Artificial Intelligence: Constraint Satisfaction Problems
Module 4: Directional Consistency
Lecture 3: Directional Arc-Consistency and Tree CSPs
Professor: Deepak Khemani
Department of Computer Science and Engineering, IIT Madras

Keywords: directional consistency, i-consistency, directional i-consistency, directional arc consistency, tree CSPs

Okay, so in the last class we looked at this notion of width and induced width and orderings associated with that. So assuming that you can find, you can use one of the algorithms, let's say min induced width algorithm or min fill algorithm to find an ordering of a of a graph. The next question that we are interested in is if you want to enforce certain amount of consistency on the ordered graph now, what is the degree of consistency that you want to enforce. That is the question we want to ask and the objective is to try and make the search backtrack free essentially. So this notion of, there is this notion of what you call as directional consistency. So there are two aspects of this, one is that it has a direction and the direction is dictated by the ordering that you have somehow chosen. And the other is that local, which means the degree of consistency that you want to enforce essentially. Now in general if you look at the diagram that we had drawn earlier that if you want a node i to be given a value, then in the ordering you may want it to be related, it may have certain parents in the ordering, then let's say this and this and there are other nodes in between which you are not bothered about for this particular node essentially. So given given this ordering, if you want to give a value to i, then in this example there are three parents, and the value that you are going to assign to i must be consistent with the values of those three parents. So essentially it means that if I take these three nodes, any value to this must be, or must allow a value for i, or $x_i$. So this is the notion of directional consistency. I am not interested in arbitrary consistency, which says that you know if you give a consistent value to any subset of variables, then you can extend it to one more variable. That's the notion of directional i-consistency. The notion of directional, that was a notion of i-consistency. The notion of directional i-consistency would be that if a node has $i - 1$ parents and if you assign a consistent value to those parents then you should be able to find a value for that node essentially. So what is the relation between width and remember that width is of an ordering or induced width of an ordering, which the amount of consistency or the degree of consistency that one wants to enforce to ensure backtrack free search. That is the main question that we are looking at now. Okay, so if you look at the graph that I have drawn here and if you talk about arc consistencies, so we will start talking about arc consistency first. You can see that there are four nodes, which are connected, the ones that we have drawn and we have this notion of arc consistency amongst all those nodes, but if I name those nodes for example, if this node is j for example, if you look at i and j, then what do I need to do, I need to do arc consistency only in one direction, in the sense that I only need to make j arc consistent with respect to i. I do not have to bother about making i arc consistency with respect to j, and the reason for that is that we have already frozen an order in which we will process nodes. It basically means that as long as j is arc consistent with respect to i, then I will always find a value for i because that is the meaning of saying that node j is arc consistent with respect to node i, that for every value that I choose for j, I will find a value for i. So if I, as long as I can prune the value in j, I don't have to bother about pruning the values in i, because I will only give a value to i after I have given a value to j. This order is giving us this notion of directional consistency. So I need to do only in some sense if you look at arc consistency only half the work, only one side of consistency I'll need to enforce, and that obviously saves me time, not only it saves me time because it is half the work, but also saves me time because propagation does not happen, that once you have deleted a node, pruned the node, it is never going to be pruned again essentially. So we will look at that.

So let's define the notion of directional arc consistency. So we will say DAC. So let's give a

definition: a network R, which means a set of variables, set of domains, and a set of constraints is, I will just use the term DAC here, with respect to an order r, order d, if for every variable or if every variable $x_i$ is AC respect to every $x_j$ where $j > i$. So that is a simple notion of, that that that if you are related to a variable $x_j$, then you better be arc consistent with respect to j essentially. So I must add of course $x_j$ that is related to i, which means $R_{ij}$ is a relation in my domain. So let's write an algorithm for this and the algorithm is called DAC(R). So we had this algorithm called Revise and we had written two notions of it either we revise $x_i$ with respect to $x_j$ or the other way of calling it was with $D_i$, $D_j$, $R_{ij}$. So we can see that there is only one loop here and you are simply going from the last to the first, so let's do a small example and let's choose a equality relation rather than inequality relation because they are kind of easier to visualize, so let's say I have r, b, g as three colors here and let's say r, g and y here and let's say r, y and b here, and let's say r, g and y here, and the relation is equality which I will write like this. So we are basically saying that the two related variables must get the same color essentially whatever the problem is, it is not map coloring it is something else, but you want the same color to be given to that essentially. Maybe they belong to the same team or something like that. so let's do this algorithm now. The first step would be to consider the last variable and when you look at the last variable it is related to the second last variable. So the first thing would be its influence on this and essentially you will prune from this anything which cannot be extended to. So let me just call them, let's say this is A, B, it is always good to give names. So first we will look at D and D is related to C, so in the first step, in the first cycle, you will prune from C all those values which do not have a matching value in D, so r has a value in D and y has a value in D, and b does not have a value in D, so we will prune this one value from this domain. Then we will come to C in the second cycle. Now j is become C so to speak. We will prune A with respect to C. So this is our second step. In the second step because it is related to that r has a value in C so it is fine. b does not have a value now in C, so b will go from here and g does not have a value in C, so g will also go from here. In the third step, we will look at B because there will be only three cycles in this. There are only four variables and it has no effect. In the sense it does not do anything. So what we get at the end of this is this new problem, CSP problem, in which A's domain has been pruned down to r, B's domain has not changed, C's domain has got r and y, and D's domain has not changed. So the number of deletions we have made is small and you can see that this is DAC, but not AC. So when I say AC, I mean full AC. It is not full AC because if I for example chose a value for B let say green, I cannot get a value for A essentially. So B is not arc consistent with respect to A, but A with respect to B and that is what matters and A is with respect to C and that is also what matters because once you choose a value for A, which is red, I can choose red for both B and C, and once I choose a value red for C then I can chose a value for D essentially. So you can see that in this particular example, it results in backtrack free search, but this is just an example, it is not the proof of the fact that we may be argue about that a little bit, but first let's talk about the complexity of this algorithm. What is the complexity? So we are going to process e edges in the graph and in each edge we may call Revise once and that will give us $k^2$. If you remember Revise complexity is $k^2$. So this algorithm is $ek^2$. If you remember the complexities we had considered when we were considering full arc consistency, we started with AC1 which had much higher complexity, then that was $nek^3$ or $nek^4$ or something. Then we looked at AC3 whose complexity was $ek^3$, and then we came to AC4 which was, whose complexity was $ek^2$ and this is basically the bottom that you can hit, it is the rock bottom, in the sense that even to verify whether the network is directly arc consistent you will have to $ek^2$ amount of work essentially. So it is an efficient algorithm essentially.

Okay, so now let's talk about handling trees, CSPs where the constraint graph is a tree. So the basic question is for tree CSPs, what is the min induced width. If you can answer this question, then we can get some clue as to how much consistency is enough. So let's try to answer this question. So let's just do this with an example, so let us say I have this tree, A, B, C. We had looked at a similar example sometime ago, but let me make it a little bit larger, F, G, H. So this is clearly a tree graph. Now, one thing you will observe here is that an ordering would be bad if A were to be at the end

because A would be connected to B and A would be connected to C, and A would be connected to D, and we had seen a similar example earlier that if we, let's say we are doing map coloring now with three colors and if you choose, so let's say all domains r b g, then you can see that this is bad ordering because if I choose red for this and blue for this and green for this, I do not get a this thing. So obviously we are not talking about bad orderings. What is a good ordering? So let's say the min, so it turns out that for trees, if I look at the min width ordering, what does it say, it basically says put the smallest degree nodes in the end. So I can choose any smallest degree node. I can put D in the end, then I can put C, then I can put A, because by then A would have become degree 1 then I can put E, and I can put H. You can verify that this is indeed a plausible min width ordering and the graph is A is connected to, A is connected to B and to C and to D and D is connected to F, where is F? Sorry, I have not put F here is it? Okay, so let's say F is here, so B is connected to F, and B is connected to E, and F is connected to B, and F is connected to G, and F is connected to H. Width, I will leave it to you to verify, is one. Every node has exactly one parent essentially. D has only A, C has only A, A has only B, E has only B, H and G have only F, and so on essentially. So the next next thing to observe is an implication of this, so if width is equal to 1, then induced width is equal to 1. Why is that? Because if width is one, has only one parent. Each node has only one parent and if each node has only one parent, you don't have to add any connections because there is only one parent and therefore induced width is equal to 1 essentially. Now, you can see that a tree can always be arranged in a ordering of width one, which also means that the ordering, the induced width of that ordering is one. What is the implication of induced width. The induced width tells you how many parents you are connected to in the worst case, right? In this case, the answer is one. Which means only one parent is influencing your value that you can choose and if induced width is one, this implies DAC is enough. Because you only need to make the parent consistent with a given node essentially. So in this new ordering as long as the value of A is such that if it is consistent with C as well as consistent with D, then whatever, then whichever values that I choose for A, I will be able to choose the value for C and D. Likewise if you start from the beginning, which is F in this case, that is the first node, if F is consistent with B and G and H, but this is only arc consistent in one direction then whatever value I choose for F, I will be able to choose the value for B because F is arc consistent with respect to B and I will be able to choose a value for G and for H and so on and so forth. So handling tree constraint satisfaction problems is a relatively easier task, but if there is no ordering which is of min width or if the width of a graph is more than one, which means there is no ordering, which has width one, which means that there are loops in the graph, there are cycles in the graph, then things become a little bit more complicated and we know that when there are cycles, a node will always have two parents, some nodes will always have two parents, which means we will have to worry about the consistency of those two parents, so we must give values to those two parents which are such that you can give a value for this node whose parents are there. It basically means we will have to do consistency which is more than arc consistency, and in fact in that case path consistency. So we will look at path consistency in the next class and we will also try to generalize that to the notion of higher order consistency and then try to come to a conclusion about how much consistency is needed given that you can produce a graph of a certain width essentially. So again when you are talking about width, you must distinguish between min width and min induced width. Min induced width is a little bit harder to find, but min width is easier to find, in fact we have a greedy algorithm. Very often min induced width algorithms will also produce good orderings, but not necessarily the minimum width ordering. That is a little bit of a problem. We will address that. So we will take that up in the next class.