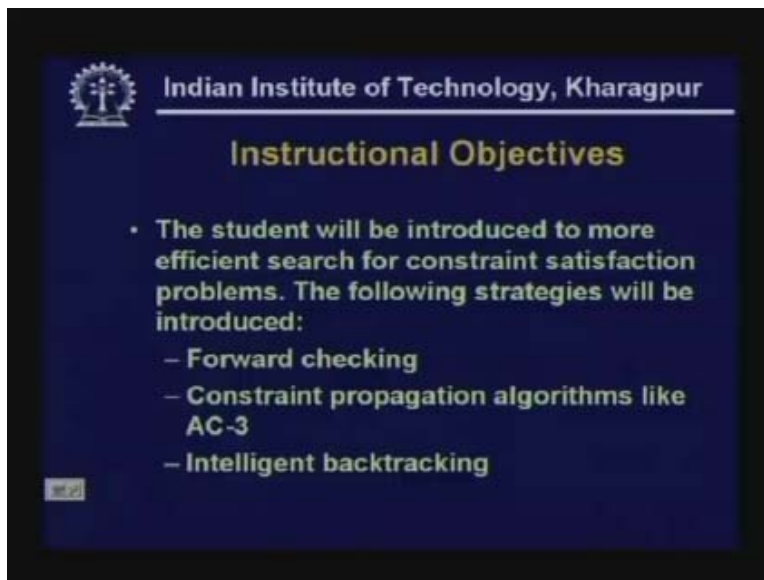


Artificial Intelligence
Prof. Sudeshna Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture - 10
Constraint Satisfaction Problems - 2

Today we start with the lecture ten of the course Artificial Intelligence. In the last class we introduced what you mean by constraint satisfaction problems and we looked at how we cast such problems as search problems and solve them by depth first search with backtracking. Today we will explore some more efficient techniques for solving constraint satisfaction problems.

(Refer Slide Time: 01:25)



The instructional objectives of today's class are as follows:

In this class the student will be introduced to more efficient search for constraint satisfaction problem. We will talk about the following strategies:

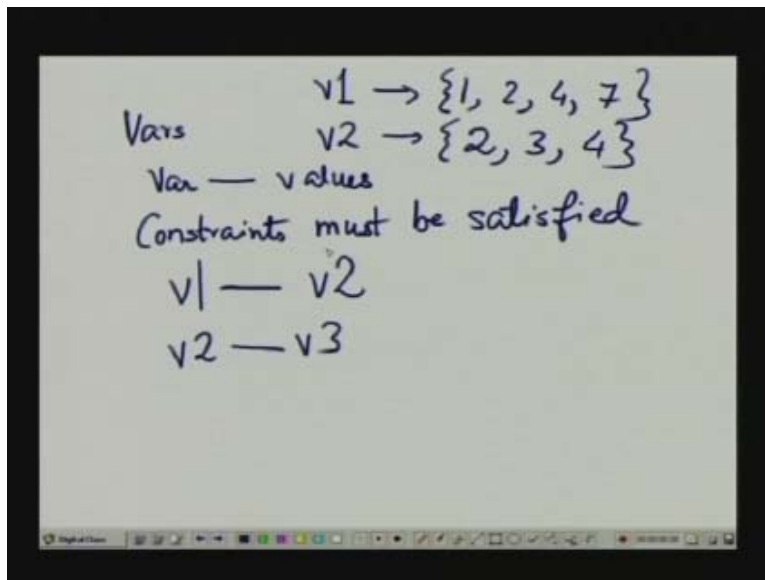
Forward checking and then we will look at constraint propagation algorithms like the AC-3 algorithm. Then we will briefly talk about intelligent backtracking or backjumping. On taking these topics the student should be able to apply these techniques to constraint satisfaction problems.

Before we start let us recapitulate the formulation we did for constraint satisfaction problems. If you remember, the sort of constraint satisfaction problems we had a number of variables and each variable has a domain. So v_1 is a variable and v_1 has a domain it can take a set of values. v_2 is another variable it can take some set of values. So we have different variables along with their domains. Our objective is to assign variables to values

and the value for a particular variable must come from its domain and the constraints must be satisfied.

We especially looked at a most common type of constraint called binary constraints. Binary constraints involve a pair of variables. Suppose we can have a constraint saying that v_1 and v_2 cannot have the same value this is one type of constraint. We can say that the constraint between the variables v_2 and v_3 is that the value of v_3 must be greater than the value of v_2 . So we have a set of variables, each variable has a domain. The variables must be assigned values from the domains such that the constraints are satisfied.

(Refer Slide Time: 04:09)



This is a general class of constraint satisfaction problems. These constraint satisfaction problems can be cast as state space search problems and we can do the search in the following manner: We pick up a variable, so this is the start, when none of the variables are assigned any values so v_1 is not assigned a value, v_2 is not assigned a value, v_3 is not assigned a value and so on. So this is the empty state where none of the variables are assigned any values. In the next step we look at the assignment of different values to variable v_1 . Suppose variable v_1 can take the values 1, 2, 4, 7 we have 4 branches corresponding to all the possible values that v_1 can take.

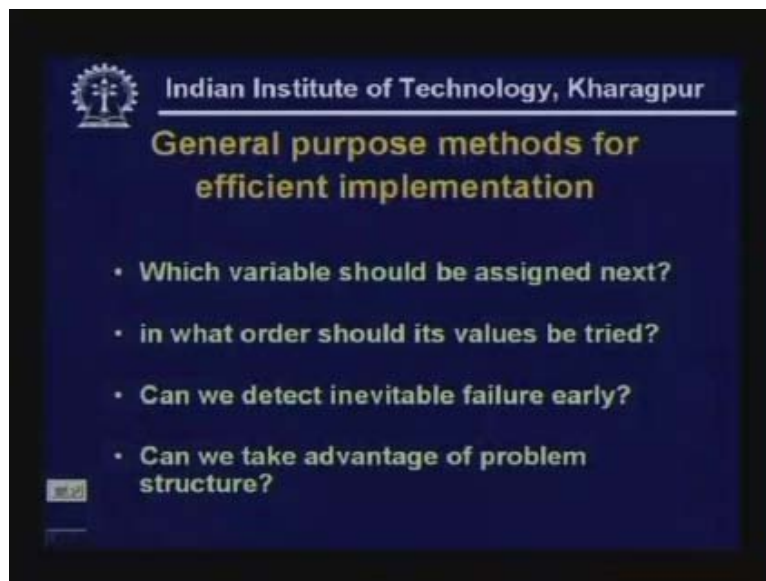
In the next state we have to pick up another variable. Let us say we pick up v_2 and we have to look at all possible assignments of values to v_2 that are consistent with the assignment that we have so far. For example, if v_1 has the domain 1, 2, 4, 7 and v_2 has the domain 2, 3, 4 then if v_1 is already one v_2 can be either 2 or 3 or 4. If v_1 is 2 then v_2 can be either 3 or 4. We cannot have v_2 is equal to 2 because any further assignment after we have v_1 is equal to 2, v_2 is equal to 2 will not undo this constraint. So if we have an assignment which is inconsistent we should not proceed further in that direction.

Next, suppose we have another variable v_3 which has a domain 1 and 5 and we have the constraint that the value of v_3 must be greater than the value of v_2 then in this case v_3 can only take the value of 5, in this case v_3 again can only take the value of 5 and so on. So, in this way we can construct the search tree corresponding to the constraint satisfaction problem and we note that a solution to the constraint satisfaction problem is a leaf in the search tree which corresponds to all variables being assigned to specific values that do not violate any constraints.

Therefore, if there are n variables all the leaves at level n are the different solutions to the search problem. And the strategy which is appropriate for solving such search problems is depth first search. Whenever we find that a constraint is violated we do backtracking. This backtracking is called chronological backtracking because we backtrack to the previous choice point. So, depth first search with chronological backtracking is a very appropriate method for solving constraint satisfaction search problems.

We also discussed that we can try to make this search efficient by looking at a proper order in which we choose the next variable that we select next for assignment. So if you take a proper order of the variables we might end up in reaching the first solution much faster. And then once we pick a variable to assign values too we will be considering in what order should we assign values to these variables.

(Refer Slide Time: 08:53)

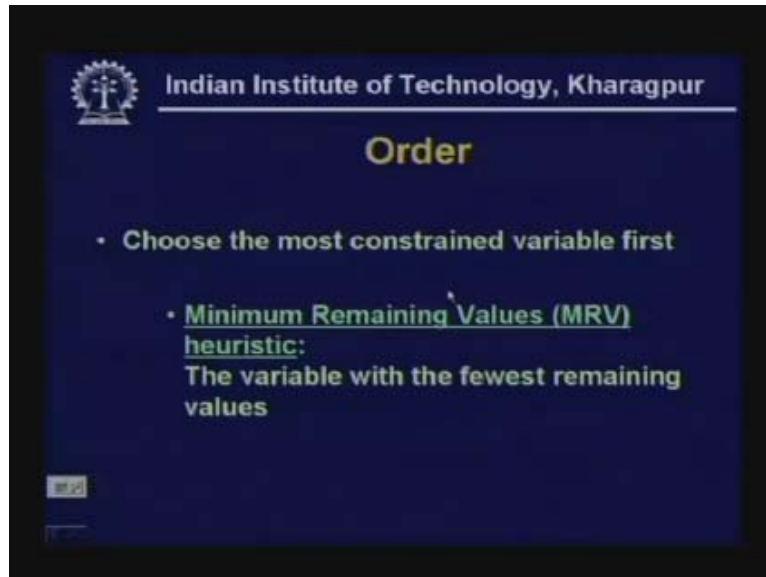


Then we will look at other questions like, can we detect inevitable failure earlier?

So, in the general search problem we said that whenever we see that a constraint is violated we terminate the search at that point and then backtrack to the previous choice point. But it may be the case that future problems can be detected even earlier on. Let us see some techniques to handle that and then there is a question whether we can take advantage of the problem structure to further prune the search specific to a particular

problem. One heuristic to choose the next variable is the minimum remaining value heuristic. In the minimum remaining value heuristic the variable with the fewest remaining values is chosen next.

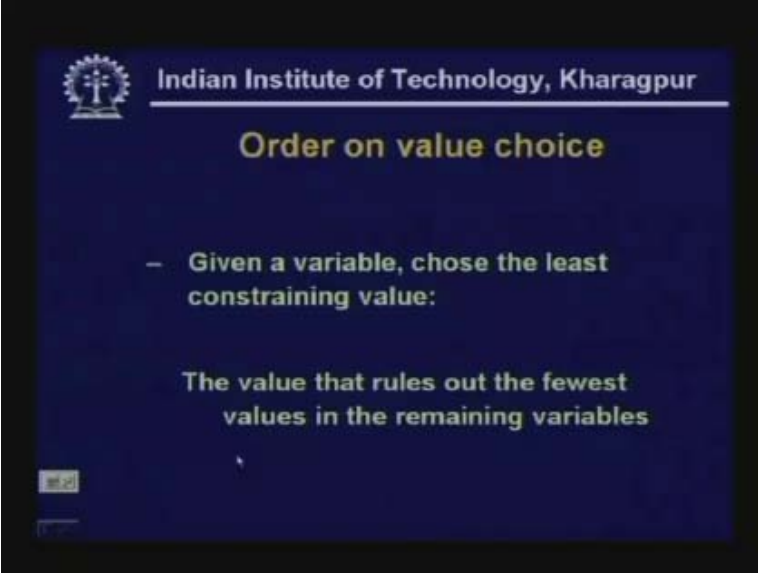
(Refer Slide Time: 09:51)



What do you mean by variable with the fewest remaining values?

Every variable has a domain. So when we have a partial search tree some of the variables have been already assigned values and as a result of assigning values to v_2 and v_3 the domain of v_1 has become restricted, the domain of v_4 has become restricted, domain of v_5 has become restricted. So we find out the size of the domains of the remaining variables and we pick the variable for which the remaining domain is smallest in size. The intuition behind this is, since in the constraint satisfaction solutions or constraint satisfaction problems all variables have to be assigned values, the variable which is most constrained will have the fewest remaining values. So we need less of backtracking in order to consider the effect of that on the search. Once you have chosen a variable we have to choose the order of value assignment.

(Refer Slide Time: 11:30)



Indian Institute of Technology, Kharagpur

Order on value choice

- Given a variable, chose the least constraining value:
The value that rules out the fewest values in the remaining variables

And here the heuristic is, choose the least constraining value. All variables have to be assigned value and the solution but each variable has to be assigned only one value. So our intuition is to choose a value which is most likely to take you to a solution. And consider that value which constraints the domains of the remaining variables the least. That is the value that rules out the fewest values in the remaining variables. Apart from the depth first search formulation we are considering we can propagate constraints earlier on.

(Refer Slide Time: 12:25)



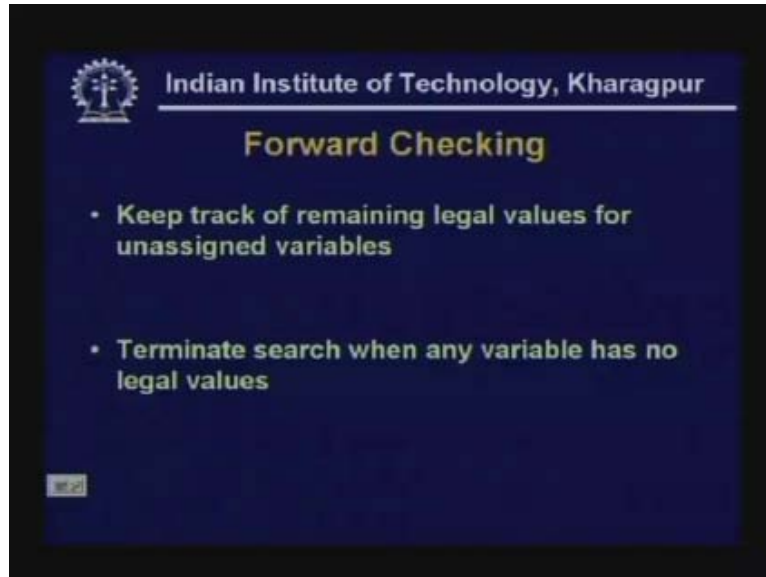
Indian Institute of Technology, Kharagpur

Propagating constraints

- Instead of considering variables one at a time, look at the constraints earlier on to reduce the search space.
- Simplest idea: forward checking

Instead of considering variables one at a time we look at the constraints early on to reduce the search space. And there are several ideas which people have formulated which gave rise to different algorithms for constraint satisfaction search and the simplest of these ideas is forward checking.

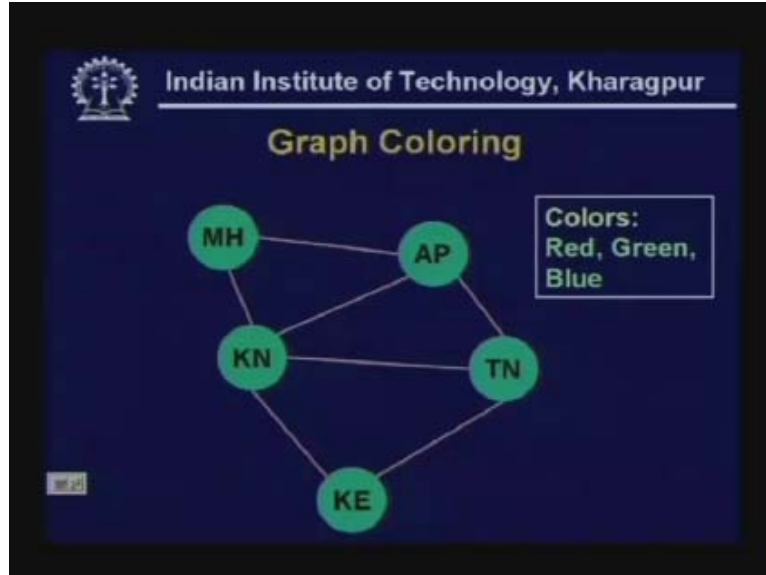
(Refer Slide Time: 12:50)



What is forward checking?

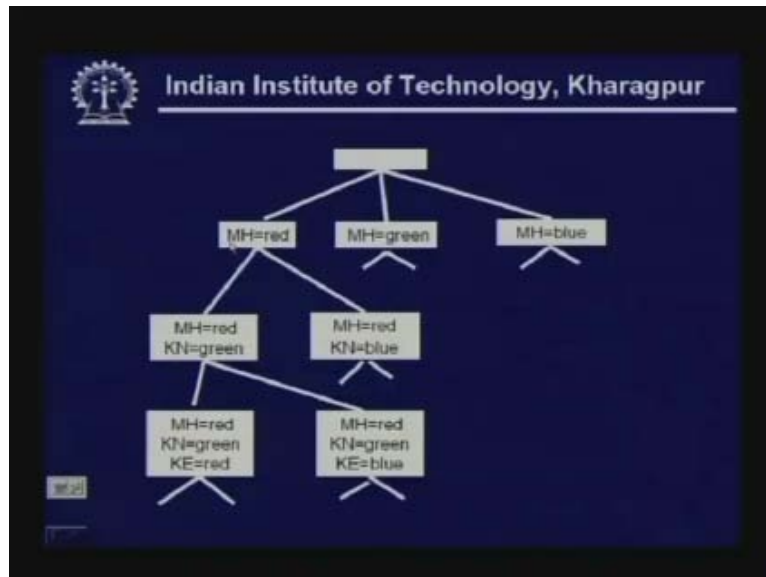
We keep track of the remaining legal values for the unassigned variables. So, when we assign values to a variable we suitably constrain the domains of the remaining variables. And then if we detect that there is a variable which does not have any legal values left then we can immediately terminate that path because finally we have to assign values to every variable along a correct solution path. So, if you find that there is a variable which does not have any legal values left we will not be able to assign values to the variables so that search path is bound to fail. If we can detect this early on we can terminate search at that point of the search tree. This is an example of a graph coloring problem where we have five nodes and there are three colors red green and blue and these are the links in the graph.

(Refer Slide Time: 13:57)



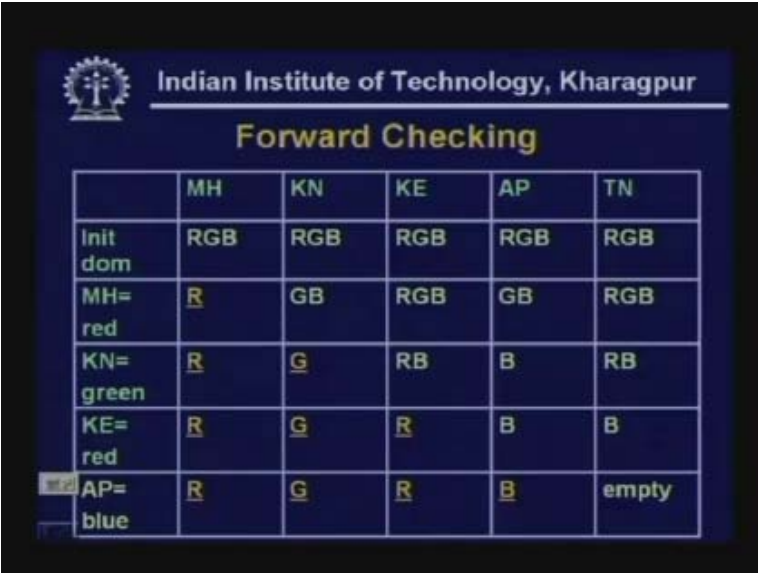
Corresponding to this problem suppose we show how a constraint satisfaction search proceeds.

(Refer Slide Time: 14:11)



Initially suppose we pick the node MH we can paint is red green or blue we paint MH is equal to red then if we pick KN next KN is a neighbor of MH so KN can be either green or blue. If we pick KN is equal to green and then we pick KE next KE is a neighbor of both MH and KN so KE is a neighbor of KN so it has to have a different color than KN so it can be either red or blue. So this way we can construct the search tree and we can do a depth first search.

(Refer Slide Time: 14:56)



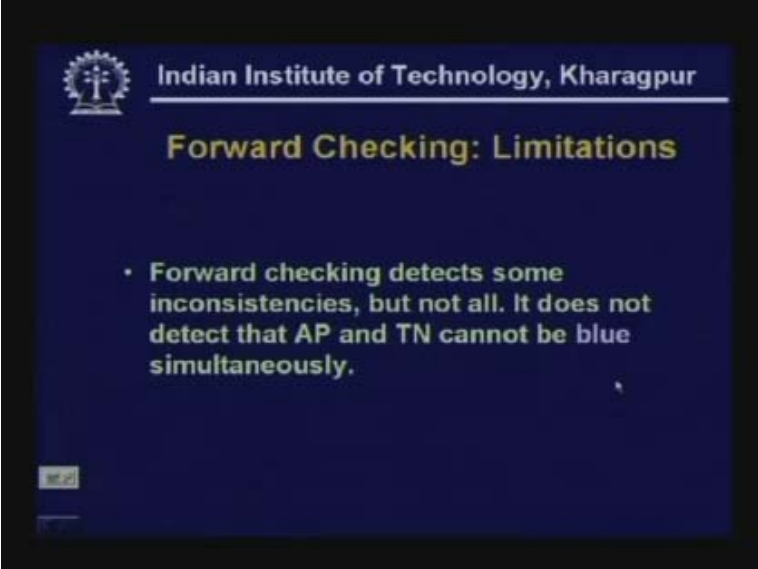
The slide features the IIT Kharagpur logo and title at the top. Below is a table with 6 rows and 6 columns. The columns are labeled MH, KN, KE, AP, and TN. The rows represent the domain of each variable. The table shows the following values:

	MH	KN	KE	AP	TN
Init dom	RGB	RGB	RGB	RGB	RGB
MH= red	R	GB	RGB	GB	RGB
KN= green	R	G	RB	B	RB
KE= red	R	G	R	B	B
AP= blue	R	G	R	B	empty

Now, if we do forward checking let us see how we will proceed. Initially we have five variables MH KN KE AP and TN. Each of these variables can take each of the three values red green and blue. When we set MH is equal to red KN and AP are neighbors of MH so they can be either G or B, the others can be RGB. If we pick now KN is equal to green then AP which is a neighbor cannot be green so it can only be blue, KE and TN can be either RB or RB they are also neighbors of KN. Then we pick KE is equal to red as a result AP and TN can be only blue. Now if you take AP blue we find that the domain of TN is empty because TN is a neighbor of AP. So we detect here that there is a problem with the assignment and we backtrack. This is what forward checking does.

Let us look back at the figure. At the 4th step we found that KE and TN have only blue left in their domains. Now KE has a legal value left, TN has a legal value left but we also know that there is a constraint between KE and TN. KE and TN cannot be of the same color. We are talking about AP and TN. They cannot be of the same color because they are neighbors. So in this step you see that AP and TN have only one legal value left. Even though they are individually legal together AP is equal to blue and TN is equal to blue cannot happen. So a smart algorithm might detect this early on and terminate search. This is a limitation of forward checking.

(Refer Slide Time: 17:04)



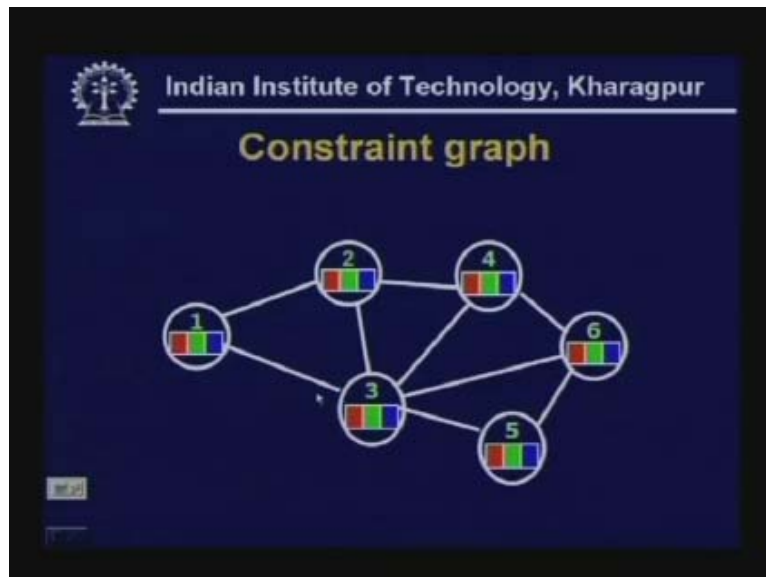
Indian Institute of Technology, Kharagpur

Forward Checking: Limitations

- Forward checking detects some inconsistencies, but not all. It does not detect that AP and TN cannot be blue simultaneously.

Forward checking checks that every remaining variable has at least one legal value but it does not explore further whether these values can be assigned to this variable subject to the constraints between these variables. Forward checking detects some of the inconsistencies but it does not detect all inconsistencies. For example, in the previous example it does not detect that AP and TN cannot be blue simultaneously.

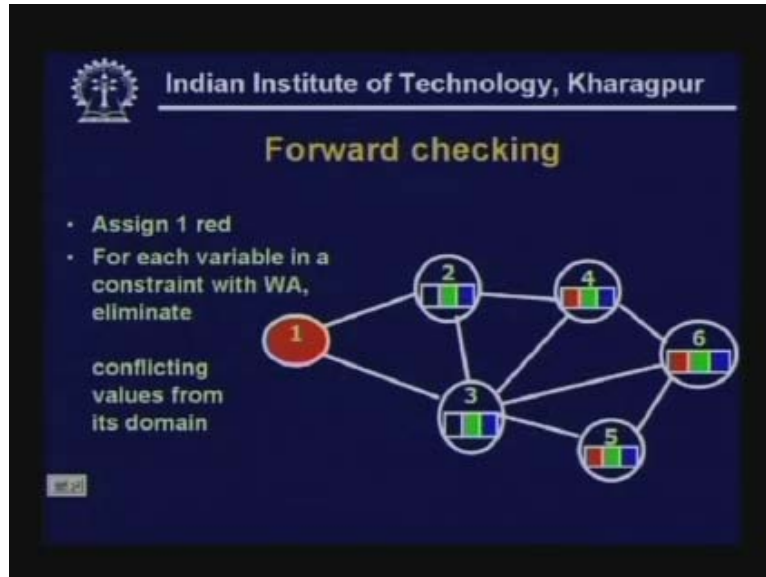
(Refer Slide Time: 17:45)



Now let us look at another constrained graph on which we will run the next algorithm. Again this is a graph coloring problem. We have 6 nodes which we have numbered 1, 2, 3, 4, 5 and 6 and this is the connectivity information between the nodes. Each of the

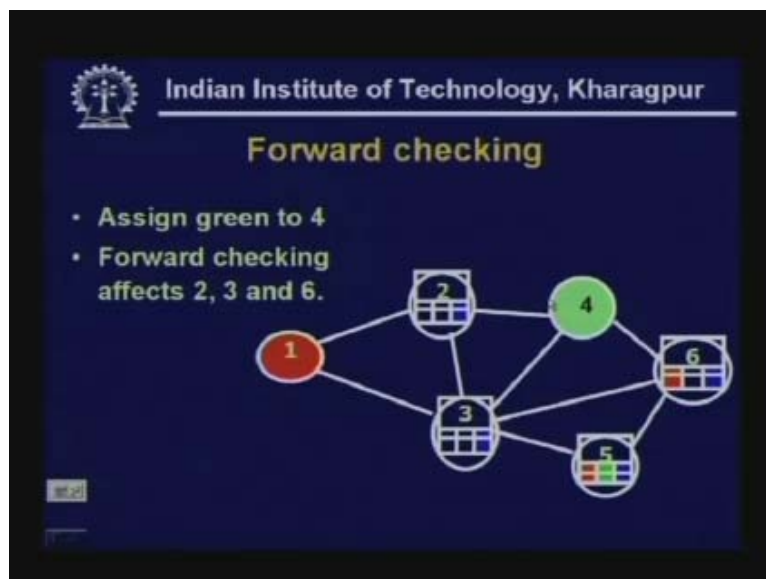
nodes can be colored red, green or blue but two neighboring nodes cannot be assigned the same color. Now, if we do forward checking let us see what happens.

(Refer Slide Time: 18:24)



Suppose we first pick up 1 and we assign 1 to red. Now, if I assign 1 to red 2 and 3 cannot be red so we reduce the domains of 2 and 3 we have blue or green. So, for each variable which is in constraint with 1 we eliminate the conflicting values from the domain. Next is, suppose we assign green to 4 then 2, 3 and 6 cannot be green so we remove green from the domains of 2, 3 and 6.

(Refer Slide Time: 19:03)



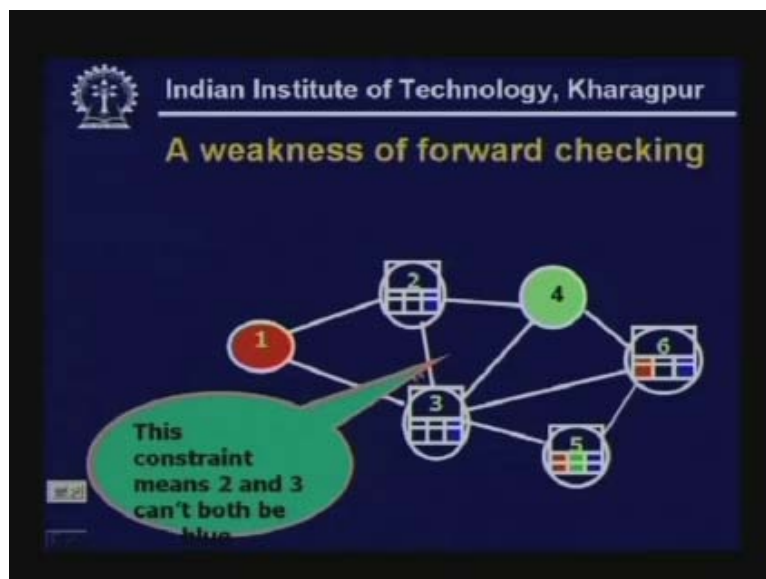
Now 2 can be only blue, 3 can be blue, 6 can be red or blue. Next we assign blue to 5 and as a result it affects the domains of 3 and 6. So we remove blue from the domains of 3 and the domains of 6.

(Refer Slide Time: 19:44)



Now as a result we see that the domain of 3 becomes empty. So this is impossible so we have to back track at this point. Let us now find out the weakness of forward checking. When we assigned for green we saw that 2 and 3 have only one legal value left which is blue but these two legal values cannot be assigned together so we should have detected this problem early on, forward checking does not detect this.

(Refer Slide Time: 20:29)



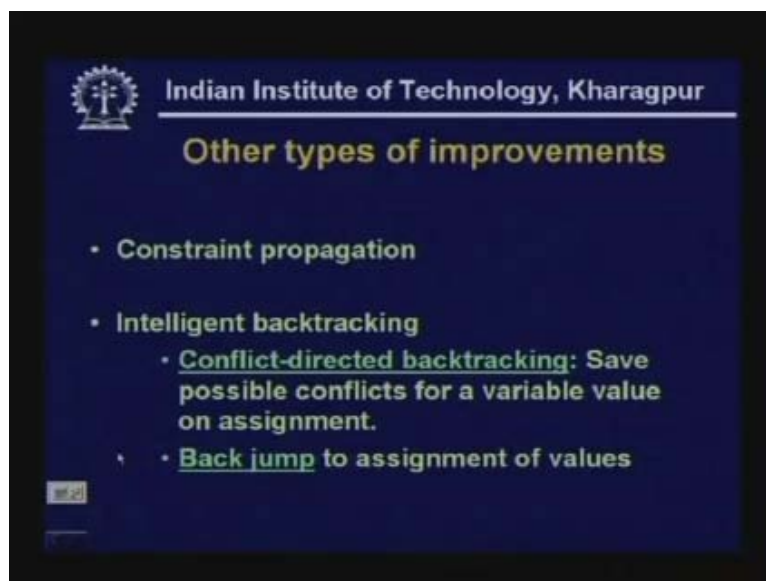
So this is not detected by forward checking. We should have backtracked at this step.

(Refer Slide Time: 20:37)



Now let us see what further improvements we can do. There are several things that we could do. One is we could try to propagate the implications of the constraint on one variable on to the other variables. We propagate not just values but constraints between these variables. **This is the idea of constraint propagation we will explore next.**

(Refer Slide Time: 21:12)



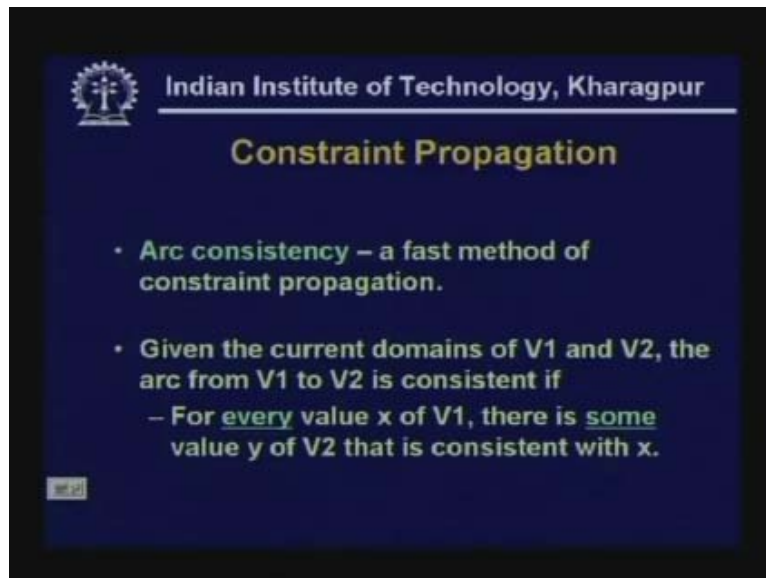
Secondly what we will see after that is, we can backtrack intelligently which we call conflict directed backtracking. So we will save possible conflicts for a variable value

when we assign that variable value and we will back jump to that assignment which gave rise to this conflict.

In chronological backtracking on normal depth first search we backtrack to just the previous choice of point.

In backjumping when we detect a constraint violation we backtrack to that assignment which led to the conflict which could have taken several decisions earlier. So first, let us explore constraint propagation.

(Refer Slide Time: 22:06)



Arc consistency is a fast method for constraint propagation. This is how arc consistency works:

Given the current domains of v_1 and v_2 the arc from v_1 to v_2 is consistent if, for every value x of v_1 there is some value y of v_2 that is consistent with x . In forward checking check if the domain of in the remaining legal the legal domain of v_1 is empty or not. Here we are not only checking whether the domain has a value but we are just checking whether for every value left in the domain there is some value y in the domain of another variable so that these two assignments of values are together consistent. This is what constraint propagation does.

There are several constraint propagation algorithms. We will look at AC-3 and some algorithms like that. Let us see an example of the same graph that we discussed earlier.

(Refer Slide Time: 23:24)

Indian Institute of Technology, Kharagpur

Arc consistency

- Assign green to 4, so check all arcs from 4
- Check 4→2 and eliminate green from 2
- Remember to check 2 later

We assign green to 4. After we assigned green to 4 we check all the arcs from 4, 2, 3 and 6. First we check 2. So when we check 4 is equal to 2 we eliminate green from the domain of 2. But we have to check back later with the variables with which 2 has some constraint.

(Refer Slide Time: 23:54)

Indian Institute of Technology, Kharagpur

Arc consistency

- Check other arcs from 4
- Check 4→3 and eliminate green from 3
- Remember to check 3 later

Next we check the constraint 4 to 3 and eliminate green from the domain of 3. After that we check the constraint 4 to 6 and eliminate green from the domain of 6. After we have finished checking the neighbors of 4 we go back to those variables whose domains have

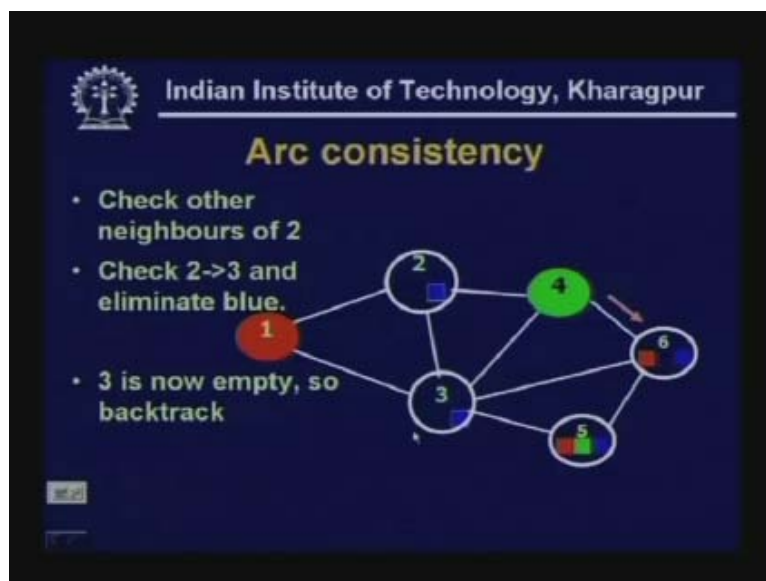
been changed to check if their current remaining values are consistent with the values of their neighbors.

(Refer Slide Time: 24:27)



We now go back to 2 and check the neighbors of 2. First we check 2 with 1 there is no violation, there is no problem. Then we check 2 with 3 and then we notice that if 2 is blue 3 cannot be blue so we eliminate blue from 3.

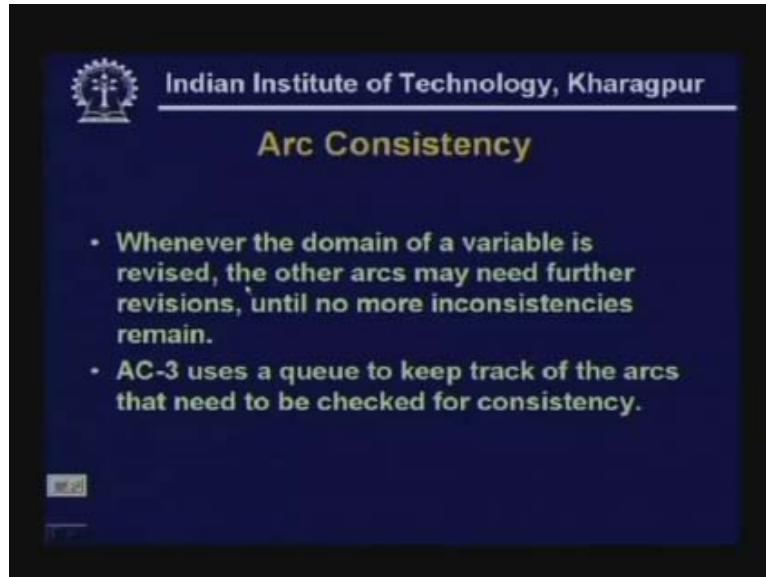
(Refer Slide Time: 24:45)



If we do that the domain of 3 becomes empty which cannot happen so we backtrack at this point. This is the arc consistency algorithm and we can backtrack when we detect

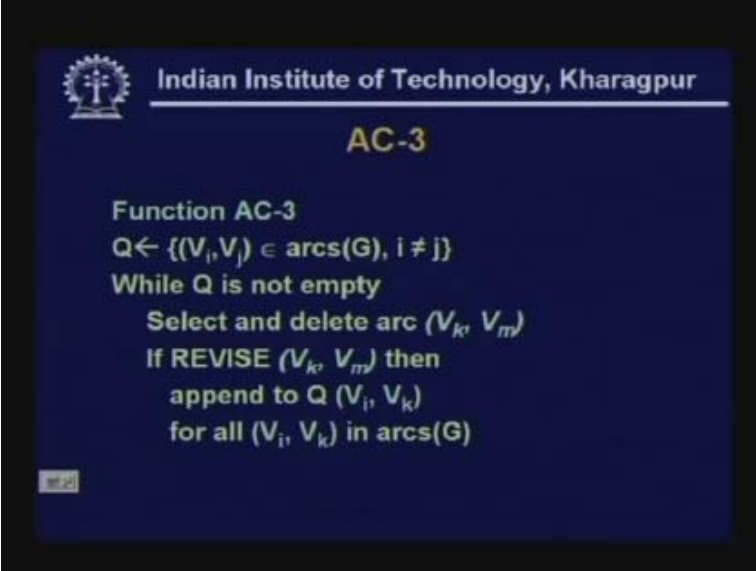
that there are variables which do not have domains which are consistent together because of the constraints between those variables. This is the idea of arc consistency.

(Refer Slide Time: 25:18)



Whenever the domain of a variable is revised the other arcs may need further revisions until no more inconsistencies remain. One implementation of the arc consistency algorithm is the AC-3 algorithm. The AC-3 algorithm uses a queue to keep track of the arcs that need to be checked for consistency because when we assign 4 to green it is not enough that we check the domains of 2, 3 and 6 which is neighbors of 4 but if these domains did change we have to check 2 with its neighbors and do this recursively until there is no further constraint propagation. So AC-3 does this constraint propagation so whenever the domain of a variable is reduced according to this algorithm you propagate the constraints to the neighbors of that variable. This is done recursively in the AC-3 algorithm. Now let us look at an implementation of the AC-3 algorithm.

(Refer Slide Time: 26:42)



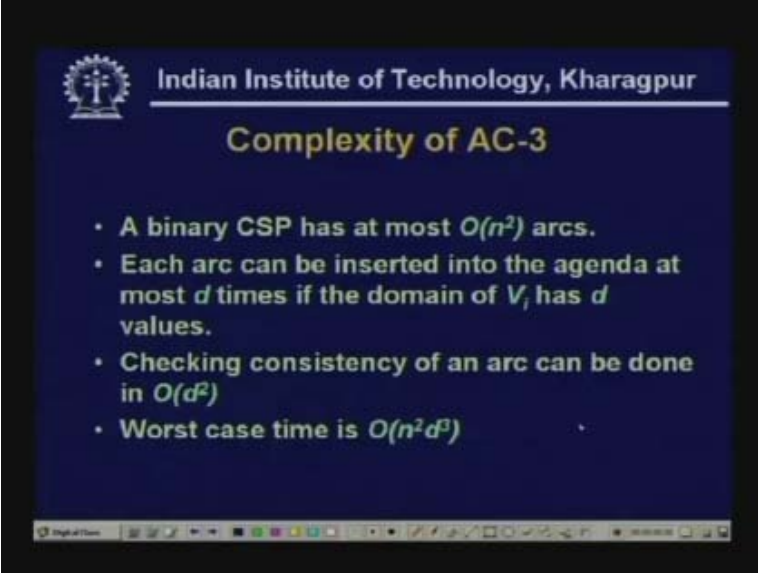
The slide features the IIT Kharagpur logo and name at the top. Below that, the title "AC-3" is centered. The main content is the pseudocode for the AC-3 function, which starts with a queue Q containing all arcs of the graph. It then enters a while loop that continues as long as Q is not empty. In each iteration, an arc is selected and removed from Q. The algorithm then checks if this arc's constraint is satisfied. If not, it revises the domains of the variables involved and adds all arcs incident to these variables back into the queue Q.

We have a data structure of a queue. Initially the queue contains all the arcs of the graph. So we have all the $V_i V_j$ that are the edges of the graph such that i_0 is equal to j . Then while q is not empty we select one of the arcs from this queue that is V_k, V_m and we delete V_k, V_m from queue. Then we check if $\text{REVISE } V_k, V_m$ is true. That is, when we delete this arc we see if the domains get reduced. When we consider this constraint between V_k, V_m we check if the domains of these variables get reduced. So as a result of assignment of V_m if the domain of V_k gets changed then we find all the arcs in the graph of the form V_i, V_k . That is, if V_i is a neighbor of V_k then we have to check those constraints. So we append to the queue all the arcs V_i, V_k . If as a result of an assignment to V_m the domain of V_k gets changed.

Now let us see what is the complexity of this algorithm AC-3?

Now, if we have a binary constraint satisfaction problem then the constraints are only between pair of variables. So there are utmost n square arcs in the constrained graph.

(Refer Slide Time: 28:58)



Indian Institute of Technology, Kharagpur

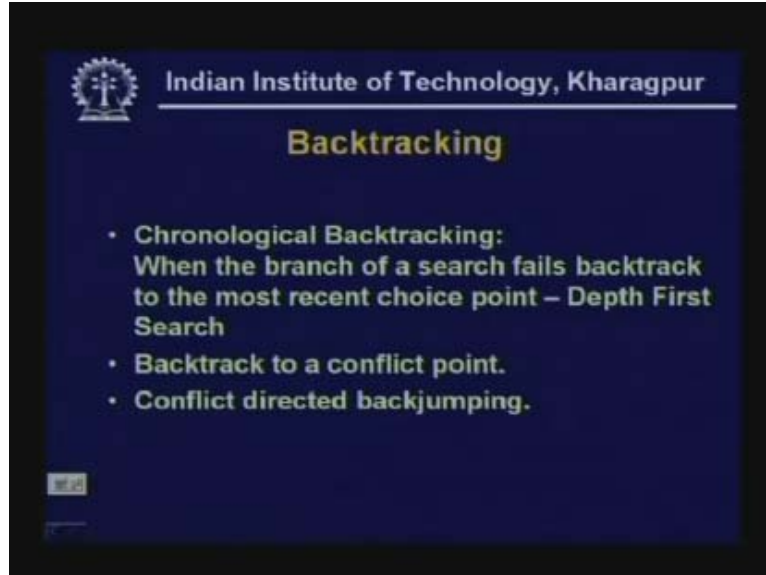
Complexity of AC-3

- A binary CSP has at most $O(n^2)$ arcs.
- Each arc can be inserted into the agenda at most d times if the domain of V_i has d values.
- Checking consistency of an arc can be done in $O(d^2)$
- Worst case time is $O(n^2 d^3)$

In the algorithm every arc can be inserted utmost d times if the domain of V_i has d values. So, when we are putting an arc involving a variable into the queue we are only doing it when the domain of the variable is reduced. Suppose initially the domain of a variable V_k has b values we consider this arc for revision only when this domain is getting reduced. Corresponding to every V_k the arc can be put into the queue utmost d times. Now, checking the consistency of an arc can be done in order d square times because we check it against all the arcs in the graph. So the worst case time complexity of this algorithm is n square times d times d square which is $O(n$ square d cube). So $O(n$ square d cube) is the worst case complexity of the AC-3 algorithm. This is what constraint propagation is about.

In constraint propagation as a result of reduction in the domain of a variable we look at the implications in terms of the constraints of the variables with other variables. And there are a set of algorithms which do constraint propagation. One of the well known algorithms is AC-3 which we discussed. Next we will consider another technique which we mentioned as intelligent backtracking.

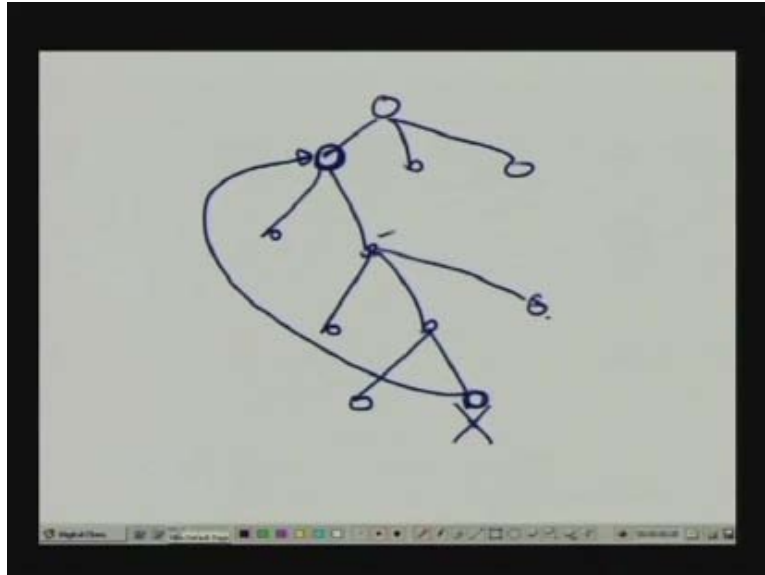
(Refer Slide Time: 31:22)




We have already seen backtracking which has a very important role in solving the constraint satisfaction problems. So what you have in plain depth first search is chronological backtracking. When the branch of a search fails we need to backtrack to the most recent choice point. This is called depth first search. In depth first search we backtrack to the previous choice point but intelligent backtracking involves backtracking to the point because of which the conflict has arisen.

Suppose we have a search tree like this and we have a constraint violation here, in chronological backtracking what we would do is, if there is a violation here we will go back here and then go back here which has a choice point. So we will go back to this node which is the next choice point. But in chronological backtracking if we find that this violation is because of an assignment here we will backtrack directly to this place. This is called backjumping or conflict directed backjumping.

(Refer Slide Time: 32:58)



(Refer Slide Time: 33:07)

 Indian Institute of Technology, Kharagpur

Chronological Backtracking

Consistency check performed in the order in which vars were instantiated

If c-check fails,
try next value of current var

If no more values,
backtrack to most recent var

In chronological backtracking consistency check is performed in the order in which the variables were instantiated. If consistency check fails we try the next value of the current variable. If the current variable has no more values we backtrack to the most recent variable that has a choice left. This is what is done in normal depth first search. Backjumping is also a type of backtracking but it is more efficient when there is no consistent instantiation to be found for the current variable.

(Refer Slide Time: 33:51)



Indian Institute of Technology, Kharagpur

Backjumping (BJ)

- Similar to BT, but
 - more efficient when no consistent instantiation can be found for the current var
- Instead of backtracking to most recent var...
 - BJ reverts to deepest var which was checked against the current var

And in this case we go to the deepest variable which was constraint checked against the current variable. So, when we find that the current variable is inconsistent and has no possible instantiation left instead of backtracking only to the previous choice point we find out what is the last variable which constrain the current variable. The current variable has no legal values left. In order that the current variable does get some legal value some constraint which participated with the current variable has to be changed. So we find the deepest node in the path from the current variable to the root where a variable was assigned a value and that constrains the domain of the current variable, we have to find that and that is what intelligent backtracking does.

Backjumping reverts to the deepest variable which was checked against the current variable in which introduce the constraints. Now let us look at this example. So here we have the 6 queens problem and we have put queen 1 in row 2, queen 2 in row 5, queen 3 in row 3 and queen 4 in row 6. After that we have put queen 5 in row 4. Now we see that, after we have placed these 4 queens the last queen cannot be placed in any of the rows. When we cannot put queen 5 we really do not need to look at any further assignment to the value of queen 5.

(Refer Slide Time: 36:22)

Indian Institute of Technology, Kharagpur

Backjumping (BJ)

BJ Discovers
(2, 5, 3, 6) inconsistent
with x_6
No sense trying other
values of x_5

Q
		Q

We have looked at some efficient constraint satisfaction problems. We have revised forward checking then we have looked at constraint propagation for which we looked at arc consistency algorithms and then we looked at backjumping.

What is the idea behind these algorithms?

In forward checking when we assign values to some of the variables we look forward and then we update the domains of the remaining variables. And in forward checking we just checked whether the domain of any of the remaining variables become empty.

(Refer Slide Time: 37:41)

→ FC
→ CP - Arc Consistency Algs
→ BJ

update domains

```
graph TD; A(( )) --- B[V1=1]; A --- C[V1=2];
```

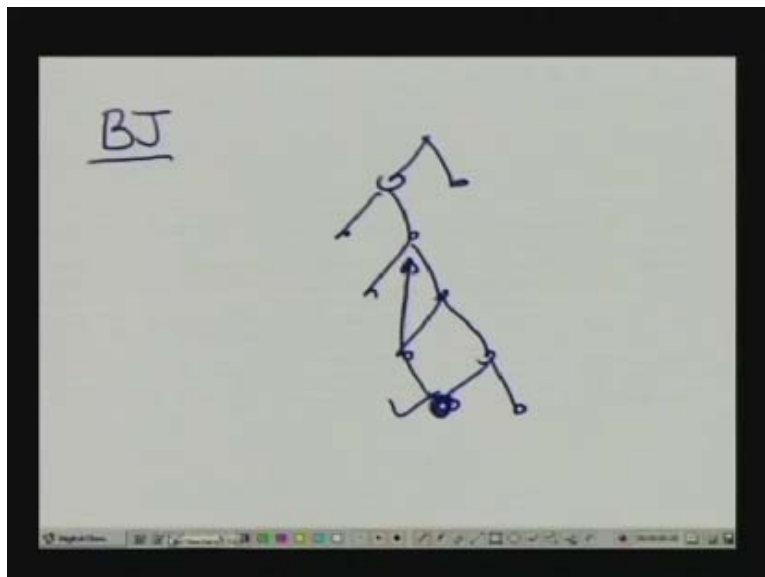

If they do become empty then we terminate search at that point. In constraint propagation, in the general constraint satisfaction class of problems what we do is that, we not only propagate the constraints to restrict the domains of the unassigned variables but we also do a consistency check on those variables which have constraints between them. That is we check whether every variable whose domain gets affected, whether it has a legal value for a legal value of its neighbor with which it has some constraints. AC-3 is a constraint propagation algorithm which does this recursively. Whenever it reduces a domain of a variable it takes up those variables and looks at its constraints with its neighbors.

If there are constraints with its neighbors which reduce the domains of the neighbors then again it checks those with their neighbors and so on. So AC-3 is quite an efficient algorithm and is able to detect constraints earlier. As you make your constraint checking algorithms very intelligent you are reducing your reducing the number of nodes in the constraint satisfaction tree that you are going to search but you are spending more time at every step. So you have to select a right trade off about whether these intelligent processing does give you benefit.

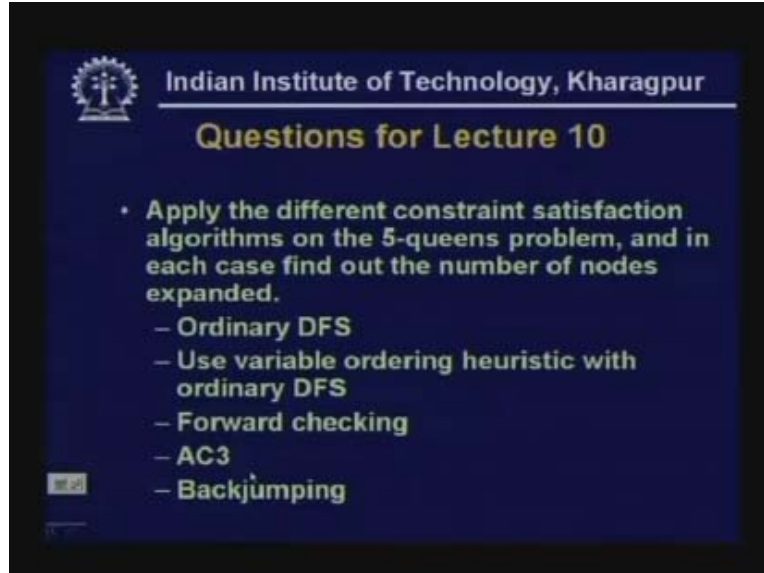
Does this take up time? Whether the benefit that you get in terms of reduced search space if they warrant that sort of benefit?

Finally we looked at backjumping. What backjumping does is that, when in your search tree some constraint or some conflict is discovered that is the domain of this variable has become empty. In backjumping we find the earliest variable which had been checked with this variable in which we introduced “c”. Now we must undo those effects in order to get out of this situation. So instead of going to the most recent choice point we jump to a conflict point.

(Refer Slide Time: 41:05)



(Refer Slide Time: 41:15)



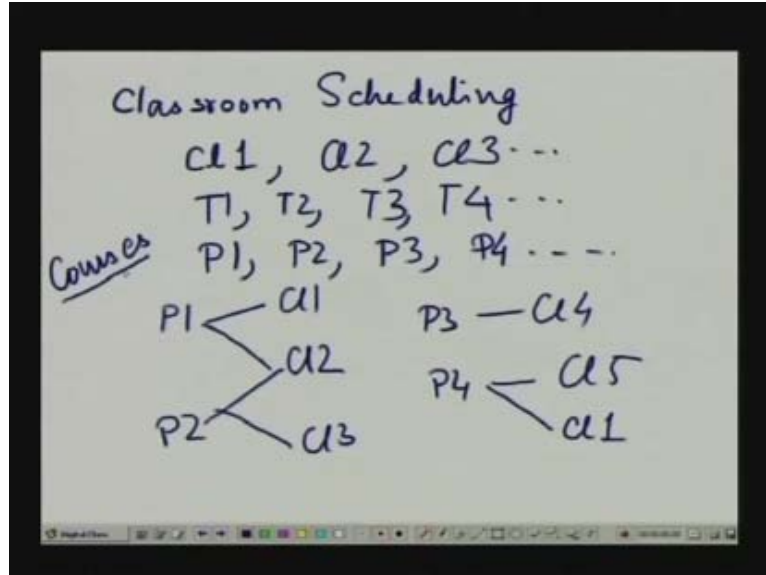
Now we look at the questions for this lecture:

The first question: Apply the different constraint satisfaction algorithms we discussed in today's class on the 5 queens problem. In each case you should find out the number of nodes that are expanded. So first you will try ordinary depth first search. Second; you will try variable ordering heuristic along with ordinary depth first search. In the third instance you try forward checking. The 4th instance you work out AC-3 and finally you work with backjumping.

Let us look back briefly at the questions we have set for lecture nine. If you remember the first question asked you to do the class room scheduling, to look at the class room scheduling problem and formulate it as a constraint satisfaction problem. So let us briefly look at the class room scheduling problem. We have a number of class rooms. Suppose $c11, c12, c13$ are the class rooms and we have some times slots 43:02

Let $T1, T2, T3, T4$ be the time slots and we have some teachers. So let $P1, P2, P3, P4$ etc be the teachers. Now for each teacher we have a set of classes which the teacher should be able to teach. So let us say $P1$ should be able to teach $c11$ and $c12$, $P2$ should be able to teach let us say $c12$ and $c13$, $P3$ should be able to teach $c14$, $P4$ should be able to teach $c15$ and $c11$. So we have a set of teachers and each teacher should be able to teach a set of classes. We have to assign courses so we have a set of courses.

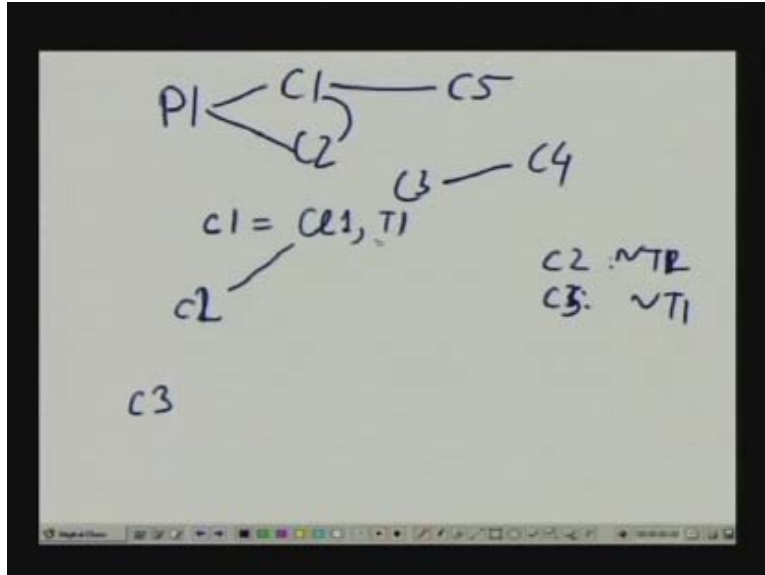
(Refer Slide Time: 44:16)



Now this is a problem which involves a large number of variables of different types. Now let us look at the basic types of constraints which are involved. Now, if a teacher has to teach both the courses namely course 1 and course 2 then it means that course 1 and course 2 must be in different time slots. If a teacher is teaching c1 and c5 it means c1 and c5 must be in different time slots. Now each class room at a slot can have only one class. This is something you have to follow. And if a teacher is teaching two classes those two classes cannot be of the same time slot. Now what we have to do is, we have to assign classes to class rooms and time slots.

So we can start with picking class c1 and assigning it to class room 1 time slot T1. Now, if c2 and c5 cannot have the same time slot c2 domain will not have this time slot T1 and c5's domain will also not have T1. So when we consider the next instantiation, when we look at c2 we have to look at the choices for c2 in time slots other than T1.

(Refer Slide Time: 46:34)



When we look at $c3$, if after $c1$ we look at $c3$, $c3$ can be in time slot $T1$ but in time slot $T1$ it cannot be in a class room $c1$. So any two courses cannot share the same room and slot. So both rooms and slots cannot be the same for two courses. And two courses that can be taught by the same professor cannot share a time slot. So, based on these constraints we have to do the assignment of values. You can work out these for a small set of values.

(Refer Slide Time: 47:38)

Any two courses cannot share the same $\langle \text{Room, Slot} \rangle$
2 courses that can be taught by the same professor cannot share a time slot

The second problem we said was a crypt arithmetic problem. Specifically we were asked to work out the values of the different letters for this crypt arithmetic problem.

I will discuss briefly how you go about the solution to this problem. Sometimes for these problems if we are working out manually you can use certain heuristics from your knowledge. The value of M in this case cannot be more than 1. So even if S and M have the highest values this can be utmost 1. This is not 0 because if this value is 0 we would not be putting this here so this is very easy to see that M is equal to 1. So what we can do is, we can simplify this problem by putting 1 in this position. Now we have some other variables left.

What are the variables we have?

The variables are D, E, Y, N, R, O and S so we have seven variables in this problem. Let us say we first pick up D.

Now what are the values that D can take?

D can be 1 or 2 or 3 or 4 or 5 or 6 or 7 or 8 or 9 or 0 so we try the different values of D. There are many possible values of D so we try all possible values of D. Now let us say first we try D is equal to 1. Now if D is equal to 1 we have to try the different values of E. We first try the value E is equal to 1. Now if E is equal to 1 then Y has to be is equal to 2 if D is 1 and E is 1 and Y is 2. So Y is fixed and here we have E is 1, this E is 1, these are the values we have assigned and Y has become 2.

Now we have to choose N and R so that we have 1 here. Suppose so we choose the different possible values of N, suppose you first choose N is equal to 1, if N is equal to 1 then R has to be 0 to satisfy this constraint so we have this assignment. And then E is 1 along this path and we do not know the value of O and N is 1 so E is 1, O is not known, N is 1 so it must be is equal to 0. So here we have o is equal to 0 and we have this is 1 so S must be is equal to 9. Therefore a solution to this problem is S is equal to 9, E is equal to 1, N is equal to 1 and D is equal to 1, M is 1, O is 0, R is 0, E is 1 and M is 1, O is 0, N is 1, E is 1 and Y is 2 and you can verify whether this is correct.

It is 1 plus 1 is equal to 2, 1 plus 0 is equal to 1, 1 plus 0 is equal to 1, 9 plus 1 is equal to 10. So this is a solution to the crypt arithmetic problem send plus more is equal to money. And we have solved this problem by simple depth first search and in this case this is a simple problem with many solutions so we could get this problem by only exploring one path.

(Refer Slide Time: 52:40)

The image shows a handwritten mathematical derivation on a whiteboard. At the top, there is a sequence of assignments: $SENDI$ and $M=1$. Below this, a vertical addition is shown: $+ 10REI$ followed by a horizontal line, then $10NEY2$ followed by another horizontal line. To the right of these additions are the letters $D, E, Y, N,$ and R, O, S . Below the second horizontal line, the letter $D=1$ is written. A tree diagram branches from $D=1$ to $E=1$, which then branches to $Y=2$. From $Y=2$, three branches lead to $N=1$, $R=0$, and $O=0$. Below these, $S=9$ is written. A green circle highlights a vertical addition: 9111 followed by a horizontal line, then $+ 1001$ followed by another horizontal line, resulting in 10112 . At the bottom of the whiteboard, there is a small toolbar with various icons.