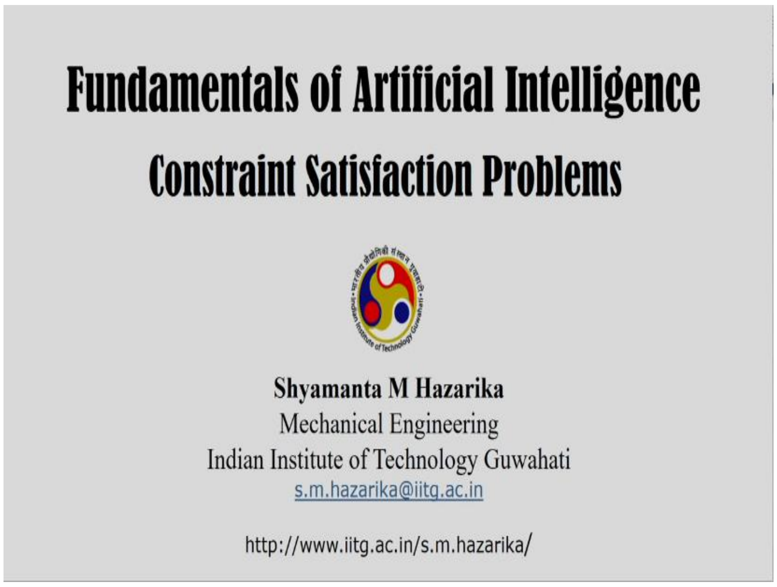
Fundamentals of Artificial Intelligence

Prof. Shyamanta M Hazarika
Department of Mechanical Engineering
Indian Institute of Technology - Guwahati

Module - 2
Lecture - 6
Constraint Satisfaction Problems

Welcome to fundamentals of artificial intelligence. In today's lecture, we would look at constraint satisfaction problems.

(Refer Slide Time: 00:51)



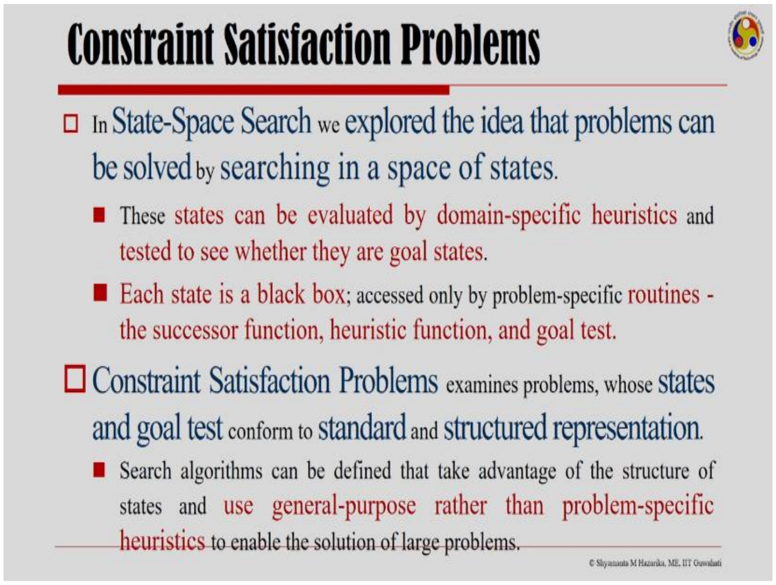
Fundamentals of Artificial Intelligence
Constraint Satisfaction Problems

Shyamanta M Hazarika
Mechanical Engineering
Indian Institute of Technology Guwahati
s.m.hazarika@iitg.ac.in

http://www.iitg.ac.in/s.m.hazarika/

Constraint satisfaction is another way of solving problems in AI, where the main idea is to get to the goal state by satisfying a group of constraints. Many problems can be posed as constraint satisfaction problems. And all the self algorithms or constraint satisfaction problem solvers could then be used to get to the solution.

(Refer Slide Time: 01:21)



Constraint Satisfaction Problems

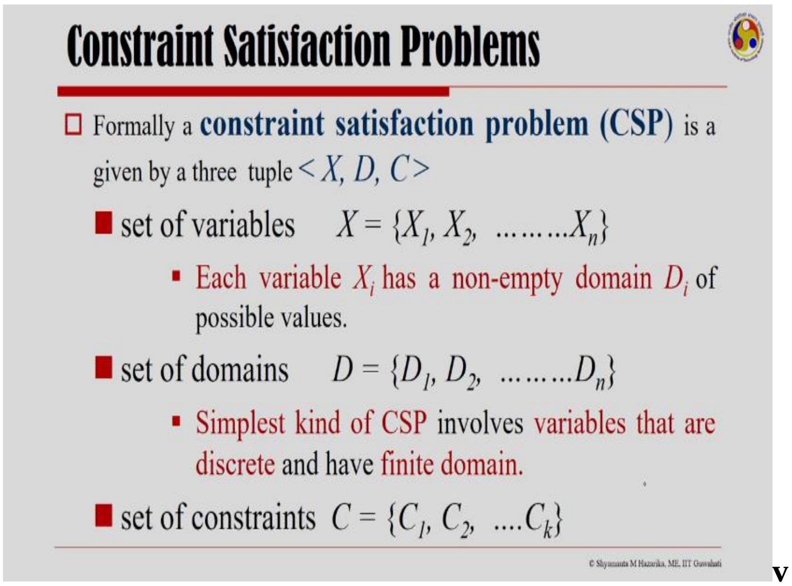
- In State-Space Search we explored the idea that problems can be solved by searching in a space of states.
  - These states can be evaluated by domain-specific heuristics and tested to see whether they are goal states.
  - Each state is a black box; accessed only by problem-specific routines - the successor function, heuristic function, and goal test.
- Constraint Satisfaction Problems examines problems, whose states and goal test conform to standard and structured representation.
  - Search algorithms can be defined that take advantage of the structure of states and use general-purpose rather than problem-specific heuristics to enable the solution of large problems.

© Shyamanta M Hazarika, ME, IIT Guwahati

While we were doing state space search, we explored the idea that problems can be solved by searching in a space of states. In such a situation, this states can be evaluated by domain-specific heuristics and we tested to see whether they are goal states or not. Most of these states that we explored or we have looked at in problem solving as state space search, we can think of each state as a black box, which is assessed only by problem-specific routines: the successor function, the heuristic function and the goal test.

In contrast to them, in constraint satisfaction problems, we examine problems who's states and the goal test itself confirm to standard and structured representation. Such a representation then allows search algorithms to be defined that take advantage of the structure of the states and then use general-purpose rather than problem-specific heuristics to enable solution of large problems.

(Refer Slide Time: 02:40)



Constraint Satisfaction Problems

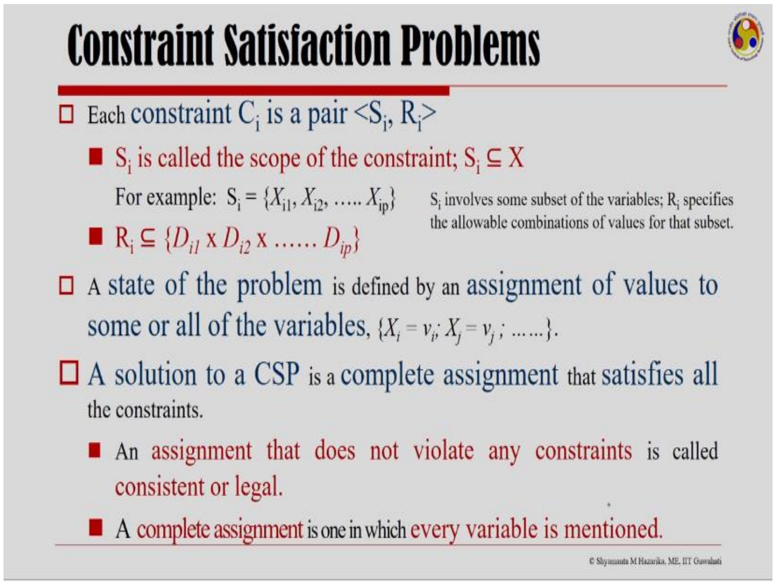
- Formally a **constraint satisfaction problem (CSP)** is a given by a three tuple $< X, D, C >$
  - set of variables $X = \{X_1, X_2, \ldots\ldots X_n\}$
    - Each variable X_i has a non-empty domain D_i of possible values.
  - set of domains $D = \{D_1, D_2, \ldots\ldots D_n\}$
    - Simplest kind of CSP involves variables that are discrete and have finite domain.
  - set of constraints $C = \{C_1, C_2, \ldots C_k\}$

© Shyamanta M Hazarika, ME, IIT Guwahati

v

Formally, a constraint satisfaction problem is a 3 tuple. The set of variables, each variable has a non-empty domain of possible values. A set of domains, simplest kind of CSP involves variables that are discrete and have finite domains. Then we have a set of constraints.

(Refer Slide Time: 03:10)



Constraint Satisfaction Problems

- Each constraint C_i is a pair $<S_i, R_i>$
  - S_i is called the scope of the constraint; $S_i \subseteq X$

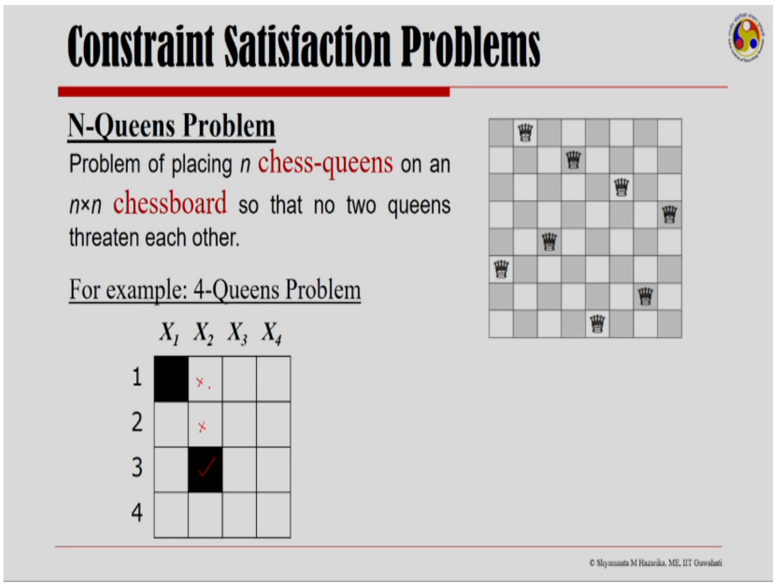
    For example: $S_i = \{X_{i1}, X_{i2}, \ldots X_{ip}\}$

    S_i involves some subset of the variables; R_i specifies the allowable combinations of values for that subset.
  - $R_i \subseteq \{D_{i1} \times D_{i2} \times \ldots\ldots D_{ip}\}$
- A state of the problem is defined by an assignment of values to some or all of the variables, $\{X_i = v_i; X_j = v_j; \ldots\ldots\}$.
- A solution to a CSP is a complete assignment that satisfies all the constraints.
  - An assignment that does not violate any constraints is called consistent or legal.
  - A complete assignment is one in which every variable is mentioned.

© Shyamanta M Hazarika, ME, IIT Guwahati

Each constraint is a pair. S 1 is called the scope of the constraint and involves some subset of the variables. R i specifies allowable combination of values for that subset. A state of the problem is defined by an assignment of values to some or all of the variables. A solution to a CSP is a complete assignment that satisfies all the constraints. A complete assignment is one in which every variable is mentioned. And an assignment is said to be legal or consistent if it does not violate any constraint. A solution is a consistent complete assignment.

(Refer Slide Time: 03:54)



Constraint Satisfaction Problems

N-Queens Problem

Problem of placing *n* chess-queens on an *n*×*n* chessboard so that no two queens threaten each other.

For example: 4-Queens Problem

X_1 X_2 X_3 X_4
1
2
3
4

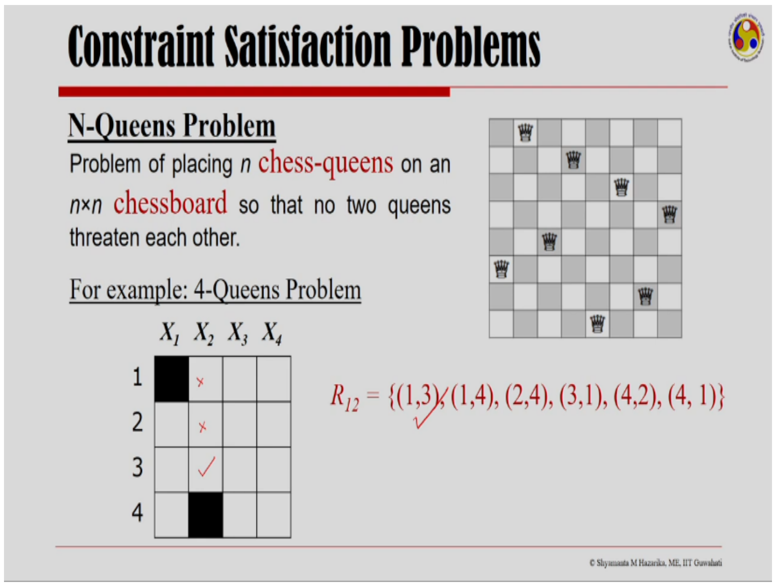
© Shyamanta M Hazarika, ME, IIT Guwahati

Now, let us look at a very popular problem from AI that uses constraint satisfaction problem solving technique, the N-Queens problem: The N-Queens problem is a problem of placing n chess-queens on a n cross n chessboard, so that no queen, 2 queens threaten each other. For example, if we are talking of the 4-queens problem, we have 4 queens to be placed on a 4 cross 4 board.

And we will mark these variables as X 1, X 2, X 3, X 4. And we have 1, 2, 3, 4, the different rows. Now, how do you specify constraints in such a scenario is that the constraints could be explicitly mentioned. Like, if we start with a queen here and look for what could be the position of the second queen; so, we will specify constraints between 1 and 2. Then, a queen cannot be here, a queen cannot be here, but we can place a queen here.

So, that is what we are looking for. And therefore, a constraint that says 1 and 3 is satisfied between 1 and, the first and the second queen could be explicitly stated. So, we have 1 and 3 here.

(Refer Slide Time: 05:38)



Constraint Satisfaction Problems

N-Queens Problem

Problem of placing *n* chess-queens on an *n*×*n* chessboard so that no two queens threaten each other.

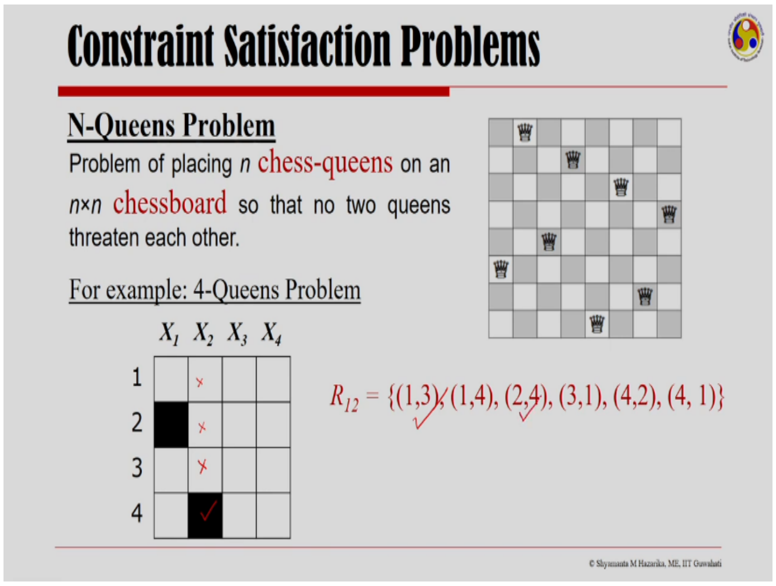
For example: 4-Queens Problem

$R_{12} = \{(1,3), (1,4), (2,4), (3,1), (4,2), (4,1)\}$

© Shyamanta M Hazarika, ME, IIT Guwahati

Similarly, we could include 1 and 4.

(Refer Slide Time: 05:42)



Constraint Satisfaction Problems

N-Queens Problem

Problem of placing *n* chess-queens on an *n*×*n* chessboard so that no two queens threaten each other.

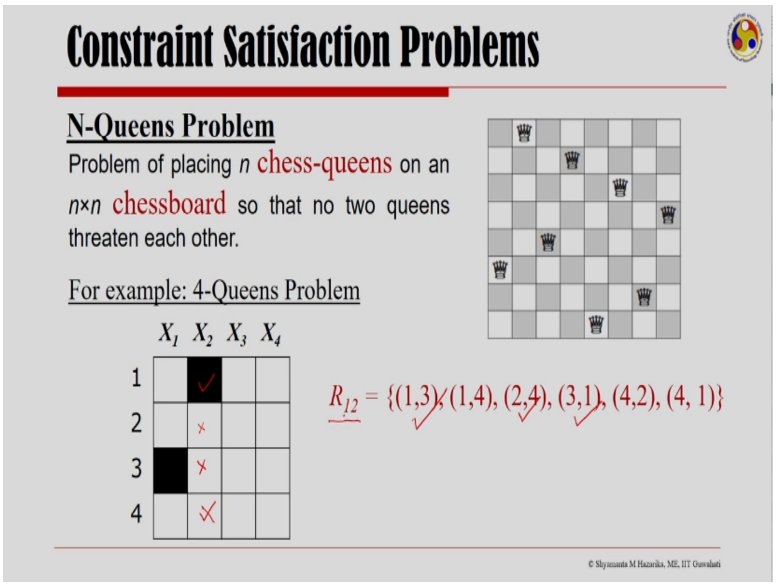
For example: 4-Queens Problem

$R_{12} = \{(1,3), (1,4), (2,4), (3,1), (4,2), (4,1)\}$

© Shyamanta M Hazarika, ME, IIT Guwahati

And then, we could think of the second row for the first queen. And then, the positions that could be possible is also not this, as the only position that would be possible is this. So, we could explicitly state that and include 2 4 here.

(Refer Slide Time: 06:02)



Constraint Satisfaction Problems

N-Queens Problem

Problem of placing n chess-queens on an $n \times n$ chessboard so that no two queens threaten each other.

For example: 4-Queens Problem

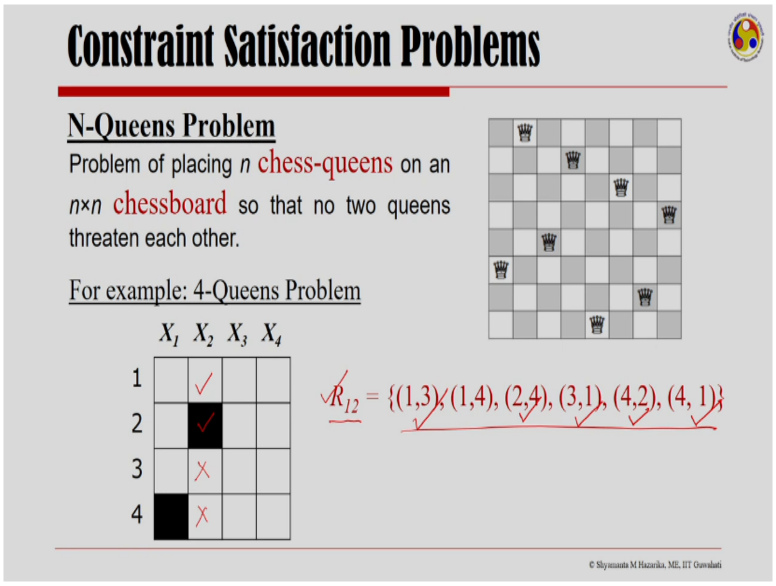
	X_1	X_2	X_3	X_4
1		■ ✓		
2		×		
3	■	×		
4		×		

$R_{12} = \{(1,3), (1,4), (2,4), (3,1), (4,2), (4,1)\}$

© Shyamanta M Hazarika, ME, IIT Guwahati

We could then think of going for the third one, in which case this also is not possible. The only possibility in the third case is having a queen there. And therefore, we would explicitly include 3 1 to be there in the set of R 1 2. Here, 1 and 2 refers to the first and the second queen, a relationship between them.

(Refer Slide Time: 06:33)



Constraint Satisfaction Problems

N-Queens Problem

Problem of placing n chess-queens on an $n \times n$ chessboard so that no two queens threaten each other.

For example: 4-Queens Problem

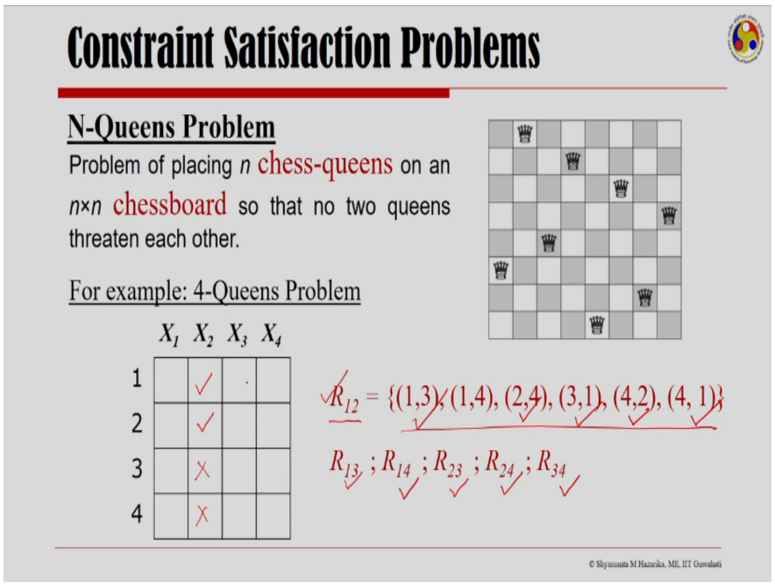
	X_1	X_2	X_3	X_4
1		✓		
2		■ ✓		
3		×		
4	■	×		

$R_{12} = \{(1,3), (1,4), (2,4), (3,1), (4,2), (4,1)\}$

© Shyamanta M Hazarika, ME, IIT Guwahati

Thereafter, we could look at the fourth row and see that these possibilities do not exist, that I cannot have a second queen here, I cannot have a second queen here, but I could have a second queen at 4 2, which is marked there. And I could have a second queen at 4 1, which is marked here. So, as we see here, we could state the constraints that I won between the first and the second queen as an explicit list. And this list is the list of allowable values between the first and the second queen.

(Refer Slide Time: 07:11)



Constraint Satisfaction Problems

N-Queens Problem

Problem of placing *n* chess-queens on an *n*×*n* chessboard so that no two queens threaten each other.

For example: 4-Queens Problem

	X_1	X_2	X_3	X_4
1		✓	·	
2		✓		
3		✗		
4		✗		

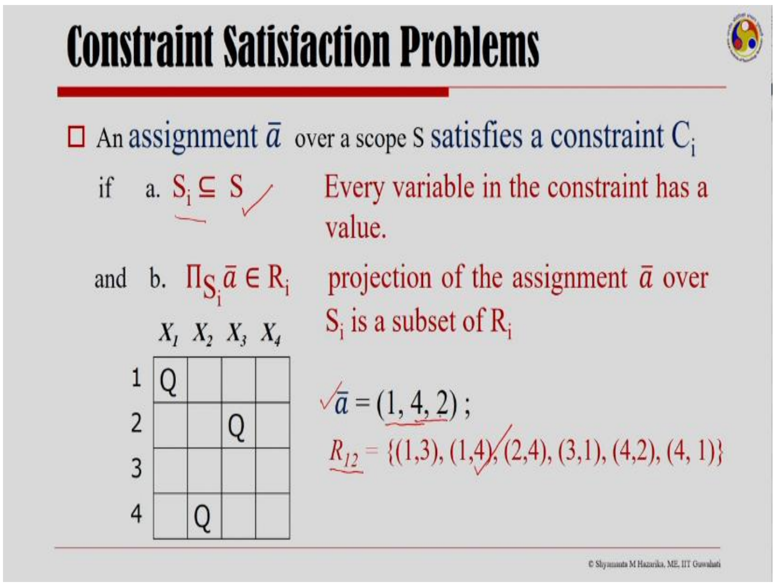
$\checkmark R_{12} = \{(1,3), (1,4), (2,4), (3,1), (4,2), (4,1)\}$

$R_{13}\checkmark ; R_{14}\checkmark ; R_{23}\checkmark ; R_{24}\checkmark ; R_{34}\checkmark$

© Shyamanta M Hazarika, ME, IIT Guwahati

Similarly, I could look for values that would be possible between the first and the third; the first and the fourth; the second and the third; the second and the fourth; and third and fourth variables of this 4-queens problem.

(Refer Slide Time: 07:35)



Constraint Satisfaction Problems

☐ An assignment $\bar{a}$ over a scope S satisfies a constraint C_i

if a. $S_i \subseteq S$ ✓ Every variable in the constraint has a value.

and b. $\Pi_{S_i} \bar{a} \in R_i$ projection of the assignment $\bar{a}$ over S_i is a subset of R_i

	X_1	X_2	X_3	X_4
1	Q			
2			Q	
3				
4		Q		

$\checkmark \bar{a} = (1, 4, 2)$;

$R_{12} = \{(1,3), (1,4)\checkmark, (2,4), (3,1), (4,2), (4, 1)\}$

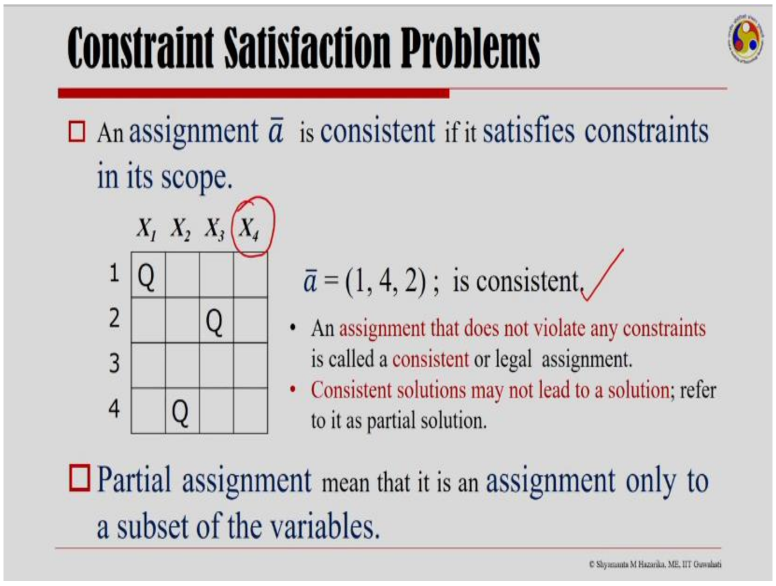
© Shyamanta M Hazarika, ME, IIT Guwahati

An assignment a bar over a scope S satisfies a constraint C if S i is a subset of S. That means, the every variable in the constraint has a value. And a projection of that assignment over the scope is a subset of R i. Now, we need to take a minute and try to understand what we mean by a projection of the assignment. So, let us go back and look at our 4-queens problem once again.

We could have a queen at the first position. The second queen that I can place is definitely either this or this. So, I could place at 4. And the other queen now at position second for the variable X 3 is what I can place. So, this assignment could be written as 1, 4, 2. Now, if you look at this assignment and if you look at the constraints that I was talking of and look at the queen 1 and the queen 2, then I could see that I have 1 4 which is a possibility included in R 1 2.

Similarly, if I had written the other constraints, I could have seen that I could have covered the possibility of having 4 2 between 2 and 3 as well.

(Refer Slide Time: 09:16)



Constraint Satisfaction Problems

- An assignment $\bar{a}$ is consistent if it satisfies constraints in its scope.

	X_1	X_2	X_3	X_4
1	Q			
2			Q	
3				
4		Q		

$\bar{a} = (1, 4, 2)$; is consistent ✓

- An assignment that does not violate any constraints is called a consistent or legal assignment.
- Consistent solutions may not lead to a solution; refer to it as partial solution.

- Partial assignment mean that it is an assignment only to a subset of the variables.

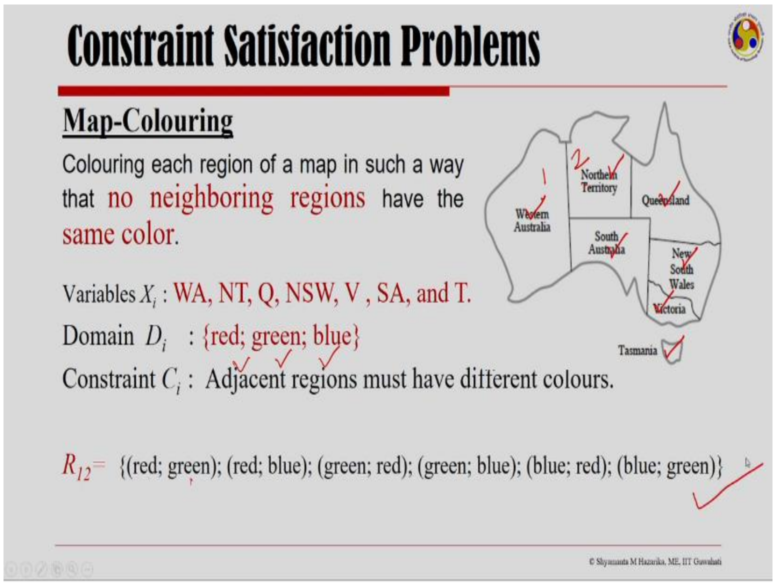
© Shyamanta M Hazarika, ME, IIT Guwahati

An assignment is said to be consistent if it satisfies constraints in its scope. Like in the previous example that we were looking at, where I had an assignment of 1, 4, 2 for 3 queens placed on a 4 cross 4 grid, I could see that it satisfies the constraints. And therefore, that assignment is said to be consistent. An assignment that does not violate any constraint is called a consistent or legal assignment.

Now, this is interesting to note here that consistent assignments may not lead to a solution. Let us look back at this example one more time. We have queen at this position, which is perfectly okay with queen at 4 for X 2 and queen at 2 for X 3. But, when we want to have a queen for a fourth variable, we are unable to have a queen here, we are unable to have the queen at the second location, neither in the third row nor in the fourth row.

So, we are not able to arrive at a solution even if initially this assignment up to the third queen was consistent. Such an assignment is called a partial assignment or a partial solution. A partial assignment is one that is an assignment only to a subset of the variables. In this case, we could assign only up to X 3 and we could not do for X 4.

(Refer Slide Time: 11:00)



Constraint Satisfaction Problems

Map-Colouring

Colouring each region of a map in such a way that no neighboring regions have the same color.

Variables X_i : WA, NT, Q, NSW, V , SA, and T.

Domain D_i : {red; green; blue}

Constraint C_i : Adjacent regions must have different colours.

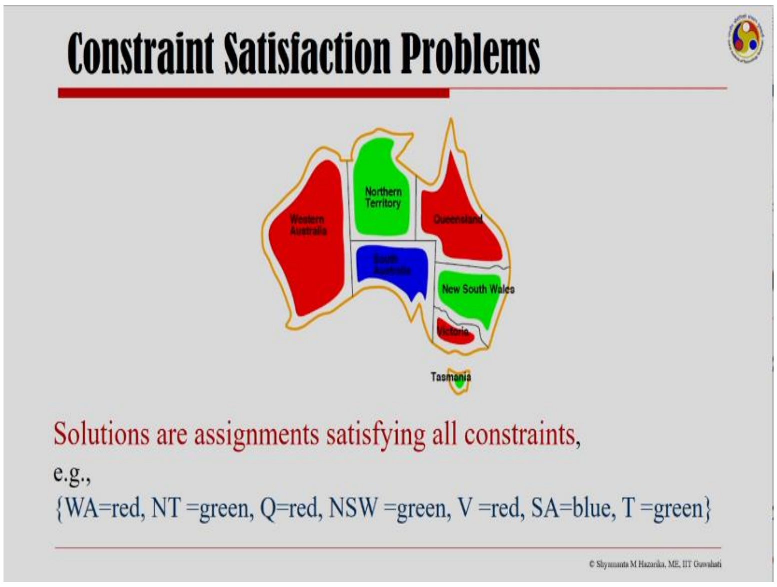
R_{12} = {(red; green); (red; blue); (green; red); (green; blue); (blue; red); (blue; green)}

1
2
Western Australia
Northern Territory
Queensland
South Australia
New South Wales
Victoria
Tasmania
© Shyamanta M Hazarika, ME, IIT Guwahati

We look at another example from constraint satisfaction problems, called the map coloring problem, which is about coloring each region of a map in such a way that no neighboring regions have the same color. So here, if you notice, we have 1, 2, 3, 4, 5, 6, 7 variables. And let us say we are interested in coloring them in either of 3 colors. So, we would have a domain of 3: red, green and blue.

Now, if we look and what is the constraint for this problem, the constraint for this problem is that any 2 adjacent regions must have different colors. So, if you look at the first 2, let us say this is 1 and this is 2. The possibilities could be that I could have red, green or red, blue or green, red or green, blue; blue, red or blue, green. Now, this is how explicitly we could state the constraints or we could state what is allowed and the rest of the things are not allowed in mapping the regions 1 and 2, map coloring them 1 and 2.

(Refer Slide Time: 12:22)



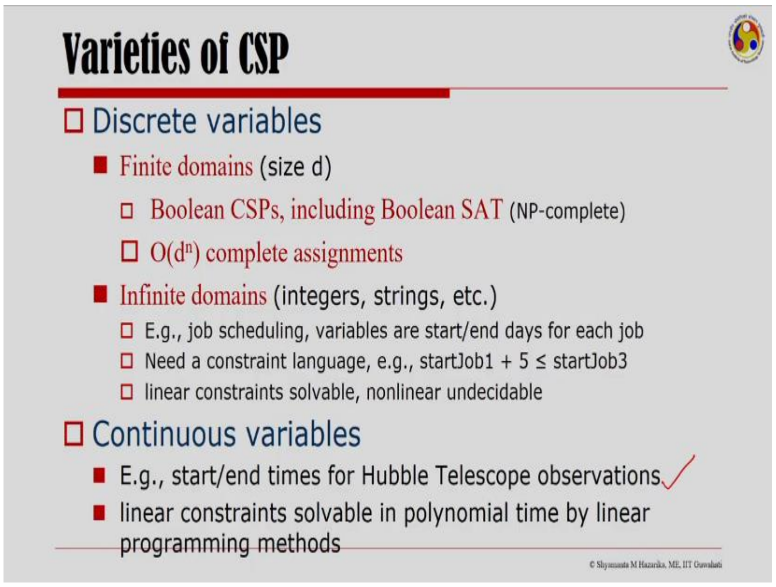
Constraint Satisfaction Problems

Solutions are assignments satisfying all constraints,
e.g.,
{WA=red, NT =green, Q=red, NSW =green, V =red, SA=blue, T =green}

© Shyamanta M Hazarika, ME, IIT Guwahati

So, here is a solution for the map coloring problem, where we could have every region colored different. We will come back to this in our course of discussion today.

(Refer Slide Time: 12:35)



Varieties of CSP

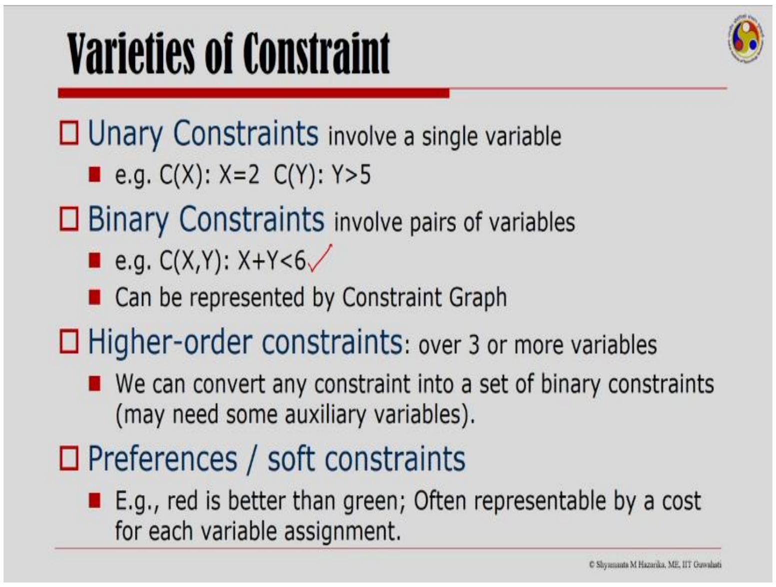
- Discrete variables
  - Finite domains (size d)
    - Boolean CSPs, including Boolean SAT (NP-complete)
    - $O(d^n)$ complete assignments
  - Infinite domains (integers, strings, etc.)
    - E.g., job scheduling, variables are start/end days for each job
    - Need a constraint language, e.g., startJob1 + 5 ≤ startJob3
    - linear constraints solvable, nonlinear undecidable
- Continuous variables
  - E.g., start/end times for Hubble Telescope observations ✓
  - linear constraints solvable in polynomial time by linear programming methods

© Shyamanta M Hazarika, ME, IIT Guwahati

Now, depending on the types of variables, we could have varieties of constraint satisfaction problems. The most common of them are the ones with discrete variables. So, they have finite domains. The Boolean CSPs including the Boolean satisfiability problems are of this nature. We have infinite domains, domains that have integers, strings, etcetera. For example, the job scheduling; they need a constraint language and linear constraint are solvable, nonlinear are undecidable.

Then, we have continuous variable constraint satisfaction problems, where for example, the start end times for the Hubble Telescope observation. Linear constraints are solvable in polynomial time for such CSPs.

(Refer Slide Time: 13:31)



Varieties of Constraint

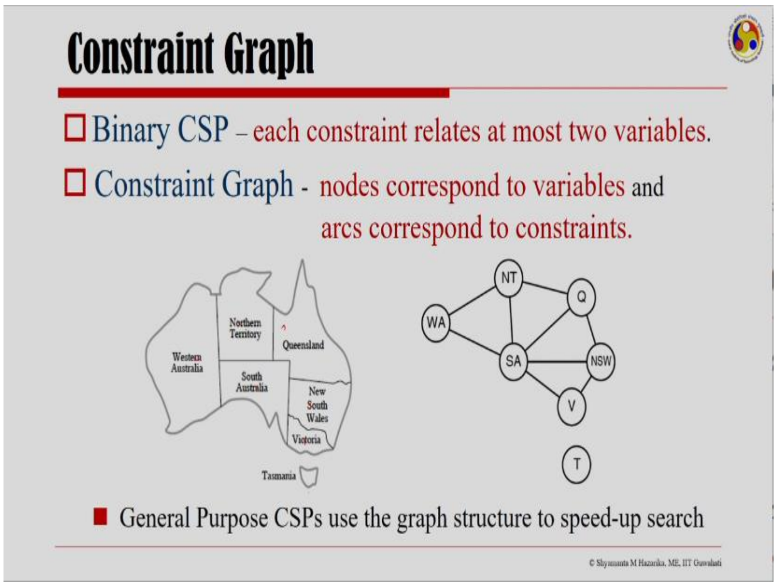
- Unary Constraints involve a single variable
  - e.g. C(X): X=2 C(Y): Y>5
- Binary Constraints involve pairs of variables
  - e.g. C(X,Y): X+Y<6✓
  - Can be represented by Constraint Graph
- Higher-order constraints: over 3 or more variables
  - We can convert any constraint into a set of binary constraints (may need some auxiliary variables).
- Preferences / soft constraints
  - E.g., red is better than green; Often representable by a cost for each variable assignment.

© Shyamanta M Hazarika, ME, IIT Guwahati

Varieties of constraints may be possible. We may have unary constraints, constraints that involve a single variable. Or, we may have binary constraints, constraints that involve pair of variables. For example here, I have a constraint between X and Y and the sum of X + Y must be < 6. Binary constraints can be represented by constraint graph. Then, we could have higher order constraints, constraints over 3 or more variables.

Now, it is interesting to note that we can convert any constraint into a set of binary constraints. They may need some auxiliary variables to be introduced. The fourth type of constraint that we could have are called preference or soft constraints. Things like, we could have constraints stating red is better than green. They are often representable by a cost for each variable assignment.

(Refer Slide Time: 14:37)



Constraint Graph

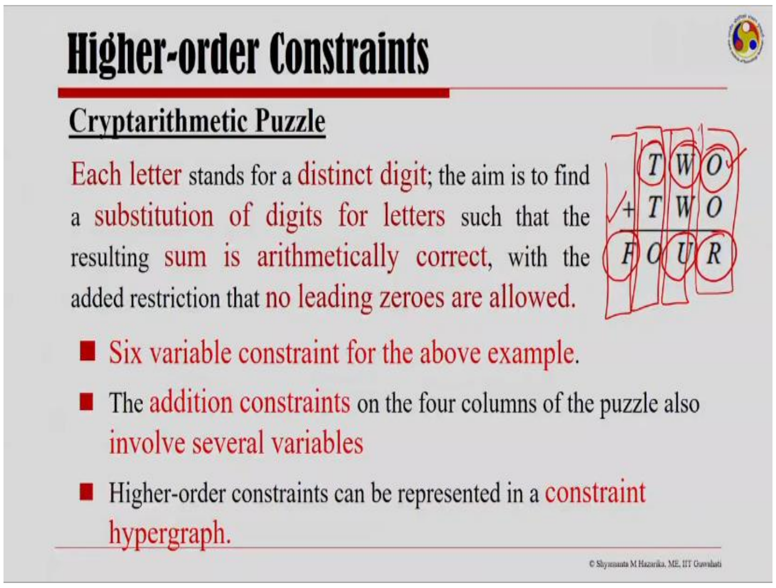
- Binary CSP – each constraint relates at most two variables.
- Constraint Graph - nodes correspond to variables and arcs correspond to constraints.

- General Purpose CSPs use the graph structure to speed-up search

© Shyamanta M Hazarika, ME, IIT Guwahati

As I was telling you, binary constraint satisfaction problem is one in which each constraint relates at most 2 variables. We could create a constraint graph where nodes correspond to variables and arcs correspond to constraints. Coming back to the map coloring problem here, we had 7 constraints: 1, 2, 3, 4, 5, 6, 7. And we could draw a constraint graph here to represent relation between the variables on the constraint graph, each of the nodes corresponding to the variables. General purpose constraint satisfaction problems use the graph structure to speed up search. We would look at this in one of our lectures.

(Refer Slide Time: 15:30)



Higher-order Constraints

Cryptarithmetic Puzzle

Each letter stands for a distinct digit; the aim is to find a substitution of digits for letters such that the resulting sum is arithmetically correct, with the added restriction that no leading zeroes are allowed.

T W O
+ T W O
F O U R

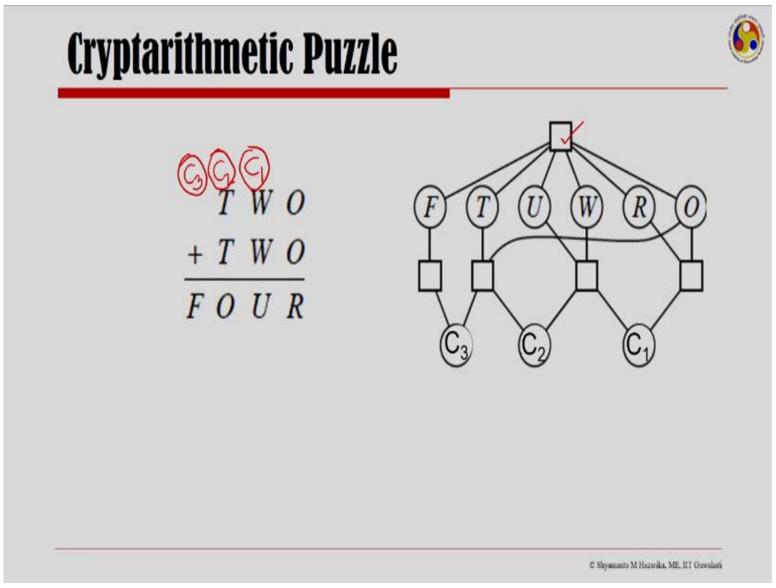
- Six variable constraint for the above example.
- The addition constraints on the four columns of the puzzle also involve several variables
- Higher-order constraints can be represented in a constraint hypergraph.

© Shyamanta M Hazarika, ME, IIT Guwahati

While we are talking of higher order constraints, let us focus our attention on a very interesting cryptarithmetic puzzle. Cryptarithmetic puzzles are puzzles where you write things in letters and operate using mathematical operators and expect that each letter, if it stands for distinct digit, we can find some substitution for the letters in terms of the digits, such that the sum is arithmetically correct, with the added restriction here, that this leading letter should not have a 0.

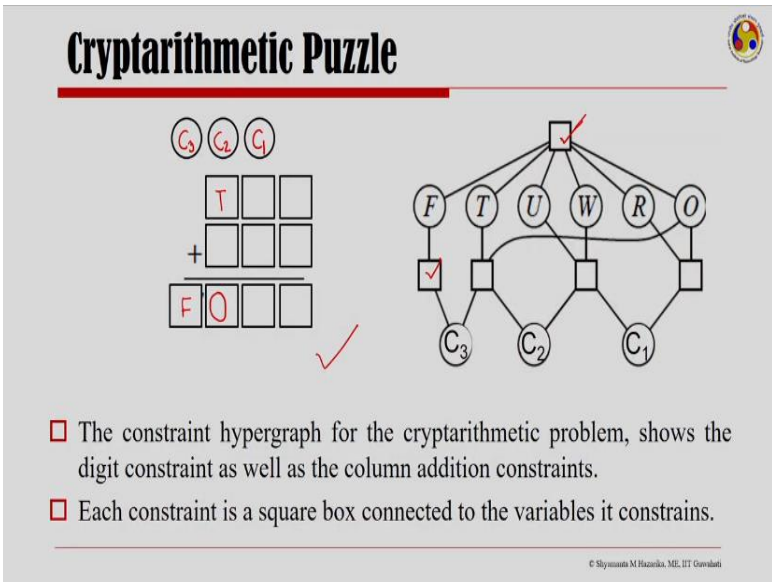
When we are trying to solve such problems, we can see that the small problem of saying 2 + 2 is 4, we have 6 variable constraints for the above example. This literally means that all 6 letters that we can think of: T, W, O, R, U and F are together in a constraint. And the constraint is that all of them needs to be a different digit. The addition constraints on the 4 columns of this puzzle: 1, 2, 3, 4; the 4 columns of this puzzle, they also involve several variables. All of these create a higher order constraint and they could be represented in a constraint hypergraph.

(Refer Slide Time: 17:18)

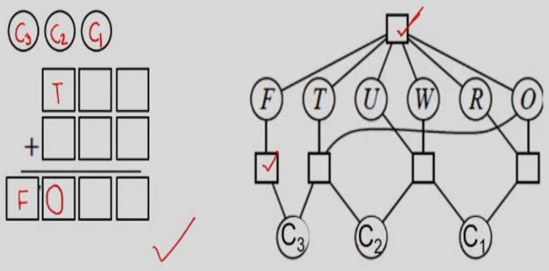


So, here is the cryptarithmetic puzzle 2 + 2, 4. On your right is the hypergraph. We could see that the 6 variables F, T, U, W, R and O are all together constraint. And this C 1, C 2, C 3, are the carryovers possible at these columns.

(Refer Slide Time: 17:49)



Cryptarithmetic Puzzle



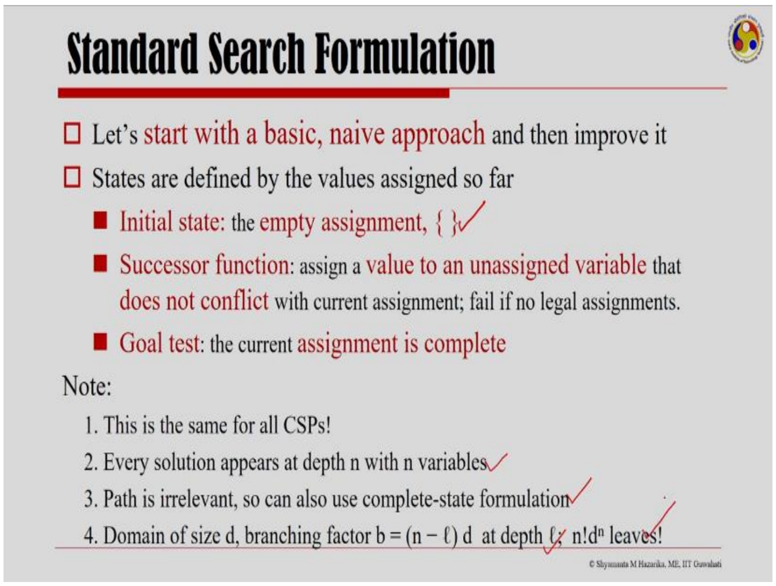
- The constraint hypergraph for the cryptarithmetic problem, shows the digit constraint as well as the column addition constraints.
- Each constraint is a square box connected to the variables it constrains.

© Shyamanta M Hazarika, ME, IIT Guwahati

So, when we want to solve them, this could be like 6 placeholders and I could have 4 placeholders for the answer. And I could have these placeholders for the carries. And the constraint hypergraph for this problem shows both the digit constraints as well as the column addition constraints. The constraint that all of them needs to be different digit is coming from this square box.

And the constraint that F and C 3, if this is the carryover C 1, carryover C 2 and carryover C 3. If you remember this was F. So, the constraint between F and C 3 is coming here. C 3 and F have a constraint. Similarly, and C 2 as well as O here have a constraint. So, T, C 2 and O have a constraint, so and so forth. So, the constraint hypergraph for the cryptarithmetic problem shows the digit constraints as well as the column addition constraints. And each constraint is a square box connected to the variables it constrains.

(Refer Slide Time: 19:21)



Standard Search Formulation

- Let's start with a basic, naive approach and then improve it
- States are defined by the values assigned so far
  - Initial state: the empty assignment, { }
  - Successor function: assign a value to an unassigned variable that does not conflict with current assignment; fail if no legal assignments.
  - Goal test: the current assignment is complete

Note:

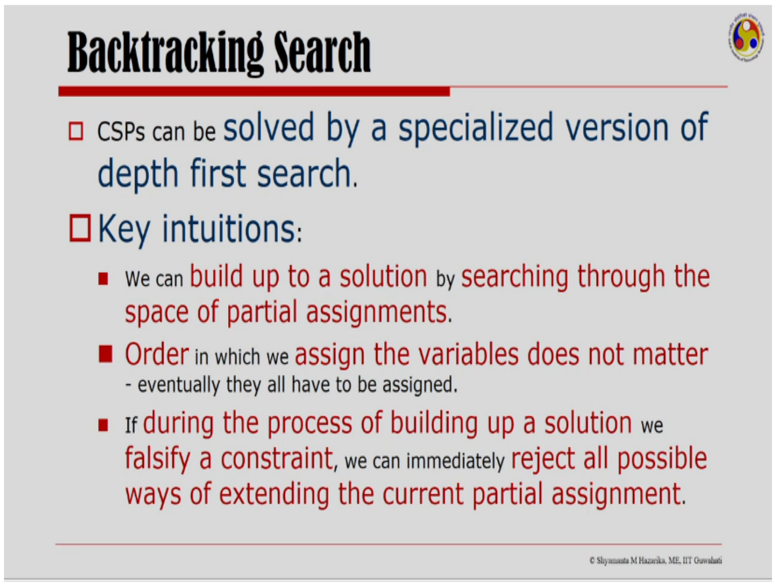
1. This is the same for all CSPs!
2. Every solution appears at depth n with n variables
3. Path is irrelevant, so can also use complete-state formulation
4. Domain of size d, branching factor b = (n − ℓ) d at depth ℓ, n!d^n leaves!

© Shyamanta M Hazarika, ME, IIT Guwahati

So, how do you look for solutions under such problems. Under standard search formulation, we can start with a basic naive approach and then improve it. States are defined by the values assigned so far. So, we could have the initial state given by the empty assignment. And then, a successor function, like assign a value to an unassigned variable that does not conflict could be defined as a successor function.

And the goal test could be if the current assignment is complete. Now, we should know that this is same for all CSPs. Every solution appears at depth n with n variables; path is irrelevant; so, can also use complete state formulation. And for a domain of size d, we have a branching factor b of n – l into d at depth l. So, that leads to factorial n to d to the power n leaves. And this is a huge space to search. Could we do better than this?

(Refer Slide Time: 20:35)



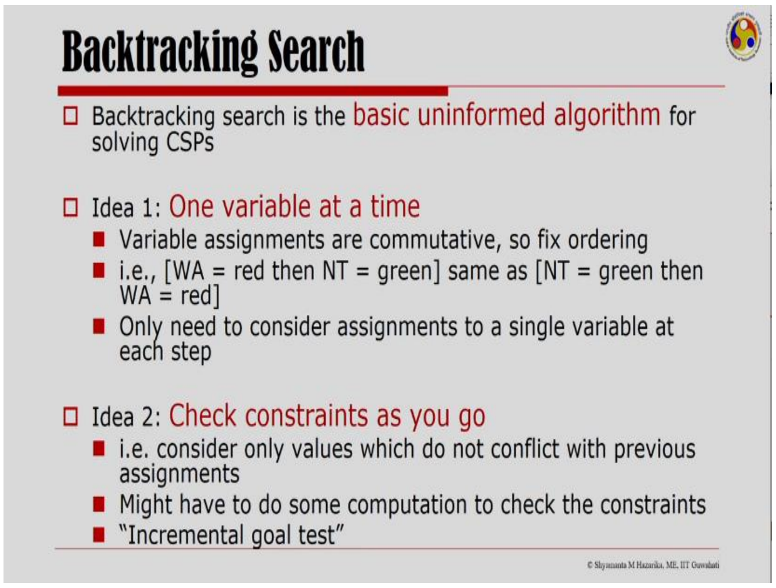
Backtracking Search

- CSPs can be solved by a specialized version of depth first search.
- Key intuitions:
  - We can build up to a solution by searching through the space of partial assignments.
  - Order in which we assign the variables does not matter - eventually they all have to be assigned.
  - If during the process of building up a solution we falsify a constraint, we can immediately reject all possible ways of extending the current partial assignment.

© Shyamanta M Hazarika, ME, IIT Guwahati

So, we can look for a specialized version of depth-first search. CSPs are solved by this and the search is called backtracking search. The key intuition is that we build up to a solution by searching through the space of partial assignments. Order in which we assign the variable does not matter; eventually they all have to be assigned. If during the process of building up a solution we falsify a constraint, we realize that this cannot be on the part of the solution. So, we can immediately reject all possible ways of extending the current partial assignment and backtrack.

(Refer Slide Time: 21:23)



Backtracking Search

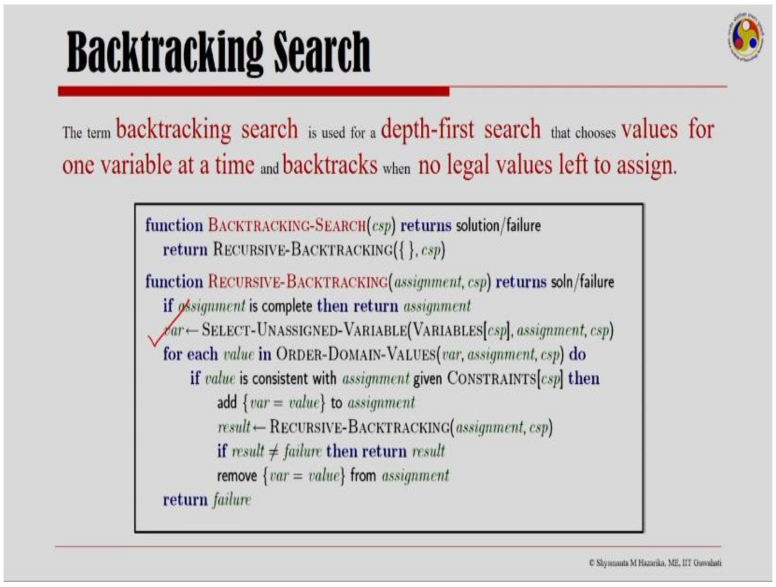
- Backtracking search is the basic uninformed algorithm for solving CSPs
- Idea 1: One variable at a time
  - Variable assignments are commutative, so fix ordering
  - i.e., [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each step
- Idea 2: Check constraints as you go
  - i.e. consider only values which do not conflict with previous assignments
  - Might have to do some computation to check the constraints
  - "Incremental goal test"

© Shyamanta M Hazarika, ME, IIT Guwahati

Backtracking search is the basic uninformed algorithm for solving constraints satisfaction problems. We look at one variable at a time. So, variable assignments are commutative. No fixed ordering is there. Only need to consider assignments to a single variable at each step.

And the second idea is to check for constraints as we go about. That is, we consider only values which do not conflict with the previous assignment. For that, we might have to do some computation to check the constraints. But then, we get one step at a time, towards the goal.

(Refer Slide Time: 22:06)



Backtracking Search

The term backtracking search is used for a depth-first search that chooses values for one variable at a time and backtracks when no legal values left to assign.

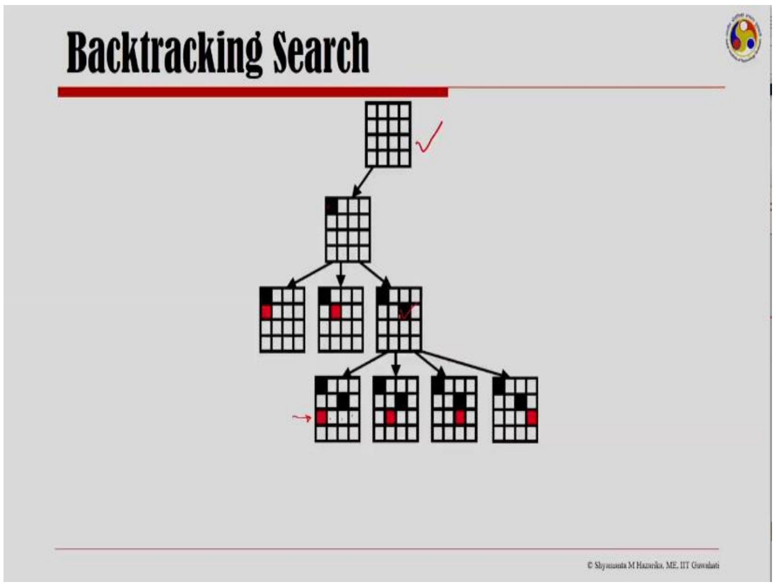
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
   return RECURSIVE-BACKTRACKING({ }, csp)
function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment given CONSTRAINTS[csp] then
         add {var = value} to assignment
         result ← RECURSIVE-BACKTRACKING(assignment, csp)
         if result ≠ failure then return result
         remove {var = value} from assignment
   return failure
```

© Shyamanta M Hazarika, ME, IIT Guwahati

Here is the backtracking search algorithm. The only thing I want to focus here or point you out is the idea of selecting the unassigned variable. We have a routine to select the unassigned variable. That is done in a way that no constraints are violated. Is used for a depth-first search. Values for one variable at a time and backtracks when no legal values are left to be assigned.

(Refer Slide Time: 22:43)

Backtracking Search

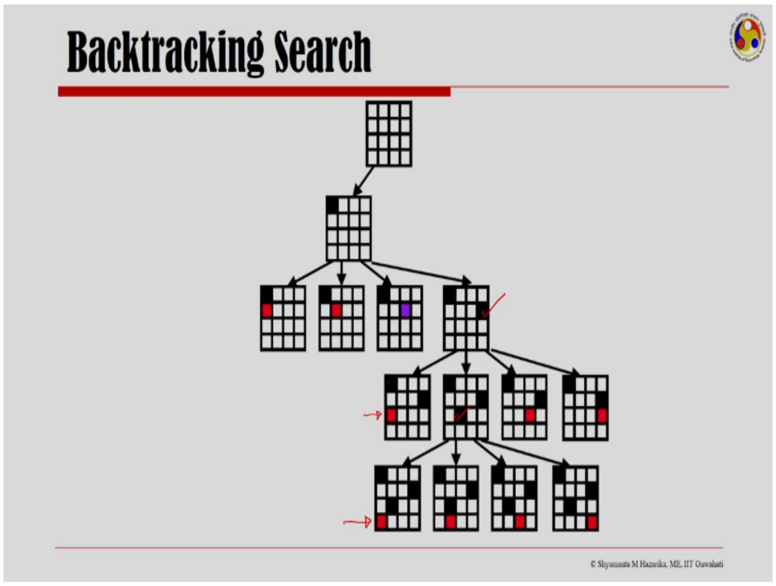


© Shyamanta M Hazarika, ME, IIT Guwahati

So, let us look at our 4-queens problem and try to understand backtracking search. So, here is the 4 cross 4 grid. And we start with placing our queen at this location. So, from the constraints that we have placed our self that no 2 queens should attack each other, we know that this is not a legal position for the second queen, this is not a legal position for the second queen, but the third position on the second row is a legal position for the second queen.

So, we put the second queen here and move one step further to look for solution for the third row. This is no longer a legal position, this is not a legal position, this is no longer a legal position, neither is this a legal position. So, what we have realized at this point is that after we have assigned a second queen, we are not left with any legal position for the third queen. We have to backtrack.

(Refer Slide Time: 24:03)



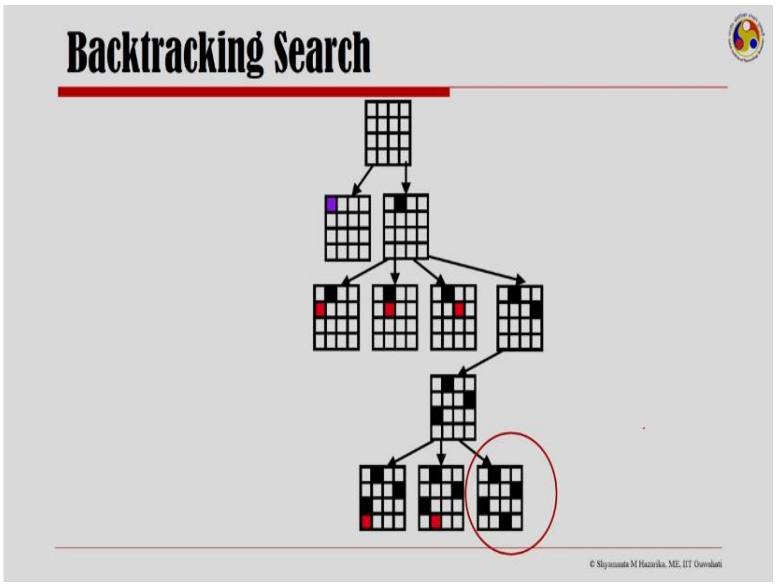
Backtracking Search

© Shyamanta M Hazarika, ME, IIT Guwahati

So, we remember what we started with, backtrack and get a new position for our second queen. With the new position for the second queen, we start once again with the third row to place our third queen. The first position is not legal because we have a queen here which will attack. But the second position is a legal position. Therefore, we move forward and look for the fourth row.

So, we cannot have a queen here, neither we can have a queen here, nor a queen at the third position, nor at the fourth in on the fourth row. That means, after we have assigned this queen, we are not left with any legal positions in the next step. So, we have to abandon this path and backtrack. So, we backtrack.

(Refer Slide Time: 25:04)



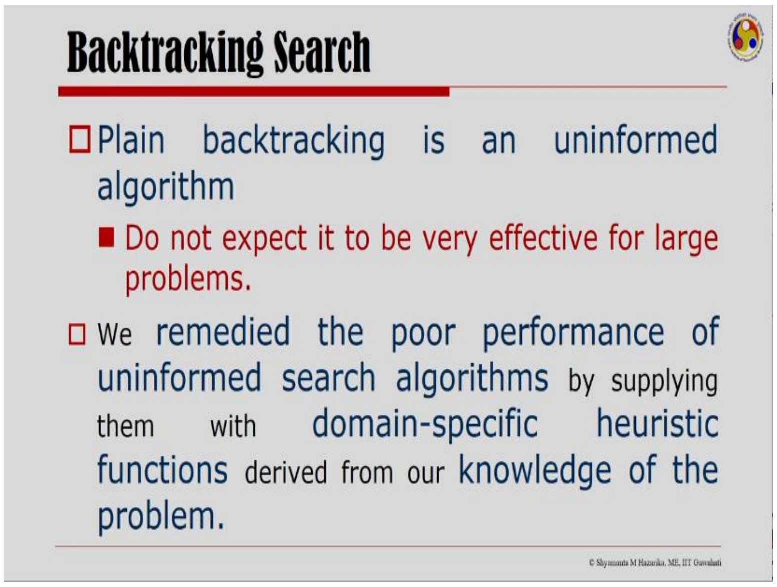
Backtracking Search

And now, where do we backtrack. We need to understand that we have already looked at all possibilities for the second queen, given the first queen. So, we must backtrack to starting with another position for the first queen. So, we backtrack to a point where we put our first queen on the second column. Then we have the queen here. The third row, we could have a legal position at this.

Given these 3 queens, we could get a legal position for our fourth queen. So, that is our solution. So, that is how backtracking search works. Let me repeat this very quickly one more time. So, we had some legal positions up to this level. And when we tried for the third level, we could not have a legal position. So, we backtracked and started with an alternative in the second position.

We could have a third position very nicely. But we could not have a fourth position placed. But then, we have realized that we have exhausted all possibilities of our first queen itself and therefore backtrack to removing our first queen to a different position. And once we had done this, we could arrive at a solution.

(Refer Slide Time: 26:38)



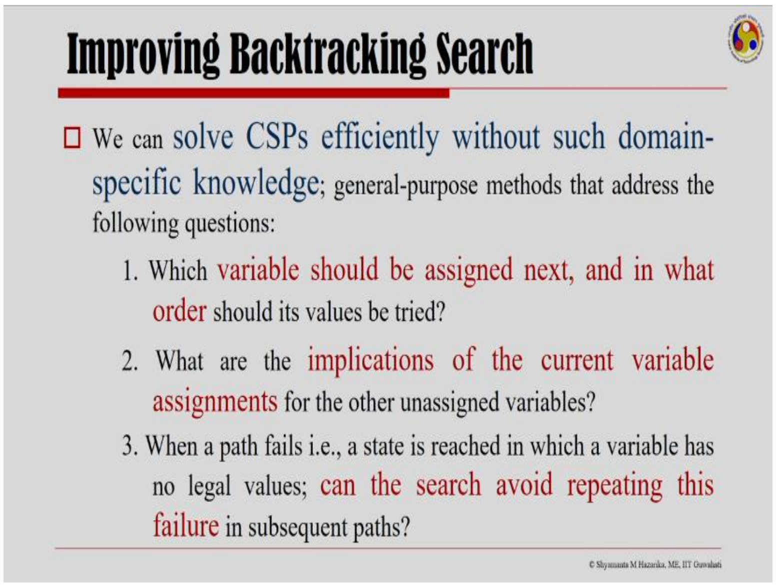
Backtracking Search

- Plain backtracking is an uninformed algorithm
  - Do not expect it to be very effective for large problems.
- We remedied the poor performance of uninformed search algorithms by supplying them with domain-specific heuristic functions derived from our knowledge of the problem.

© Shyamanta M Hazarika, ME, IIT Guwahati

So, playing backtracking as you have realized is an uninformed algorithm. We do not expect it to be very effective for large problems. We remedied the poor performance of uninformed search algorithms in our previous discussion of state space search by supplying them with domain-specific heuristic functions, derived from our knowledge of the problem.

(Refer Slide Time: 27:15)



Improving Backtracking Search

- We can solve CSPs efficiently without such domain-specific knowledge; general-purpose methods that address the following questions:
  1. Which variable should be assigned next, and in what order should its values be tried?
  2. What are the implications of the current variable assignments for the other unassigned variables?
  3. When a path fails i.e., a state is reached in which a variable has no legal values; can the search avoid repeating this failure in subsequent paths?

© Shyamanta M Hazarika, ME, IIT Guwahati

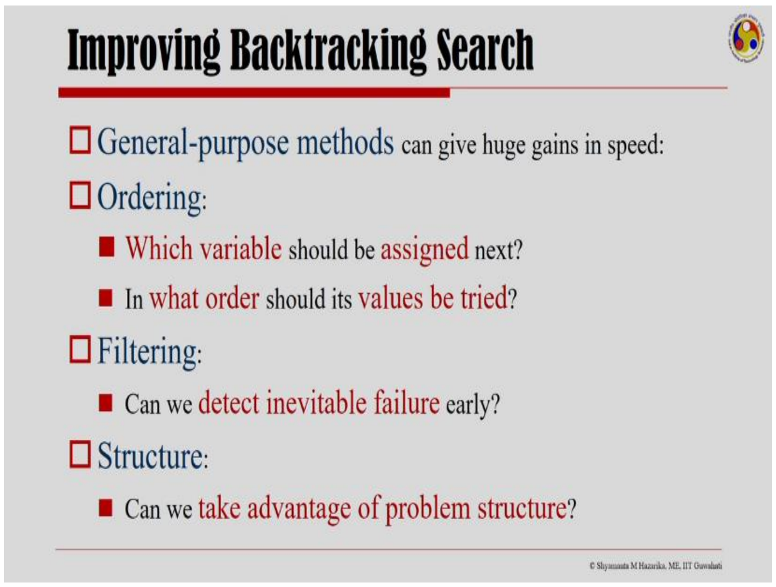
But because constraint satisfaction problems have in themselves some structure or representation, we can solve CSPs efficiently, without such domain-specific knowledge. General purpose methods that address the following questions improve backtracking search.

a. Which variables should be assigned next and in what order should these variables be tried?

b. What are the implications of the current variable assignment for the other unassigned variables?

c. When a path fails, a state is reached in which a variable has no legal values. Can the search avoid repeating this failure in the subsequent paths?

(Refer Slide Time: 28:01)



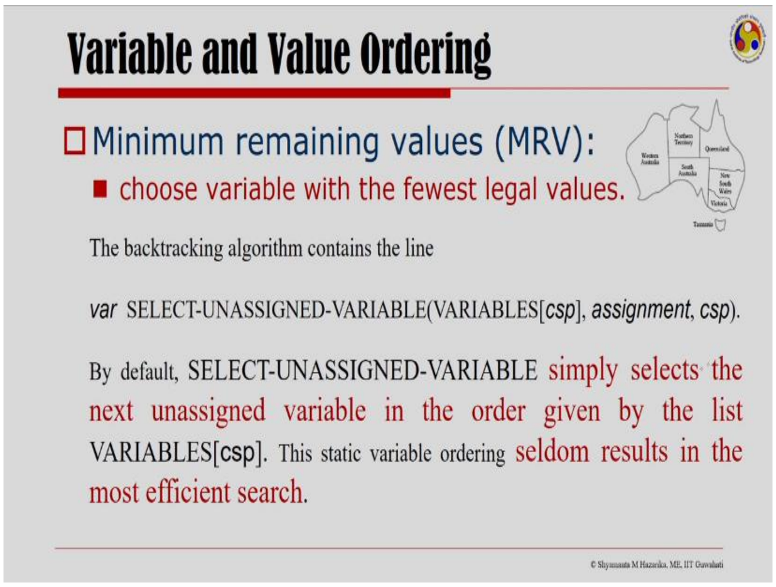
Improving Backtracking Search

- General-purpose methods can give huge gains in speed:
- Ordering:
  - Which variable should be assigned next?
  - In what order should its values be tried?
- Filtering:
  - Can we detect inevitable failure early?
- Structure:
  - Can we take advantage of problem structure?

© Shyamanta M Hazarika, ME, IIT Guwahati

General purpose methods can give huge gains in speed and in improvement of backtracking search. Ordering: Which variable should be assigned next? In what order should its values be tried? Or, filtering: Can we detect inevitable failures early? Or, structure: Can we take advantage of the problem structure? These can lead to improving backtracking search.

(Refer Slide Time: 28:31)


Northern Territory
Queensland
Western Australia
South Australia
New South Wales
Victoria
Tasmania

Variable and Value Ordering

- Minimum remaining values (MRV):
  - choose variable with the fewest legal values.

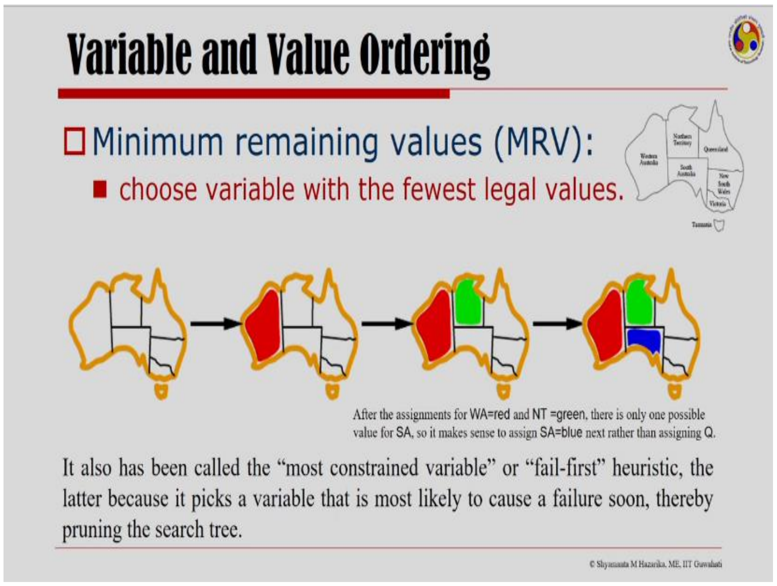
The backtracking algorithm contains the line

var SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*], *assignment*, *csp*).

By default, SELECT-UNASSIGNED-VARIABLE simply selects the next unassigned variable in the order given by the list VARIABLES[csp]. This static variable ordering seldom results in the most efficient search.

© Shyamanta M Hazarika, ME, IIT Guwahati

Let us now look at the variable and value ordering heuristics. The backtracking algorithm contains a call to select the unassigned variable. And it simply selects the next unassigned variable in the order given by the list of variables. Now, this static ordering would seldom result in efficient search. Therefore, the idea is to choose variables with the fewest legal values. **(Refer Slide Time: 29:08)**



Variable and Value Ordering

- Minimum remaining values (MRV):
  - choose variable with the fewest legal values.

Western Australia
Northern Territory
Queensland
South Australia
New South Wales
Victoria
Tasmania

After the assignments for WA=red and NT =green, there is only one possible value for SA, so it makes sense to assign SA=blue next rather than assigning Q.

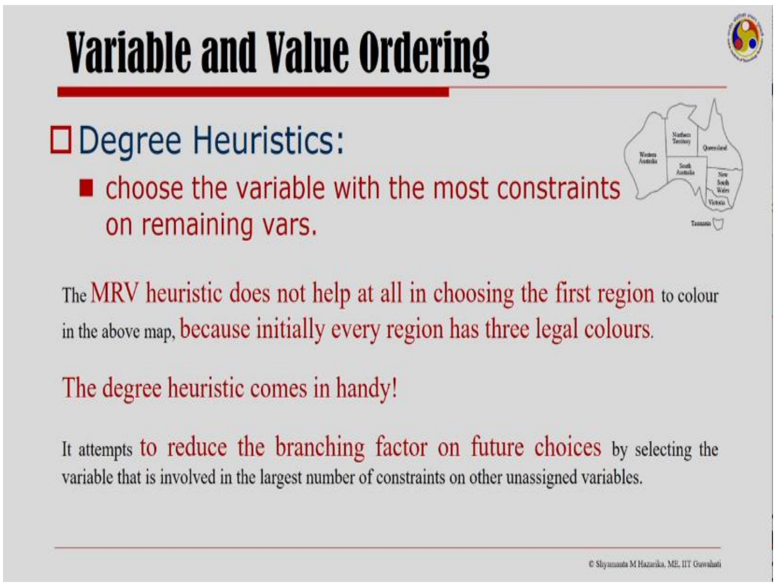
It also has been called the "most constrained variable" or "fail-first" heuristic, the latter because it picks a variable that is most likely to cause a failure soon, thereby pruning the search tree.

© Shyamanta M Hazarika, ME, IIT Guwahati

Let us try to understand this with the help of our map coloring example. So, on your right, we have the map coloring problem with 7 variables. Let us say, after the assignment of WA to red, we assign NT as green. Now, there is only one value possible for SA here. And that is blue. So, it makes sense to assign blue here, rather than assigning the value for Q. This is what the minimum remaining value heuristic does.

Now, it has also been called the most constrained variable or the fail-first heuristic. Because it picks up a variable that is most likely to cause a failure soon, thereby pruning the search tree.

(Refer Slide Time: 30:10)



Variable and Value Ordering

- Degree Heuristics:
  - choose the variable with the most constraints on remaining vars.

Western Australia
Northern Territory
Queensland
South Australia
New South Wales
Victoria
Tasmania

The MRV heuristic does not help at all in choosing the first region to colour in the above map, because initially every region has three legal colours.

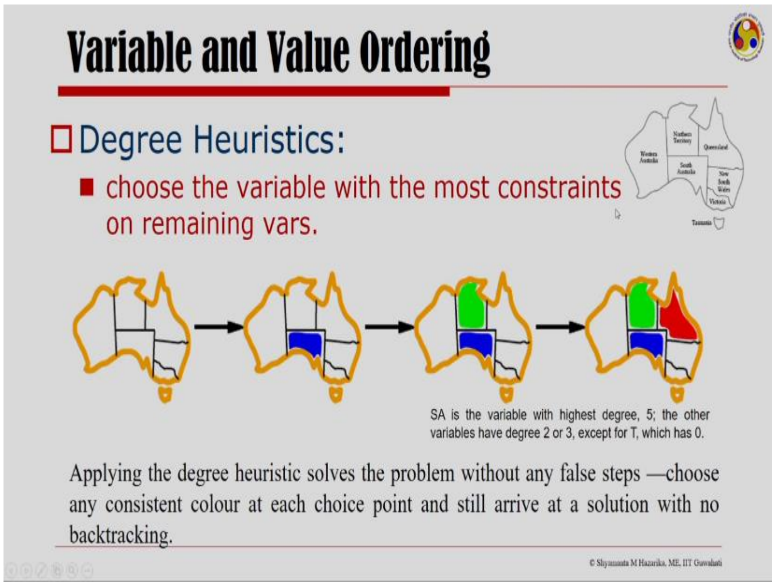
The degree heuristic comes in handy!

It attempts to reduce the branching factor on future choices by selecting the variable that is involved in the largest number of constraints on other unassigned variables.

© Shyamanta M Hazarika, ME, IIT Guwahati

The minimum remaining value heuristic does not help at all in choosing what would be the first region to color in our map coloring problem. Because, all of these 7 variables, when we start, can be colored in any one of red, blue or green. Now, this is where the degree heuristic comes in handy. The degree heuristic lets us to choose a variable with the most constraints on remaining variables. And it does attempts to reduce the branching factor by choosing the variable that is involved in the largest number of constraints on other unassigned variables.

(Refer Slide Time: 31:02)



Variable and Value Ordering

- Degree Heuristics:
  - choose the variable with the most constraints on remaining vars.

SA is the variable with highest degree, 5; the other variables have degree 2 or 3, except for T, which has 0.

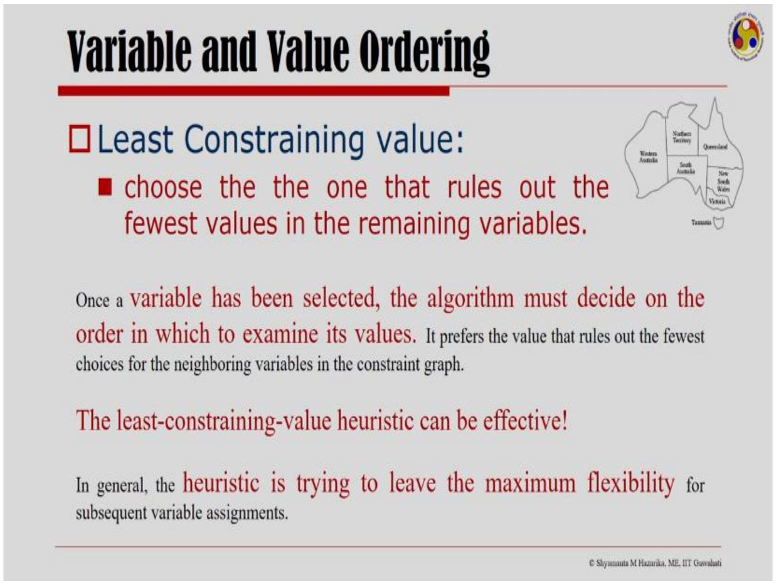
Applying the degree heuristic solves the problem without any false steps —choose any consistent colour at each choice point and still arrive at a solution with no backtracking.

© Shyamanta M Hazarika, ME, IIT Guwahati

For the map coloring problem, if you look at the 7 variables, SA is the variable with the highest degree, which is 1, 2, 3, 4 and 5. The other variables have degree either 2 or 3, except TA here, which has a degree 0. So, we choose the variable with the most constraints on the remaining variable. That is, we first choose SA. And thereafter, we choose NT, and so on and so forth.

Applying the degree heuristic actually solves the problem without any false step. If we keep on choosing the consistent color at each choice point, we can still arrive at a solution with no backtracking.

(Refer Slide Time: 32:03)



Variable and Value Ordering

- Least Constraining value:
  - choose the the one that rules out the fewest values in the remaining variables.

Once a variable has been selected, the algorithm must decide on the order in which to examine its values. It prefers the value that rules out the fewest choices for the neighboring variables in the constraint graph.

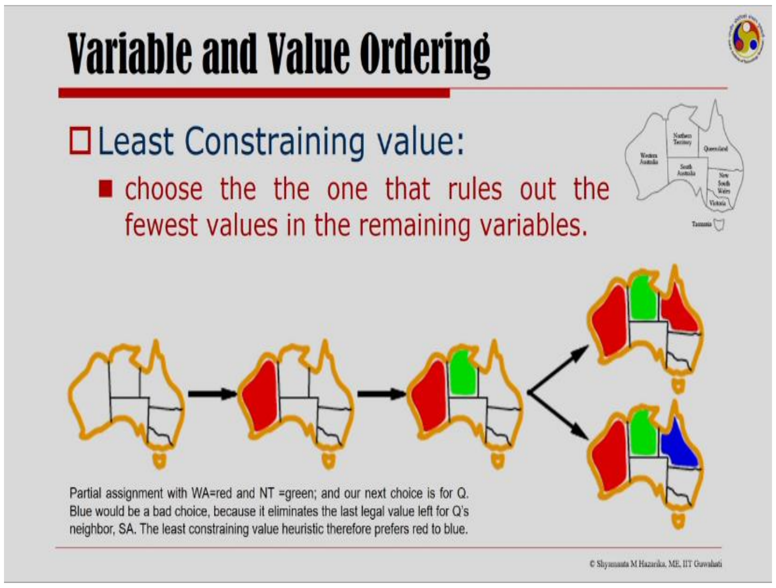
The least-constraining-value heuristic can be effective!

In general, the heuristic is trying to leave the maximum flexibility for subsequent variable assignments.

© Shyamanta M Hazarika, ME, IIT Guwahati

These 2 heuristics that we have discussed, actually when looking at which variable to select. But then, once a variable is selected, the algorithm must decide on the order in which to examine the values of the variable. This is where the least constraining value heuristic comes into play. The least constraining value heuristic allows us to choose the variable that rules out the fewest values in the remaining variables.

(Refer Slide Time: 32:37)



Variable and Value Ordering

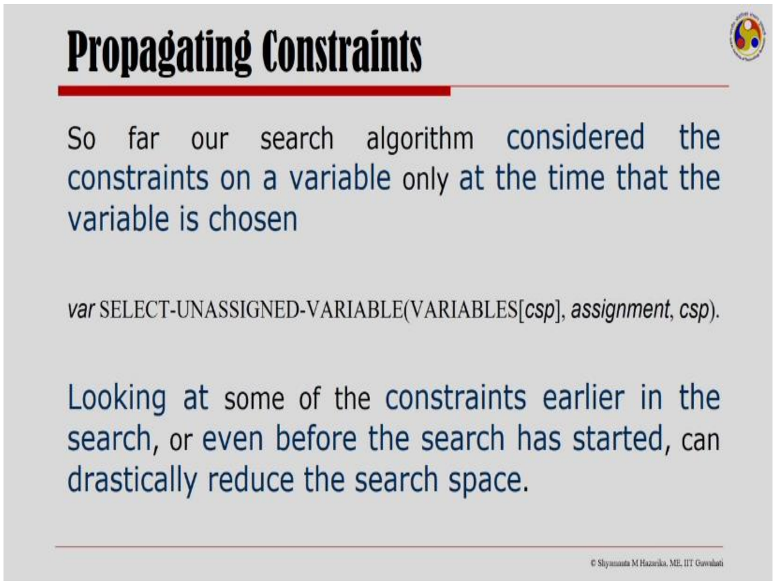
- Least Constraining value:
  - choose the the one that rules out the fewest values in the remaining variables.

Partial assignment with WA=red and NT =green; and our next choice is for Q. Blue would be a bad choice, because it eliminates the last legal value left for Q's neighbor, SA. The least constraining value heuristic therefore prefers red to blue.

© Shyamanta M Hazarika, ME, IIT Guwahati

Coming back to our map coloring example, let us see that after a partial assignment of red and green to WA and NT respectively, what is our choice for Q. We could now choose to color Q, either blue or red. But, we need to understand that blue is a bad choice. Because, it would then eliminate the only legal value possible for SA. So, the least constraining value heuristic prefers to color Q red, rather than coloring it blue.

(Refer Slide Time: 33:34)



Propagating Constraints

So far our search algorithm considered the constraints on a variable only at the time that the variable is chosen

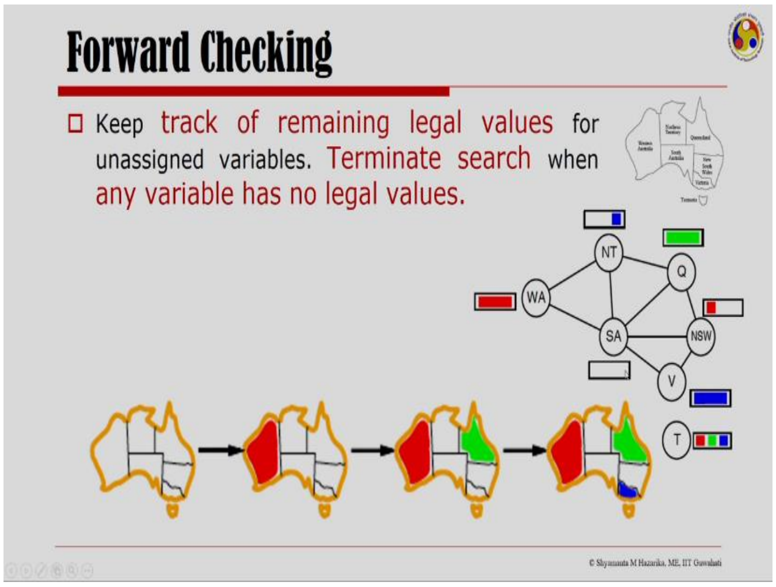
var SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*], *assignment*, *csp*).

Looking at some of the constraints earlier in the search, or even before the search has started, can drastically reduce the search space.

© Shyamanta M Hazarika, ME, IIT Guwahati

So far, our search algorithms that we have considered, the constraints on them only looked at the variable when it is chosen. It would be a good idea to look at some of the constraints earlier in the search or even before the search has started. This can then drastically reduce the search space. Let us look at one such idea called the forward checking.

(Refer Slide Time: 34:05)



Forward Checking

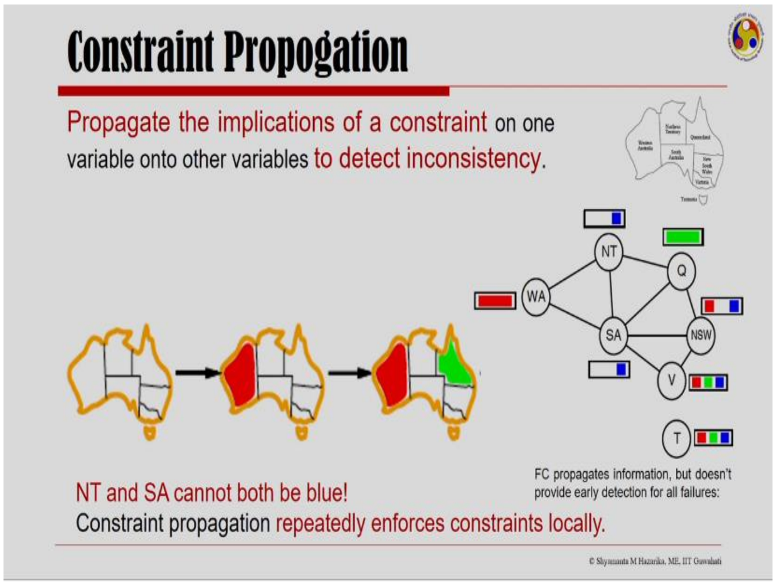
- Keep track of remaining legal values for unassigned variables. Terminate search when any variable has no legal values.

© Shyamanta M Hazarika, ME, IIT Guwahati

The forward checking keeps track of the remaining legal values for any unassigned variable. And then, it terminates search if any variable has no legal values to color from. So, let us look at this map coloring example to understand forward checking. Here, on your right is the constraint graph and we have 7 variables as we have been discussing. To start with, all these variables could be colored in any one of the 3 colors.

Now, let us say that we have colored WA red. When WA is colored red, then the variable NT and the variable SA, the only legal colors left for them is green, blue. And now, if we color Q as green, then the only legal values left for NT and SA is blue, whereas NS could be colored red and blue. After we have now decided to color V as blue, we see that we are left with no legal colors for SA. And the search must thus terminate.

(Refer Slide Time: 35:46)



Constraint Propogation

Propagate the implications of a constraint on one variable onto other variables to detect inconsistency.

NT and SA cannot both be blue!

Constraint propagation repeatedly enforces constraints locally.

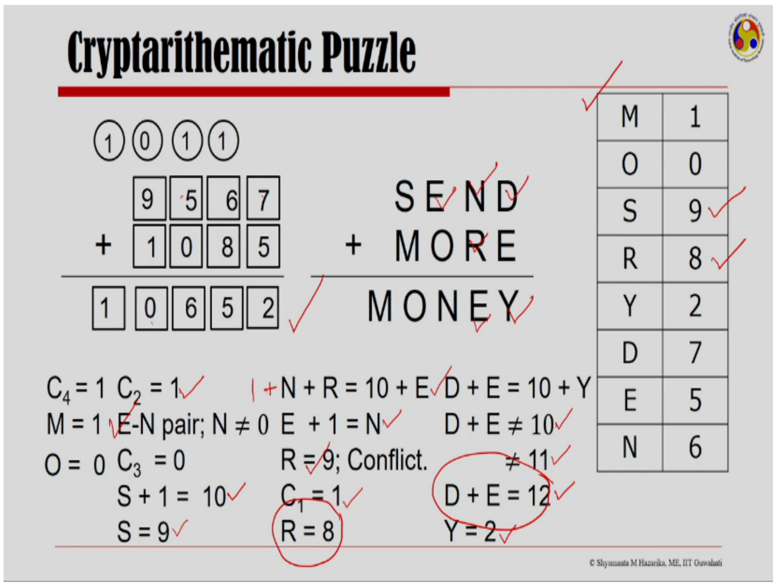
FC propagates information, but doesn't provide early detection for all failures:

© Shyamanta M Hazarika, ME, IIT Guwahati

The forward check propagates information, but does not provide early detection for all failures, as we will see when we look at this newer idea of constraint propagation. Constraint propagation propagates the implications of a constraint onto the variable, other variables to detect inconsistency. So, continuing with our map coloring example, we initially have all 3 colors for each of the 7 variables.

Now, after we color WA red, the only legal values left for NT and SA is green and blue. At that point, if we color Q green, the legal values that is possible for NT and SA is blue. But because we are propagating this information locally, from one variable onto other variables, we now immediately realize that NT and SA cannot both be blue together. And therefore, we know we will not be left with a solution, a path to a solution, if we carry forward our coloring this way. So, constraint propagation repeatedly enforce constraints locally and could detect failure early.

(Refer Slide Time: 37:22)



Cryptarithematic Puzzle

$$\begin{array}{ccccc} & 1 & 0 & 1 & 1 \\ & \boxed{9} & \boxed{5} & \boxed{6} & \boxed{7} \\ + & \boxed{1} & \boxed{0} & \boxed{8} & \boxed{5} \\ \hline \boxed{1} & \boxed{0} & \boxed{6} & \boxed{5} & \boxed{2} \end{array} \checkmark \qquad \begin{array}{ccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline M & O & N & E & Y \end{array}$$

M	1
O	0
S	9 ✓
R	8 ✓
Y	2
D	7
E	5
N	6

$C_4 = 1$ $C_2 = 1$ ✓ $1 + N + R = 10 + E$ ✓ $D + E = 10 + Y$

$M = 1$ E-N pair; $N \neq 0$ $E + 1 = N$ ✓ $D + E \neq 10$ ✓

$O = 0$ $C_3 = 0$ $R = 9$; Conflict. $\neq 11$ ✓

$S + 1 = 10$ ✓ $C_1 = 1$ ✓ $D + E = 12$ ✓

$S = 9$ ✓ $R = 8$ $Y = 2$ ✓

© Shyamanta M Hazarika, ME, IIT Guwahati

We will look at a cryptarithmatic puzzle in order to do this. So, here is a puzzle which says send plus more is equal to money. As I had told you earlier, I could think of placeholders for S, E and N, D; and I could have placeholders for M, O, R and E, on which I will put my guess of digits between 0 and 9. And then, I would have placeholders for M, O, N, E and Y. And then, as discussed earlier, we have carryovers 1, 2, 3 and 4, which we call C 1, C 2, C 3 and C 4.

And we create a placeholder table where the guess for M, O, S, R, Y, D, E, N, the 3 + 3, 6, 7, 8 variables that I have to identify is to be recorded. Now, the first thing that I must realize in this problem to be solved as constraint satisfaction is that, here, only search will not give me a solution. I have to do some reasoning along with the search. And when I am doing this, I have to think of taking the constraints forward.

When I am making a guess, I make a guess and then take this constraint forward and see if it violates or conflicts. If it violates or conflicts, then I should come back and do a search again for another value for that variable. So, to start with, we should realize that here C 4 should be 1. Because the carryover can be either 1 or 0. But a cryptarithmetic puzzle requires that the first place holder is not a 0. Therefore C 4 must be 1. And in that case, I have M = 1.

So, my first answer of getting to a constraint being satisfied is the fact that the constraint between C 4 and M gives me an answer that M is = 1. Given M is = 1, I update my table by putting 1 here and putting 1 again in a place where M is on ring. So, I put a 1 there. And next, I focus on looking for an answer to either the second column or the third column. So, if I focus on the second column, I could see that I have a constraint between C 3, S and O.

M already is known to me to be 1. And it is required that I have to carry forward a 1. So, the number here would be 10 or something more than 10. So, my initial guess could be that, let that be a 0. That means, O is a 0, in the sense that this is 10. So, I start with O being 0 and take O = 0 and put 0s in this placeholders, 0 and 0. Given O = 0, I then have a interesting relationship that I need to understand, third column here.

I have E here and I have N here. And this is a 0. And we should realize that E + N, there must be some relation. The E N is a pair. And I should realize that N cannot be 0 even if E N is a pair, because already 0 is taken by O. And the E N pair for E to be different than N, C 2 must be at least 1. Otherwise, if C 2 is 0, E + 0 + 0 is actually E. So, E and N will be the same number. That gives me a very interesting realization that the carry forward C 2 is 1.

Once I realize the carry forward C 2 is 1, I then take one step further and assume C 3 to be 0. Once I take C 3 to be 0, I have done S + 1 is = 10, because that is a carryover. So, S + 1 is = 10, gives me S is = 9. So, I have my third variable satisfied, S = 9. I update my variables S = 9 there. And then, now I have to look for R, E, N, D and Y. So, I focus on my fourth column and I realize that N + R must be E + 10.

Because, I need to carryover 1 there. So, this value that I get in my fourth column must not be E alone, but E + 10. Then only I can carryover C 2. So, I write that N + R is = 10 + E. And already from here, my third column, I know the fact that E + 1 is = N, given these 2, I can substitute E + 1 for N here. And then, I will have R = 9. But the moment I have R = 9, I realize that S is already 9. So, there is a conflict.

At that point of time, C 1 comes to my rescue. So, I say C 1 is actually not 0 but 1. If C 1 is 1, then I have C 1 + N + R is = 10 + E. So, this one is like plus 1 here. 1 + N + R is = 10 + E. In that case, R will come out as 8 and it does not have a conflict. So, I update this C 1 as 1 and I realize R is 8. So, put my 8 value there. Now, it is interesting to note that I have a relation between E and N already there as a pair.

But my last column also makes it clear that there is a relation between D and E. But only thing that I must realize is that the Y that I have is actually the relationship that D + E is = 10 + Y because I have a carry of 1 here. So, once we look at this, then I know that D + E is 10 + Y. But we know that Y cannot be 0, because 0 is already taken by O here. So, D + E is not equal to 10, neither is D + E = 11, because if D + E is = 11, Y is = 1.

But we already know M is = 1. So, there would be a conflict. So, the only possibility at this point of time is that D + E is = 12. If D + E is = 12, then Y is = 2. With Y = 2, I already have a carryover 1. So, that is pretty good. So, this is what I populate my placeholder with, Y = 2. So now, all I have is to find out the values for D, E and N. Now, it is interesting to note that I know E + 1 is N.

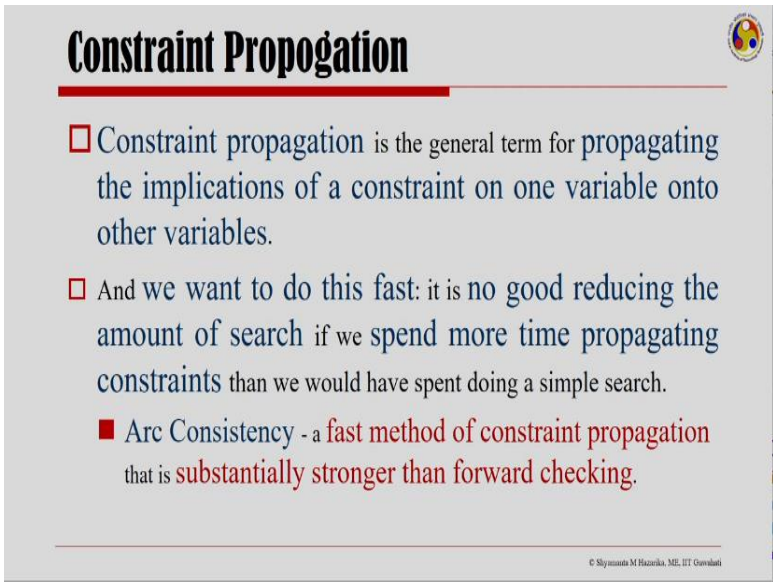
And from here, I know that D + E is 12. D + E could be 12 for digits 0 to 9 in a number of ways. One could be that I could start with D = 3. But if I start with D = 3, I would have E = 9. But S is already 9. So, D = 3 is not a possibility. Then I could start with D = 4. But if I take D = 4, U would be equal to 8, in which case it would conflict with R = 8. So, D is not 4. Then, I start with D = 5.

I could see that if I take D = 5, I will have E = 7. This is satisfied, no conflict. But then, let me check what happens if I have E = 7. E = 7 would mean that N is = 8, in which case it will conflict with R. Therefore, I have a realization that, actually D is not 5 but 7 and I have D = 7, in which case E is = 5 and N is = 6. So, I could populate these numbers there. And now, I have all the variables satisfied.

None of the constraints violated. And if you add them now, I have 7 + 5, 12; 2 carry over 1; 6 + 1, 7 + 8, 15; 5 carryover 1; 5 + 1, 6; and then, 9 + 1, 10; 0, carryover 1 and 1. So, this is the solution for the cryptarithmetic puzzle, send plus more equal to money, with these values for the different letters. What one needs to understand here is that, when this puzzle was being solved, we have done search and reasoning intermingled.

It is not that I was only searching and trying to satisfy constraints. Searching was done to satisfy constraints, but also what was being done was that, I was doing some amount of reasoning to understand what should be my next guess.

(Refer Slide Time: 48:50)



Constraint Propogation

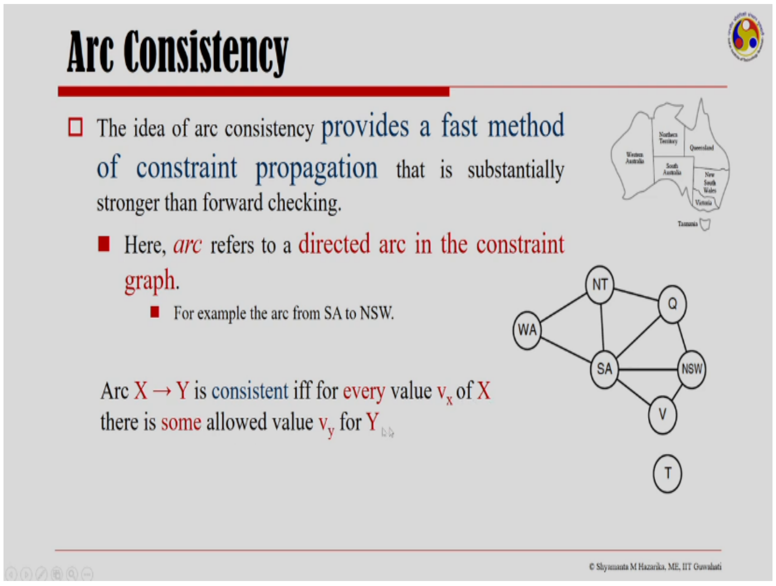
- Constraint propagation is the general term for propagating the implications of a constraint on one variable onto other variables.
- And we want to do this fast: it is no good reducing the amount of search if we spend more time propagating constraints than we would have spent doing a simple search.
  - Arc Consistency - a fast method of constraint propagation that is substantially stronger than forward checking.

© Shyamanta M Hazarika, ME, IIT Guwahati

So, these type of things is what is usually done in constraint propagation. Constraint propagation is the general term for propagating the implications of a constraint on one variable onto other variables. And we want to do this fast. So, it is no good. If I want to reduce the amount of search, but if we spend more time propagating constraints, then we would have spent doing a simple search.

There is no point of propagating the constraint. So, you want to have very fast propagation of constraints. A very fast method of constraint propagation, that is substantially stronger than the forward checking that we have discussed today is arc consistency.

(Refer Slide Time: 49:42)



Arc Consistency

- The idea of arc consistency provides a fast method of constraint propagation that is substantially stronger than forward checking.
  - Here, *arc* refers to a directed arc in the constraint graph.
    - For example the arc from SA to NSW.

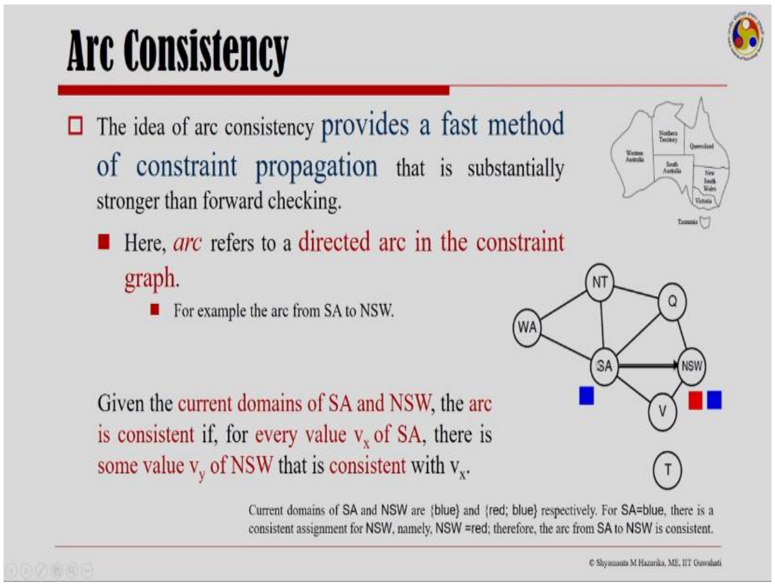
Arc $X \rightarrow Y$ is consistent iff for every value v_x of X there is some allowed value v_y for Y

Northern Territory
Queensland
Western Australia
South Australia
New South Wales
Victoria
Tasmania
NT
Q
WA
SA
NSW
V
T

© Shyamanta M Hazarika, ME, IIT Guwahati

Let us now try to understand the idea of how arc consistency provides a fast method of constraint propagation that is substantially stronger than forward checking. So here, an arc refers to a directed arc in the constraint graph. An arc X to Y is consistent. If for every value v x of X, there is some value v y for Y.

(Refer Slide Time: 50:16)



Arc Consistency

- The idea of arc consistency provides a fast method of constraint propagation that is substantially stronger than forward checking.
  - Here, *arc* refers to a directed arc in the constraint graph.
    - For example the arc from SA to NSW.

Given the current domains of SA and NSW, the arc is consistent if, for every value v_x of SA, there is some value v_y of NSW that is consistent with v_x.

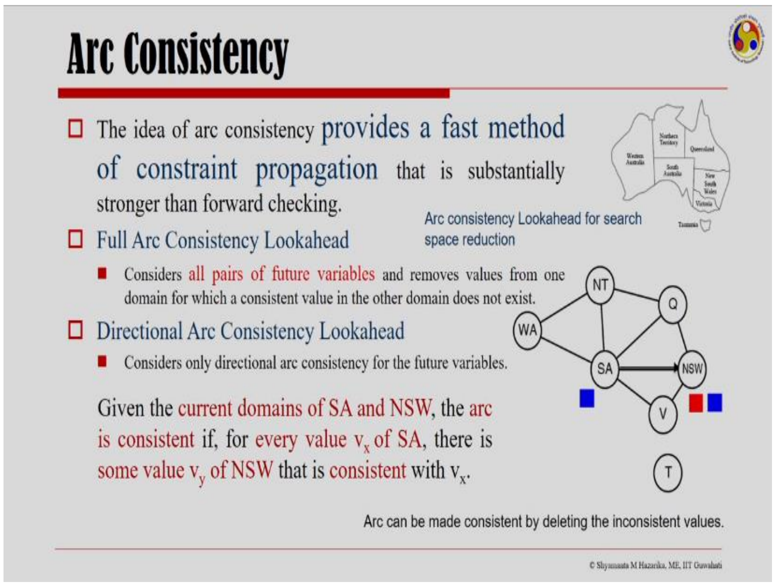
Current domains of SA and NSW are {blue} and {red; blue} respectively. For SA=blue, there is a consistent assignment for NSW, namely, NSW =red; therefore, the arc from SA to NSW is consistent.

© Shyamanta M Hazarika, ME, IIT Guwahati

So, let us say, in our example of the map coloring, the current domains of SA and NSW are blue and red blue respectively. Given the current domains of SA and NSW, we could see that there is a consistent assignment for SA = blue, I could always speak up NSW = red and I can therefore have a consistent assignment from SA to NSW. Whereas, if I am looking for arc consistency from NSW to SA, if I pick up red, I could have a legal value blue here. But if I pick up blue, I do not have a legal value in SA.

Therefore, the arc from SA to NSW is consistent, whereas NSW to SA is not. All that the arc consistency tells me is that I have an consistent arc from SA to NSW.

(Refer Slide Time: 51:31)



Arc Consistency

- The idea of arc consistency provides a fast method of constraint propagation that is substantially stronger than forward checking.
- Full Arc Consistency Lookahead
  - Considers all pairs of future variables and removes values from one domain for which a consistent value in the other domain does not exist.
- Directional Arc Consistency Lookahead
  - Considers only directional arc consistency for the future variables.

Given the current domains of SA and NSW, the arc is consistent if, for every value v_x of SA, there is some value v_y of NSW that is consistent with v_x.

Arc consistency Lookahead for search space reduction

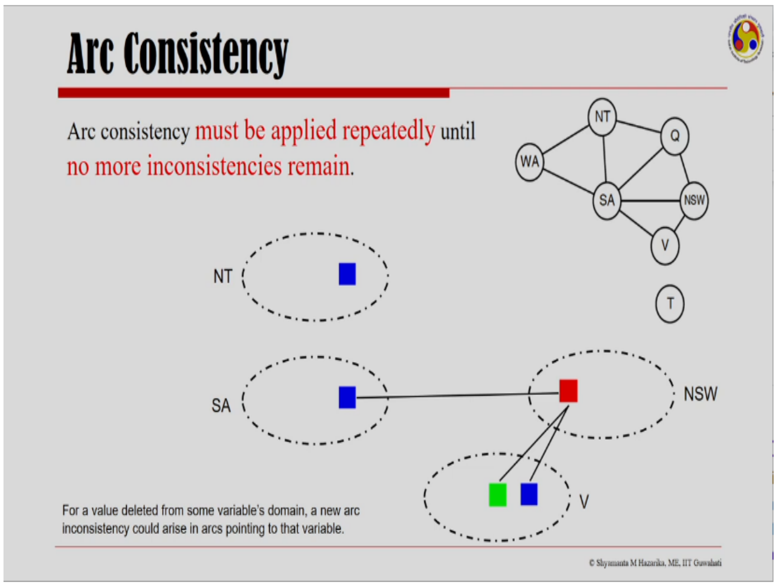
Arc can be made consistent by deleting the inconsistent values.

© Shyamanta M Hazarika, ME, IIT Guwahati

Arc consistency can be used as a lookahead to search for space reduction. And the idea is in 2 ways. Either we look for full arc consistency lookahead or we look for directed arc consistency lookahead. The full arc consistency lookahead is about considering all pairs of future variables and then removing values from one domain for which a consistent value in the other domain does not exist.

The directional arc consistency lookahead is about considering only directional arc for the future variable. Like in the example that we were discussing for the map coloring problem, SA to NSW the arc, is a directional arc consistency. Arcs that become inconsistent could be made consistent by deleting the inconsistent values. Like, if here, we think of making arc consistency, we can think of deleting certain value from NSW. That is the value blue and we would end up having consistency from NSW to SA.

(Refer Slide Time: 52:59)



Arc Consistency

Arc consistency must be applied repeatedly until no more inconsistencies remain.

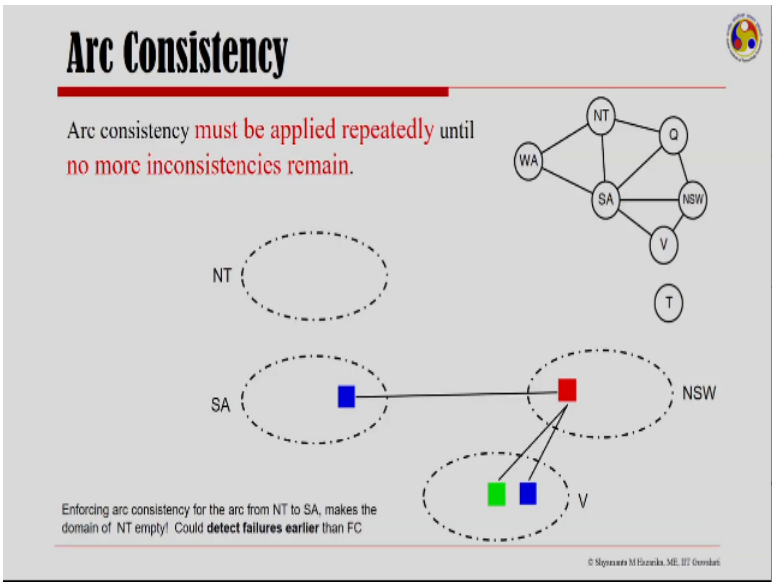
For a value deleted from some variable's domain, a new arc inconsistency could arise in arcs pointing to that variable.

But the problem is, if I delete a value from some variable's domain, a new inconsistency could arise in arcs pointing to that variable. So, arc consistency must be applied repeatedly until no more inconsistencies remain. This is what we will try to understand in this example here. So, on your right is the constraint graph. Let us assume the current domains that we have for the different variables NT, SA, V and NSW as shown.

So, as you can see, there is consistency from SA to NSW, because for a value from SA, I can pick up a value from NSW. But the other way, it is not correct. So, I can delete the blue value. And then, what I will have is a consistent arc between NSW and SA. But then, if you have realized this, earlier, the arc between V and NSW was consistent. But because of deletion of blue, if I now pick up red this side, I will have inconsistency.

Because, I will have no legal value for NSW available. And therefore, to maintain arc consistency, I need to delete red from here. And then, I have enforced arc consistency between NSW and V one more time. This has to be applied repeatedly until I have no more inconsistencies in the constraint graph. Now, if you look at these 2 domains of NT and SA, if I somehow apply arc consistency between them.

(Refer Slide Time: 55:01)



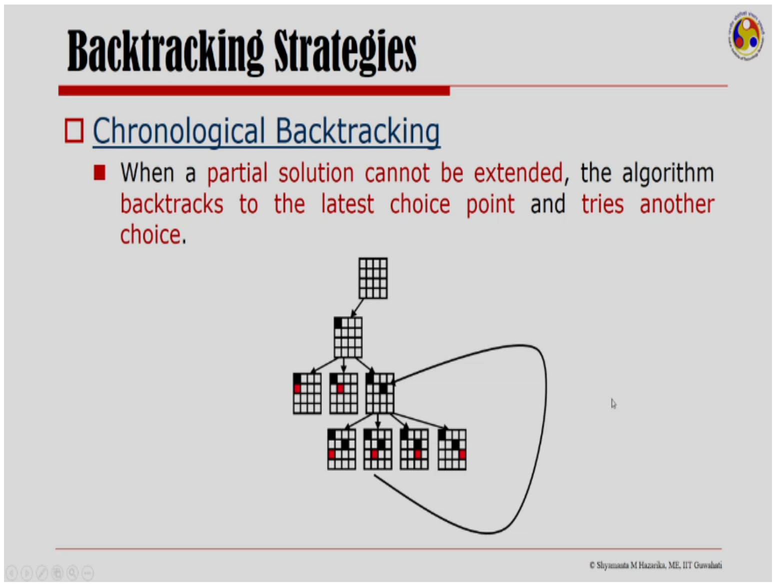
Arc Consistency

Arc consistency must be applied repeatedly until no more inconsistencies remain.

Enforcing arc consistency for the arc from NT to SA, makes the domain of NT empty! Could **detect failures earlier** than FC

Then I would need to delete blue from NT in order to have arc consistency between SA and NT. So, enforcing arc consistency from the arc from NT to SA would make the domain of NT empty. And this is how arc consistency could detect failures earlier than forward checking.

(Refer Slide Time: 55:34)



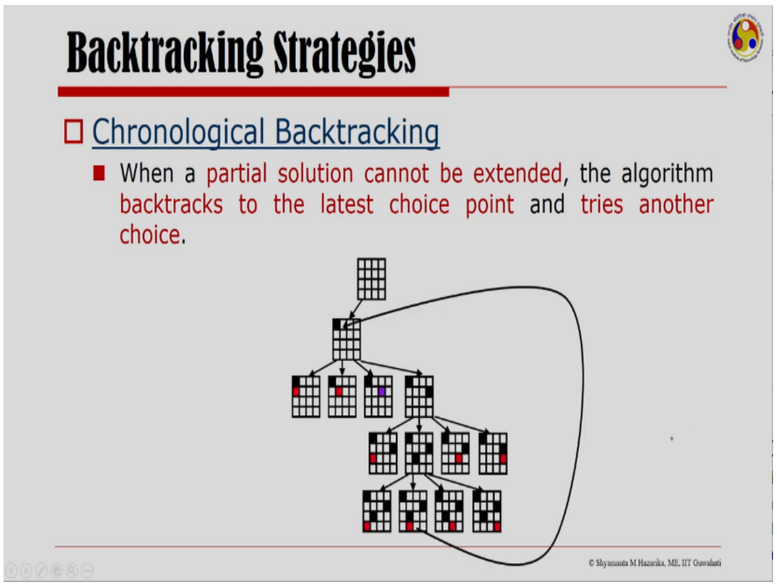
Backtracking Strategies

- Chronological Backtracking
  - When a partial solution cannot be extended, the algorithm backtracks to the latest choice point and tries another choice.

Now, we will focus our attention in a couple of backtracking strategies. We will work through 2 important backtrackings. The chronological backtracking is the simplest of them. It is about going back to the latest choice point and try another choice when a partial solution cannot be extended. So, here is what the chronological backtracking does. We look at this through our 4-queens problem.

So, we place our first queen. And then, the only possibility is here now, to place either in the third or in the fourth column in the second row. So, we place it here. And then, we realize that we are not left with any legal choice for our third queen. According to chronological backtracking, we would now need to backtrack from here to the point where we could get the second choice.

(Refer Slide Time: 57:01)



Backtracking Strategies

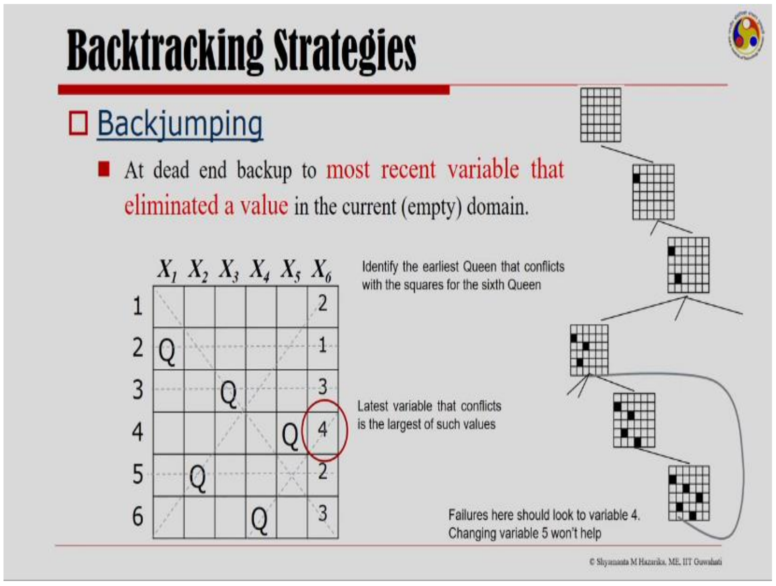
- Chronological Backtracking
  - When a partial solution cannot be extended, the algorithm backtracks to the latest choice point and tries another choice.

© Shyamanta M Hazarika, ME, IIT Guwahati

So, we would backtrack from here and we will look for using the other choice at this choice point which enables us to have a legal value for our third queen. But if we go one step forward, we realize that now it is no longer possible to have any legal position for our fourth queen. So, we would need to backtrack again. But that backtracking would be to the last choice point, which we have not yet exhausted.

So, this one is already exhausted. So, we will go back here. And that would be chronological backtracking. This is something that we have also looked at when I was talking to you about the backtracking algorithm.

(Refer Slide Time: 57:55)



Backtracking Strategies

- Backjumping
  - At dead end backup to most recent variable that eliminated a value in the current (empty) domain.

Identify the earliest Queen that conflicts with the squares for the sixth Queen

Latest variable that conflicts is the largest of such values

Failures here should look to variable 4. Changing variable 5 won't help

© Shyamanta M Hazarika, ME, IIT Guwahati

Now, we will look at one of a slight variance of this called the back jumping. So, the back jumping, whenever we arrive at a dead end, it backs up to the most recent variable that eliminated a value in the current domain. So, let us try to understand that with a 6-queens example. So, here is our 6 cross 6 grid. Let us place our first queen in row 2. Then, we place our second queen in row 5.

And now, the legal positions possible for our queen: We pick up one and place it in row 3. You should realize at this point that either I could place my fourth queen here or somewhere down there, here. So, I take that option and place my fourth queen here. Now, I am left with only one legal position for my fifth queen. I cannot place it here, because this queen will be in direct attack.

I cannot place it here, it will be attacked by this queen. I cannot place it here, because this will be then attacked by the third queen. I have only possibility of placing it is in this position and which is on the fourth row. Now, I am looking for a position for the sixth queen. If I go for the first row, that is not a legal position, because this queen will then attack. If I go for the second, this first queen will attack.

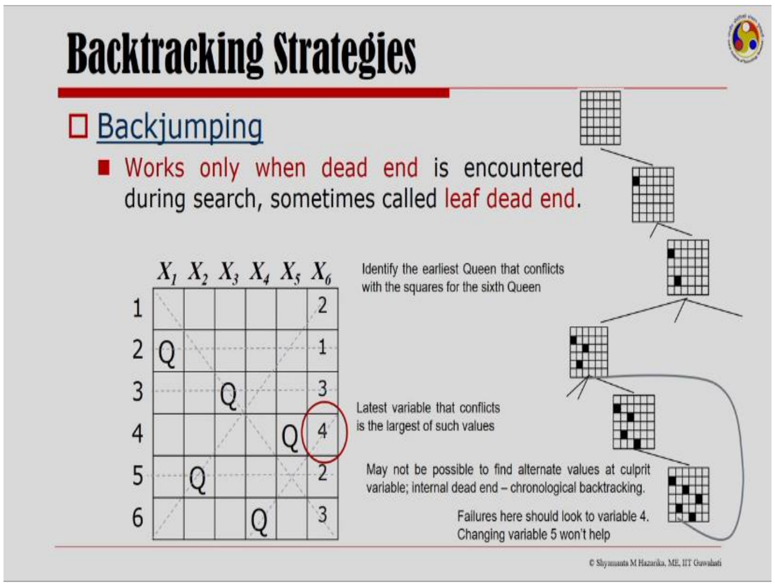
If I go for the third row, I already have a queen sitting on the third row. I cannot use the fourth row neither can I use the fifth or the sixth row. So, I now do not have a legal move for my sixth queen. I do not know which position to place, because I now have an empty domain.

Now, when I want to back up on this and go back, I would rather love to go to the most recent variable that eliminated a value in the current domain.
So, how do I go about doing that is, I try to look at the earliest queen that conflicts with the squares for the sixth queen. Like, let me see what I mean by this in the given example. So, if I want to now place something in this square, I would love to identify which is the queen that conflicts with the square for the sixth queen. So, if I do this, I realize that it is my second queen sitting there, which is actually not allowing a queen in the first position on the sixth column.

Next, if you look for the second position, you see it is the first queen. So, I try to identify the queens that do not allow me these positions and try to record each of the earliest ones. So, the third position is covered by the third queen. The fourth position is covered by the fourth queen sitting here. So, that is the earliest queen that conflicts with the square for the sixth queen. Then, for the fifth square, I have the second queen.

And for the final, I have the third queen. I record all these queens the earliest of them. And then, I take the largest of these values. The largest of these values actually gives me the latest variable that conflicts with making the current domain empty. So, I take this value which is 4. And now, I know that if I have to jump back, changing 5 would not help. I have to go back and look for making changes at the variable level 4. This is back jumping.

(Refer Slide Time: 01:02:45)



Backtracking Strategies

- Backjumping
  - Works only when dead end is encountered during search, sometimes called leaf dead end.

Identify the earliest Queen that conflicts with the squares for the sixth Queen

Latest variable that conflicts is the largest of such values

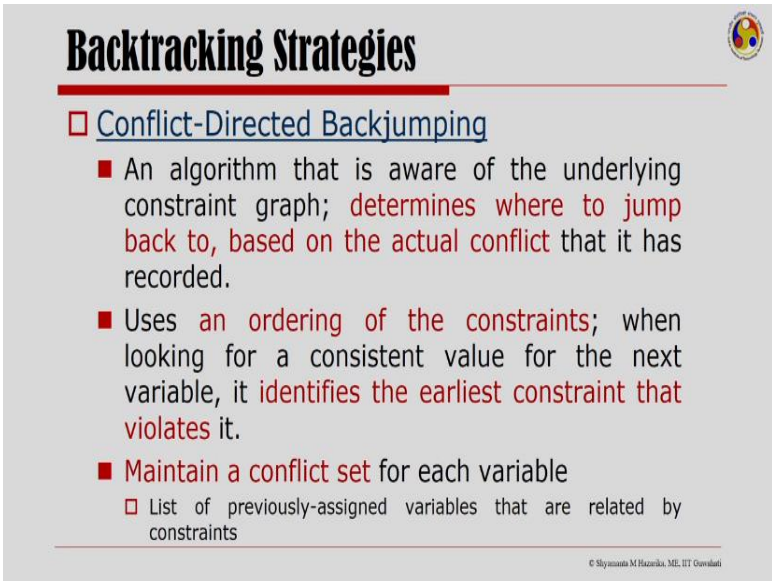
May not be possible to find alternate values at culprit variable; internal dead end – chronological backtracking.

Failures here should look to variable 4. Changing variable 5 won't help

© Shyamanta M Hazarika, ME, IIT Guwahati

Now, back jumping works only when the dead end is encountered during search. That is sometimes called the leaf dead end. If, by chance, if you go there and you are not able to get an alternate value at the culprit variable, in our case, the fourth variable here itself. That is called an internal dead end. Beyond that point, backjumping only does chronological backtracking.

(Refer Slide Time: 01:03:21)



Backtracking Strategies

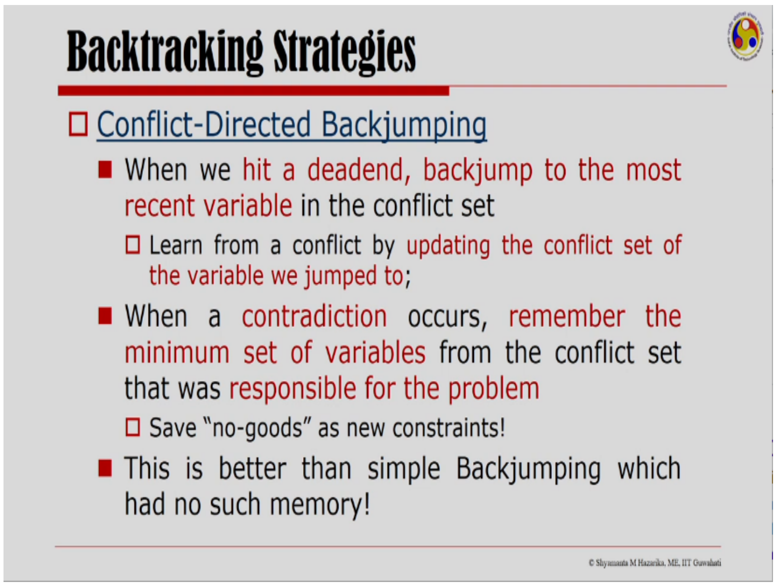
- Conflict-Directed Backjumping
  - An algorithm that is aware of the underlying constraint graph; determines where to jump back to, based on the actual conflict that it has recorded.
  - Uses an ordering of the constraints; when looking for a consistent value for the next variable, it identifies the earliest constraint that violates it.
  - Maintain a conflict set for each variable
    - List of previously-assigned variables that are related by constraints

© Shyamanta M Hazarika, ME, IIT Guwahati

Now, we will look at one more variant of back jumping called the conflict-directed back jumping. We will simply introduce the idea here and working through an example I leave it is an exercise for the readers. The conflict directed back jumping is an algorithm that is aware of the underlying constraint graph. It determines where to jump back to based on the actual conflict that is, it has recorded.

So, it uses an ordering on the constraints and identifies the earliest constraint that violates it. And it maintains a conflict set for each variable. That is, list of previously assigned variables that are related by constraint.

(Refer Slide Time: 01:04:16)



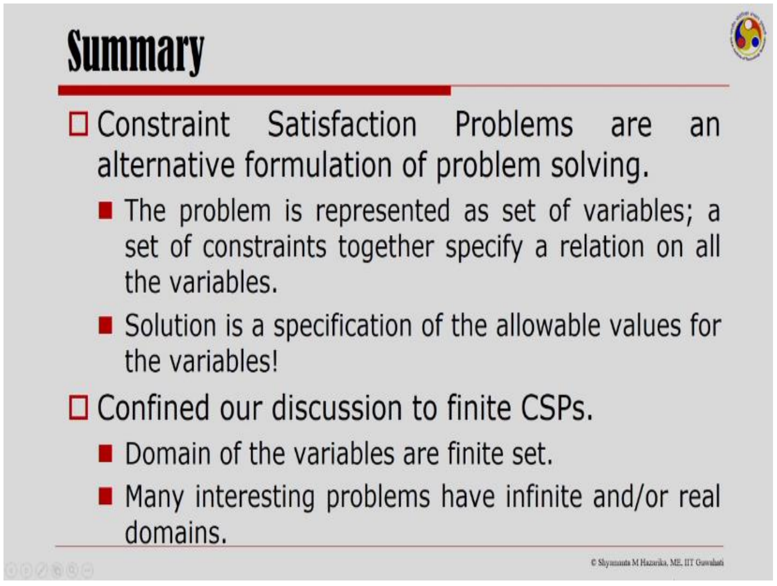
Backtracking Strategies

- Conflict-Directed Backjumping
  - When we hit a deadend, backjump to the most recent variable in the conflict set
    - Learn from a conflict by updating the conflict set of the variable we jumped to;
  - When a contradiction occurs, remember the minimum set of variables from the conflict set that was responsible for the problem
    - Save "no-goods" as new constraints!
  - This is better than simple Backjumping which had no such memory!

© Shyamanta M Hazarika, ME, IIT Guwahati

When we hit a dead end, it back jumps to the most recent variable in the conflict set. Interesting is that it learns from a conflict by updating the conflict set of the variable we jump to. When a contradiction occurs, conflict directed back jumping which remembers the minimum set of variables from the conflict set that was responsible for the problem and takes a decision. In fact, it says no-goods variables as new constraints. This is better than simple back jumping which had no such memory.

(Refer Slide Time: 01:05:01)



Summary

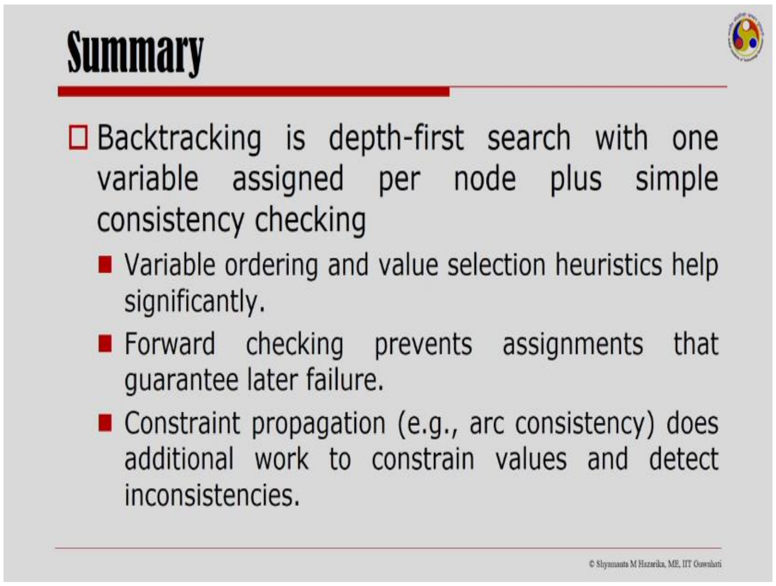
- Constraint Satisfaction Problems are an alternative formulation of problem solving.
  - The problem is represented as set of variables; a set of constraints together specify a relation on all the variables.
  - Solution is a specification of the allowable values for the variables!
- Confined our discussion to finite CSPs.
  - Domain of the variables are finite set.
  - Many interesting problems have infinite and/or real domains.

© Shyamanta M Hazarika, ME, IIT Guwahati

So, to conclude, we had looked at constraint satisfaction problems which are an alternate formulation of problem solving. Each problem is represented as set of variables, set of constraints which together specify a relation on all the variables. Solution is a specification of the allowable values for the variables. We have confined our discussion to only finite CSPs,

CSPs whose domains are finite set. There are many interesting problems that have infinite or real domains. We have not looked at them.

(Refer Slide Time: 01:05:47)



Summary

- Backtracking is depth-first search with one variable assigned per node plus simple consistency checking
  - Variable ordering and value selection heuristics help significantly.
  - Forward checking prevents assignments that guarantee later failure.
  - Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies.

© Shyamanta M Hazarika, ME, IIT Guwahati

We have looked at backtracking which is a depth-first search with one variable assigned per node. And then, we add to it consistency checking. Variable ordering and value selection heuristics, we have seen helped significantly. We have looked at the idea of forward checking that prevents assignments that guarantee later failure. And then, we have looked at constraint propagation including arc consistency that does additional work to constrain values and detect inconsistencies. Thank you very much.