Artificial Intelligence: Constraint Satisfaction Problems
Module 4: Directional Consistency
Lecture 2: Min-Width and Min-Induced-Width Ordering
Professor: Deepak Khemani
Department of Computer Science and Engineering, IIT Madras

Keywords: ordering, parent, fixed ordering, dynamic ordering, width of a node, width of a graph, min-width algorithm, induced graph, induced width, min-induced-width algorithm, min-fill algorithm

Okay, so let's continue with study of solving constraint satisfaction problems. In the last class, we had started talking about the effects that order of variables has on the process of finding the solution. So let's investigate that a little bit more today. So, in general if you see, solving a CSP would involve search, that's one of the techniques that we are going to apply, and the general idea here is that, for each variable assign a value which is consistent of the partial solution constructed so far. This is a simplest thing that we can do. Later on, we will see that it's not just in the past that you will look for consistency, but maybe also in the future; those techniques would be called lookahead techniques, but we'll come to those later. Now, what we saw, or, in the last class, was that if we had a tree network, which means that the constraint graph was a tree, then certain orders were better than other orders essentially. So we want to explore that idea. So there are two things that we are worried about, that whether some orders are better than others, and secondly, how much of consistency enforcement do we need to do to achieve backtrack free search essentially. So remember that backtrack free search essentially says that you don't have to backtrack; that for every new variable, has a value, and then further it can be extended. We'll look at those concepts again. But in general if you look at the order, that if you're given some order of variables, let's say we call it x_1, and x_2, and so on, upto some x_n. So we have decided that this is the order in which we will process variables essentially. Which means we will look for a value for x_1, we'll look for a value for x_2, then look for a value for x_3, and so on and so forth. And then, if you're looking in general for a value for x_i, then what you need to do is to look at all the variables which are related to x_i, let's say these three. So these are parents of x_i essentially. So what is the effect of order? The effect of order is that if you, and we saw this in the last example, and we'll try to, kind of, formalize that notion today, is that certain orders are better than other orders because, for example, good orders, or good orderings let me say, have fewer parents. Now since since the value that you want to assign to x_i should be consistent with the partial solution, that we will interpret to saying that consistent with parents essentially: choose value that is consistent with parents. So you must remember that parents already have a value. So there are two things here: one is that if a node has fewer parents, that means there are fewer nodes which are constraining what values it can take, so orderings which have fewer parents are better essentially. And we had started talking about that in the last class, we hadn't yet started talking about the notion of width, and we will continue with that today. The other thing which affects this ability to find a value for x_i, if if, or shall I say if the parents are well behaved essentially, and by this I mean that they have taken values which will allow a value for the child x_i to take place. And that is basically the idea of consistency enforcement. So we want to enforce certain amount of consistency, we want to make parents only take those values which will enable the child to take a value, and we will look at that, but we do not want to do full consistency. For example, we do not want to do full arc consistency because that involves more work essentially, especially because we have chosen a given order. Now this order may be fixed, and that's what we will explore to begin with, or it may be dynamic. So in fixed order, of course, before you start searching, you'll freeze the order, and that's what we're going to look at to start with. In dynamic ordering, you may choose the ordering as as the search goes along, and we will see that there are many factors that play which help you decide which orders are better essentially. So the fixed orders that we are looking at are typically going to be based on graph properties, so we will start looking at some graph-based properties like width, and and decide orders based on that. But dynamic ordering may also depend upon what are the values that are allowed essentially. For example, the simplest case is that if a variable has only one value in its domain, then it's better to assign that value so that everything else will be consistent with what you can give to that, because you can not give any other value. So dynamic techniques will try to do this sort of a thing, that look at the more constrained variables first, and then move to other variables. But we'll come to that later essentially.

So, what we want to look at is these graph concepts which help us find good orderings. And we had started talking about this in the last class. We had said that, given a graph: a set of vertices and a set of edges, and an ordering d, which say we will call as x_1, x_2, x_n, we had defined the following notions: width of a node, and when we say width of a node then we always here mean with respect to an ordering d, because the ordering d determines who is the parent of who essentially. Because if two nodes are generally connected, the one which is evaluated first becomes the parent of the second one essentially. So the ordering decides who are the parents, so, and the width of a node basically counts the number of parents essentially. So clearly, it is a property of the ordering as well as the graph connectivity essentially. Then we had said width of a graph, a graph given the ordering, which I will write as (G,d), which means you are given a graph, and you are given an ordering, is the maximum of the width of its nodes, or vertices essentially. So, so, width of a graph given an ordering basically tells you, in the worst case, how many parents a node will have essentially, and that will basically determine how easy it is for you to assign a value to that or not essentially. And then, finally we say that a width of a graph is the minimum over all orderings. So obviously this is just a property of the graph essentially. So the task is to now find, now this is easy, because a greedy algorithm works. So this is not a hard task, so let's write that algorithm. That algorithm is called MIN-WIDTH. Or we will also use the term Min W. And it says given a graph, so input $G = <V,E>$, and $|V| = n$. So given a graph with n nodes, you simply do the following: for starting with n down to two, you start with the last node which will be my last node in my ordering, and then which will be my second last node, and so on and so forth. Which should be the last node? The last node should be the one which has the fewest number of parents essentially, which means it has the lowest degree possible essentially. So, you choose a node r with the smallest in G. Then you put r in position j, and delete r and its edges from the graph essentially. In other words, $G \leftarrow G - \{r\}$, where you have to understand this minus means that you delete the node as well as you delete the connected edges essentially. So this simple greedy algorithm works very well for finding a minimum width ordering of a graph essentially. And as you can see its simply linear time ordering; you just have to look at all the nodes, and keep picking them one by one essentially.

So let's look at a small example which will allow us to do that. Let us say we have, we have these nodes, this is A, B, C, D, E, F essentially. So let's first take the lexical ordering. Let's call this d_1, and the ordering is simply A first, then B, then C, then D, E, F essentially. I must connect this A with B, A with C, A with E, then B with E, B with A, C with D, and C with A, and D with F. I have some ordering. And let's look at the reverse ordering for this, let's call it d_2, which basically begins from the other side: A, B, C, D, E, F. A with B, A with C. Okay, I think I've got all of them. So let's just look at these two orderings. We have done something like this. B and D, okay. So B is confused with, so in both of them I've missed out on B and D okay, fine. So, let's look at the width of these two orderings. So remember width is the number of parents, so it's one for this, two for this; you simply count the edges which are going towards the left, two for this, one for this, one for this, and root is always width zero; it has no parents. In this case, it is three, two, one, one, zero, zero essentially. So these are the widths of the nodes. The widths of the graph is the maximum of these, which is two here, and here it is three here. So clearly one of them is better than the other essentially. So now, let's do the MW algorithm, and see what it does. So maybe I'll make a copy of this, and then we'll keep deleting nodes from the graph as we go along. Now, you can see that that there is one node which has degree one, so we put F in the end, and then I delete, so I'll do this on this, I've deleted F, now I have, this is what remains of the graph. Now, there are several nodes with degree two, you can see E, D, and C are all of degree two, so let's randomly choose let's say C here, which means I delete C and its edges. Then I choose D because that's a node with degree one, and I delete its edges. Then I can choose any one of them, so let's say A. Then I delete A, then B, and then E. So I produce some ordering which is based on the MIN-WIDTH algorithm that we have. So let's just redraw the connections now. So A is connected to B, A is connected to E, and A is connected to C. B is connected to E, A, E, and D. Then C is connected to A and D. D is connected to C, B, and F. E is connected to, where is E? Oh it's in the corner. E is connected to A and B, that's already done. And F is connected to this. So I have this ordering now. And if you look at the the width, this is one, this is two, this is one, this is two, this is one, and this is zero. So this gives me a two. So it has, and and you can verify this, that in this graph any ordering will have at least, I mean you can't go below two, and one of the results that we will see is that if a graph has a ordering of width one, then it must be a tree essentially, and this is not a tree clearly, it has cycles. It will have these things essentially.

We also have a notion of something called an induced graph. So given a graph G, and an ordering d, you can induce another graph which we'll call G^*, with the ordering d, and basically, the algorithm for constructing an induced graph is that for j starting with n down to two, connect all parents of node j. So such a graph is called an induced graph, and for an induced graph we have a notion of an induced width, which is analogous to this. It's just that the graph has changed a little bit. So let me just go back to the previous example and see what the three different induced graphs are like. So, let me use some slightly lighter colour here, so this is the green colour. So F has, in the first graph on the top which is d_1, F has only one parent, so there's nothing to do here. E has two parents: A and D, sorry, A and B, but they're already connected. D has two parents: C and B, but they're not connected. So in the induced graph, so this is the induced graph, I will connect these. And C has two parents: D and A, which are connected. So I'm adding one edge here essentially. If you look at the second example, I have A is connected to B, C, and E. So I must connect B and C in the induced graph. Now notice one thing, that as I am constructing the induced graph, the width of nodes may increase essentially, because you're adding edges to them. So, for example, the width of B in the second graph has gone up from two to three because of this new edge essentially. So B is connected to C. Sorry, A was connected to B and C, so I connected C, and A was also connected to E and B, but that's already connected, and A is connected to E and C, so I must have another edge here: E and C. Now B is connected to, B is now connected to C and E, but there's already an edge there. It's connected to C and D, and there's already an edge there, and it's connected to E and D, there's no edge there, so I must add this edge. So three edges I have added here. C is connected to D and E, and there's an edge there. D is connected to F and E, so I must add an edge there. So you can see a lot of edges have been added and the width of this node has now become three, width of this node which was one initially has become two, width of this node which was one has also become two, and width of this has become one, and this anyway is zero essentially. Likewise in this. Anyway, so that's an example of an induced graph. And we have this notion of an induced width essentially. Now the trouble is, finding a min induced width ordering, is NP complete. Finding a min width ordering is easy; you can do it by a greedy algorithm, but finding a min induced ordering is NP complete, that's because you're adding new edges as you go along and that complicates matters. But good, reasonably good greedy algorithms, exist. So let's look at the simplest of them, which is the min induced width. So I'll just write MIW, which is min induced width. So you're given a graph, and you're given an ordering, and you want to find, sorry, you're given a graph, and you have to find a min induced width ordering essentially. For, so given a graph. So it's very similar to the min width ordering. For j starting with n down to two, r gets the smallest degree node, which is the same, in G, then put r in position j. The new step is connect r's neighbours, which basically means that for the set of edges, you augment with all those edges $\langle v_i, v_k \rangle$ such that $\langle v_i, v_j \rangle$ belongs to E, and $\langle v_k, v_j \rangle$ belongs to E. And then delete r. So I'll just use the older terminology of saying $G \leftarrow G - \{r\}$. So everything is the same except this new step that we have added, which says that as you are doing this task of ordering, now once you have ordered, you know what are the parents of a, once you have fixed the order of a node, you know that its, who its parents are, the parents are the ones that are connected to it. If they're not connected to each other, connect them essentially. So that's basically producing the induced graph essentially. And this ordering has been known to produce good results. There is a variation to this, which I will just quickly mention, which has empirically performed better. So, let me use this. It's called MIN-FILL, and the only difference here is this step here, that S is the same, and here you choose the node r such that the number of connections needed by its parents, or in other words, to make it an induced graph, is smallest. Slightly different from the min induced width. Min induced width was like a min width algorithm, which simply said choose a node with the smallest degree, and put it in the end. Then choose the next smallest degree and put it at the second last position, and so on and so forth. MIN-FILL is a variation, in which this one step is changed, is how do you choose a node r to be put in the position. And this one says that look for that node whose parents are already connected so to speak essentially. They require a minimum number of connections to be added to that. And empirically, this has been better. Both are, you can see, greedy algorithms, and both give you reasonably good orderings essentially. I mean, they're, by this I mean that most of the time they give you actually the minimum width ordering. And then in the next class onwards, we will try to look at the relation between width and the amount of consistency that you need to enforce for search to be backtrack free. Our goal is backtrack free search, and we are trying to find orderings which will allow us to do that essentially. So we'll stop here, and take that up in the next class.