Constraint Satisfaction Problems

Prof. Deepak Khemani

Department of Computer Science

Indian Institute of Technology – Madras

Module - 3, Lecture - 01

Let's move towards solving of Constraint Propagation Problems or Constraint Satisfaction Problems and this is going to be an interplay between two kinds of processes. One is search which will try to find values for variables and the other is propagation which will try to restrict the values that variables can take.

Let's focus on the propagation part today and we will look at the idea called constraint propagation which basically at the simplest level says that if a certain variable takes certain values, then a related variable can take only some related values. We also use the term consistency enforcement. For this you need to revise the notion of consistency and a partial solution. We had used the phrase "a' is consistent if it satisfies all the constraints whose scope is covered" essentially or whose scope is a subset of the scope of this partial solution or instantiation a' essentially.

If you remember we had tried to draw a small diagram where we said that we have x1, x2, x3, x4, x5 and so on and let's say xn and we have a partial solution, say (2, 1, 3, 6). We have given values to x1, x2, x3 and x4 and then if there is any constraint whose scope lies within that, the values must satisfy the constraint. x1 and x3 must satisfy constraint C1 and x2, x3 and x4 must satisfy constraint C2 which means (2, 3) must be a subset of the relation C1 and (1, 3, 6) must be a subset of the relation C2. Then we say that this partial solution is consistent.

The idea of constraint propagation is to allow a partial solution to be extended, so essentially we are going to be interested in those partial solutions. Let's say I have a partial solution with four variables x1, x2, x3, x4. I should be able to extend that partial solution to the fifth variable. So I must not work with an example partial solution where I can't choose a value for x5 because then I'm resorting more to search than to propagation essentially.

We have this notion of i-consistency –a partial solution of i-1 variables can be extended to i variables. Now this is a more general statement than it looks like from the example that I have written. In the example on the right, I've kind of assumed that the order in which you extend the solution is x1, x2, x3, x4, x5. Now in practice we don't have that constraint. We don't have that restriction. Nobody is saying that after finding values for x1, x2, x3 and x4 it's x5 that you must find the value for. But typically, of course algorithms will work in a deterministic manner so we'll end up doing like that but the notion of i-consistency is more general than that. In practice, of course our algorithms will be more restricted and therefore we will assume that we are extending them in a predetermined order. So, after x4 I will do x5. That's not a part of the definition of i-consistency.

The simplest case is of one consistency which my definition says that at the very beginning if I choose any variable, I'll be able to give a value for that. Now you might ask as to why this is the case. This may be required in situations where there are unary constraints or unary relations which means there is a constraint only on the value of one variable essentially.

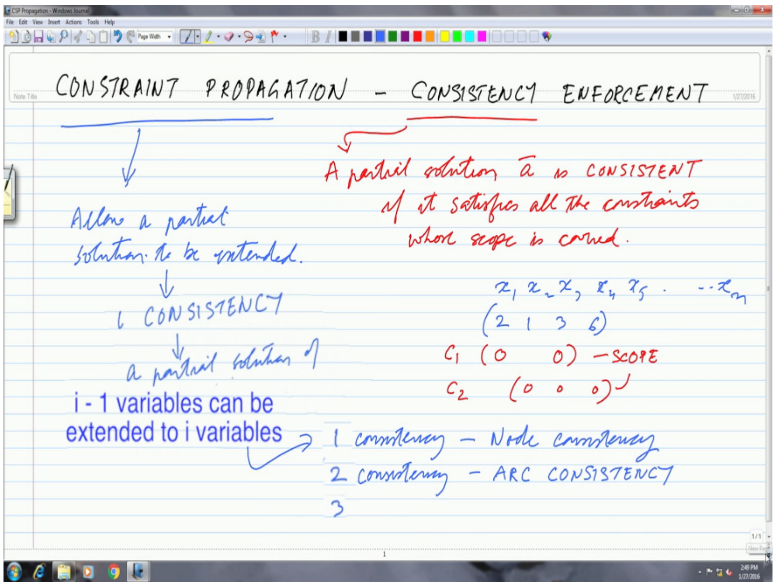
So, for example if you are doing a map colouring problem and let's say you will colour the map of India only in the three colours a, b and c, whereas the domain may have more than these three colours. I have an additional constraint that I'll only colour it white or something like that. And then if my domain has red, blue and white then it's not one consistent because I cannot extend my solution to any value from red and blue. Anyway, we'll come back to that in a moment.

This is called node consistency and implementing node consistency essentially means that you prune values which do not satisfy some constraint. So for example if some colour is disallowed, you remove it from the domain of that variable. Anyways that's a very simple kind of a thing. What's more interesting is two consistency and that's what we're going to look at today. It's also called arc consistency and it says that if I choose a value for one variable then I should be able to choose a value for any second variable. If that is the case, then we say that the network is arc consistent.

Why are we interested in this? Because if we have enforced two consistency on the network then it means that you can give a value for the first variable and you're sure that you will be able to give a value for the second variable. You don't have to backtrack at that stage. And then of course you have one then two then three and so on, higher and higher orders of consistency. So, five consistency means that four variables can be extended to five variables.

i consistency or n consistency would mean that if you have a partial solution of n-1 variables, you will always find a value for the nth variable essentially. You can see that the higher the level of consistency, the less the amount of backtracking you must do because essentially consistency is saying that you will be able to find a value for the next variable.

(Refer Slide Time : 9:09)



CONSTRAINT PROPAGATION – CONSISTENCY ENFORCEMENT

Allow a partial solution to be extended.

i CONSISTENCY

a partial solution of i - 1 variables can be extended to i variables

A partial solution $\bar{a}$ is CONSISTENT if it satisfies all the constraints whose scope is covered.

$x_1\ x_2\ x_3\ x_4\ x_5\ \ldots x_n$

(2 1 3 6)

C_1 (0 0) – SCOPE

C_2 (0 0 0)

1 consistency – Node consistency

2 consistency – ARC CONSISTENCY

3

Let's look at arc consistency in a little bit more detail. We said that for a relation or an edge R, Rxy is consistent, if variable x is, so we will use the short form AC here, if variable x is AC with respect to variable y and vice versa.

To define this, we say that for every value a ϵ Dx that we can choose, there exists a value b ϵ Dy such that the pair (a, b) ϵ Rxy. If this is the case then we say that the variable x is arc consistent with respect to the variable y. If y is also arc consistent with respect to x, then we say that the relation Rxy is arc consistent and then finally we say that a network is AC if all pairs of variables are AC.

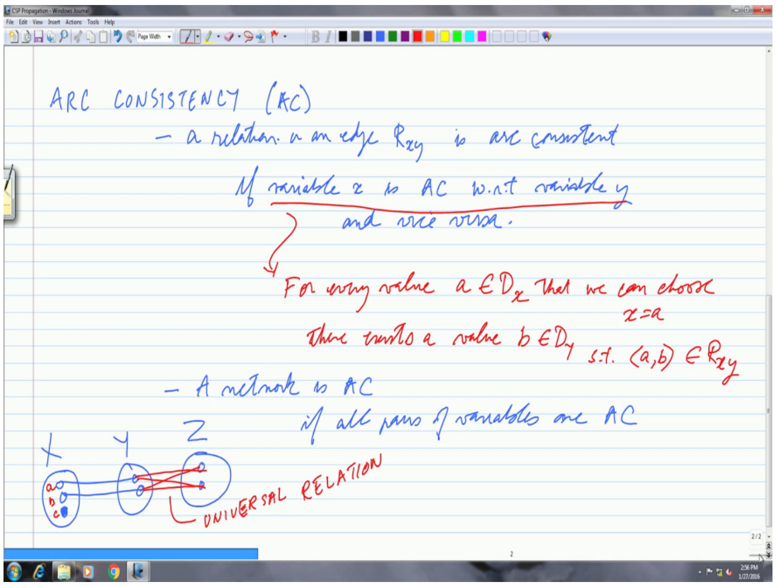
So we have first defined consistency in one direction, then in other direction and then over all edges. So you must not confuse with the fact that when I say all pairs, it's equivalent to saying all edges because if you are talking about a binary constraint network, if you don't have an edge between two variables or two nodes, it basically signifies the universal relation. And when it's a universal relation then of course you can choose any value from one variable and any value from another variable essentially.

So to just look at an example, if we have X, Y and Z, and you have values for the variables as shown in the diagram, then you can see that in this example at this stage X is not arc consistent with respect to Y because there is a value, the coloured element, which does not have a corresponding value in Y but Y is arc consistent with respect to X.

If we think of Z as a universal relation that means implicitly everything is related to everything else. If you have not specified a constraint it means that you are allowing any combination and therefore that is by definition arc consistent. So, you can see that this network is not arc consistent. Y is arc consistent with respect to Z and Z is arc consistent with respect to Y. So the edge YZ is arc consistent but the edge XY is not arc consistent because X is not arc consistent with respect to Y.

Consistency enforcement will basically involve removing values form domains of variables so that it becomes arc consistent so that, if these three values were called a, b and c, then a search algorithm should not start with value X = c because then it will not find a consistent value in Y. So, we want to avoid that kind of dead end in search and that's the whole idea of doing arc consistency.

(Refer Slide Time : 15.40)


ARC CONSISTENCY (AC)

– a relation on an edge R_{xy} is arc consistent if variable x is AC w.r.t variable y and vice versa.

For every value $a \in D_x$ that we can choose $x=a$ there exists a value $b \in D_y$ s.t. $(a,b) \in R_{xy}$

– A network is AC if all pairs of variables are AC

So how do we go around doing this? We start by making a variable arc consistent and that is done by an algorithm which traditionally everybody calls as revise and one way of saying this is that you're revising x with respect to y, i.e., Revise((X), Y). It's just a way of defining this function or module or subprogram. I prefer to use this notation – Revise(Dx, Dy, Rxy). In both the cases, it's just a different notation. You're trying to prune the domain of variable x which is indicated here within brackets. The first notation is used in Rina Dechter's book, the second notation is used in my book but essentially the algorithms are still the same.

Revise is the core of the algorithm for doing arc consistency which is called AC-one. We will see it shortly. The core of the algorithm is the module called revise and what revise does is that it prunes the domain of x so that it has only consistent values with respect to y. So as you can imagine this is a simple algorithm and we can write this as follows. For each a ϵ Dx, if there is no, writing it in Dechter's style, in high level, b ϵ Dy such that (a, b) ϵ Rxy, then delete a from Dx. It's a very simple algorithm and we can illustrate it.

So supposing I have some constraints as shown, then if I make a call Revise(Dx, Dy, Rxy), then this will delete c from Dx which basically means that this value will get deleted from the domain and so new Dx will be {a, b, d} only.

One call to revise x with respect to y has deleted one value from the domain of x. Likewise if I call revise y with respect to x then it will delete three values as you can see. Let the values be {1, 2, 3, 4, 5}. So, revise y with respect to x will delete the values 2, 4 and 5 from here essentially. And then the edge xy will become arc consistent.

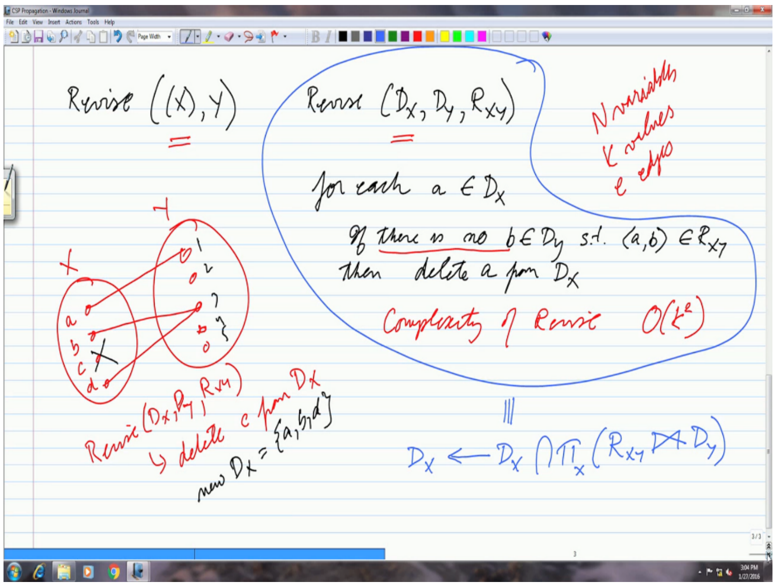
What is the complexity of revise? In the worst case, what will you end up doing? For checking that there is no such value, you may have to look at k values in the domain. In the worst case what is the decision? x has k values and y has k values. So whenever we're doing complexity analysis we'll assume that every variable has k values. In the worst case x has k values and y has k values and let's assume that it's an empty relation which means there are no consistent pairs allowed. That's the worst case situation. What will you do? You look at the first value for x, then you will look at all the k values for y and then you will remove that value for x. Then you will look at the second value for x, look at all the k values for y and remove that value. So, in the worst case you will look at k^2 entries. So the complexity of revise is $O(k^2)$.

So this is something we will always use whenever we talk about complexity - n variables and k values. So our complexity will always be in terms of n variables and k values. Sometimes we will use the notion of count of number of edges in the binary constraint network but not in this example.

In the best case, for every value you look at, the first value you will look at on the other side will be a related value. So you will only look at k values in the best case but generally we talk about worst case complexity.

Another way of expressing this whole algorithm is to say that we're doing the join operation. So, we can express the whole algorithm in terms of one relational statement – Dx ═ Dx ∩ πx (Rxy ∞ Dy) and then from that you're taking the values for x. Whichever values in the original domain were there, you're keeping only those where the projection and the join returns a value.

(Refer Slide Time : 23.12)


Revise ((X), Y)

Revise (D_X, D_Y, R_{XY})

```
for each $a \in D_X$
    If there is no $b \in D_Y$ s.t. $(a,b) \in R_{XY}$
    then delete a from $D_X$
```

Complexity of Revise $O(k^2)$

N variables
K values
e edges

Revise(D_X, D_Y, R_{XY})
↳ delete c from D_X
new $D_X = \{a, b, d\}$

$$D_X \leftarrow D_X \cap \pi_X (R_{XY} \bowtie D_Y)$$

Now let's talk about making a network arc consistent. The first algorithm that we will see is called AC-1. It's a very commonly known algorithm and before we do that let us try to work at an example. This example is very similar to what we have been looking at. Let's say we have 3 variables and let's say we have 4 or 5 values in each domain. Let's call these variables X, Y and Z.

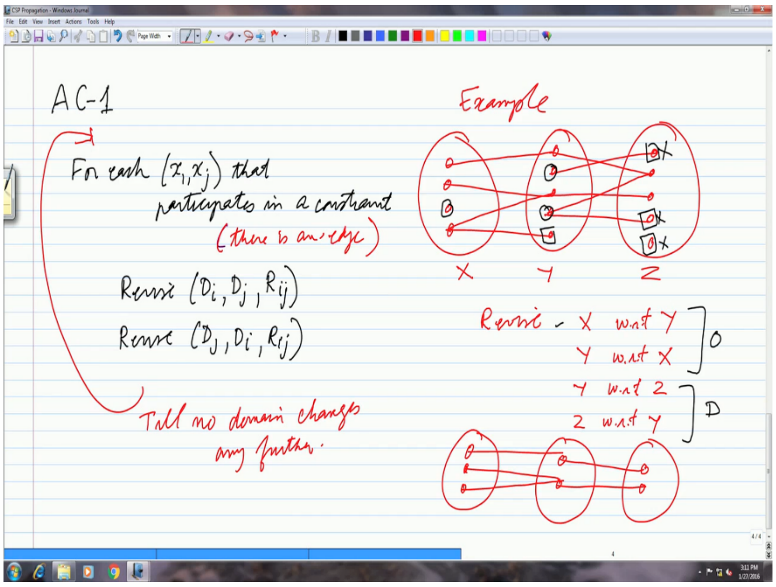
What do I need to make this whole network arc consistent? Obviously, I want to revise X with respect to Y. I want to revise Y with respect to X. And we will do this only for the relations which are explicit. So Rxy is explicit and Ryz is explicit. For Rxz anything is allowed. So we don't need to revise there essentially. We need to revise Y with respect to Z and Z with respect to Y.

So this we can write as follows that for each pair (xi, xj), that participates in a constraint, which is equivalent to saying that there is an edge, we have to make two calls to revise. One is Revise(Di, Dj) and the second one is Revise(Dj, Di). We will use the same relation Rij because that basically contains both ways. So we'll not go into the integrities of how we represent this relation. Essentially we want to make two calls. So for each pair of variables XY, we'll make these two calls and for YZ we will make these two calls because by definition we have to first make each variable arc consistent with respect to the other variable in both directions and then we have to do it for all possible pairs of variables.

The question is if I do this, whatever I've written on the right hand side as an algorithm, that for each pair I call revise, call X with respect to Y, Y with respect to X, Y with respect to Z and Z with respect to Y, will I get a network which is arc consistent?

It turns out that in this case it did become arc consistent but you can see that when we delete some values, that you may be disrupting the consistency of some other variable. In the next class I will choose an example where this doesn't really happen and then we will see that we need to put revise in a loop. And the condition for that is till no domain changes any further. I'll take up another example in the next class and we'll just try to see that just one round of revisions or one round of revise calls is not enough. We have to make multiple rounds. So we'll make a case for this and then we'll talk about efficiency in the next class.

(Refer Slide Time : 30.31)



AC-1

```
For each $(x_i, x_j)$ that
        participates in a constraint
        (there is an edge)
    Revise $(D_i, D_j, R_{ij})$
    Revise $(D_j, D_i, R_{ij})$
Till no domain changes
any further.
```

Example

Revise - X w.r.t Y] 0
Y w.r.t X
Y w.r.t Z] D
Z w.r.t Y

CSP Propagation - Windows Journal
File Edit View Insert Actions Tools Help
X
Y
Z