Constraint Satisfaction Problems

Prof. Deepak Khemani

Department of Computer Science

Indian Institute of Technology – Madras

Module – 3, Lecture – 05

We have been looking at arc consistency and before we move on to path consistency, let me give a notion of generalized arc consistency for non-binary networks. Very often in the real world we have to deal with problems which are not binary constraint networks and which have to be expressed as higher order networks. We need to define a notion of arc consistency over such networks. Remember that basically arc consistency says that if you choose a value for a variable, then you can extend it to the next variable and so on.

Look at an example like this. Suppose we say $X+Y+Z \leq 15$. You can see that this is a constraint over three variables X, Y and Z. Now suppose we also have another constraint which says that $X \geq 13$. That's a unary constraint. That's a separate constraint essentially. Now what happens because of this? We know that we cannot have arbitrary values in the domains. In particular, the impact of this constraint is that $Y \leq 2$ and $Z \leq 2$. So it has the same flavour of arc consistency which says that you take a variable and because of a constraint you prune the domain of the variable. This is what generalized arc consistency does. In this example, we are looking at a constraint over three variables but we can extend it to any number of variables and that's the definition we are looking for essentially.

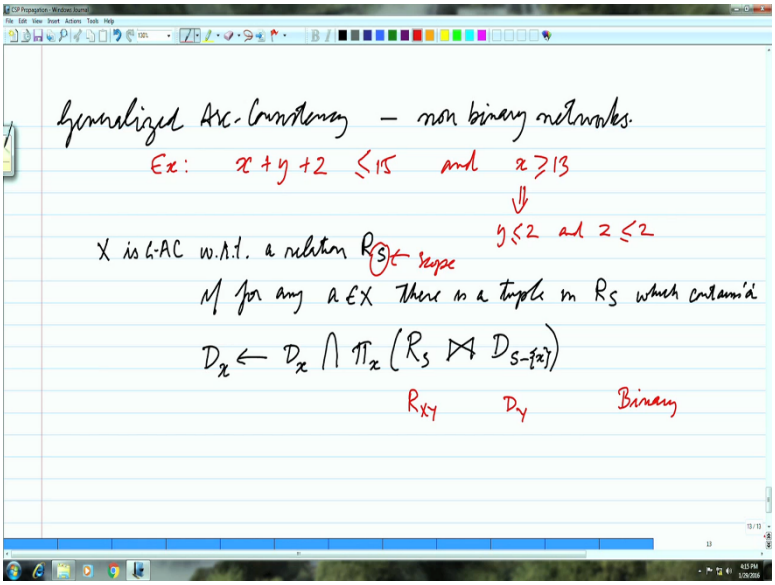
So we say that a variable X is generalized arc consistent with respect to a relation or a constraint R, whose scope is S, if for any $a \in X$, there is a tuple in Rs which contains a. When I say contains a it basically means in the proper position, i.e., where ever the variable's position is in that tuple, it contains that value a. So if that is the case, there is at least one tuple in the relation such that a finds a value. That means if we had given a value to a, then that constraint is satisfied and we could have given values to other variables. So that's the generalized notion of arc consistency that we are looking at and we could do a revise operation for this by simply saying revise(Dx), prune the domain of X. So this is very similar to the definition that we gave for revise over a binary constraint network except that in this

example it is not binary and you are working with a larger constraint and you are doing a join with the domain which contains S minus X. $D_x = D_x \cap \pi_x (R_s \infty D_{S-\{X\}})$.

So when we are looking at non binary, its simply extended to more variables, that instead of binary constraints we have a variable over S variables and the join is done by removing the variable X and seeing whichever other values we have. We are pruning the domain of variable X essentially.

There is a slightly different notion of arc consistency in which we see what is the influence of X on the relation amongst the remaining variables. We can prune the remaining variables and this is called relational consistency. At some point, later I will mention it again but this kind of gives us a nice stepping stone into path consistency because the key thing that we do in path consistency is to prune relations whereas in arc consistency we are pruning domains. We can also think of them as inducing relations of arity one or unary relations. In path consistency we'll go back to binary constraint networks to start with. We will see that we are inducing binary relations on networks essentially.

(Refer Slide Time: 7.37)



One thing that you should notice as we describe at path consistency is that when we were talking of arc consistency we were talking about variables which are related to each other. We said that if there is a constraint between X and Y, or in other words if there is an edge in the constraint graph between X and Y, we will do the pruning of domains at either end of the

edge but when we look at path consistency we'll see that we will even look at pairs of variables which don't have an explicit constraint. And this makes sense because if we don't have an explicit constraint it basically means that everything is allowed. We'll see an example there but let me first start with the definition.

We say that the binary constraint network is said to be path consistent or three consistent if every pair of consistent assignment of any two variables, can be extended to a third variable. Take a look at the example that we had seen in the previous class of the network of three countries with two colours. This network that we saw in the last example is a map colouring problem.

We saw that this network was arc consistent but not path consistent. Why is that? If I give names to these variables – X, Y and Z, you can choose pair of values for X and Y. For example, you can choose X = red and Y = blue. That's consistent enough and in fact the network is arc consistent and we say that for any value you choose for a variable, you will be able to choose another value for the related variable. You can choose X = red and Y = blue but we cannot extend it to variable Z because we can choose neither red nor blue for Z and in fact this problem has no solution and in this example, we can see that this network is not path consistent.

Another way of looking at this is to say that every edge in the matching diagram can be extended, well extended is not quite the right word but I hope you will understand what I'm trying to say, to a triangle with every other variable. If you compare this to what we said about arc consistency, we said that arc consistency says that every point in the matching diagram has an edge which takes it to every other domain. So for every variable, every point is connected to a value for every other variable. So, every point can be extended to an edge in the sense that you start with any point in any variable and you will find an edge which takes you to another point in the domain and so you have found an edge essentially.

What we're saying here is that you find any edge in the matching diagram and you take any other variable and you will find that there is a point there which makes a triangle with the edge that you have chosen. So that's what I mean by saying that every edge in the matching diagram can be extended to a triangle with every other variable essentially.

Let me give another example here. This is also something that we have seen. It also has three countries and two colours as before. What I've drawn is a constraint diagram or a constraint graph. I've not drawn the matching diagram here. So there are three countries. X is adjacent

to Y and Y is adjacent to Z but we don't know anything about X and Z or there is no constraint. Nobody has said that they must be coloured differently. Is this network arc consistent and is it path consistent?

It's definitely arc consistent. When I ask this question, you must keep in mind that sitting here between X and Z is a universal relation. What does that mean? We are not constrained what pairs we can choose essentially. So if I want to actually draw the matching diagram for this problem, then what I would see between X and Z is that r is allowed with b, r is allowed with r, b is allowed with r and b is allowed with b. So you must keep in mind that if a constraint is not explicitly specified, then we are not barring any tuples which means you are allowing all combinations of tuples. You are allowing X = r and Z = r, X = r and Z = b, or any other combination. So that's the universal relation. So implicitly whenever there is no constraint specified it means anything is allowed. So keeping that in mind, is this network arc consistent and is it path consistent? If you notice there are only two edges between X and Y and there are only two edges between Y and Z but there are four edges between X and Z because it's a universal relation.

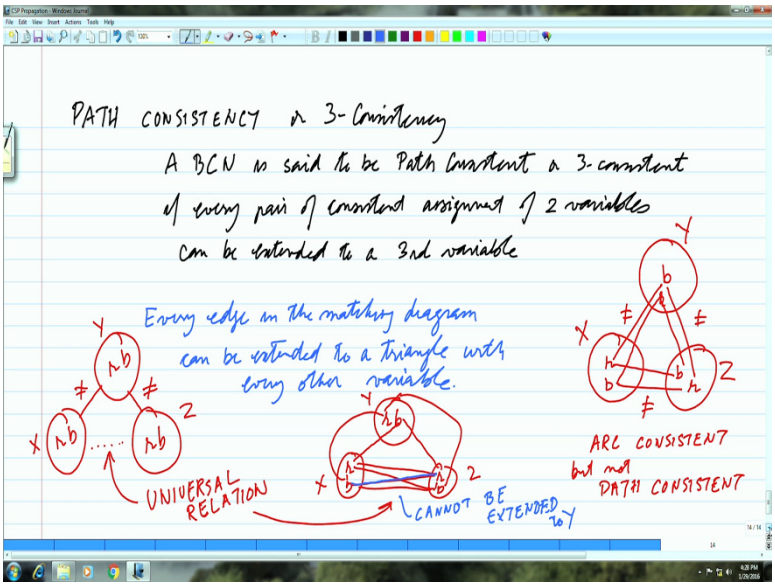
Now if you look at the definition of arc consistency, this network is clearly arc consistent. Even the other example we saw which didn't have a solution was also arc consistent. This luckily has a solution. And it is also arc consistent because for every value in X or Y or Z we can choose a corresponding value in X or Y or Z in any other variable but it is not path consistent because there are edges we can choose between X and Z which cannot be extended to Y. For example, if you choose X = b and Z = r, it cannot be extended to Y. The search algorithm that we will be looking at will essentially say choose a value which is consistent with all the constraints that are given to you.

You choose for the first value X = blue. Then you, let's say, go to Z and that's the order in which you are processing the variables and you say I choose Z = red, which is fine as far as the constraint goes but then I cannot choose a value for Y. I cannot choose red because Z already is red and I cannot choose blue for Y because X already is blue essentially.

What will enforcement of path consistency do? Its analogous to what we did in arc consistency. In arc consistency we inferred unary relations. So that is, from two consistency we inferred unary relations. In path consistency, which is three consistency, we will infer binary relations. So we will impose a relation between X and Z and what is the relation? The relation is that X = Z. Its that they must be coloured with the same colour essentially and we

had actually seen this. When we looked at composition of relations we said that if you compose Rxy with Ryz, you get a new relation Rxz which is in this case is the relation that we are looking for. And that is what path consistency is all about essentially. The interesting thing about path consistency is that it changes the constraint graph. It adds edges to the constraint graph.

(Refer Slide Time: 20.12)



What we need is a corresponding operator which we will call as revise three. There are two different notations that you can come across. The notation given in Rina Dechter's book is that you're revising an edge XY with respect to another variable Z and in my book I use a slightly different notation which is similar to what we used for revise two. It's that you are looking at the domain of Z and you're looking at the relation XY but you're also looking at YZ and XZ. You're looking at all three relations essentially.

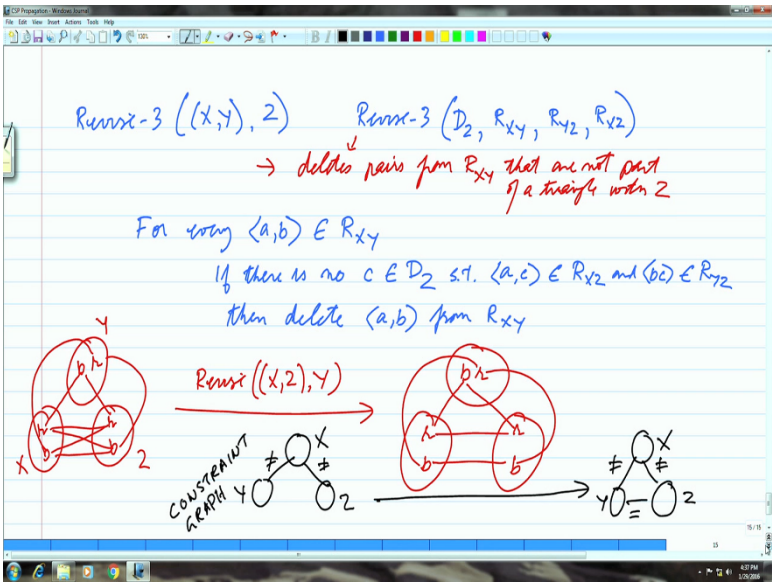
And what does this do? It basically deletes edges or pairs from Rxy that are not part of a triangle with the third variable. We can express this as an algorithm. It's that for every pair (a, b) ϵ Rxy, if there is no c ϵ Dz such the pair (a, c) ϵ Rxz and the pair (b, c) ϵ Ryz, then delete (a, b) from Rxy. So the fact that we are representing relations explicitly makes life reasonably simple for us. And now you know a relation is just a collection of tuples and we are simply deleting tuples from there.

Lets go back to the map colouring example that we just looked at where we had X Y and Z and the colours red and blue. If I use Dechter's notation, then we revise XZ with respect to Y.

I will get a network where you don't have (b, r) and (r, b) in Rxz. So I've changed the constraint graph. This did not happen when we did arc consistency. When we did arc consistency we had changed domains and domain changing doesn't change the graph because some values just vanish from the domain but when you do path consistency then you may end up adding edges to the constraint graph. So originally what was seen as a universal relation is no longer a universal relation and now it has become a equality relation. Earlier in the problem statement we said that you can choose any value for Y and any value for Z. That was in the problem statement but as we think about the problem a little bit or as we do propagation, we end up inducing a constraint YZ and it says that the value you choose for Z must the same as the value you choose for Y. So we have added a constraint to that. So we're changing the constraint graph.

As we look at algorithms for search we will see that the complexity of those algorithms will depend upon how many edges are there in the constraint graph. We saw that for AC-3 that property was true and it'll also be true for some search algorithms that we will see later. So there are two things happening at the same time. As we do propagation or consistency enforcement networks are becoming tighter and tighter. They may express the same solution but the networks are tighter which means that they have a smaller number of constraints between them. The number of edges in the constraint graph may increase and sometimes there is a trade-off that some algorithms may become more expensive.

(Refer Slide Time: 28.24)



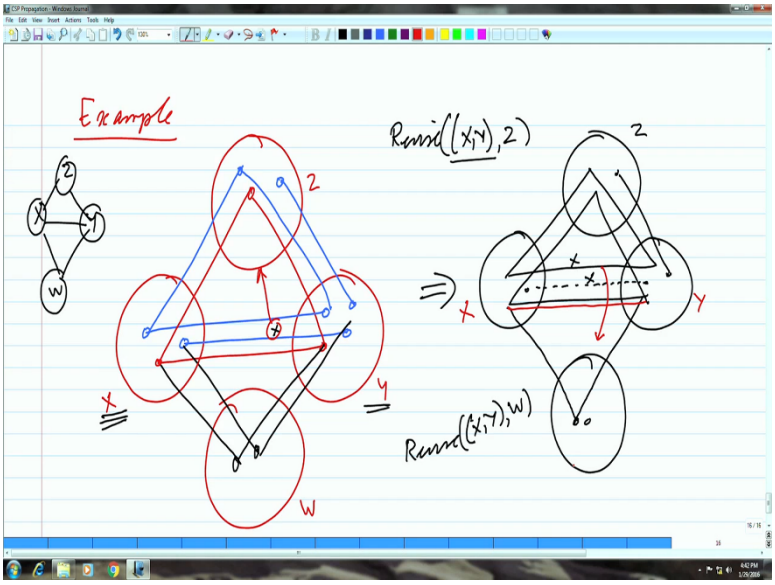
Let me try to give you a flavour of what's happening here by looking at another example. Let's say there are four domains –X, Y, Z and W. Let WZ be a universal relation. I do revise XY with respect to Z which means I am looking at every edge between X and Y and seeing whether I can find a point in Z. If there is a point in Z then I'm happy that I can keep this edge between X and Y but on the other hand if I see another edge between X and Y which doesn't have a point related in Z, then I will delete that edge. Notice that when I am making this call to revise I'm revising XY, so I'm only revising the edges between X and Y.

So if you see there was one edge which got deleted because there was no point in Z for that and the remaining would be as it was. Let us say the remaining thing has only two points and they are connected as shown, then the edge has been deleted. So there were just two edges and if I now do revise XY with respect to W, then I will end up with only one edge.

So at the end of these two calls to revise, I would be left with only one edge between X and Y. The other two edges have got deleted. One got deleted in a call with respect to Z which is shown and the second one got deleted because of a call with respect to W because it couldn't find a triangle in W. So at the end of these two calls to revise, essentially the only relations that will survive are the two triangles and if I do more calls other edges will vanish.

So essentially in path consistency the edges start vanishing from the graph and at the end every edge in the matching diagram will be extendable to every other variable with a triangle. The simple algorithm to do that is PC-1, which is similar to AC-1, which is brute force and simply says that as long as no relation is changed, or as long as some relation changes call all combinations of revise. In the next class, we'll take this up again and then we'll look at an improvement which is analogous to AC-3 in which we do selective calls to revise and we'll try to analyse the complexity of both revise three as well as the PC-1 and PC-2 algorithms.

(Refer Slide Time: 33.07)


Example
Z
X
Y
W
Remd((X,Y),Z)
Remd((X,Y),W)