Artificial Intelligence: Constraint Satisfaction Problems
Module 6: Search Methods: Look-Back
Lecture 2: Gaschnig's Backjumping: The Culprit Variable
Professor: Deepak Khemani
Department of Computer Science and Engineering, IIT Madras

Keywords: dead-end, Gaschnig's backjumping, conflict set, culprit variable, leaf dead-end, leaf dead-end state, safe jump, maximal jump, internal dead-end

Okay, so we have started looking at algorithms which jump back intelligently when they reach their dead-ends. And we want to start by looking at one of the most famous algorithms in this, which is called Gaschnig's backjumping, and as you can guess this is due to the name of the person Gaschnig who devised this algorithm. Okay, so what Gaschnig's backjumping does: it maintains information of conflict sets, and it has a notion of a culprit variable. Okay? And so let me illustrate that with an example here. So if you are looking at trying to find a value for this $x_i$, and you have not been able to find any value, in the sense that all these crosses they represent the fact that you could not find each of these values as a value for $x_i$, and you have found values of previous variables, and so on. And let us say that the partial solution that we have constructed is something like this, so this is $a_{i-1}$ essentially. Then the definition of a culprit variable is: let $a_{i-1}$ is equal to $(a_1, a_2, .. a_{i-1})$ be a leaf dead end. So, sometimes we call this a leaf dead end state, because in the space of partial assignments this is a state essentially. So the culprit index $j$ relative to $a_{i-1}$. So, observe that this observation that we are making. So, what are we trying to do, we are trying to decide where to jump back from this variable $x_i$ which we could not find a value for $x_i$ and we are trying to jump back from $x_i$ essentially. And we are trying to identify as to which is the variable which I need to go back and change essentially. And we call that variable as a culprit variable. So essentially we are looking for the index of that culprit variable as to what is the $j$ such that if I go back and change $x_j$, I may get a new solution essentially. Or in other words, if I jump back to anything later than $j$ it is not going to help me essentially. So in that sense what is the maximum safe jump that I can make essentially. And that will be defined by this notion of a culprit variable. It is defined as $b = \min \{j \leq i-1 \mid a_j \text{ conflicts with } x_i\}$, and let's call this index b okay. So, let me again illustrate this. So this notion that we are looking for the smallest previous variable, the earliest previous variable such that it still conflicts with that essentially. So, essentially what we are saying is that, if you look at this for example, then you can see that this whole this entire partial instantiation, we know is a conflict because we could not find a variable for $x_i$ essentially. But if we exclude this last one, and if you look at only the previous one essentially. So if you just keep doing that, if we exclude this and if we exclude this and so on. And if you can exclude this one, so the red ones are what we are excluding and the remaining part

is still a conflict essentially, okay. Now what these red circles indicate here are that you can you can avoid that, you can remove that part of the partial instantiation and still the remaining part is conflict essentially. So, of course including all these are also culprits, but excluding these also culprits essentially. So, in some sense then what you cannot exclude is the culprit. Which means, that if I had excluded the culprit variable, let us call it $x_b$,is that if I had excluded the culprit variable, then $x_{b-1}$ is consistent with $x_i$. Which means that although $x_{b-1}$, I can try a different value or if I just take that $a_{b-1}$, is consistent with $x_i$ that if I just take those remaining values in this example $x_1$ and $x_2$ then I will be able to find a value for $x_1$, $x_i$ essentially. So $x_b$ is a minimal values amongst all these values which is still not consistent with that essentially and essentially what the algorithm Gaschnig's backjumping says is that you must jump back to this culprit variable essentially. And because supposing a jump back to here, is not maximal. So this is safe but not maximal. It is safe because I am not going to miss out to any solutions if I jump to this, but it is not maximal because I can jump back further essentially. And if I jump back to this, it is not safe. So the culprit variable is safe and maximal to jump back. If you can identify the culprit variable then we know where to jump back to essentially. And identifying the culprit variable may not be such a hard task. In fact, it can be done while you're constructing the while you are trying to find a value for the $x_{i+}$ variable essentially. So let me illustrate that with the same example.

Identifying the culprit variable. Now remember that we have this function called select-value function, in backtracking which basically takes a partial intantiation and finds the value for the next variable essentially. Now just try to see what this select value function does. In this case we will call it select value **GBJ** which stands for Gaschnig's backjumping. So, let me draw that same diagram again. You are looking at the $x_i$ variable and you are essentially trying to see if any of these values is going to work essentially. And what you have is this existing $a_{i-1}$ essentially and you want to extend this $a_{i-1}$ to some value from this domain of $x_i$ essentially. So you will try for each value one by one. So the way that this select-value **GBJ** will work is that it will try out increasing, so instead of instead of checking $a_{i-1}$ and $x_i = a$ for consistency. You incrementally try larger values or larger instantiations. And the idea here is to try and identify at which point the partial instantiation that we have has become inconsistent essentially. And this will of course be useful when we cannot find a value essentially. So, this incremental search let me denote by a red line here. So let's say we are looking for the first value so you try this the first value in the domain of $x_i$ and you incrementally, so let us say this this arrow represents the fact that you could come up to this point, when it became inconsistent okay. So, that means you saw essentially this much, this thing and here it became inconsistent. The partial solution that I have shown in this red shaded area became inconsistent with

this value essentially. So what this algorithm does is mark this. So if this is k then you say $latest_i$ is key. We will maintain a variable called $latest_i$ which will identify the culprit variable. But, we don't know yet of course, we don't even know whether there is a culprit or whether we will find the solution or not. When we look at the first value for $x_i$ we look at larger and larger of this partial instantiation till the point where it becomes inconsistent. And then we say okay keep that k in mind essentially so remember that k essentially. And then we do the same thing for the others so the second one may become inconsistent here, then the third one way become inconsistent here and then the forth one may go up to let's say here and then the fifth one may go up to here and the sixth one is here. So what is happening we are doing all this search we are trying out this values one by one and we find that that this doesn't work, this doesn't work, this doesn't work, this doesn't work, nothing works. And at for each of these attempts we have marked where it became first, it first became inconsistent essentially and then we keep incrementing this. So, inside the loop, if k is greater than $latest_i$ so let me call this one as $latest_1$. So, for each value that we are trying we look at larger and larger part of this partial instantiation, and if we find a larger one which is inconsistent then we just mark that as the $latest_i$. So, essentially what will happen at the end of this is that this $latest_i$ will come to the one which is the latest amongst all these arrows, or longest if you want to think of them in these arrows. And essentially you know that that is a variable that you need to jump back to essentially. So, if you just think about this a little bit, that is the basic idea behind Gaschnig's backjumping essentially.

So, let me try to illustrate that before we write the algorithm. So, we will call this **GBJ,** Gaschnig's backjumping an example. I don't know whether we need to write the algorithm. But, we can at least see this. So, let us let us go back to one of our favorite examples, which is the N Queens example and let us look at the six Queens example. So, these are the Queens: 1, 2, 3, 4, 5, 6 and this is the value of latest for each Queen that I will find. so initially of course I put this Queen and its value will be 0 because I do not need to jump back at all from here. Now one observation is that if we find, and we have not written the algorithm yet but we will do that if we find a value for $x_i$, then $latest_i = i-1$ because you would have incremented that partial instantiation and eventually come up to i-1 and we would have found the value for $x_i$. So, if we find a value for the next variable then the $latest_i$ would be the just the previous index essentially. Which means that if I find a value for the second Queen which is here, $latest_i$ would be 1. I mean it will just get incremented to 1 and then I will find a value for the third Queen and the $latest_i$ would be 2 and then I would find the place for the fourth Queen and the $latest_i$ would be three and then I will be able to find the value for the fifth queen here the $latest_i$ would be 4. So, that is because we are finding these values and and the process of finding the value the $latest_i$ would have got incremented

to the previous variable. But, we aren't able to find a value for this last Queen essentially. So, what is the which is earliest conflict which is happening here. For this it is 1 it is conflicting with Queen 1. For this it is conflicting with Queen 4. So, what do I mean by this, I mean for this value of of Queen six the first value is conflicting with Queen 1, the second value is conflicting with Queen 4, the third value is conflicting with Queen 2 that is our earliest place. So supposing I were I was trying to find the value whether I can place the Queen here. So I would see is it conflicting with 1 then is it conflicting with 1 and 2 and at that point I would have discovered that yes it is conflicting with 1 and 2 and therefore the value for this is 2 here. Likewise for this one the value would be the earliest conflicting is with 3 because you know it is conflicting with this here. For this one the earliest conflicting would be 2, sorry 3. I am mixing up things here. This would have been 4 and this would be 3 because this is the third Queen it is conflicting with, and this one is 1 because it is conflicting with this. So, essentially what are the numbers that we have written here is where did my partial instantiation stop. For the first value it stopped at Queen 1 for the second value it stopped at Queen 4 which is this one. And then for the third value it stopped with Queen 2 because it is in this one and so on and so forth essentially. And so MAX is equal to latest. $latest_6$ in this case is equal to 4. So, this is a maximum value essentially. So jump back to Queen 4 essentially which means this one. Now, if you remember the forward checking algorithm that we had done, let me quickly do that. We have this 6 Queens and we are trying to place them and we are cancelling future this thing. So, if you place this we cancel this. So instead of cancelling it let me write the Queen number here 1, 1, 1, 1, 1, 1, 1, 1, and I place the Queen here. Then I write 2, 2, 2 here 2. Because all these squares are being attacked by two and all the empty squares are being attacked by none so far. Then I place a third Queen which is here and I right 3 here 3 here and 3 here. And I place the ok so I missed out there so this was actually 3 because it is being attacked by the third Queen. And then when I place the fourth queen here I will write four here and four here and at that point you can see that forward checking backtracks. It is doing something very similar; after trying the fourth queen it backtracks from here; it does not even try the fifth queen essentially. But they are doing something similar in nature essentially. This Gaschnig's backjumping is not looking ahead, but when it is trying to find the value for the sixth queen it can identify that the fourth queen was the culprit and instead of trying to find the new value for Q5 it will just jump back to Q4 and try a new value for Q4. The forward checking algorithm with the numbers that I have written here, are essentially doing something similar essentially, that it is looking ahead trying to see which Queen will not be consistent with this value, so it is in advance trying to do that. And it can sort of, the moment it realizes that this is empty, the domain of this $D_6$ is empty then it backtracks and it back tracks to Queen 4 because it has only placed this, it has only placed these four Queens. And for that

backtracking thing naturally means backtracking trying to get a new value for Queen 4 essentially. So, they are doing something quite similar in nature essentially. But the approach is totally different essentially. Both of them will try to look for a new value for queen 4 essentially. The other thing I want to point out is that the latest value here was 4 essentially. So, it could jump back; but jump back is only for leaf dead ends. And this example will illustrate this whole idea. Now $x_6$ or Queen 6 is a leaf dead end. We could not find a value for that, but we knew that we should jump back to Queen 4. Queen 4 is this one. But all the previous queens; now if you look at Queen 4, there is if you look at this other diagram for Queen 4 it jumps back here, but this is an internal dead end. This second diagram, the forward checking diagram makes it quite clear that Queen 4 has become a dead end now because the only value that we had, we tried and it did not work. So it's so we have to and there is no other value left so it is a dead end so we have to backtrack from here. But the latest values that are stored with these Queens for which we found the values was that for Queen 5 the latest was 4. Queen 4 the latest was 3. For Queen 3 the latest was 2. For Queen 2 the latest was one. Because we found values for these variables and therefore that algorithm which we will write next simply extended the latest value. So the algorithm we will say jump back to the latest but the latest will always be there. So at the point where this where we do the backtrack the jump back for 6 is 4 but the jump back for 5 is 4 for 4 is 3 and 3 is 2 and for 2 is 1 essentially. Because that those are the values which are stored in the latest variable essentially. So which is one of the things about Gaschnig's backjumping is that you can jump back from the leaf dead end but you will not be able to jump back from internal dead ends, because the latest values would simply point to the previous variable essentially. So we will, in the next class we will write this algorithm and then we will try to look at a variation which will allow us to jump back from internal dead ends essentially.