

**Constraint Satisfaction Problems**  
**Prof. Deepak Khemani**  
**Department of Computer Science**  
**Indian Institute of Technology – Madras**

**Module - 1, Lecture - 02**

Okay so, we had started looking at Constraint Satisfaction Problems and we had said that a Constraint Satisfaction Problem is a set of variables, a set of domains and a set of constraints on the variables. Now these are something which we have encountered all our lives. In some sense we always encounter problems which can be easily seen as Constraint Satisfaction Problems because that's how we tend to think of them. For example, if you are going with your family in a train, then your daughter says that, "I want a window seat". So that's a constraint on the value of seat that will be assigned to her. Or if you are having a birthday party or something, one child might say, "I want to sit next to Suresh". These are small constraints that we use to solve real world problems and they often occur in many situations.

But before coming to more examples I want to get something out of the way which is the case of numeric constraints; because we will not be addressing numeric constraints in this course and it's a very well-studied kind of a field. For example, as a young student, you might have looked at linear equations. So you might say that  $x + 2y = 12$ . So I can say this is a Constraint Satisfaction Problem. Can you find me a value for  $x$  and  $y$  such that this constraint is satisfied? And this is the kind of thing that we have done very often. Or you can add more equations. You can say  $3x + y = 13$ . So, you can have one or more equations. We know that with one equation there are infinite solutions. And I'm talking about two variables. So two variables, two equations, typically will have one solution. These are the kind of things that we have encountered in our studies and there are well defined ways of solving, for example, sets of linear equations, or you can say linear inequalities. So you can say  $x + 2y > 12$  and  $3x + y < 13$ . So this is another kind of Constraint Satisfaction Problem. So, you're saying that you have variables  $x$  and  $y$  and you're specifying certain number of constraints on these variables. And you're saying find me values of those variables which will solve this set of constraints.

And the reason we will not be studying such constraints is that you know there are extremely well studied methods which have attacked these kind of problems. So we know that there is

linear programming. Integer programming is in some sense talking about domains. Its saying that domains are integers. That solutions must be only integers. So there is a whole host of things, including quadratic optimization. So there is a whole area which deals with numeric variables and numeric constraints and real valued variables and these have been very well studied.

We do not want to look at that. What we want to do instead is to look at finite domains. What do we mean by this? The moment you talk about numbers, essentially you are talking about infinite domains and there are these other well studied methods for solving those kind of problems. So we want to look at finite domains, where each domain is basically a small set of values and then of course, we won't specify what is the nature of constraints. We can say you just give us any constraint, they are relations in general. It basically means they're a subset of combination of values that those values can take and we will restrict ourselves to these essentially.

(Refer Slide Time : 5:59)

The image shows handwritten notes on a piece of lined paper. At the top, it defines CSP as  $\langle X, D, C \rangle$ . Below this, it lists several types of constraints and optimization problems, with some circled in red. A diagonal line separates the list of problems from a section titled 'FINITE DOMAINS'.

**CSP =  $\langle X, D, C \rangle$**   
 $\hookrightarrow$  NUMERIC CONSTRAINT

Linear equations  $x + 2y = 12$  1 eqn.  $\rightarrow$  Infinite solutions  
 $3x + y = 13$  2 Var 2 eqn  $\rightarrow$  1 solution.

Linear Inequalities  $2x + 2y > 12$   
 $3x + 5y < 13$

Linear Programming  
 Integer Programming (domains are integers)  
 Quadratic  
 Optimization ...

**FINITE DOMAINS**  
 each domain is FINITE  
 Constraints - relations in general

So let me take some more examples. What are we interested in? We are interested in finite domain CSPs. And I want to just take a few more examples to see that many different kinds of problems can be formulated as CSPs quite easily. So let's take the example of timetable scheduling. Every semester, in every institute, somebody has to take up this task of saying

that this course will be in this classroom and in this slot. So how can we see this problem of timetable scheduling? What are the kind of constraints that we have?

So we're interested in finite domain CSPs, domains in which the number of values are finite and we want to look at general purpose ways of solving such CSPs. Let's look at some examples.

We begin with classroom scheduling. When you say classroom scheduling, what are the entities involved? You have teachers, classrooms and slots and of course students. So students don't really play a role in classroom scheduling. It would be nice if we could have the students in earlier so that students could choose courses first and then the scheduling could be done. But typically that's not how it is done, typically what we do is, we look at the teacher, classroom, slot combination and try to do scheduling and then present the schedule to students and say you can now choose courses.

So what is the problem? The problem is that there has to be an association between teachers and classrooms, and classrooms and slots, and of course teachers and slots. So every course should have a teacher, should have a classroom and should have a slot.

What are the kind of constraints that we talk about? I can say, for example, teachers teach only one course in one slot. Or a teacher can teach only in one room. So you cannot have a course taught by a teacher in the same slot in the same room.

Then you may have other kind of preferences. So for example, teachers may prefer non-consecutive slots. That is, I may not want to take one subject and then immediately start teaching another subject and so on. And I may have other constraints.

So likewise, there are constraints between classrooms, slots and courses that only one course can be assigned to a classroom and only one course can be assigned to a classroom and a slot combination. So we can pose this as a CSP and then of course we can have some algorithm which will go through all the constraints and try to solve the constraints essentially.

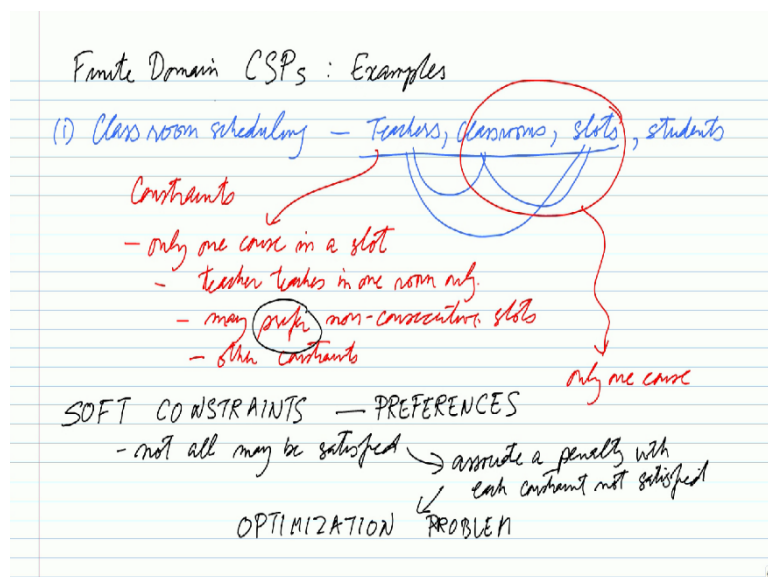
So this also brings me to the idea of what we call as soft constraints, which some people call as preferences. So the idea of a soft constraint is that not all of them may be satisfied. So somebody might say it's not possible to schedule some course so you must teach in two consecutive slots and there is no other option. So I can still teach two courses but this preference that I had that I don't want consecutive slots may not be satisfied. So typically the

way that soft constraints or preferences are handled is by associating a penalty with each constraint that is not satisfied.

So of course you have to distinguish between hard constraints and soft constraints. A constraint like a teacher can only teach one course in a slot is a hard constraint. There's no way you can make a person teach in two different classes at the same time. But there may be soft constraints which you may be allowed to violate but you may associate a penalty with that.

So what happens is that when you convert this problem into saying that you can associate a penalty, then you want to minimize the penalty. And then this becomes an optimization problem. Find a solution where the penalty is minimum. We are not going to look at soft constraints or preferences in this course because optimization is again a well-studied and a harder problem than satisfaction. We are only looking at satisfaction. Constraint satisfaction problems. So we'll only look at hard constraints and say given these constraints give me a solution. But there are many interesting problems where you may want to talk about preferences and soft constraints.

(Refer Slide Time : 13:26)



So let's look at examples again. Logical deduction as constraints. You must have studied logical deduction at some point. You might say something like this that if you have P and if you have P implies Q, and if you have Q implies R, then can you show that R is true? So we know that we can apply modus ponens repeatedly and we can solve this. Now it's interesting to note that both working with logic and with constraints are kind of different aspects of the same kind of thing that we are doing. Both are talking about giving values to variables.

If you want to see this as a CSP, then the set of variables is the set P, Q, R and the domains is the set true, false for each of these variables because this is now a boolean problem for each of those distinct variables.

And what about the relations? The relations are given by what's given to us. So the relation is between P and Q. I will use this as a way of saying that the scope of this relation is between P and Q. So there are three variables.

Let me also introduce the notion of a constraint graph here. Here nodes are variables and there is an edge between n and m if n, m participate in a constraint. This idea is more straightforward when we talk about binary constraints because only two variables always participate in a binary constraint. But it can be extended to higher order constraints. So essentially if two variables participate in a constraint we say that there is an edge between them.

For this particular problem, we have these three variables P, Q and R. So we draw three nodes for them and we have edges between them and the constraint between P and Q is given by  $R_{PQ}$  and the constraint between Q and R is given by  $R_{QR}$ . There is a third constraint which I will introduce in a moment, but if you want to look at these two constraints, what is  $R_{PQ}$ ?  $R_{PQ}$  is saying what is a set of allowed values for the variables P and Q.

One of the things that we will do, when we look at finite domain CSPs is that we will have this relation as an explicit relation. Or as an extension, instead of an implicit relation. So for example if I have numbers 1, 2 and 3, and I have a relation between x and y that  $x < y$ , so instead of writing that  $x < y$ , I will give all the allowed combinations, which is the way that we will specify relations - as a subset of the cross product. I'll say x can be 1, y can be 2; x can be 1, y can be 3; or x can 2 and y can be 3. So I will explicitly specify that.

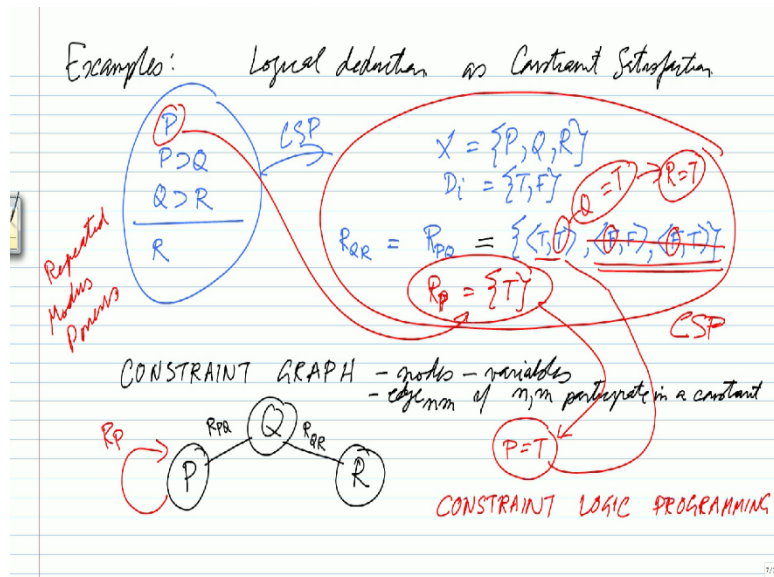
Likewise, I can specify the relation between the variables P and Q. Here what is the variable? That P implies Q is a true statement. And what does it mean? That I look at the truth table for

implication and see in which of those 4 rows the statement is true. So you know that it's the case that when P is true and Q is true, of course P implies Q is true. When P is false and Q is false, also P implies Q is true. And also when P is false and Q is true. And this also happens to be the relation between the other two variables which is  $R_{QR}$ . It's the same relation. So now you can see that I can convert my logical reasoning problem into a constraint satisfaction problem.

There is of course one more constraint which I've not mentioned which is on the variable itself. So let's call it  $R_P$  and we have one more constraint  $R_P$  and this basically says that you are only allowed one value. Why is that? Because we have said P is true and therefore the constraint is that variable P can only take one value which is true. So that's an equivalent way of saying this that we have imposed a unary constraint on the CSP problem. But now what do we have? We have a CSP problem and we can try to solve this CSP problem. So we can try solving by search. Now of course one very nice, simple way to do search would be to start with the most constrained variable and we will look at the search algorithm as we go along but the first thing we will say is that  $P = \text{true}$  because I have a constraint which says that P can only be true.

Once I say  $P = \text{true}$ , then I can look at the constraint between P and Q. So which means that the consequence of this is that only the first of the three triples is allowed. The second one is not allowed and the third one is also not allowed. So that means as a consequence of this  $Q = \text{true}$ . And as a consequence of that  $R = \text{true}$ . So this is the example of constraint propagation. I started by saying that I have three constraints, one on P, one on Q and R and one on P and Q and this is the constraint graph that we have drawn below. And then I tried to search for a solution and it turns out that once I fix that P is true, then I am eliminating this combination, the other two possible values. So once we know that the first variable is true, then the second one must be true. So if P is true, then Q must be true and when Q is true then R must be true and then we have solved the whole CSP essentially. So you can see that logical deduction can also be seen as a CSP. And in fact they are variations of constraint programming. So for example there is a language called constraint logic programming. So there is logic programming that you must have studied, there is constraint programming that we are studying and then there is constraint logic programming and there are lots of variations of these things that we can study.

(Refer Slide Time : 21:40)



Let's look at some more examples. A standard example that naturally falls as a constraint satisfaction problem is map colouring problem. What's a map colouring problem? That you are given some map of a region and that you can colour any country with any colour or some set of allowed colours that are given to you. For example, you might say this country prefers red, blue or green and this country prefers green, blue and yellow and this country prefers yellow, green and something, white and so on and so forth. Which means basically every country is a variable and the value that it can take is the colour that you are going to use in the map. And there are constraints between them.

So we can naturally draw a constraint graph on top of this and say that if a country shares a border with another country then there is an edge between them. And what is a constraint? The constraint is that the same colour cannot be used for adjacent countries. Then if we use red for one country, then we can't use red for the next country. And that's the relation between them. And then you can draw a constraint graph for every country; two countries which have an edge between them and you have this problem to solve. So map colouring is naturally a constraint satisfaction problem and we will use this as an example as we go along for, looking for algorithms for solving CSPs.

Another nice problem which I like is the n queens problem which also is a constraint satisfaction problem, which basically says that place n queens, and when we say queens we

mean chess queens, on a  $n$  cross  $n$  board such that none attacks each. So of course you must be familiar with this problem as well.

If I want to look at a small four by four problem, I've to place four queens on four cross four chess board. So I have four queens and I have decided the way I am formulating this problem. I have decided that I'll place one queen in column one and one queen in column two and so on and so forth because that is necessary. We know from the rules of chess that you can't place two queens in a column so we'll sort of encode this as part of the problem. And what we want is the row number. So these are the values and these are the variables.

So in encoding it in this fashion we have already encoded a constraint that you can't place two queens in one column because you have kind of implicitly assumed that's the case. Now only you want to find which row to place it in and what are the kind of constraints that we can talk about? So supposing we are posing it as a binary constraint satisfaction problem. Then I might say something like, okay let me write  $q_1$   $q_2$ , but it's easier to write just  $R_{12}$  if the context makes it clear. I'm not specifying the solution, I'm only saying what's the constraint between two queens, because that's how the high level statement of the problem is, that no queen must attack another queen. So what is  $R_{12}$ ? It says that if I place queen one in row one then I cannot place queen two in row one or two, it can only be in row three. Or if I place queen one in row one, I can place queen two in row four. That's allowed. Then if I place queen one in row two, I can't place queen two in rows one two or three so it must be in row four only. Likewise, if I place queen one in row three, I can only place queen two in row one or if I place queen one in row four, I can place it in row one or two.

I've specified completely the constraint between  $q_1$  and  $q_2$ . Likewise, I can specify constraint between  $q_2$   $q_3$  and so on. I can pose this as a binary constraint satisfaction problem. Now obviously you can imagine that I could have posed it as a constraint satisfaction problem in which the scope is all four queens. So if the scope is equal to  $q_1$ ,  $q_2$ ,  $q_3$  and  $q_4$ , then actually what am I doing? If I'm specifying how all four queens have to be placed, then I'm specifying the solution. In fact, a relation on all four queens is called a solution relation. And in fact that's the task that we have set ourselves, to find the solution relation. Which means, what are the combinations of the four queens that we can place? You know that, for example, there are two possible solutions to this problem. So the solution relation has two tuples essentially and actually our task is to find one of them. And this is a general idea behind



CSPs, that you don't specify all the constraints. You specify some constraints and then you find solutions which satisfy those constraints. Of course the key trick there is to satisfy enough constraints so that what you get is only the solutions that you want. In this case of course it's easy to do as a binary CSP because the problem itself is stated like a binary constraint. That no queen must attack another queen. So we can easily do that. And then of course you go looking for solutions essentially.

Now if you look at the two solutions that I have, you can see that in the solution equation, if I place queen one in row two, then the queen two must be in row four. Or if I place queen one in row three, then queen two must be in row one. So in fact these are the only two things which participate in the solution. Even though the others are valid constraints, they do not participate in the solution. And in some sense our task is to uncover or prune away those spurious constraints, relations between, things like that.

Another nice thing about this problem is again the idea of propagation that we saw when we were looking at the cryptarithmic puzzle. So if you're looking at a slightly larger problem, so let's say we are looking at a six by six problem. So we have six queens and they have to be placed. So one thing we can do is we can, as we place queens, we do search, for example. Let's say we place this queen here and then what we do is we mark out or cross out squares where we cannot place another queen. And the reason is because that's what the problem states, that no queen must attack another queen. So having placed the first queen in the top left corner, we can place the second queen only in the third row essentially. So let's proceed with that. We place the second queen here. And then like before, we cross out places where we cannot place any other queen. And the third queen can be placed in the second row. No it cannot be placed. Sorry I should have crossed it out. So the third queen can be placed only in the fifth row and once I do that, I can now place the fourth queen in the second row. And having placed this fourth queen in the second row, we can see that no place for  $q_6$ , because the entire column for queen six has been crossed out essentially by this process of propagation forward or looking ahead forward. Even before we try queen four, there is a place for queen four to try, we can actually backtrack from this place itself and then say that okay you cannot place this on  $q_4$  and then, essentially of course backtracking would mean you have to undo some of the crosses that you have to do so that's part of the algorithm that we will see later.

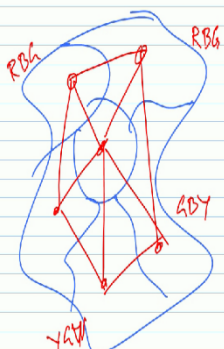
But you can see that we were doing essentially search here. We were saying, let me place a queen at the earliest location I can. But we could backtrack earlier than normal search would have done essentially. Normal search would have backtracked when it reached a dead end. This one has not reached the dead end. It still has a place for queen five to place. But it will not do that. Because the moment the domain for queen six has become empty; so, placing these crosses is like removing elements from the domains of the different queens. So the moment the domain of  $q_6$  or the domain  $d_6$  becomes empty, the algorithm will backtrack from here essentially.

So this again illustrates that these problems like map colouring or logical reasoning or n queens or scheduling can be naturally posed as constraint satisfaction problems. So I'll stop here and we'll have one more class in which we will look at some more examples of constraint satisfaction problems, before we move on to actually looking at algorithms to solve them essentially.

(Refer Slide Time : 32:36)

EXAMPLES

**MAP COLOURING**



**N-QUEENS**  
place  $N$  queens on  $N \times N$  board such that none attacks another.

**VARIABLES**  
 $Q_1, Q_2, Q_3, Q_4$

1	0	✓	
2	✓		0
3	0		✓
4	✓	0	

**VALUES**  
**BCSP**

$R_{Q_2} = \{(1,3), (1,4), (2,4), (3,1), (4,1), (4,2)\}$

$S_{Q_6} = \{Q_1, Q_2, Q_3, Q_4\} \rightarrow$  SOLUTION RELATION

Q	X	X	X	X	X
X	X	X	Q	X	X
X	Q	X	X	X	X
X	X	X	X		X
X	X	Q	X	X	X
X	X	X	X	X	X

no place for  $Q_6$