

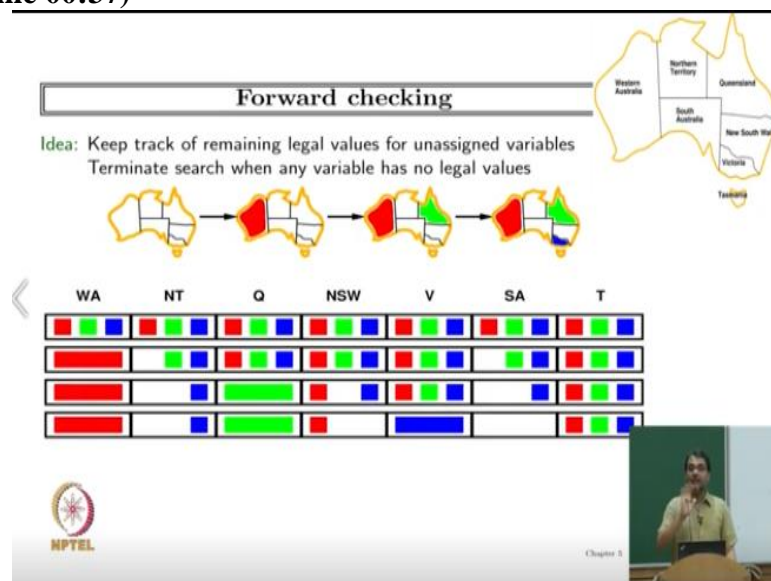
Artificial Intelligence
Prof. Mausam
Department of Computer Science and Engineering
Indian Institute of Technology – Delhi

Lecture-40

Constraint Stratification Problems: Inference for Detecting Failures Early, Part -5

Now, we will now come to the more interesting part, I mean this was interesting too, but we will come to the even more interesting part of the constraint satisfaction literature which is constraint propagation or inference.

(Refer Slide Time 00:37)



And the first idea that we will study is the forward checking idea and this forward checking idea is very simple. It is basically saying that whenever you assign a value, you figure out which values are no longer possible. In fact you need to do this for the least constraining value heuristic anyway, so initially you have all the variables and all the values and let us say we assigned Western Australia red.

Now as soon as we assign Western Australia red we remove that from Northern Territory, we remove it from South Australia and we know that there is no way we can assign these particular values? So therefore we will not even try the wrong value at all and we will also detect failure early as you can see now, that when we then assign let us say Queensland in some order, let us say we assigned Queensland so as soon as they assigned Queensland green.

We are left with we remove green from Northern Territory and we remove green from South Australia this is called forward setting. Whenever I assign a value, anything that is

unassigned we remove the values that are no longer possible and now last but not the least, let us say we assigned Victoria blue, for whatever reason, be going in some order. Let us say we assigned Victoria blue now is the state possible.

This state is not possible because South Australia no longer has a value. Why? because of Western Australia, it cannot be red because of Queensland it cannot be green and because of Victoria it cannot be blue but we and we get to know this because we were doing our checking in forward checking, we were maintaining which values are possible and as soon as a variable becomes empty no value is possible we backtrack it is possible that.

We were doing you know we did not do this and so we then went to Northern Territory assigned it some value, we went to New South Wales and assigned it some value went to Tasmania and assigned it some value and then came back to South Australia and said, no value possible that would have been crazy. We will be doing so much search in this sub tree and finally, we will come back and say no solution possible at all.

We have been able to detect this ZASAP. ZASAP? So we will come back to that, but at least we have been able to detect it in 1, 2, 3 steps. Now, I claim that you can detect it in 2 steps in fact, even this state is not possible in terms of finding a solution. Do you guess see this is very important. So if you do not see this you know, we will wait so that everybody can see this. How many of you think that there will be a solution or there can be a solution from this state?

How many of you think there will be no solution from the state. I see some people who are sleeping that is all right but people are awake should have raised their hands anyway everybody feels that there is no solution here. Why is there no solution here?

(Refer Slide Time: 04:20)

Constraint propagation

Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

NT and *SA* cannot both be blue!

Constraint propagation repeatedly enforces constraints locally

NPTEL

Chapter 5

There is no solution here, because at this point, both Northern Territory and South Australia have blue and they both cannot be blue. However does my forward checking procedure do this? No. Actually intuitively, if you think about it this constraint between Northern Territory and South Australia is between 2 assigned variables and that is actually an important observation. See up until now in for checking us saying I assigned something.

And then they propagate this information to all the unassigned variables and remove their values that are no longer possible. This was propagating the constraint from assigned to unassigned. But now we are saying that we want to even improve this algorithm and we want to get to a place where constraints between 2 unassigned variables can be taken care of. We can figure out that look this is not possible.

Because they both have the values remaining which are no longer only one value are remaining, which is no longer possible together. In fact, there is an algorithm which is much more interesting, which allows us to propagate it propagate constraint between unassigned variables very, very well and that algorithm is called arc consistency. So this is exactly what this slide says that NT and SA cannot both be blue.

But forward checking could not figure this out you need to somehow propagate constraints between unassigned variables and the algorithm that leads us there is called the arc consistency algorithm and this is the main thing that I would like you to learn today.

(Refer Slide Time: 06:21)

Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff
for every value x of X there is some allowed y

If X loses a value, neighbors of X need to be rechecked

Arc consistency detects failure earlier than forward checking

Can be run as a preprocessor or after each assignment

NPTEL

Chapter 5

So let us say and so I will tell you how to do this, it will be hard to come up with this in the next minute or 2 it is like I cannot ask you a question but I hope you learn this very well. So it is not very hard. We say that an arc X to Y is consistent if and only if, for every value x of X , there is some value y for Y . An arc X to Y is consistent if for every value of X , there is a value of Y if for some value of X , there is no value of Y , what can you say?

Somebody raise their hand. If for some value of X there is no value of Y , what does that tell me? that value of X is not possible and if that value of X is not possible, we remove it very simple and now that some value of X has been removed, which all arcs might see some more removes other variables Z to X , Y because for every value of Z there should be some value of X , but now I have reduced 1 value of X .

If I have reduced a value of X it is possible that some value of Z which was dependent on this value of X is no longer consistent and we can remove those values of Z and so on and so for that this is an inference procedure. Whenever we remove a value we have proven that this value of X cannot exist. We are not searching, we are not trying it out and backtracking searching is more like, I have many options.

Let me try this option it did not work. Let me try that option or did not work inferences I have these options this particular option I can prove will not work. Therefore, I will go there an arc consistency is the approach where I enforce these constraints between unassigned variables. Let us take an example, let us say we are at a point where Western Australia is red

Queensland is green and we want to somehow propagate this to say that they will have this is not going to lead to a solution.

Let us try to just enforce some marks. So let us first Australia to New South Wales, for every value of South Australia, do we have a value for New South Wales? Yes or no? Yes please, answer because then I would know you get it. So for every value of South Australia do we have a value for New South Wales? Yes. Can we remove any value of South Australia? No. This is consistent.

But now let us look in the other direction for every value of New South Wales. Do we have a value for South Australia? No. For value New South Wales blue we do not have a value for South Australia so we can remove, New South Wales being blue we have proven the New South Wales cannot be blue. This keeps going. Let us say we check Victoria to New South Wales, now for every value of Victoria, do we have a value for New South Wales?

For Victoria green do we have a value for New South Wales? Yes, for Victoria blue do we have a value for New South Wales? For Victoria red, do we have a value for New South Wales? No. So Victoria cannot be red and this keeps going so basically at some point we will check the arc South Australia the Northern Territory we will realize that for every value of South Australia blue. We do not have a value for Northern Territory.

We remove South Australia blue as soon as we remove South Australia blue, South Australia has no value as possible we have detected failure, we will backtrack. So in other words arc consistency see detects failures much earlier than forward checking. Forward checking is only enforcing constraints from assigned to unassigned are consistencies enforcing constraints between an asset yes there was a question no. So neighbouring in what?

No not let us let ask the question and not the map colouring vocabulary. So non neighbouring in which graph in the constraint graph, we talked about constraint graphs last time. So, will we also employ this in non neighbours in the constraint graph, we have already said that all constraints are binary because we can convert any general CSP into binary CSP. Therefore all constraints are between 2 variables. Whenever there is a constraint between 2 variables.

We are added an edge between the constraint graphs between them in the constraint graph and so they are neighbours and so on. While there will be constraints between non neighbours, for now we will not worry about it. We will simply check them locally between neighbours between the 2 variables which are linked to each other by a constraint and then keep enforcing it. See, as soon as 1 value goes down, then its neighbour becomes active and its neighbour becomes active. So, the constraints will propagate.

(Refer Slide Time: 12:23)


Arc consistency algorithm

```

function AC-3(csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables  $\{X_1, X_2, \dots, X_n\}$ 
  local variables: queue, a queue of arcs, initially all the arcs in csp
  while queue is not empty do
     $(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\text{queue})$ 
    if REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) then
      for each  $X_k$  in NEIGHBORS[ $X_j$ ] do
        add  $(X_k, X_i)$  to queue

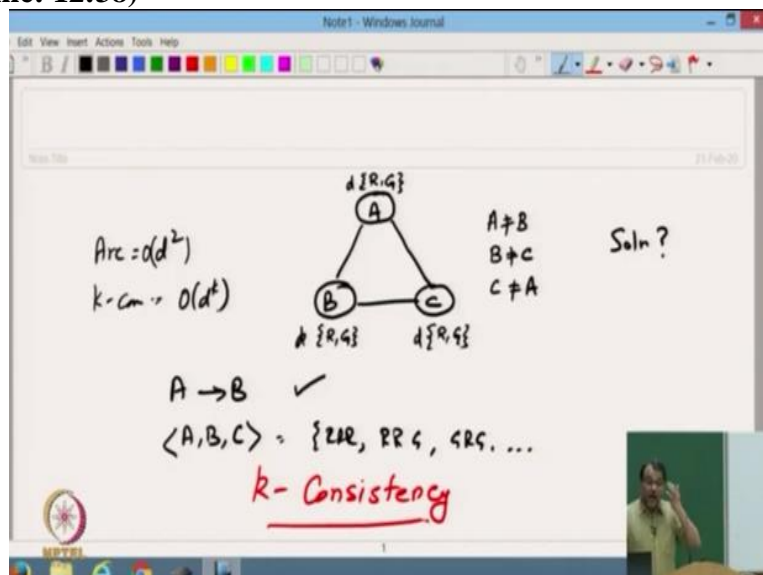
function REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) returns true iff succeeds
  removed  $\leftarrow$  false
  for each  $x$  in DOMAIN[ $X_i$ ] do
    if no value  $y$  in DOMAIN[ $X_j$ ] allows  $(x, y)$  to satisfy the constraint  $X_i \leftrightarrow X_j$ 
      then delete  $x$  from DOMAIN[ $X_i$ ]; removed  $\leftarrow$  true
  return removed
  
```

$O(n^3)$, can be reduced to $O(n^2d^2)$ (but detecting all is NP-hard)



One question you should ask, is will this solve all problems? Can you come up with a problem where arc consistency is not sufficient?

(Refer Slide Time: 12:38)



Somebody so let us look at this very simple problem I have 3 variables A, B and C. I have inequality constraints and I give everybody their red, green. Now, is there a solution to this problem? Is there a solution to this problem? No. Does arc consistency figure it out? So, let

us check the arc A to B for every value of A, for every value of B? Yes so no problem. Similarly, you can check D to A, A to C, B to C everything.

You will never get a problem so arc consistency see will not get rid of all possibilities will not detect all failures. In fact if I want to detect failure here, what do I need to do? Now, you can always do search but inference wise what do I need to do? So I need to see that while A to B, B to C, C to A independently possible A, B, C together there is no assignment, whatsoever.

So, you can think of AB as tiring red green, red blue, red green, green red, red red, green green possibilities and seeing that some possibility exists. Similarly in A B C, I can try you know that red red red, I can try red red green, I can try green red green and so on so for that and I keep trying and I realized no possibility exists and if no possibility exists, then I need to detect I need to backtrack.

So, the extension of arc consistency which is pairwise consistency to general sets of k is called S it is called k consistency I just said k but of course, each arc takes how much time to check this has this is domain D, this is domain D. So each other will take the time D square each arc will take the time D square, whereas each k consistency will check subsets of size k, so each k consistency operation will require time D to the power of k. So now this is very interesting now of course, K is a constant.

(Refer Slide Time: 15:24)

Arc consistency algorithm

```

function AC-3(csp) returns the CSP, possibly with reduced domains
inputs: csp, a binary CSP with variables  $\{X_1, X_2, \dots, X_n\}$ 
local variables: queue, a queue of arcs, initially all the arcs in csp
while queue is not empty do
   $(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\text{queue})$ 
  if REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) then
    for each  $X_k$  in NEIGHBORS[ $X_i$ ] do
      add  $(X_k, X_i)$  to queue


```


```

function REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) returns true iff succeeds
removed ← false
for each  $x$  in DOMAIN[ $X_i$ ] do
  if no value  $y$  in DOMAIN[ $X_j$ ] allows  $(x, y)$  to satisfy the constraint  $X_i \leftrightarrow X_j$ 
  then delete  $x$  from DOMAIN[ $X_i$ ]; removed ← true
return removed

```

$O(n^3)$, can be reduced to $O(n^2d^2)$ (but detecting all is NP-hard)

 Chapter 5



But as k increases you are doing more inference and you are doing more constraint propagation. So essentially what did we learn in today we talked about arc consistency we said that, you know this particular algorithm removes inconsistent values between pairs of variables. So if I have 2 check the arc $X_i X_j$ for each value of X in domain X_i I will check value of Y .

And if there is no value in Y then I will delete X_i from the domain X_i and if I delete X_i from domain X_i then I will add all neighbours have $X_i X_k$ to the queue and I will check the $X_k X_i$ this particular algorithm is called the AC-3 algorithm, arc consistency version 3 algorithm and it takes time, let us think about how much time it takes. So each arc consistency algorithm takes D^2 each arc consistency may be invoked D times Y .

Because X to Y when will X to Y be involved when some value of Y goes away then I will say check X to Y . So, each and how many values of Y we have D , so D times I may want to check a particular arc and how many arcs, do I have I have n^2 arcs. So overall I have n^2 into D^2 as my time complexity again just to remind you, I have n^2 arcs, each arc will be checked D times and each checking takes D^2 time.

So, therefore, it will be $n^2 D^3$, it can be reduced to $n^2 D^2$ if I do more interesting book keeping. So if I remember the support for each where the value of a variable, how many values in X , we are supporting this value, and if the value goes down from a certain below a point, then I invoke that arc and I even remove the member which values was supporting etc. Such as if I do a lot of book keeping.

Then I can reduce it to $n^2 D^2$, which is the best we can do and that algorithm is called AC. Guess 4 now, these are engineers, so scientists are developing algorithms. And then there is AC-2 also which is even weak. So, we discussed the consistency algorithm, we also discussed that it will not figure out all failures, because detecting all failures is actually NP hard. Why? Because if we can detect all failures.

Then I can sort of solve the CSP problem, and CSP problem is exponential problem in practice. It is an NP hard problem, otherwise, you will not be studying it in the engineer. Now let us think about what is the final algorithm that we came up with, like the backtracking search algorithm. The backtracking search algorithm says I am at a node, run arc consistency,

make sure some values which are not possible are removed, then I cannot remove any other value.

If I cannot remove any other value, I do some search, so I take the best of that variable, assign the best value and then run arc consistency algorithm and then if I still have some, if there is no failure, and if we have not found a solution, then I keep doing this keep doing this and at some point, either I detect failure or I have so few values remaining, maybe 1 value remaining for each variable that I keep assigning it and I finally get to my solution this is a combination of search and inference.

We are doing search, we are trying out ideas, but we are doing it intelligently we are saying that every point I try out an idea I back I propagate all information, I remove any possible values that are no longer possible and then I do intelligent search. So then I do search next. So I have divided by work I have said when I do not know what to do I will do search, but when I can do some moving I will do some moving and I will keep going back and for that.

And in fact, you can even have an algorithm which is an inferential algorithm, which is which does not do any search whatsoever. That when it chooses a value, you know there is a solution there you do not need to backtrack at all. But that algorithm would be exponential, that we will be doing K consistency for $k = n$ sort of a thing. So now you are in this very interesting place where if you do search, you can solve the problem.

If you do inference, you can solve the problem, if you do search, it is very, very slow because you are uninformed. If you do inference, it is very, very slow. It is exponential. Both of them are exponential. What do you do? You combine the 2. So you do a little bit of search, and then you do a little bit of inference, then you do a little bit of search, and then you do a little bit of inference and it was found that this would lead to the best solution in general.

We have done one other version of this in the past, get somebody make the connection to what we have done so far, which is related to this. What is your name? Ash sir? Calculating the admissible heuristic, earlier we talked about we may not have any heuristic function whatsoever. We will have to do a uniform cost search. We can have that will be equal to $x = 0$ heuristic.

We may have a perfect heuristic, but that will take a long time to compute, we may want to find the best place where our heuristic is polynomial computable. So that it gives you enough information so that the search moves faster and there was the trade-off too expensive or too inexpensive heuristics are was in the middle is the balance in the very same way. You can do forward checking, that is not enough you can do k consistency for $k = 4$, $k = 5$ that will be too slow.

Because now we are doing D to the power of 4, D to the power 5. In practice guess what people do most of the time $k = ?$ This is this is practical common sense. You may or may not have that yet us just in a 30 years students, $k = 2$ and for some problems $k = 3$ is what people in practice code. People are with quadratic algorithms for these problems in some of their cases, they are with cubic.

But they almost never go beyond $k = 3$, everything else is for theory, in practice and you will see this again and again there is a topic we do not do called you know classical planning. In classical planning, there is again this constraint propagation that happens with a mutual exclusion, and you can do mutual exclusion with 2 variables, you can do mutual exclusion between 3 variables, you can do mutual exclusion between 4 variables, but what mutual exclusion do they do?

It is almost become like a common sense rule now, for me that if you have an algorithm which can operate at different levels operated at the edge level, not at the triangular clique of size 3 levels, not at bigger sizes, not lower, do edge this sort of shows up again and again. Again, I do not know why I do not have a theory for this, but in practice quadratic. In the context of a bigger algorithm, which is expensive works more often.

Than not the best wishes you can predict the possible optimal value of n have a node possible optimal value of k , but I already told you optimal value. So, you should try some more interesting CSP and see when this will require you to go read up some literature, but I am sharing this based on you know, my general experience of problems and not everybody only solves map colouring problems.

There are CSP competitions whether those competitions have benchmarks those variable CSP problems may be of the 1000, 10s of 1000, 100,000 variables, very complicated problems.

But still people only use either consistency or 3 consistency in practice. Now I am not a CSP expert to give you the very accurate answer of can we look at the graph and figure out the optimal value of k .

But my guess is that my advice to you have $k = 2$ or 3 will hold 100% of the time if I mean 99 plus percent of the time is not 100% this is my APR, yes. So, we have 2 more things to discuss in this lecture. So we will discuss in the next class and then we will move on to logic which is a very important and very traditional field of a and that will be the last traditional topic it will study in effect stop here.