Artificial Intelligence: Constraint Satisfaction Problems
Module 6: Search Methods: Look-Back
Lecture 1: Look-Back Methods: Definitions
Professor: Deepak Khemani
Department of Computer Science and Engineering, IIT Madras

Keywords: Lookback search, dead-end, conflict set, no good, safe jump, maximal jump

So we have been looking at search methods for solving CSPs. And today we want to look at methods which look back onto the search that they have done to try and backtrack in an intelligent fashion essentially. So generally speaking such methods are called lookback methods. And you might say, smart backtracking. So let us see what is the problem with the chronological backtracking first. So, if you look at this backtracking algorithm that we have seen. Then let us look at a small example of a search graph, let us say a map coloring problem, and we have x_1, which has let us say colors red and blue and green let us say. And we have x_2 which is also red, blue, green and we have let us say x_3. So I will use some unique color names here. So colors are 1, 2 and 3. Just for the sake of this example. Then we have x_4 which has totally unique colors, 4, 5 and 6 and let us say it is constrained here. And x_5 with let us say 7, 8 and 9 and x_6 with red and blue. So we have the six countries or six regions and we want to color them using the colors which are shown inside each node, and let us assume that we are looking at the backtracking algorithm. So, I want to illustrate what is the difficulty with chronological backtracking essentially. So, what is the search tree that this will explore? So, you start with some root node and then you look for values for x_1. So, the values for x_1 are red. So, let us say it is doing, we are simulating the algorithm. So we will not draw the other part of the tree. This is blue, but some parts we will draw just to give us a feel of the tree. And then we will go to x_2. And x_2 also we have three choices as you can see. So first you will try red but that is a dead end because there is a constraint between x_1 and x_2 that they cannot both be colored red. And then you will try blue, and if blue fails then you will try green and the same thing below these, which we will not draw the graphs for the moment. So, we have we are on the second color in x_2, and when you go to x_3, we have three colors, 1, 2 and 3. And likewise we would have the same three colors here 1, 2, 3 and so on. So all the tree grows like that. So x_3 let us say we choose 1, and then we come to x_4. And x_4 has three colors again 4, 5, 6 and the same three colors would be here and here and here everywhere esentially, because that is basically the options for x_4 and then we have x_5. So we have 7, 8, 9 and the same three colors would be here 7, 8, 9 and so on so there is a large tree sitting out here. And then we come to x_6. That is node we are interested in. First, we try red and that is a dead end because of the fact that x_6 has a constraint with x_2 as well as with x_1 esentially. So since x_1 has got red we can't choose red for x_6. So then we try blue, the second color that is available for. So you can't try red and then we will come to blue and blue also we cannot try because x_2 has got blue esentially in this. So this is a situation in which the backtracking algorithm finds itself in. It's got values for five colors x_1, x_2, x_3, x_4, x_5. It is trying to get a value for the sixth color which is x_6, and it has two options red and blue none of them works essentially. So, obviously the algorithm needs to backtrack essentially. And look for some other options for other colors for which you will be able to choose either red or blue for x_6 essentially. Now obviously when we look at this problem, we can see that the problem with not being able to find a value for x_6 is because it is conflicting with variables x_1 and x_2. x_1 has got red and x_2 has got blue, and because x_6 is constrained by x_1 and x_6, we cannot choose red or blue. So, now we want to look at algorithms which try to behave in what you can say that is a more intelligent fashion, that instead of jumping back to the previous variable, you should jump back to some earlier variable so as to solve the problem with this thing. So what does chronological backtracking do? Chronological backtracking will then try the same values red and blue for this. And they will obviously fail then it will try same values red and blue for this. So it is basically. Once this dead end is been encountered it will try a new value for x_5 and then the values red and blue for x_6, it won't work. Then it will go back and try a new value for x_5 which is 9. And then again it will come to red in blue which will fail. And then it

will go and try a new value for x_5 then x_6. And everything will fail, as you can see. This entire subtree, and it is a it is a large subtree is bound to fail. And the reason for that to us is obvious, because looking at the constraint graph we can see that x_6 is constrained by x_1 and x_2. So if you want, if you are not able to find a value for x_6, it is because we don't have matching values in x_1 and x_2 that have been chosen esentially. So here backtracking is explicit. Backtracking exhibits the behavior of thrashing. We call this behavior as thrashing. It basically it is trying out different values for x_3, x_4, x_5. Without making any headway with a value for x_6 essentially. And essentially we want to look at an algorithm which will allow us to do that. In fact we will look at two or three algorithms.

One algorithm which is kind of similar to the kind of reasoning that we have been doing here will be based on the graph topology. It will look at the graph topology and try to decide where to jump back. But you will also see that there are other approaches. The graph topology based algorithm would not look at what are the values that are conflicting essentially. So it is really because the values that we have chosen for previous variables that are conflicting. And in some way if we can sort of work with values, maybe we will get a different approach to jumping back smartly and in fact those algorithms are called back jumping algorithms essentially. So we will look at two or three approaches, one which will look at values, one which will look at topology, and then finally one which tries to combine topology with values essentially. We will see that each of them has some advantages each essentially. But before we do that we need some definitions. So, let a_i be a partial instantiation such that there is no value $b \in D_{i+1}$ and the vector a_i and $x_{i+1} = b$ is consistent. And we say that a_i is a DEAD END STATE. and we say that x_{i+1} is a dead-end variable. So, both these of course go together. Whenever we have a DEAD END STATE, we have a dead end variable. Which is the next variable essentially. Then we have this notion of a CONFLICT SET, which is similar to this notion that we have just defined, but it is a little bit more general. So we say that let a be a consistent assignment to some variables. then if there is no, the same definition $b \in D_x$, such that a, x = b is consistent we say that a conflicts with the variable x. Or that a is a CONFLICT SET of x. So, obviously in a dead end we have a CONFLICT SET. But that is just one example of a conflict set. Dead end is with respect to a particular search algorithm, which is in our case the backtracking algorithm and therefore we are identifying the next variable. A CONFLICT SET is a more general notion it simply says that the partial instantiation is not consistent with some variable x essentially. Then we have this notion of a minimal CONFLICT SET. We say that a is minimal if there is no subset of a which is a CONFLICT SET. If there is no subset of a which is a CONFLICT SET as well essentially. Then, we have this notion of a Leaf dead-end is actually what we defined as a dead end earlier. And so if a_i is equal to <a1, a2 .. ai> is consistent and is a CONFLICT SET of $x_{i+1.}$ Then we call it as a leaf dead end essentially. So, what we called as a dead end earlier is actually what we call as a leaf dead end. But as we go along we will see that you can have dead ends which are not leaf dead ends and that would happen is there if you backtrack to something and then you are at a dead end state essentially. So, this is essentially, the idea behind leaf dead end is when search is progressing, and a new variable or the next variable is a dead end. This is as opposed to INTERNAL DEAD ENDS, which are variables we have jumped back to. We will see that the first algorithm that we want to look at will be able to jump back from leaf dead ends so when I say jump back I essentially am trying to emphasize the fact that you are backtracking more than one level, you are not going to the previous variable but to some earlier variable essentially. And we will see that the first algorithm which is called Gaschnig's back jumping algorithm will be able to jump back from leaf dead ends, but not from internal dead ends. And then we will look at some graph based methods which will be able to also jump back from internal dead ends essentially. We have another notion which is called NO-GOOD. So a consistent partial instantiation a is a NO-GOOD if it cannot be extended to a solution. So, we can make an observation that conflicts are NO-GOODs by definition, because that is a notion of a conflict that that the conflict is with respect to a variable essentially. And conflicts are NO-GOODs because they cannot be extended to that variable that we are referring to. But NO-GOODs need not be conflicts. And the reason for that is with respect to a

variable. And NO-GOODs may not be a conflict in the sense that that for every individual variable you may be able to extend that no-good to a larger partial instantiation. But, you may not be able to find consistent value for the remaining variables essentially. So, can we... it is possible okay that we that we can extend a NO-GOOD to, so I must emphasize need not be. That you can extend it to let us say you have an NO-GOOD of five variables you can extend it to the six variable or to the seventh variable or to the eight variable individually, but not as a set essentially. So if you extend it to the seventh variable, you will not be able to extend it to the eight and so on and so forth essentially. NO-GOODs are of interest to us because these are consistent instatiations, which will always be dead ends in any kind of a search algorithm, and if you could somehow find them and remember them, then it might be worthwhile activity. And then we have similarly a minimal NO-GOOD, similar to minimal conflict that there is no subset of a NO-GOOD which is also NO-GOOD essentially. minimal NO-GOODs would be more interesting, because you could probably spot them earlier. So just imagine a search algorithm which kind of learns NO-GOODs on the way and then every time it sees a NO-GOOD it says okay I do not need to extend that solution any further. So it will save on search essentially.

Then we are interested in back jumping. That is the idea behind back that that if you are doing x_1, x_2, x_3, x_4, x_5 and if you are looking at x_6 in the example that we saw you want to jump back to x_2 and not to x_4 if you cannot find a value for x_6 essentially. So we have two notions associated with jumping back. We say that a jump is safe. So let us say that x_{i+1} is a dead end. It may be or may not be a leaf dead end. Then a jump to x_j which is less than x_i is SAFE if it does not preclude any solution. Okay so what is the idea here. Let me try to illustrate this. So you're... you are trying to find a value for x_{i+1} and you found a value for x_i and let's say this is x_1 and you got some value here then some value here then some value here and then so on and then you some value x_i essentially. And you are not able to find any value for x_{i+1}. what chronological backtracking would have done is that it will have gone to x_i. How do I show this, it would have gone to x_i and tried the next value for x_i and then the next value for x_i and so on essentially. So this is what chronological would have done. What a jump back does, is that it jumps back to x_j and try the next value for x_j essentially. Which means that it has not explored all this tree which would have been lying below x_1 but which had different values for the different variables essentially. And we say that.. we say that this jump is safe if there is no solution with all these things which are inside this tree. So, essentially you're jumping back to some variable x_j instead of jumping instead of backtracking to x_i and we say that the jump is safe, if you have not missed out on any solutions because you have this area which is in this shaded region which is actually a quite a large sub tree here. We are not seeing it as an exponentially growing tree. You are just excluding all that and you are going back to x_j and trying the next value for x_i which means that all those value for x_{j+1}, x_{j+2} up to x_{i+1} that you could have tried, you have not tried essentially. So if I go back to my example that I started with, you can see that this is a safe jump. The moment you see that x_6 R and B are not working, if you jump back to x_2 and try the green value there, you are not going to have missed out on any solutions, because all this region that is again shown in this enclosed area you would have been searching fruitlessly. So, that is a notion of a safe jump and any back jumping algorithm should only make safe jumps because we want our algorithms to be complete, and then we want to talk about maximal jumps. It basically says jump back as much as possible. So, ideally an algorithm should do safe and as well as maximal jumps. We will see that this notion of a safe jump is algorithm independent. Because the notion of a safe jump simply say that you are not missing out on any solution but this algorithm is... but this notion of a maximal jump is algorithm dependent. Because it depends on what data you are working with. Okay so we are interested in this safe and maximal jumps and we will look at three algorithms. In the next class, I will start by looking at an algorithm called Gaschnig's back jumping algorithm which does not look at the topology of the graph, but rather tries to collect some data about which values are working and which values were not essentially.