for some value of D, I do not have a value of F I removed that value of D. So, if I have any value of D remaining. What can I do?

I can simply assign it to D. And if I assigned it to D based on our consistency, I know that there is some value of E for it, and there is some value of F for it. So that means I will be able to solve my CSPs. And I keep doing this again and again back. So, now that I have some values of D, I check arcs between B and D, C can take any value of course, I can check arcs between B and D.

And I say, I need some direct value of D that is consistent with the remaining values of D. Similarly, I need that value of B that is consistent with the remaining values of C which are all of them. And if B has any value that that means, my CSPs still solvable and I go back and I check A to B and I remove any value of A that is inconsistent with B and now that I have done this.

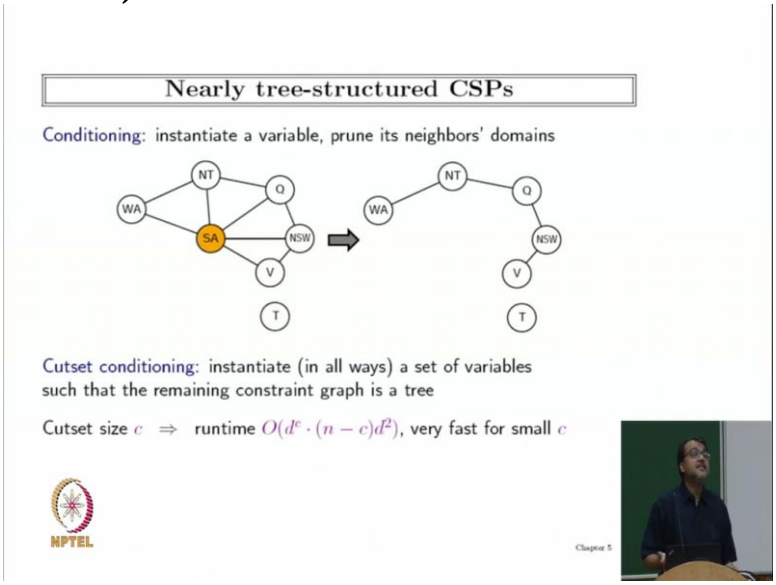
If A has some value, then I know that for that value, there is a value of B and I know that for that value of B, there is a value of C and D. And I know that for that value of D, I have some value of E and F. So, I can just assign them in a forward pass from A to F and I am done. Everybody with me any confusions. What will be the running time of this algorithm? Number of edges into d square but the number of edges is bigger of n because it is a tree.

In fact, there are exactly n - 1 edges. So, it takes, each arc requires D square computation and it takes n order and n X. So, I get n d square in the back part and just order and in the front pass I guess or something like that. So therefore I have been able to solve this problem in

such a beautiful and low polynomial time if the structure is a tree but the structure may not be a tree of course.

However, there is something very beautiful we can do and for that I do not want to show you the answer let us look at the old problem here. Is there something you can do something small you can do to make it a tree. What is your name? Shivam says to remove SA. You cannot completely remove SA from a problem? How will you remove SA but there is one way to remove a SA. Such way SA search SA branch over all possible values of SA. So let us say I say I am going to assign SA first, and I am going to try SA red as a blue as a green. But as soon as I set it, my remaining constraint graph.

(Refer Slide Time: 16:33)



Becomes a tree. Now that it becomes a tree, I can solve it in linear time. And I will have to try it d times because there are d values of SA. In general, if there are c variables that I need to set to make my graph a tree, then these c nodes are called cut sets. This is a technical term for it. It is a graph theoretic term. It says that there are cut set is those that as soon as I removed them and associated edges, my remaining graph becomes a tree.

Some people use the word cycle cutsets. There may be other kinds of sets. I do not know. cycle cuts it says and when I cut them I the cycles have been cut. So this is how you think about them. It is the set of nodes, which I cut so that cycles are gone. So I have to first find a cycle cutset. Now there are algorithms to find cycle cutsets, which also take their own sweet time.

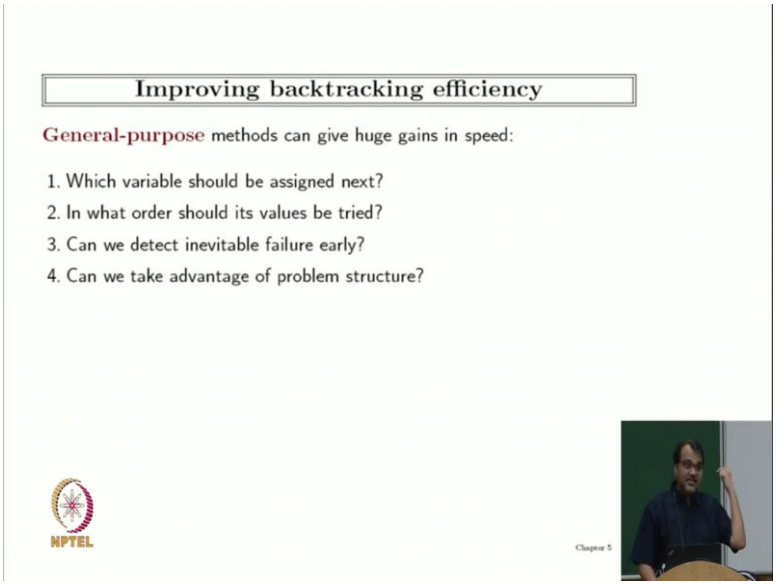
And finding the optimal cycle cutset is also, you know, hard. So life is never easy. If it is an exponential problem, it will mostly say exponential problem unless NT becomes free. So we are going to stay that way. But in practice, you know that these kinds of ideas really help because if I do some algorithm to find a decent in a cycle cutset, then I will search over those. And then the remaining problem I can solve extremely faster overall, I might get a huge speed up.

So if my cycle cutset is our size c then how many searches do I have to do? How many searches how many different possible ways to assign these c variables d to the power c, but as soon as I have done this, my remaining problem, which is our size n - c can be solved in time n - c times d square. So, my total learning time becomes bigger of d to the power c times n - c times d square.

Which is bigger of n - c in times d to the power d + 2, which is still if c is small cutset size is small then this is a huge speed up. So, therefore, what we have done, we looked at a problem we said if I have this beautiful structure in this problem, in this case a tree again solved extremely efficiently. Then I said typical CSPs do not have that structure. But then I said let us make that structure artificially by getting rid of some nodes and doing search.

So, now this is again a combination of search and inference but a very different kind of combination. See earlier we were doing a little bit of search, a little bit of inference, a little bit of search a little bit of inference. Now, we are saying first find the cutset. However that gets done, then search at the top level on those and then do inference and solve the problem. So it is a different way of splitting my problem between search and inference. So, this completes our story of systematic search. Again we did 4 things.

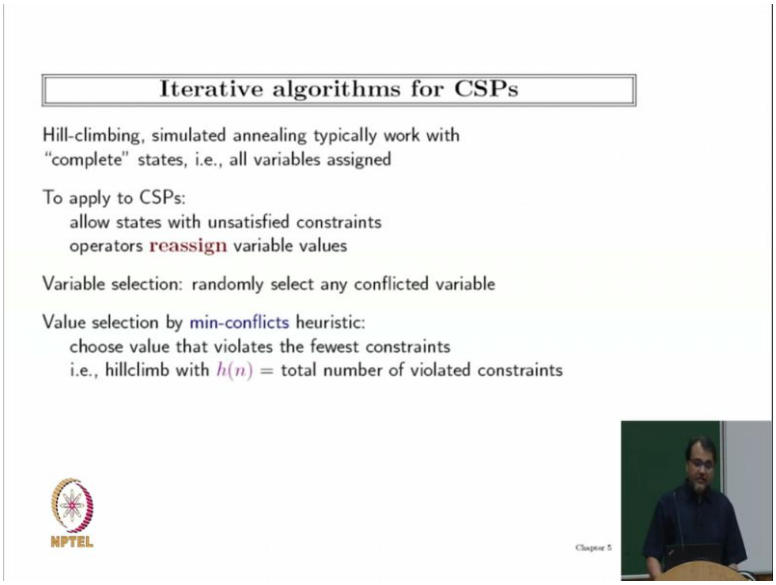
(Refer Slide Time: 19:44)



Branching policy on variable branching policy and value strength propagation to detect failure early and using the problem structure to solve my problem much more efficiently. The last bit we need to do here is very different from systematic search. We have earlier done 2 kinds of search algorithms, systematic search and local search. And we said that any problem can be solved a systematic search.

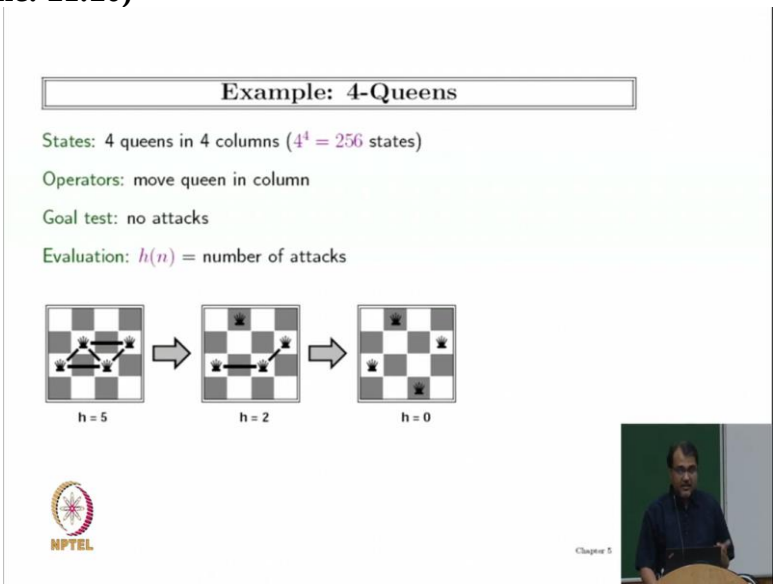
Usually we do partial assignments and solve and any problem can be as local search if we search in the solution space. So, what would be solution space here we will be searching in a space of each state would be full assignment of all the variables not partial full assignment, everything is a solution, but in some solutions, sometimes may be violated and then we have to give it a score and our score can be number of constraints violated. And we want to minimize this.

(Refer Slide Time: 20:54)



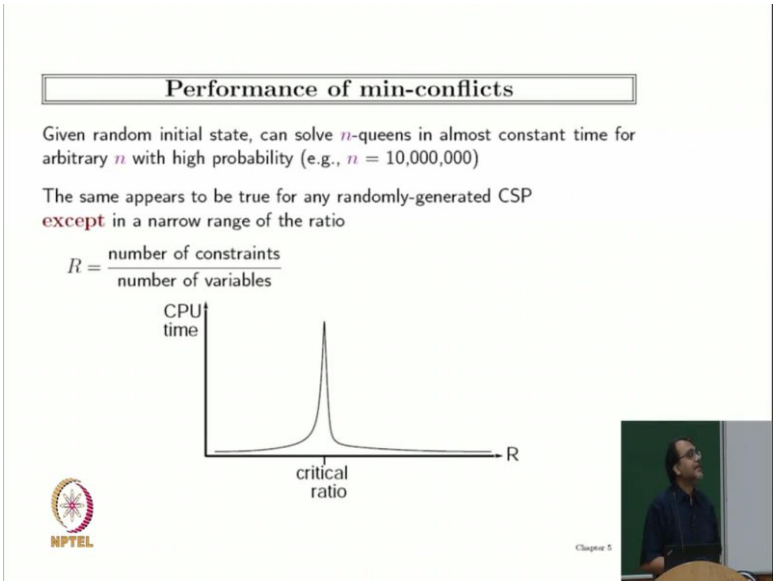
So, we can use hill-climbing this is what the slide states we will work with complete states to apply CSPs will allow states with unsatisfied constraints and operators will re assign variable value. So, what will be my neighbourhood, pick a value pick a variable change its value. That would be a natural neighbourhood function. You can think of more and we can hill climb on top of it on the total number of violated constraints.

(Refer Slide Time: 21:20)



And we have already done the problem of 4 queens in local search. So, let us keep this particular slide it basically says I can do this using local search.

(Refer Slide Time: 21:28)



Now, 2 very interesting things that I want to share with you. Earlier we said that if I use systematic search, I can solve 1000 queens. And we were amazed that we can go from 25 queens to 1000 queens. Believe it or not, if you do local search, we can get to 10 million queens problems. We can solve in almost constant time with very high probability. So you may not always give me a solution, but more often than not you will solve it.

And that I have been always saying is the power of local search. Local search does not always give you guarantees, but it often solves problems in practice. The other thing that is extremely exciting and is a very confusing phenomenon, which we will talk about after the end of next lecture slides is that suppose I make the amount of time a graph with y axis being the amount of time taken to solve the problem.

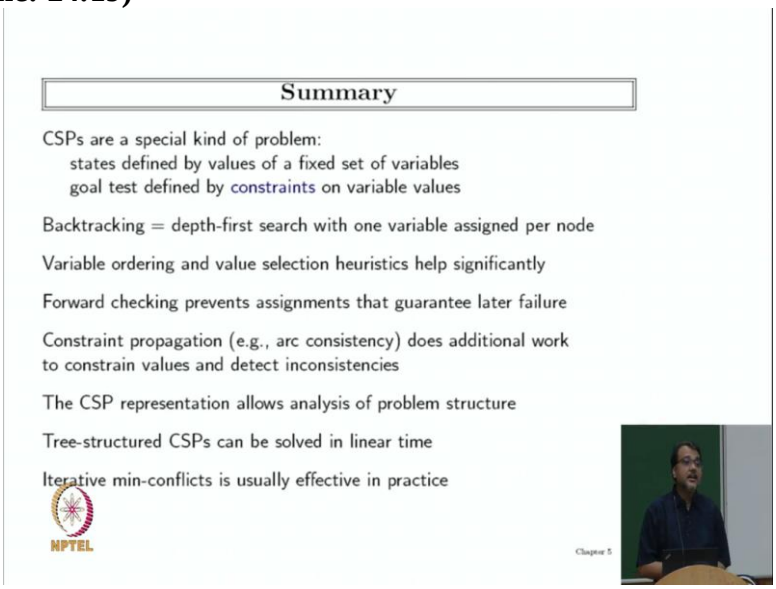
And on the x axis I keep number of constraints divide by number of variables. If for the small number of variables, I have the highest number of constraints and the problem is more constraint. So, basically as my X axis becomes very high the problem is more constrained, if my x axis becomes lower my problem is less constrained. And the beauty full phenomenon that was observed is that if I draw this curve.

The amount of time taken to solve the problem of the problem unsolvable using local search on the random problems of a domain follows this kind of heavily in the middle class. Initially, if the problem is under constraint, it is very easy to solve it later. If the problem is over constrained, it is very easy to solve it up to it is unsolvable in the middle. And where this peak happens is called the critical ratio.

In the middle is extremely slow. In fact, sometimes it just goes so high that you spend a lot of time doing local search and are never able to solve it. This is surprising. And this phenomenon in a literature or in other physics literature is called does anybody know? This is called phase transition. And we will talk more about it later. So I am just appetising you right now. And just saying it is a phenomenon. We do not know where this is happening.

Actually, we do know a little bit. But for now, we do not know you and I do not know what is going on. And actually later when we studied logic and talk about phase transitions in logic, we studied a little bit more deeply.

(Refer Slide Time: 24:19)



So let us summarize the lecture on constraint satisfaction. We started out on the trajectory that ai was search long time back. And later, we have taken a d 12, not a d 12, we have backtracked, and we have said, search is awesome, it is the core. But search is not enough, we need to communicate with the search engine, and the medium of communication is going to be some representation.

And that representation is actually a big deal. In fact, a lot of magic of ai is in the representation. And we defined a very simple representation of CSPs. It is a special kind of search problem where states are defined by value is a fixed set of variables. Goal test is defined by constraints satisfied on variable value complete assignments. We talked about backtracking search, which is depth-first search with only one variable assigned for node.

Variable ordering and value of selection restricts helps significantly forward checking prevents assignments that guarantee later failure. Constraint propagation does additional work to detect failures much earlier than forward checking. CSP representations can also allow faster algorithms if there are specific kind of problem structures and we talked about 3 structured CSPs that can be solved in linear time.

And we also talked about cut set conditioning, where I can convert a general CSP into tree by a little bit of effort. And then we said that well, we did all the systematic search, but we can also do this with local size, and typically it will scale better. There is not that much more to tell you about local search, so we did not go into the detail of that.